6. Classes

Como em diversas outras linguagens, podemos declarar classes em JavaScript como uma forma de gerar um padrão para nossos objetos. Um objeto gerado a partir de uma classe é chamado de "instância".

Classes possuem construtores, que são funções especiais que servem para gerar um novo objeto. Um padrão utilizado dentro da programação é nomear classes utilizando a primeira letra maiúscula, assim como no PascalCase.

Para declarar uma classe, utilizamos a seguinte sintaxe:

```
class Classe {
        atributo;
        #atributoPrivado;

        function funcao(param) { /* implementação */ }

        function #funcaoPrivada()  { /* implementação */ }
}
```

E para gerar uma instância de uma classe, podemos simplesmente chamar a palavra reservada "new", seguida do nome da classe:

```
const item = new Classe();
```

Caso desejemos modificar o comportamento padrão do nosso construtor para, por exemplo, inicializar uma nova instância com uma valor passado via código, podemos declarar um construtor da seguinte forma:

```
class Classe {
        atributo;

        constructor(atributo) {
                this.atributo = atributo;
        }
}
```

Atributos Privados

Apesar da semelhança com outras linguagens, em JavaScript existem apenas atributos públicos e privados. Atributos públicos podem ser acessados diretamente através das sintaxes mostradas nas aulas de Objetos, já atributos privados, podem ser acessados somente internamente pela classe.

Atributos Estáticos

Assim como em outras linguagens, JavaScript também possui atributos estáticos, esses valores e funções podem ser utilizados sem a necessidade de criar uma instância de uma classe, porém, não podem ser acessadas através de instâncias. Um bom exemplo disso são as funções associadas à classe Math, nativas do JavaScript:

```
Math.random();
```

Como pudemos ver, não foi necessário instanciar um novo objeto "Math", para acessar a função "random".

Geralmente, utilizamos o padrão de atributos e funções estáticos para classes utilitárias, onde temos uma classe que não será instanciada, servindo apenas como um agrupamento de funções comuns à um contexto.

Herança

Por fim, podemos criar classes mais genéricas, que podem ser utilizadas por outras classes que irão estendê-las, vejamos os seguintes exemplos:

```
class Animal {
  constructor(nome) {
    this.nome = nome;
  }

  fazerBarulho() {
    console.log(`${this.nome} faz algum barulho.`);
  }
}

// Definição da classe filha Cachorro que herda de Animal
class Cachorro extends Animal {
  constructor(nome, raca) {
    // Chamada para o construtor da classe pai usando super()
    super(nome);
    this.raca = raca;
  }
```

```
  // Sobrescrita da função da classe pai
  fazerBarulho() {
    console.log(`${this.nome} (raça: ${this.raca}) late.`);
  }

  // Nova função da classe filha
  correr() {
    console.log(`${this.nome} está correndo.`);
  }
}
```

Podemos ver que é possível sobrescrever funções de classes filhas simplesmente escrevendo uma nova implementação, ao chamar o método construtor da classe pai através da função "super", todos os atributos e funções são transferidas para a classe filha.

Desafios

Fáceis

Primeiro:
Crie a classe veículo com os seguintes atributos:

- Marca
- Modelo
- Ano
- Cor
  E as funções:
- Buzinar

Segundo:
Crie a classe "Retangulo" com a função estática "calcularArea"

Médios

Primeiro:
Crie a classe "ContaBancaria" com os seguintes atributos:

- Número
- Saldo
  E as funções:
- Sacar

- Depositar
- Ver Saldo

Segundo:
Crie uma classe Pessoa com os seguintes atributos:

- Nome
- Idade
- Sexo
  Crie funções para definir e visualizar cada um dos atributos.

Difícil

Primeiro:

1. Crie a classe Animal com a função "falar"
2. Crie a classe Cachorro que estende a classe Animal
3. Sobrescreva a função "falar" de cachorro para retornar "latido"