escola britânica de artes criativas & tecnologia

Engenheiro Front-End

Orientação a Objetos com JavaScript

Abstração

A programação orientada a objetos é composta pelos pilares: **herança, polimorfismo e encapsulamento.**

Através da programação orientada a objetos (POO) podemos representar os itens do mundo real na programação.

Pense em um carro, ele é composto por rodas, pneus, volante, portas e outros itens. Essa ideia que temos de um carro é uma abstração.
Um carro é composto por muito mais itens, temos a parte elétrica, mecânica, em carros mais modernos temos a programação presente.

Embora um carro seja composto por diversas partes não é necessário conhecer os seus detalhes para dirigir um, toda essa complexidade é abstraída para um objeto com rodas.

Objetos

Em outras linguagens de programação possuímos classes. Uma classe é um código que será utilizado para criar os objetos.

Por exemplo, uma classe carro é utilizada para criar um objeto do tipo carro.

Uma classe será composta por atributos e métodos.

Atributos são as características que formam o objeto, no exemplo do carro temos como características a cor, modelo, marca, entre outros.

Já **os métodos** são as ações que o objeto é capaz de executar, como acelerar e frear.

No JavaScript não possuímos nativamente as classes, mas temos funções construtoras que possuem o mesmo papel, construir objetos.

```
function Carro() {}
```

Objetos

É uma função normal, por convenção utilizamos a letra maiúscula na primeira letra do nome da função.

Os atributos numa função construtora são passados como argumentos da função e armazenados no próprio contexto do objeto, utilizando a palavra reservada this:

```
function Carro(nomeModelo) { // nomeModelo = argumento
    this.modelo = nomeModelo; // this.modelo = atributo
    this.acelerar = function() { // this.acelerar = método
        console.log("vruum");
    }
}
```

Objetos

A diferença de um método com um atributo, é que **o método tem como valor uma função.**

Para criar um objeto a partir de uma função construtora utilizamos a palavra reservada new, ele irá retornar o objeto construído:

```
const carroDaAna = new Carro("HB20")
```

Para acessar os atributos e métodos do objeto criado utilizamos a **notação ponto**, que nada mais que um ponto com o nome do atributo, depois do nome do objeto:

```
const carroDaAna = new Carro("HB20")
console.log(carroDaAna.modelo) // HB20
carroDaAna.acelerar() // vruum
```

Herança

A herança na POO nos permite criar funções construtoras que são derivadas de outras funções.

No exemplo do carro, para sermos mais genéricos, podemos pensar num veículo, um veículo pode ser aéreo ou terrestre, logo nossa função construtora deveria ser herdeira de uma função mais genérica, ser uma função filha de Veículo.

Herança

Para fazer com que uma função construtora herde de outra devemos chamar a **"função construtora mãe",** para isso utilizamos um método que existe a todas as funções, chamado call.

```
function FuncaoMae(a, b) { }
function FuncaoFilha() {
    FuncaoMae.call(this, "argumento A", "argumento B");
}
```

O **método call** recebe alguns argumentos, o primeiro é **o this atual**, os outros são os argumentos existentes na função mãe.

Polimorfismo

Polimorfismo é a capacidade de se realizar a mesma tarefa de maneiras diferentes.

Pensando num veículo, um carro pode ir de 0 à 100km/h em um tempo diferente de uma motocicleta, ambas abstrações estão realizando a mesma tarefa de uma maneira diferente, com uma implementação diferente.

Polimorfismo

Para aplicar o polimorfismo no JavaScript basta sobrescrevermos o método:

```
function Veiculo() {
    this.segundosTempoAte100Km() { return 20 } // implementação
padrão
}
function Carro() {
   Veiculo.call(this);
    this.segundosTempoAte100Km() { return 15 }
}
function Aviacao() {
    Veiculo.call(this)
    this.segundosTempoAte100Km() { return 5 }
}
function Motocicleta() {
    Veiculo.call(this)
     this.segundosTempoAte100Km() { return 10 }
}
```

Encapsulamento

O encapsulamento no permite ter maior controle sobre as os atributos de um objeto, podem até mesmo validar os valores que o atributo irá receber.

Por padrão os atributos utilizando this. NOME_ATRIBUTO são públicos, podem ser atribuídos ou lidos de qualquer maneira.

Encapsulamento

Para aplicar um controle de acesso às propriedades precisamos muda-las para variáveis e criar métodos que alterem e acessem esta variável.

```
function Funcionario() {
    let _salario = 0;
    this.retornaSalario = function() {
    return _salario;
}
this.atribuiSalario = function(valor) {
    _salario = valor;
    }
}
const funcionarioA = new Funcionario();
```

Encapsulamento

Acessamos o atributo e fazemos a atribuição de valor através de métodos:

```
funcionarioA.atribuiSalario(1000)
funcionarioA.retornaSalario() // 1000
```

O _ antes do nome da variável é uma convenção utilizada para quando o atributo é privado.

Dessa maneira podemos até adicionar uma validação na atribuição do valor, verificando se o número é positivo e se o tipo do dado é **number.**

Encapsulamento

Os métodos quando possuem a tarefa de retornar e atribuir valor a um atributo, são chamados de getters e setters, respectivamente.

Dessa maneira o nome correto para os método seria:
getSalario e **setSalario**, também podemos formatar o retorno do atributo:

```
getSalario() {
    If (_salario === undefined) return "O salário ainda não foi
atribuído";
    return `O salário é de R$ ${_salario}`;
}
```