STEP Javascript homework 017

1. Baseando-se no código de sala de aula que versa sobre **classes** no javascript, construa uma veterinaria que atende animais de vários tipos. Os animais podem ser cães, gatos, papagaios e lagartixas.*Todos esses animais emitem um som quando examinados, menos a lagartixa. Esse método deve exibir algo na página, p.ex. "Eu, cao, lati quando fui examinado"*. Opcionalmente, exiba infos adicionais (atributos, chamadas de método, etc) sobre a criação e sobre estados desses objetos na página (p.ex. "Um cao foi criado"). Em resumo, seu código deve criar vários animais, encaminhar esses animais para a veterinária, a veterinária deve atendêr um animal por vez através de um veterinário, e quando este animal for atendido, deve emitir um som.

Requerido: usar Herança (use *classes*, e opcionalmente, se for necessário, *prototypes* e o método *call*).

Requerido: todos os animais devem ter um método 'emiteSom' *que é herdado pela herança.*

Requerido: você deve ter um botão para criar cada tipo de animal (os atributos de cada animal, p.ex., nome, podem ser 'hard-coded' na hora da criação do animal, ou se quiser, faça input boxes para serem definidos pelo usuário quando criar o animal.

Requerido: ao criar um animal, ele deve ser inserido em uma lista que *pertence à veterinária*. É a partir dessa lista que um veterinário atende um animal de cada vez no código principal.

.

DICA: as classes são bem parecidas com o Java (se você já teve aulas sobre Java, não terá muita dificuldade em defini-las). Entretanto, olhe o código ao final do documento: há pequenas diferenças

Requerido em todos os exercícios: os scripts devem estar separados, na subpasta js (não vamos mais fazer scrips 'embebidos no html' mas importados pela tag *script* - veja no código ao final deste doc

==========
Entregue a pasta inteira (a pasta com o arquivo .html e a subpasta 'js' com o arquivo .js dentro)
==========

Segue um código desenvolvido em sala de aula

HTML

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <script src="./js/auloa017.js"></script>
    <title>Document</title>
</head>
<body>

</body>
</html>
```

JAVASCRIPT

```javascript
function Pessoa (primeiro, ultimo, interesses) {
    this.nome = {primeiro, ultimo};
    this.interesses = interesses;
}

// O prototype é uma maneira de generalizar métodos
// para vários objetos diferentes
// MÉTODO DA CLASSE PESSOA
Pessoa.prototype.saudacao = function () {
    document.write("Ola! Eu sou o " + this.nome.primeiro + ".<br>")
}

// Herança prototípica: Um professor é uma pessoa
// Aqui, prototypes e construtores são copiados, não executados
function Professor (primeiro, ultimo, interesses, assunto) {
    Pessoa.call(this, primeiro, ultimo, interesses);
    this.assunto = assunto;
}

// CÓPIA DE PROTOTYPE
Professor.prototype = Object.create(Pessoa.prototype)
```

```
//Professor.prototype.constructor = Pessoa

// Professor "herda" de Pessoa. O método 'call' chama o construtor
// da outra função construtora, sendo que a assinatura tem que ser
// respeitada E o primeiro parametro é adicional e obrigatoriamente 'this'
professor = new Professor("dudu","xxxx", ["javascript", "dinheiro"])

// MÉTODO EXCLUSIVO DO PROFESSOR
Professor.prototype.darNota = function () {
    console.log("estou lançando notas")
}

let pessoa = new Pessoa(
    "Arthur",
    "Souza",
    ["Futebol", "Cozinha"]
)

// teste
pessoa.saudacao()

// teste
professor.saudacao()

// teste
professor.darNota()

// inspeciona os atributos
console.log(Object.getOwnPropertyNames(professor))
console.log(Object.getOwnPropertyNames(pessoa))


// SISTEMA DE HERANÇA A PARTIR DO ES6
// Cao é um Animal
// logo, tenho uma herança onde Animal é uma classe pai
// e Cao um classe filha
class Animal {
```

```
    constructor(numPatas) {
        this.numPatas = numPatas
    }
    andar() {
        console.log("andei")
    }
    exibeNumPatas() {
        console.log("eu tenho " + this.numPatas + " patas.")
    }
}

// animal = new Animal(4)
// animal.andar()

// Cao vai herdar de Animal
class Cao extends Animal {
    constructor(numpatas) {
        super(numpatas)
    }

}

// teste
cao = new Cao(4)
cao.andar()
cao.exibeNumPatas()
```