Introdução a POO – Programação Orientada a Objetos JS

A programação de sistemas informatizados evoluiu para a programação orientada a objetos, portanto vale consignar que orientação a objetos é um paradigma de análise, projeto e programação de sistemas de software baseado na composição e interação entre diversas unidades de software ou abstrações chamadas de objetos.

A utilização do paradigma de programação orientada a objetos contribui para o desenvolvimento de softwares, códigos ou algoritmos mais organizados, atenuando algumas dificuldades enfrentadas no uso exclusivo do paradigma da programação estruturada ou procedural.

Na programação orientada a objetos as classes são elementos essenciais, portanto é importante entender que classe é um gabarito ou padrão de definição de objetos, isto é, se tudo é objeto corpóreo ou incorpóreo, abstrato ou concreto, as classes detém a função de definir os objetos que serão utilizados no software ou sistema orientado a objetos.

Em JavaScript um objeto é uma coleção de dados e/ou funcionalidades relacionadas (que geralmente consistem em diversas variáveis e funções — que são chamadas de propriedades e métodos quando estão dentro de objetos)[1], valendo ressaltar que quase tudo é objeto em JavaScript.

Para melhor entendimento apresenta-se um exemplo de classe no JavaScript:

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

    toString() {
        return this.getNome()+", nasceu em "+this.getNascimento();
    }

    getNome() {
        return this.nome;
    }

    getNascimento() {
        return this.nascimento;
    }
}
```

[1] Mozilla Foundation. © 2022. Disponível em: https://developer.mozilla.org/pt-BR/docs/Learn/JavaScript/Objects/Basics. Acesso em 23 de ago. de 2022.

Nos moldes que a classe Pessoa foi criada é necessário criar um objeto para utilização da inteligência algorítmica, como exemplifica-se:

```
<!DOCTYPE html>
<html>
<body>

<p id="ver1"></p>
<p id="ver2"></p>
<p id="ver3"></p>

<script>
class Pessoa {
   // constroi ou instancia o objeto com os valores
    constructor(nome, nascimento) {
      this.nome = nome; this.nascimento = nascimento;
    }
    // método que retorna os valores do objeto
    toString() {
        return this.getNome()+", nasceu em "+this.getNascimento();
    }
   // método que retorna o valor contido no atributo nome
    getNome() {
        return this.nome;
    }
    // método que retorna o valor contido no atributo nascimento
    getNascimento() {
        return this.nascimento;
    }
}
//cria o objeto ou referência objPessoa
const objPessoa = new Pessoa("Maria da Silva", "2000-01-01");
document.getElementById("ver1").innerHTML = objPessoa.getNome();
document.getElementById("ver2").innerHTML = objPessoa.getNascimento();
document.getElementById("ver3").innerHTML = objPessoa.toString();
</script>

</body>
</html>
```

Maria da Silva

2000-01-01

Maria da Silva, nasceu em 2000-01-01

Nota-se no exemplo exposto na imagem acima que o objeto ou referência objPessoa foi criado e instanciado com os valores correspondentes aos atributos nome e nascimento.

Vale ainda salientar que a classe Pessoa do exemplo acima detém dois atributos [nome e nascimento], três métodos [toString(), getNome() e getNascimento()] e o construtor.

Uma classe padrão teria ainda mais dois métodos para inserção autônoma dos valores em cada atributo, sendo esses métodos conhecidos como setters. Não obstante os métodos que permitem a obtenção dos valores armazenados nos atributos são conhecidos como métodos getters.

Para ampliar o entendimento apresenta-se um exemplo de uma classe "completa":

```
<!DOCTYPE html>
<html>
<body>

<p id="ver1"></p>
<p id="ver2"></p>
<p id="ver3"></p>

<script>
class Pessoa {
   // constroi ou instancia o objeto com os valores
    constructor(nome, nascimento) {
      this.nome = nome; this.nascimento = nascimento;
    }

    // método que retorna os valores do objeto
    toString() {
        return this.getNome()+", nasceu em "+this.getNascimento();
    }
   // método que retorna o valor contido no atributo nome
    getNome() {
        return this.nome;
    }
    // método que retorna o valor contido no atributo nascimento
    getNascimento() {
        return this.nascimento;
    }

    // método que define o valor do atributo nome
    setNome(nome) {
        this.nome=nome;
    }

    // método que define o valor do atributo nome
    setNascimento(nascimento) {
        this.nascimento=nascimento;
    }
}
//cria o objeto ou referência objPessoa
const objPessoa = new Pessoa();
objPessoa.setNome("Maria da Silva");
objPessoa.setNascimento("2000-01-01");
document.getElementById("ver1").innerHTML = objPessoa.getNome();
document.getElementById("ver2").innerHTML = objPessoa.getNascimento();
document.getElementById("ver3").innerHTML = objPessoa.toString();
</script>

</body>
</html>
```

Maria da Silva

2000-01-01

Maria da Silva, nasceu em 2000-01-01

No exemplo acima os valores são inseridos de forma autônoma em cada atributo,

por meio dos métodos setters[setNome() e setNascimento()].

Há como incluir um formulário no documento HTML, permitindo a inserção dos dados pelo usuário, como exemplifica-se:

```
<!-- identificar o formulário para uso no JS  --
>
<form id="formAula" >
<!-- identificar o input para uso no JS  -->
   <input id="v1" type="text">
<!-- identificar o input para uso no JS  -->
   <input id="v2" type="date">
   <button type="submit">OK</button>
</form>
```

Após a criação do formulário é importante criar tags identificadas para exibição do conteúdo, como demonstra-se:

```
<p id="ver1"></p> <!--Parágrofo identificado para exibir conteúdo-->
<p id="ver2"></p> <!--Parágrofo identificado para exibir conteúdo-->
<p id="ver3"></p> <!--Parágrofo identificado para exibir conteúdo-->
```

Nas tags <script> criar variáveis para receber os elementos do formulário, conforme código abaixo:

```
<script>
//variável formulário recebe o form com id = formAula
var formulario = document.getElementById('formAula');
//variável inputNome recebe o input com id = v1
var inputNome = document.getElementById('v1');
//variável inputNascimento recebe o input com id = v2
var inputNascimento = document.getElementById('v2');
```

Necessário condicionar a exibição do conteúdo a realização de clique no botão OK, portanto o método addEventListener da variável formulario (que recebeu o form), abrigará o código correspondente a exibição do conteúdo, conforme demonstra-se:

```
//início do método  addEventListener
formulario.addEventListener('submit', function(e) {
    //cria o objeto ou referência objPessoa
    const objPessoa = new Pessoa();
    /*define o conteúdo do atributo nome do objeto com o valor da variável
     inputNome que recebeu os dados do input de id = v1 */
    objPessoa.setNome(inputNome.value);
    /*define o conteúdo do atributo nascimento do objeto com o valor da variável
     inputNascimento que recebeu os dados do input de id = v2 */
    objPessoa.setNascimento(inputNascimento.value);
    //exibe na tag <p id="ver1"> o conteúdo do atributo nome do objeto objPessoa
    document.getElementById("ver1").innerHTML = objPessoa.getNome();
    //exibe na tag <p id="ver2"> o conteúdo do atributo nascimento do objeto objPessoa
    document.getElementById("ver2").innerHTML = objPessoa.getNascimento();
    //exibe na tag <p id="ver3"> o conteúdo do objeto objPessoa
    document.getElementById("ver3").innerHTML = objPessoa.toString();
    // impede o envio do formulário(form)
    e.preventDefault();
}); // fim do método addEventListener
```

Após o addEventListener terá o código da classe Pessoa, conforme exemplo a seguir:

```
class Pessoa {
    // constroi ou instancia o objeto com os valores
     constructor(nome, nascimento) {
       this.nome = nome; this.nascimento = nascimento;
     }

     // método que retorna os valores do objeto
     toString() {
         return this.getNome()+", nasceu em "+this.getNascimento();
     }
    // método que retorna o valor contido no atributo nome
     getNome() {
         return this.nome;
     }
     // método que retorna o valor contido no atributo nascimento
     getNascimento() {
         return this.nascimento;
     }

     // método que define o valor do atributo nome
     setNome(nome) {
         this.nome=nome;
     }

     // método que define o valor do atributo nome
     setNascimento(nascimento) {
         this.nascimento=nascimento;
     }
}

</script>
```

O codificação completa da exemplo demonstra a possibilidade de inserção dos

dados pelo usuário, bem como percepção dos valores inseridos:

```
<!DOCTYPE html>
<html>
<body>
<form id="formAula" >
   <input id="v1" type="text">
   <input id="v2" type="date">
   <button type="submit">OK</button>
</form>
<p id="ver1"></p>
<p id="ver2"></p>
<p id="ver3"></p>
<script>
var formulario = document.getElementById('formAula');
var inputNome = document.getElementById('v1');
var inputNascimento = document.getElementById('v2');
formulario.addEventListener('submit', function(e) {
    const objPessoa = new Pessoa();
    objPessoa.setNome(inputNome.value);
    objPessoa.setNascimento(inputNascimento.value);
    document.getElementById("ver1").innerHTML = objPessoa.getNome();
    document.getElementById("ver2").innerHTML = objPessoa.getNascimento();
    document.getElementById("ver3").innerHTML = objPessoa.toString();
    e.preventDefault();
});
class Pessoa {
    constructor(nome, nascimento) {
      this.nome = nome; this.nascimento = nascimento;
    }
    toString() {
        return this.getNome()+", nasceu em "+this.getNascimento();
    }
    getNome() {
        return this.nome;
    }
    getNascimento() {
        return this.nascimento;
    }
    setNome(nome) {
        this.nome=nome;
    }
    setNascimento(nascimento) {
        this.nascimento=nascimento;
    }
}
</script>
</body>
</html>
```

JOSE DOS SANTOS 02/02/2000 OK

JOSE DOS SANTOS

2000-02-02

JOSE DOS SANTOS, nasceu em 2000-02-02