ORIENTAÇÃO A OBJETOS

O QUE SÃO PARADIGMAS DE PROGRAMAÇÃO?

Os paradigmas de programação são **abordagens** ou estilos que definem **a forma** como os programas de computador são **desenvolvidos** e **estruturados**.

O QUE SÃO PARADIGMAS DE PROGRAMAÇÃO?

Eles oferecem uma visão específica de como os problemas devem ser **abordados** e **resolvidos** em termos de **organização do código**, **controle de fluxo**, e **manipulação de dados**. Diferentes paradigmas servem para diferentes propósitos, facilitando a implementação de soluções dependendo da **natureza do problema**.

IMPERATIVO

O **paradigma imperativo** foca em definir **instruções claras** que dizem ao computador **exatamente como** executar cada etapa. O desenvolvedor especifica passo a passo o fluxo de controle, como **laços** e **condicionais**.

ESTRUTURADO

O **paradigma estruturado** é uma variação do imperativo, que se preocupa em organizar o código em **blocos lógicos**. Ele introduz conceitos como **funções** (ou sub-rotinas) para dividir tarefas e evitar o uso de estruturas não estruturadas como o *goto*, que torna o código confuso e difícil de manter.

ORIENTADO A OBJETOS

A **programação orientada a objetos** (OO) organiza o código em **objetos**, sendo **instâncias de classes**. Cada objeto contém **atributos** e **métodos**. O paradigma OO é útil para **modelar** situações do mundo real, criando uma hierarquia de objetos que interagem entre si.

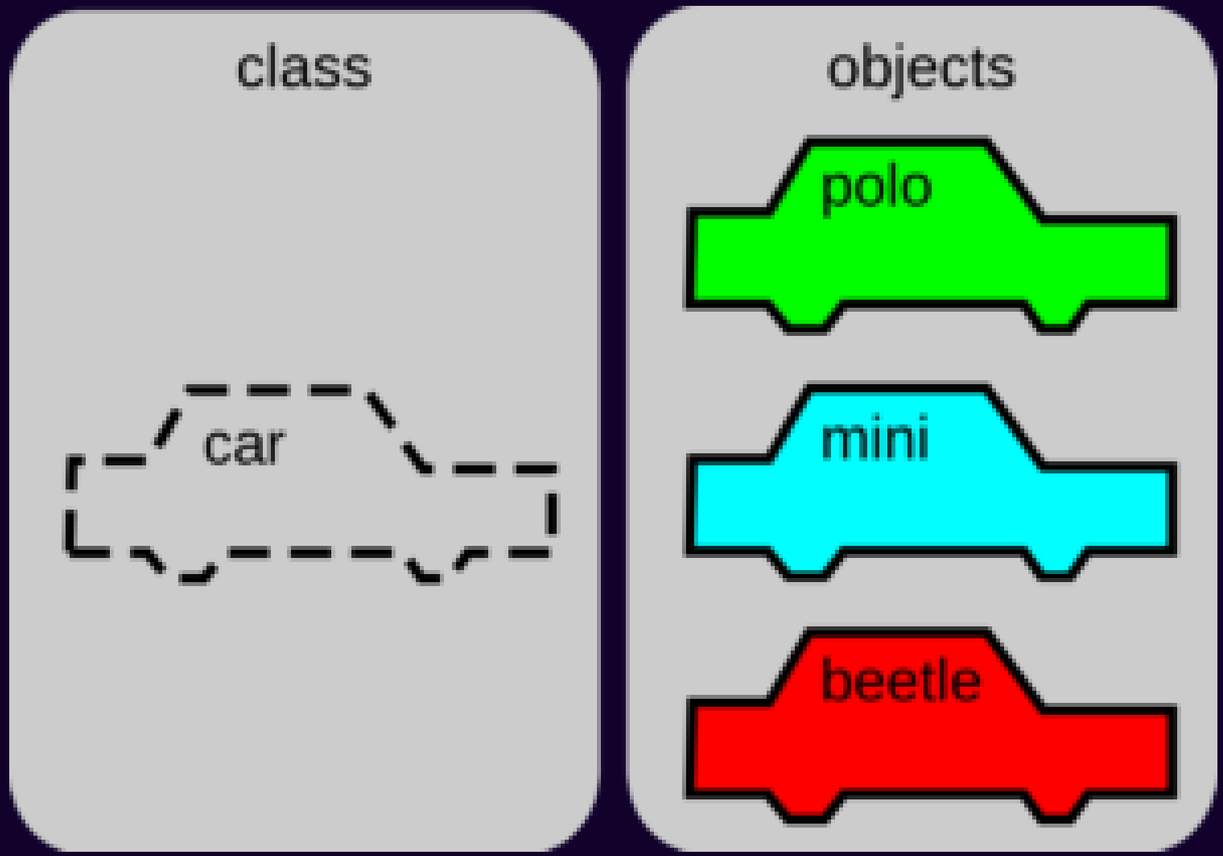
SAIBA MAIS

Artigo

PROGRAMAÇÃO ORIENTADA A OBJETOS

A **programação orientada a objetos** surgiu como uma alternativa a essas características da **programação estruturada**. O intuito da sua criação também foi o de aproximar o manuseio das estruturas de um programa ao manuseio das coisas do mundo real, daí o nome “objeto” como uma algo genérico, que pode representar qualquer coisa tangível.
Esse novo paradigma se baseia principalmente em dois conceitos chave: **classes** e **objetos**.
Todos os outros conceitos, igualmente importantes, são construídos em cima desses dois.

O QUE SÃO CLASSES E OBJETOS?



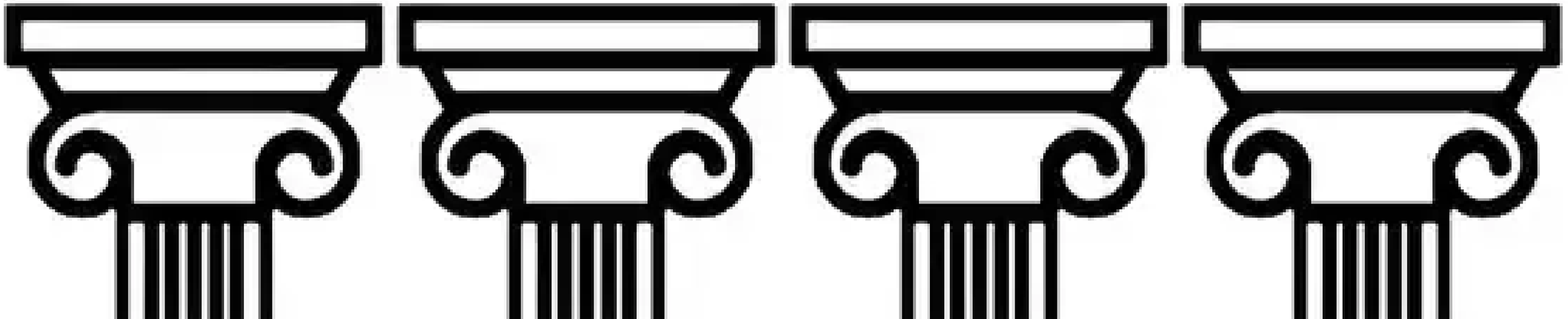
A IMPORTÂNCIA DAS CLASSES

01 Servem como um molde para a criação de objetos

02 Tornam mais fácil criar e lidar com objetos similares

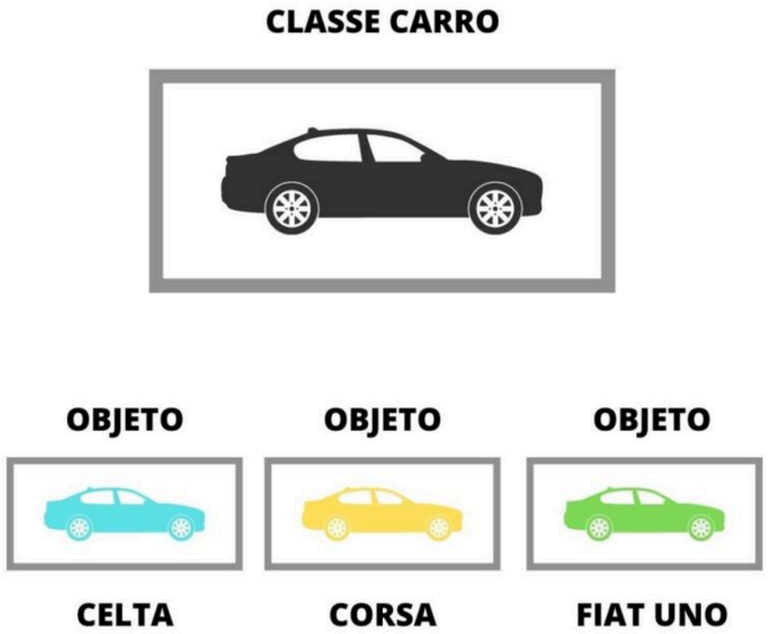
03 Evitam a necessidade de definir individualmente cada objeto

Os quatro pilares da Orientação a Objetos



ABSTRAÇÃO

Na **Programação Orientada a Objetos** (POO), abstração é um princípio que consiste em esconder os detalhes internos de implementação e expor apenas as funcionalidades essenciais de um objeto.



FUNÇÕES CONSTRUTORAS

Antes do surgimento da palavra-chave class em JavaScript, o paradigma de Programação Orientada a Objetos (POO) era implementado principalmente utilizando funções construtoras e o mecanismo de protótipos.

```
1  function Car(model, year) {
2    this.model = model;
3    this.year = year;
4  }
5
6  Car.prototype.getDetails = function() {
7    return `${this.model} - ${this.year}`;
8  };
9
10 const myCar = new Car('Toyota', 2020);
11 console.log(myCar.getDetails()); // Toyota - 2020
12
```

SURGIMENTO DA PALAVRA-CHAVE CLASS (ES6)

Com o surgimento do ECMAScript 6 (ES6) em 2015, foi introduzida a palavra-chave class, que trouxe uma sintaxe mais clara e amigável para quem vinha de outras linguagens orientadas a objetos. No entanto, internamente, as classes em JavaScript continuam usando o sistema de protótipos.

```
class Car {
  constructor(model, year) {
    this.model = model;
    this.year = year;
  }

  getDetails() {
    return `${this.model} - ${this.year}`;
  }
}

const myCar = new Car('Honda', 2021);
console.log(myCar.getDetails()); // Honda - 2021

```

O CONTEXTO DO THIS

O valor da palavra-chave **this** em JavaScript depende do contexto de execução. Dentro de uma função construtora, this faz referência ao objeto que está sendo criado. No entanto, em callbacks ou funções anônimas, o valor de this pode mudar, o que costuma causar confusão.

```
function Car(model) {
  this.model = model;

  setTimeout(function() {
    console.log(this.model);
    // `this` aqui não é o objeto `Car`,
    // e sim o objeto global ou `undefined`
  }, 1000);
}

const myCar = new Car('Ford');

```

MÉTODOS BIND, CALL E APPLY

Esses métodos servem para controlar explicitamente o valor de this em diferentes contextos.
Eles podem ser usados para garantir que this sempre referencie o objeto esperado.

- bind(): Cria uma nova função que, quando invocada, tem o valor de this pré-definido.

```
const car = {
  model: 'Nissan',
  getModel: function() {
    console.log(this.model);
  }
};

const getModel = car.getModel.bind(car); // `this` será sempre `car`
getModel(); // Nissan
```

- apply(): Permite chamar uma função em um determinado contexto.

```
const car = { model: 'Chevrolet' };

function displayModel(year, color) {
  console.log(`${this.model} - ${year} - ${color}`);
}

displayModel.apply(car, [2022, 'Red']); // Chevrolet - 2022 - Red
```

- call(): Invoca uma função, passando um valor específico de this.

```
const car1 = { model: 'BMW' };
const car2 = { model: 'Audi' };

function showModel() {
  console.log(this.model);
}

showModel.call(car1); // BMW
showModel.call(car2); // Audi
```

MÉTODOS DE CRIAÇÃO DE FUNÇÕES EM JAVASCRIPT

JavaScript oferece três formas principais de criar funções, cada uma com suas particularidades:

- Expressão de Função: Definida como uma expressão, onde a função é atribuída a uma variável. Diferente da declaração de função, não ocorre *hoisting*.

```
const greet = function() {
  console.log('Hello!');
};
greet(); // Hello!
```

- Declaração de Função: Uma das formas mais comuns e tradicionais de definir funções. Pode ser chamada antes de ser declarada devido ao hoisting.

```
function greet() {
  console.log('Hello!');
}
greet(); // Hello!
```

```
greet(); // Hello!
function greet() {
  console.log('Hello!');
}
```

- Função de Flecha: Introduzida no ES6, possui uma sintaxe mais curta e não vincula seu próprio this, herdando o this do escopo onde foi definida. Ideal para callbacks.

```
const greet = () => {
  console.log('Hello!');
};
greet(); // Hello!
```

DIAGRAMAS UML

UML (Unified Modeling Language) é uma **linguagem de notação** para modelar e documentar softwares orientados a objetos. Usando elementos gráficos, como retângulos e setas, ela cria diagramas que representam a estrutura, interações e mudanças do sistema, facilitando o desenvolvimento e a comunicação da equipe.