Seção 07 Javascript Classes – Programação Orientada a Objetos POO

* Aula 80 Criando Classes

class Pessoa {

    constructor(nome, sobrenome) {

        this.nome = nome;

        this.sobrenome = sobrenome;

    }

    falar() {

        console.log(`${this.nome} está falando.`);

    }

    comer() {

        console.log(`${this.nome} está comendo.`);

    }

}

const p1 = new Pessoa('Luiz','Miranda');

console.log(p1);

modelo de para criar uma classe de objeto, nesse caso a class Pessoa.

Nesse modelo os métodos que foram criados já foram altomaticamente lincados no prototypes dessa classe criada, não tendo a complexidade a mais de uma função construtora.

O javascript otimiza o seu código.

function Pessoa2(nome, sobrenome) {

    this.nome = nome;

    this.sobrenome = sobrenome;

}

Pessoa2.prototype.falar = function () {

    console.log(`${this.nome} está falando.`);

}

Esse seria o modelo para criar em function constructor da mesma classe

* Aula 81 Getters e Setters

class Carro {

    constructor(nome) {

        this.carro = nome;

        this.velocidade = 0;

    }

    acelerar() {

        if(this.velocidade >= 100) return;

        this.velocidade++;

    }

    freiar() {

        if(this.velocidade <= 0) return;

        this.velocidade--;

    }

}

const c1 = new Carro('Fusca');

for (let i = 0; i < 200; i++) {

    c1.acelerar();

}

c1.velocidade = 1500;

console.log(c1);

nesse caso foi criado a classe Carro, que foi instanciada a c1, contendo um método acelerar que so pode ir ate velocidade = 100.

Contudo foi feito uma passagem de parâmetro que esta dizendo que essa c1.velocidade recebera 1500, quebrando assim nosso código.

Para resolver isso você pode utilizar uma propriedade privada.

const \_velocidade = Symbol('velocidade');

essa propriedade a cada Symbol e diferente um do outro, mesmo que tenha o mesmo nome.

constructor(nome) {

        this.carro = nome;

        this[\_velocidade] = 0;

    }

Então alterei o constructor de velocidade para esse novo symbol.

set velocidade(valor) {

        if(typeof valor !== 'number') return;

        if(valor >= 100 || valor <= 0) return;

        this[\_velocidade] = valor;

    }

    get velocidade() {

        return this[\_velocidade];

    }

Agora temos o get e set.

c1.velocidade        // GET

c1.velocidade = 29   // SET

console.log(c1);

no caso de get e set, não são funções logo não são acessados por ()

eles são acessados tipo atributos, pelo ‘ . ’, isso para os get e set, os métodos continuam sendo acessados por ‘ () ’.

quando colocamos o ‘ = ’, estamos dizendo que isso e um set

class Pessoa {

    constructor(nome, sobrenome) {

        this.nome = nome;

        this.sobrenome = sobrenome;

    }

    get nomeCompleto() {

        return `${this.nome} ${this.sobrenome}`;

    }

    set nomeCompleto(valor) {

        valor = valor.split(' ');

        this.nome = valor.shift();

        this.sobrenome = valor.join(' ');

    }

}

const p1 = new Pessoa('Luiz', 'Miranda');

p1.nomeCompleto = 'Luiz Miranda Oliveira'; // SET

console.log(p1.nome); // GET

console.log(p1.sobrenome); // GET

console.log(p1.nomeCompleto); // GET

modelo para uso do get e set

* Aula 82 Herança com classes

class DispositivoEletronico {

    constructor(nome) {

        this.nome = nome;

        this.ligado = false;

    }

    ligar() {

        if(this.ligado) {

            console.log(`${this.nome} já ligado`);

            return;

        }

        this.ligado = true;

    }

    desligar() {

        if(!this.ligado) {

            console.log(`${this.nome} já desligado`);

            return;

        }

        this.ligado = false;

    }

}

class Smartphone extends DispositivoEletronico {

    constructor(nome, cor) {

        super(nome)

        this.cor = cor;

    }

}

const s1 = new Smartphone('Iphone', 'Preto');

console.log(s1);

Criamos a classe DispoitivoEletronico e também criamos a classe Smartphone.

Essa classe Smartphone esta extendendo de DispositivoEletronico, tudo que tem DispositivoEletronico ira parar em Smartphone.

Como estamos extendo essa classe que já tem os seus atributos, temos que em seu construtor indicar de onde vem os atributos.

Para isso usamos o super()

constructor(nome, cor) {

        super(nome)

        this.cor = cor;

    }

Agora quando instanciamos um objeto, nesse caso o atributo nome ira para a classe que foi extendida e o atributo cor para a classe nova criada.

class Tablet extends DispositivoEletronico {

    constructor(nome, temWifi)  {

        super(nome);

        this.temWifi = temWifi;

    }

    ligar() {

        console.log('Olha, você alterou o método ligar');

    }

 falaOi() {

        console.log('Oi');

    }

}

const s1 = new Smartphone('Samsung', 'Preto', 'Galaxy S10');

const t1 = new Tablet('iPad', true);

s1.ligar(); // Samsung ligado.

s1.ligar(); // Samsung já ligado

t1.ligar(); // Olha, você alterou o método ligar

Aqui foi criado uma nova classe Tablet que extende de DispositivoEletronico, com seu atributo de temWifi para ele e enviando a super classe o atributo de nome.

Essa classe contem um método ligar(), que também tem na super classe, quando isso acontece, ele ira procurar o método na classe criada para depois procurar e suas superclasses.

* Aula 83 Metodos de instancia e estáticos

class ControleRemoto {

    constructor(tv) {

        this.tv = tv;

        this.volume = 0;

    }

    // Método de Instância

    aumentarVolume() {

        this.volume += 2;

    }

    diminuirVolume() {

        this.volume -= 2;

    }

    // Método Estático

    static trocaPilha() {

        console.log('Ok, vou trocar');

    }

}

const controle1 = new ControleRemoto('LG');

controle1.aumentarVolume();

console.log(controle1);

ControleRemoto.trocaPilha();

Método estáticos, so poderá ser acessado pela sua classe.

Esse método estático e referente a classe

Já o método de instancia e referente a istancia em si e tem acesso aos dados da classe porque foi instanciada.

Os métodos estáticos não tem acesso ao dados da instancia, somente os métodos que são instanciados

* Aula 84 Usando Classes (validando CPF)

// 705.484.450-52

// 070.987.720-03

class validaCpf {

    // constructor(cpfEnviado) {

    //     Object.defineProperty(this, 'cpfLimpo', {

    //         writable: false,

    //         enumerable: true,

    //         configurable: true,

    //         value: cpfEnviado.replace(/\D+/g, '')

    //     })

    // }

    constructor(cpfEnviado) {

        this.cpfLimpo = cpfEnviado.replace(/\D+/g, '')

    }

    geraNovoCpf() {

        const cpfSemDigito = this.cpfLimpo.slice(0, -2);

        const digito1 = validaCpf.geraDigito(cpfSemDigito);

        const digito2 = validaCpf.geraDigito(cpfSemDigito + digito1);

        this.novoCpf = cpfSemDigito + digito1 + digito2;

    }

    static geraDigito(cpfSemDigitos) {

        let total = 0;

        let reverso = cpfSemDigitos.length + 1;

        for(let stringNumerica of cpfSemDigitos) {

            total += reverso \* Number(stringNumerica);

            reverso--;

        }

        const digito = 11 -(total % 11);

        return digito <= 9 ? String(digito) : '0';

    }

    esequencia() {

        return this.cpfLimpo.charAt(0).repeat(this.cpfLimpo.length) === this.cpfLimpo;

    }

    valida() {

        if(!this.cpfLimpo) return false;

        if(typeof this.cpfLimpo !== 'string') return false;

        if(this.cpfLimpo.length !== 11) return false;

        if(this.esequencia()) return false;

        this.geraNovoCpf();

        return this.novoCpf === this.cpfLimpo;

    }

}

let validacpf = new validaCpf('13459539763');

// validacpf = new validaCpf('999.999.999-99');

if(validacpf.valida()) {

    console.log('CPF Válido');

} else {

    console.log('CPF Inválido');

}

* Aula 85 Validando um formulário (usando classes)