



**INSTITUTO FEDERAL
DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA**
Rio de Janeiro

Disciplina

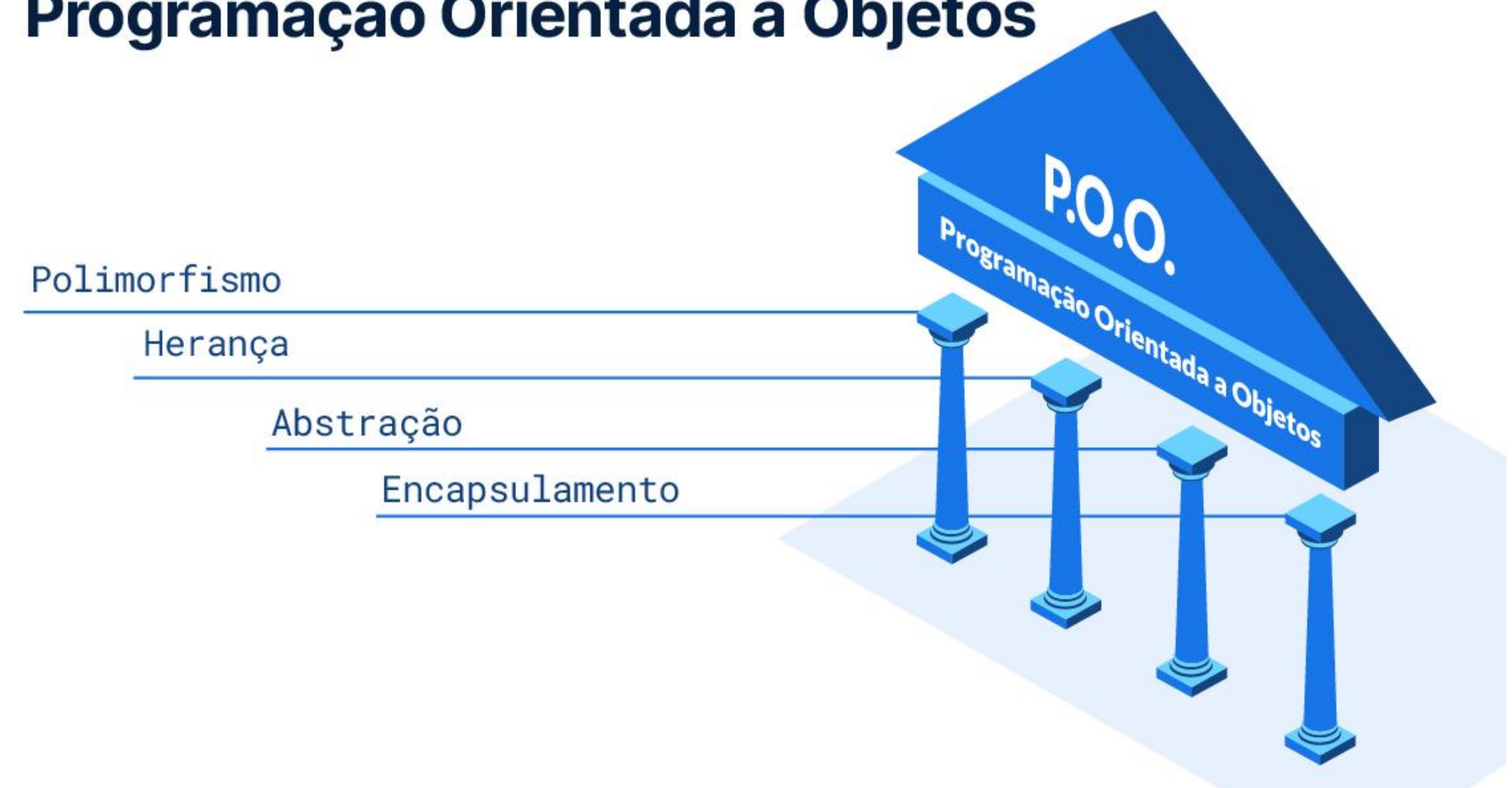
Modelagem de Domínio e
Conceitos de Orientação a Objetos

PROF. FERNANDO TAMBERLINI ALVES

Paradigma Orientação a Objetos

- ✓ Abstração
- ✓ Encapsulamento
- ✓ Herança
- ✓ Polimorfismo

Programação Orientada a Objetos



Paradigma Orientação a Objetos

✓ Encapsulamento

- A partir da versão ES2020 do Javascript foi possível declarar atributos privados incluindo “#” antes do nome do atributo na declaração das classes.

```
cobra > JS alimento.js > ...
1  class Alimento {
2      arqImagem='alimento.png';
3      #tamanho;
4      #valor;
5      #x;
6      #y;
7      #imagem;
8      constructor(valor,tamanho){
9          this.#tamanho = tamanho;
10         this.#valor = valor;
11         this.#x = Math.round(this.#tamanho+(tela.largura-2*this.#tamanho)*Math.random())
12         this.#y = Math.round(this.#tamanho+placar.altura+(tela.altura-2*this.#tamanho)*Math.random())
13         this.#imagem = new Image();
14         this.#imagem.src = this.arqImagem;
15     }
16     desenhar(){
17         canvasCtx.drawImage(this.#imagem, this.#x, this.#y, this.#tamanho, this.#tamanho)
18     }
19 }
```

Paradigma Orientação a Objetos

✓ Herança

- Objetos herdam atributos e métodos dos seus ancestrais na hierarquia.

```
cobra > JS objetoJogo.js > ...
1  class ObjetoJogo{
2      constructor(args){
3          this.tamanho = 20;
4          this.x = Math.round(this.tamanho+(tela.largura-2*this.tamanho)*Math.random());
5          this.y = Math.round(placar.altura+(tela.altura-2*this.tamanho)*Math.random());
6          if (args.length >= 1) this.tamanho = args[0];
7          if (args.length >= 2) this.x = args[1];
8          if (args.length >= 3) this.y = args[2];
9      }
10     teveColisao(obj){
11         const distX = this.x - obj.x;
12         const distY = this.y - obj.y;
13         if (((distX >= 0) && (Math.abs(distX) < obj.tamanho)
14             || (distX < 0) && (Math.abs(distX) < this.tamanho))
15             &&
16             ((distY >= 0) && (Math.abs(distY) < obj.tamanho)
17             || (distY < 0) && (Math.abs(distY) < this.tamanho)))
18             return true;
19         return false;
20     }
21 }
```

Paradigma Orientação a Objetos

✓ Herança

- Objetos herdam atributos e métodos dos seus ancestrais na hierarquia.

```
cobra > JS alimento.js > ...
1  class Alimento extends ObjetoJogo {
2      |   arqImagem='alimento.png';
3      |   #imagem;
4      |   constructor(valor,...args){
5      |       |   super(args);
6      |       |   this.valor = valor;
7      |       |   this.#imagem = new Image();
8      |       |   this.#imagem.src = this.arqImagem;
9      |       |   }
10     |   desenhar(){
11     |       |   canvasCtx.drawImage(this.#imagem, this.x, this.y, this.tamanho, this.tamanho);
12     |       |   }
13     |   }
```

Paradigma Orientação a Objetos

✓ Polimorfismo

Objetos podem responder de maneira diferente aos mesmos métodos, permitindo que um programa seja mais flexível e genérico.

```
cobra > JS jogo.js > document.addEventListener("keydown") callback
```

```
1 function jogar (){
2   tela.desenhar()
3   placar.desenhar()
4   cobra.desenhar()
5   alimento.desenhar()
6   cobra.mover()
7   if (alimento.teveColisao(cobra)){
8     placar.qtdPontos+=alimento.valor;
9     alimento = new Alimento(alimento.valor++)
10  }
11  if (cobra.vida > 0)
12    requestAnimationFrame(jogar)
13  else{
14    placar.nomeJogo = "Fim de Jogo";
15    placar.desenhar()
16  }
17 }
```

```
cobra > JS jogo.js > document.addEventListener("keydown") callback
```

```
18 let alimento = new Alimento(1)
19 //let alimento = new Alimento(1,30)
20 //let alimento = new Alimento(1,30,400)
21 //let alimento = new Alimento(1,30,460)
22 jogar()
23 document.addEventListener("keydown",(evento) => {
24   if (evento.key == 6 ) cobra.direcao=0;
25   if (evento.key == 2 ) cobra.direcao=90;
26   if (evento.key == 4 ) cobra.direcao=180;
27   if (evento.key == 8 ) cobra.direcao=270;
28   // console.log("Tecla Pressionada: " + evento.key);
29 });
```

Criando Classes

Criando a classe Triangulo

```
class Triangulo{
    constructor(base,altura){
        this.b = base;
        this.h = altura;
    }
    calcularArea(){
        return this.b*this.h/2;
    }
}

const triangulo6 = new Triangulo(5,8);
```

Criando Classes

Criando a classe TrianguloV
com parâmetros privados

```
class TrianguloV{
    #b;
    #h;
    constructor(base, altura){
        this.b = base;
        this.h = altura;
    }
    calcularArea(){
        return this.b*this.h/2;
    }
    getBase(){
        return this.#b
    }
    getAltura(){
        return this.#h
    }
}
```

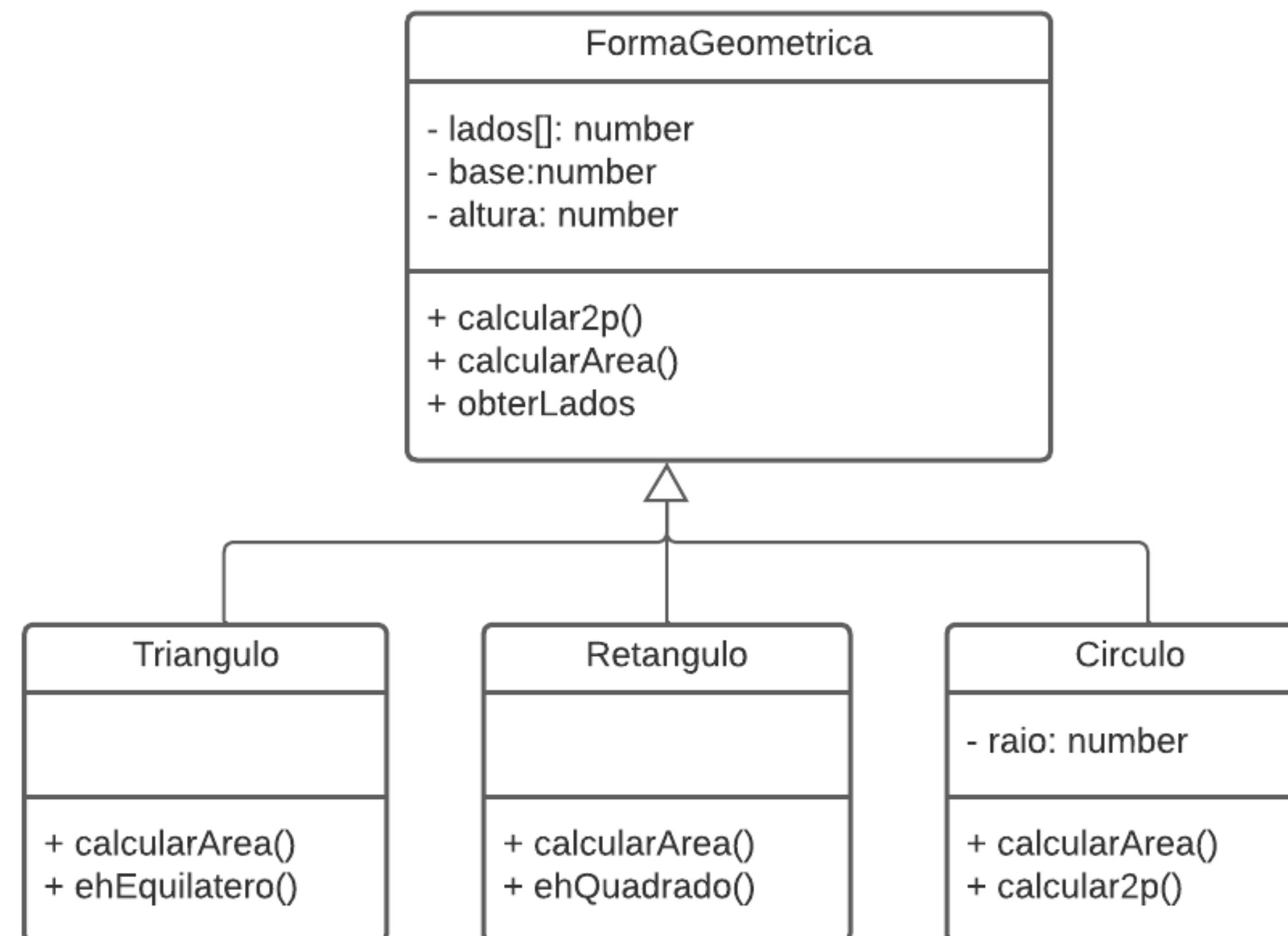
Criando Classes

Criando a subclasse
TrianguloSub que é filha da
classe Forma.

```
class TrianguloSub extends Forma{  
    #b;  
    #h;  
  
    constructor(lados,base,altura){  
        super(lados);  
        this.#b=base;  
        this.#h=altura;  
    }  
    calcularArea(){  
        return this.#b*this.#h/2;  
    }  
}
```

Exercício Prático

- Criar as classes descritas abaixo com a implementação dos seus métodos:



Exercício Prático

Criando a classe
FormaGeometrica

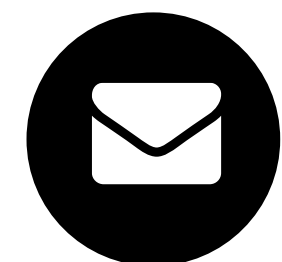
```
class FormaGeometrica{
    #lados;
    constructor(vetorLados){
        this.#lados = vetorLados;
    }
    calcular2p(){
        let perimetro=0
        for(let i=0;i<this.#lados.length;i++){
            perimetro+=this.#lados[i];
        }
        return perimetro;
    }
    calcularArea(){
        return this.#lados[0]*this.#lados[0]
    }

    obterLados(){
        return this.#lados
    }
}
```

Dúvidas?



Dúvidas?



fernando.alves@ifrj.edu.br



ftamberlini.dev.br

Referências:

- Notas de Aula do Prof. Leonardo Murta – UFF
- Livro “Object-Oriented Analysis and Design with Applications”