**Parte 1 – Classe Abstrata**

**O que são e como são simuladas em JavaScript?**

Uma **classe abstrata** é como um "molde" ou um "contrato" para outras classes. Ela define um conjunto de métodos e propriedades que as classes filhas (as que herdam dela) devem ter, mas não pode ser instanciada diretamente. Ou seja, não podemos criar um objeto a partir de uma classe abstrata com new. A ideia é que ela sirva apenas como uma base para outras classes mais específicas.

JavaScript, por padrão, não possui uma palavra-chave abstract como outras linguagens (Java ou C#, por exemplo). Portanto, precisamos **simular** esse comportamento. A forma mais comum de fazer isso é verificar no construtor da classe "abstrata" se ela está sendo instanciada diretamente e, em caso afirmativo, lançar um erro.

**Utilidade como "Contrato"**

A grande utilidade de uma classe abstrata é garantir que todas as classes que herdam dela sigam uma estrutura comum. Ela funciona como um **contrato** que obriga as classes filhas a implementarem certos métodos. Isso é muito útil em sistemas maiores, pois garante que diferentes partes do código que utilizam objetos de classes herdeiras possam contar com a existência desses métodos, tornando o sistema mais previsível e fácil de manter.

**Técnica para Impedir Instanciação e Obrigar Implementação**

1. **Impedir Instanciação:** Dentro do construtor da classe que queremos que seja abstrata, verificamos se o construtor do objeto é o mesmo da própria classe. Se for, significa que alguém está tentando instanciá-la diretamente, e então lançamos um erro. Uma forma comum de fazer isso é if (new.target === NomeDaClasseAbstrata).
2. **Obrigar Implementação de Métodos:** Nos métodos que queremos que sejam abstratos (e, portanto, obrigatórios para as classes filhas), simplesmente lançamos um erro, indicando que aquele método precisa ser implementado pela subclasse.

**Exemplo Simples**

Vamos imaginar que estamos criando um sistema para diferentes formas geométricas. Todas as formas devem ter um método para calcular a área, mas a fórmula é diferente para cada uma. Uma classe abstrata Forma é perfeita para isso.

JavaScript

// Parte 1: Classe Abstrata - Forma

// Simulando uma classe abstrata "Forma"

class Forma {

constructor() {

// Impede que a classe Forma seja instanciada diretamente

if (new.target === Forma) {

throw new Error("Não é possível instanciar uma classe abstrata diretamente.");

}

}

// Método "abstrato" que deve ser implementado pelas classes filhas

calcularArea() {

throw new Error("O método 'calcularArea' deve ser implementado pela classe filha.");

}

}

// Classe filha 1: Circulo

class Circulo extends Forma {

constructor(raio) {

super(); // Chama o construtor da classe pai (Forma)

this.raio = raio;

}

// Implementação obrigatória do método "contratado"

calcularArea() {

return Math.PI \* this.raio \* this.raio;

}

}

// Classe filha 2: Retangulo

class Retangulo extends Forma {

constructor(largura, altura) {

super();

this.largura = largura;

this.altura = altura;

}

// Implementação obrigatória do método "contratado"

calcularArea() {

return this.largura \* this.altura;

}

}

// Tentando usar as classes

try {

const forma = new Forma(); // Isso vai dar erro!

} catch (error) {

console.error("Erro esperado:", error.message);

}

const meuCirculo = new Circulo(5);

console.log(`Área do Círculo: ${meuCirculo.calcularArea().toFixed(2)}`);

const meuRetangulo = new Retangulo(10, 4);

console.log(`Área do Retângulo: ${meuRetangulo.calcularArea()}`);

**Parte 2 – Classe Interna (ou Aninhada)**

**O que são?**

Uma **classe interna** (ou aninhada) é uma classe definida dentro de outra classe. A ideia principal é agrupar classes que são logicamente relacionadas, onde uma delas só faz sentido no contexto da outra.

**Utilidade para Encapsulamento**

Elas são muito úteis para **encapsular funcionalidades**. Se temos uma classe Motor que é usada exclusivamente por uma classe Carro, faz sentido definir Motor dentro de Carro. Isso melhora a organização do código, pois a classe interna não "polui" o escopo global e fica claro que sua existência está diretamente ligada à classe externa.

**Acesso aos Membros da Classe Externa**

Em JavaScript, uma classe interna não tem acesso direto aos membros da instância da classe externa usando this. Para que a classe interna possa acessar as propriedades ou métodos da classe externa, a instância da classe externa (ou as propriedades necessárias) precisa ser passada para o construtor da classe interna.

**Exemplo de Classe com Classe Interna**

Vamos usar o exemplo de um Carro que gerencia seu próprio Motor. A classe Motor será interna à classe Carro.

JavaScript

// Parte 2: Classe Interna - Carro e Motor

class Carro {

constructor(modelo) {

this.modelo = modelo;

this.motor = new this.Motor(this); // Cria uma instância da classe interna

}

ligarCarro() {

this.motor.ignicao();

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

}

// Definição da classe interna "Motor"

Motor = class {

constructor(carro) {

// Recebe a instância da classe externa para poder acessá-la

this.carroPai = carro;

}

ignicao() {

// Acessando uma propriedade da classe externa

console.log(`Dando partida no motor do ${this.carroPai.modelo}...`);

}

}

}

// Criando uma instância da classe externa

const meuCarro = new Carro("Fusca");

meuCarro.ligarCarro();

// Não é possível (ou comum) criar um Motor sem um Carro

// let motorAvulso = new Carro.Motor(); // Isso daria erro ou seria inconsistente

**Parte 3 – Classe Anônima (Class Expressions)**

**O que são?**

Uma **classe anônima** é uma classe que é declarada sem um nome. Em JavaScript, isso é feito através de uma "expressão de classe" (class { ... }). Elas são muito parecidas com as funções anônimas.

**Vantagem para "Uso Único"**

A principal vantagem é a sua natureza de **"uso único"**. Elas são perfeitas para situações onde precisamos de uma estrutura de classe específica para uma única tarefa, como passar um objeto com uma determinada estrutura como argumento para uma função, sem a necessidade de criar um nome global para essa classe que só será usada uma vez.

**Instanciação Direta**

Com uma classe anônima, podemos instanciar um objeto diretamente na mesma linha da declaração da classe. Isso é feito colocando o operador new antes da expressão class.

**Exemplo de Uso**

Imagine que temos uma função que precisa receber um objeto "saudador" que tenha um método saudar. Podemos criar esse objeto rapidamente usando uma classe anônima, sem precisar definir uma classe Saudador separada.

JavaScript

// Parte 3: Classe Anônima

// Uma função que espera um objeto com um método "saudar"

function realizarSaudacao(saudador) {

saudador.saudar();

}

// Criando e instanciando um objeto a partir de uma classe anônima

// e passando-o como argumento para a função.

realizarSaudacao(new class {

// A classe não tem nome

nome = "Mundo";

saudar() {

console.log(`Olá, ${this.nome}!`);

}

});

// Outro exemplo: atribuindo a uma variável

const greeterPersonalizado = class {

constructor(nome) {

this.nome = nome;

}

dizerOi() {

console.log(`Oi, ${this.nome}!`);

}

};

const meuGreeter = new greeterPersonalizado("Estudante");

meuGreeter.dizerOi();