Prof. Me. Hélio Esperidião

PHP 8

PHP 8

- O PHP 8 trouxe melhorias significativas em desempenho, sintaxe e funcionalidades.
- A linguagem acompanha características importantes para manter projetos grandes
- A sintaxe é mais clara e intuitiva
- A tipagem evita problemas de conversões indevidas e padroniza a forma de desenvolvimento.

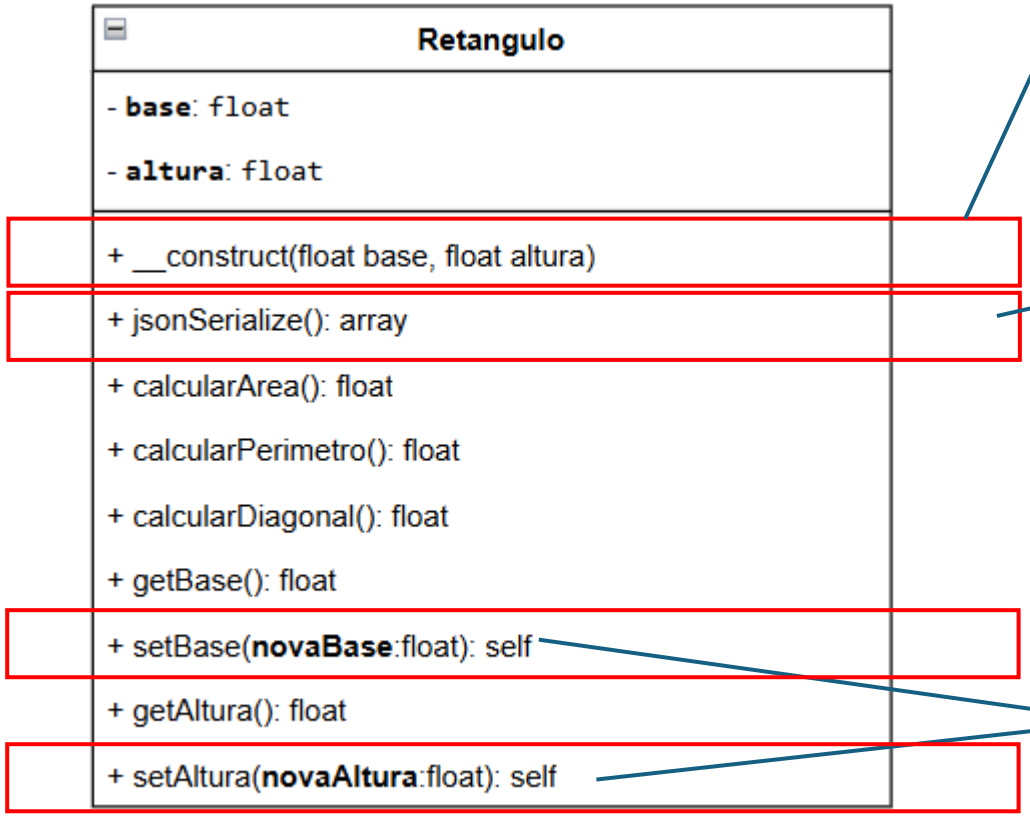
Desempenho Superior (JIT Compiler)

- Compilação Just-In-Time (JIT) introduzida no PHP 8.0, permitindo otimizações em tempo de execução para cálculos intensivos (como matemática, machine learning e processamento de dados).
- Ganhos de velocidade em benchmarks (até 3x mais rápido em alguns cenários).

Por que Migrar para PHP 8?

- Velocidade: JIT melhora desempenho em aplicações críticas.
- Menos código repetitivo
- Menos bugs: Tipagem mais forte e erros convertidos em exceções.
- Modernidade: Alinha PHP com linguagens como TypeScript e C#.
- A versão mais recente (PHP 8.3) continua adicionando melhorias incrementais, como tipagem para constantes de classe e otimizações no garbage collector.

Classe Retangulo



Definição dos atributos e definição de Valor padrão caso não seja passado valor no parâmetro

Método que é chamado quando um objeto da classe é passado como parâmetro no método json_encode($objetoRetangulo).

Isso permite que json sejam criados mais facilmente a partir de objetos da classe Retangulo.

Os métodos set agora retornam o objeto da própria classe.
Sintaxe moderna que permite o encadeamento de instruções.

Codificação do construção.

```
public function __construct(
    private float $base = 0, //Verifique a tipagem float
    private float $altura = 0 //verifique a tipagem float
) {}
```

```
//Não foi passado parâmetro,
//base então é inicializada com zero
//e altura também é inicializada com zero
$obj = new Retangulo();

//foi passado o valor 10 para a base e 20 para a altura
$obj2 = new Retangulo(10, 20);
```

O construtor é chamando quando a classe é instanciada dando origem ao objeto, ou seja a classe é chamada no momento do “new”.

A sintaxe nova declara os atributos da classe e seus valores iniciais no construtor.

```
$obj = new Retangulo();
echo $obj->getBase();//imprime 0

//foi passado o valor 10 para a base e 20 para a altura
$obj2 = new Retangulo(10, 20);
echo $obj->getBase();//imprime 10
```

O __construct() é chamado no momento do “new”

```
public function __construct(
    private float $base = 0,
    private float $altura = 0
) {}
```

```php
<?php
// Inclui o arquivo da classe Retangulo que contém a definição da classe
require_once "modelo/Retangulo.php";
// Cria uma instância de Retangulo informando apenas a base (10) usando parâmetro nomeado
// A altura assumirá um valor padrão definido na classe
$r = new Retangulo(base:10);
// Exibe a base do retângulo (será 10)
echo "Base: " . $r->getBase() . "\n";
// Exibe a altura do retângulo (será o valor padrão definido na classe)
echo "Altura: " . $r->getAltura() . "\n";
// Cria uma segunda instância de Retangulo informando ambos os parâmetros (base e altura)
// usando parâmetros nomeados explicitamente
$r2 = new Retangulo(base: 10, altura: 5);
// Cria uma terceira instância de Retangulo informando ambos os parâmetros (base e altura)
// usando parâmetros posicionais (sem nomear os argumentos)
// O primeiro valor (10) será atribuído ao primeiro parâmetro (base)
// O segundo valor (5) será atribuído ao segundo parâmetro (altura)
$r3 = new Retangulo(10, 5);
// Observações importantes:
// 1. A primeira instância ($r) demonstra o uso de valor padrão para altura
// 2. A segunda instância ($r2) mostra a forma mais legível com parâmetros nomeados
// 3. A terceira instância ($r3) mostra a forma tradicional de passagem de parâmetros
// 4. Todas as instâncias criam objetos válidos, mas de formas diferentes
// 5. A saída dos echos mostrará os valores conforme foram definidos
•  ?>
```

JsonSerializable

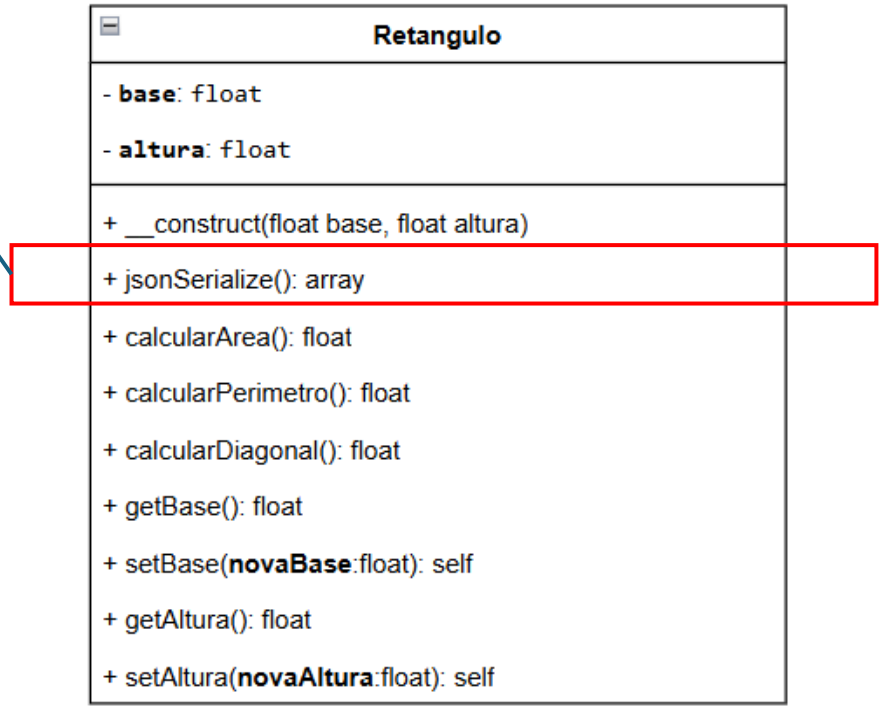
- Permite que um objeto de uma classe seja facilmente convertido para Json.

jsonSerialize()

```php
declare(strict_types=1); //implementa a tipagem estrita
class Retangulo implements JsonSerializable{
    public function __construct(
        private float $base = 0,
        private float $altura = 0
    ) {}
    public function jsonSerialize(): array {
        return [
            'base'      => $this->base,
            'altura'    => $this->altura,
            'area'      => $this->calcularArea(),
            'perimetro' => $this->calcularPerimetro(),
            'diagonal'  => $this->calcularDiagonal()
        ];
    }
}
//Considere que os métodos calcularArea(), >calcularPerimetro() e calcularDiagonal() já foram implementados
```

Utilização

```php
<?php

require_once "modelo/Retangulo.php";

//foi passado o valor 10 para a base e 20 para a altura

$obj2 = new Retangulo(10, 20);

echo json_encode($obj2);

//{"base":10,"altura":20,"area":200,"perimetro":60,"diagonal":22.360679774997898}
```



Quando o objeto é passado como parâmetro para a função json_encode($obj) o método jsonSerialize é chamado, um json é construído com base no retorno do array do método jsonSerialize()

Observe que o Json formado possui a mesma estrutura do vetor retornado no método jsonSerialize()

Implementação de métodos

```php
public function calcularArea():  float | int    {
    return $this->base * $this->altura;
}
public function calcularPerimetro(): float {
    return 2 * ($this->base + $this->altura);
}
 public function calcularDiagonal(): float {
    return sqrt($this->base ** 2 + $this->altura ** 2);
}
```

Observe que o método calcularArea() pode retornar um Real ou Inteiro

Os métodos agora possuem determinação de tipo de retorno.

Métodos get e set

```php
public function getBase(): float{
   return $this->base;
}
public function setBase(float $novaBase): self {
        $this->base = $novaBase;
        return $this;
    }
```

Observe que os métodos set agora possuem retorno.
O método set que não retornava dados, agora retorna $this que é o próprio objeto

Utilização do encadeamento de métodos

```php
<?php

require_once "modelo/Retangulo.php";

$obj2 = new Retangulo();

$obj2->setBase(novaBase: 10)->setAltura(novaAltura:20);

echo json_encode($obj2);

//{"base":10,"altura":20,"area":200,"perimetro":60,"diagonal":22.360679774997898}
```

novaBase: É recomendado explicitar o nome da variável que será passada como parâmetro.

Em uma primeira análise o desenvolvedor pode imaginar que o método setBase() chama o método setAltura(), mas não é isso que acontece.

Observe que na implementação do método setBase() é retornado o objeto $this que é a própria variável $obj2 analisando o exemplo ao lado.

```
[$obj2->setBase(novabase: 10)]->setAltura(20);
[$this]->setAltura(novaAltura:20);

$obj2->setAltura(novaAltura:20);
```

Observe que a execução se dá em etapas, primeiramente é realizado o setBase(), o setBase() retorna o $this que é o próprio objeto.

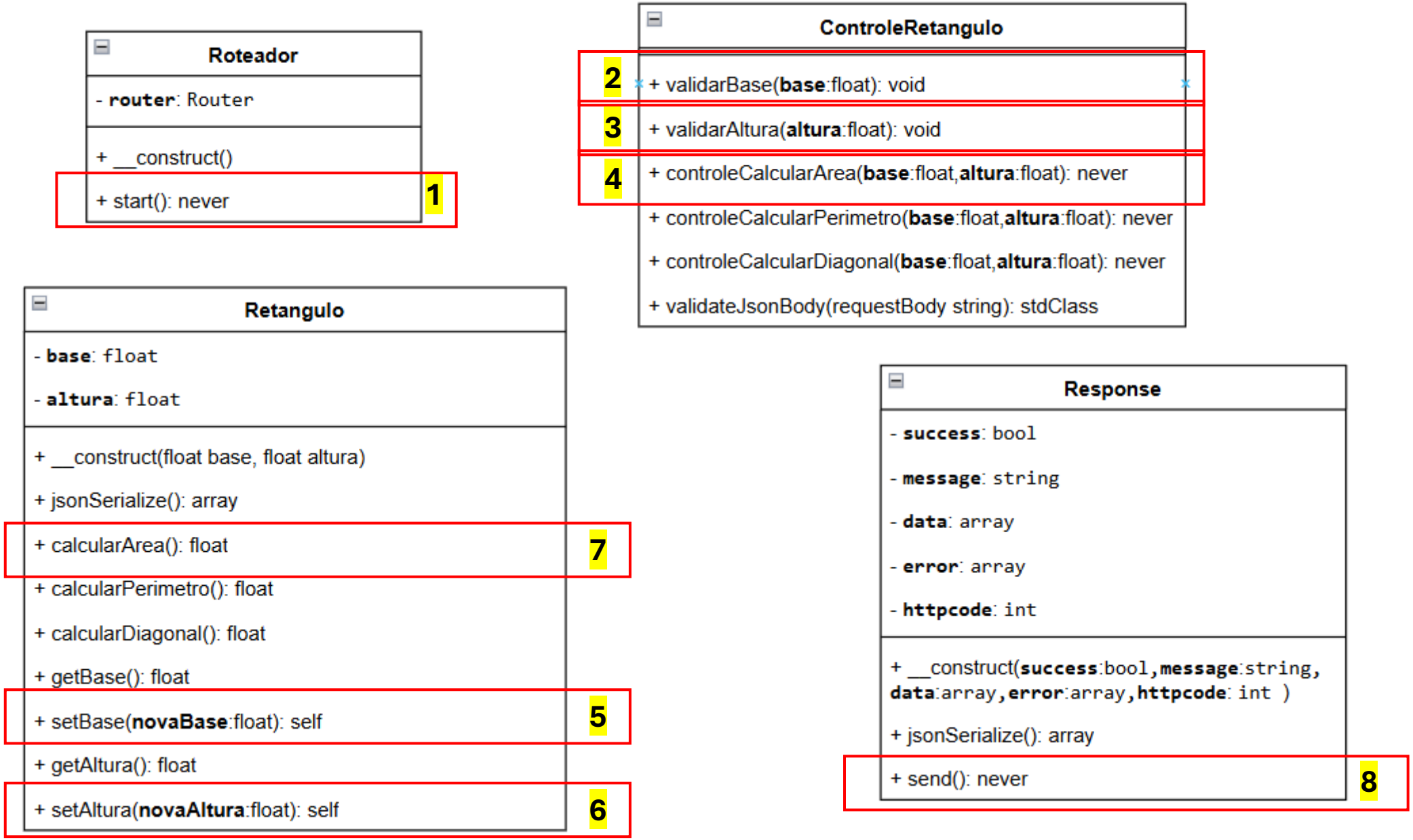
Restruturação da API Retângulo.

- Controle
  - ControleRetangulo.php
- http
  - Response.php
- Modelo
  - Retangulo.php
- Index.php
- Roteador.php
- Router.php

API – Retângulo PHP 8

Demonstrativo da rota:

- http://localhost:8080/retangulos/areas/10/5


Roteador
- router: Router
+ __construct()
+ start(): never
1
ControleRetangulo
2
+ validarBase(base:float): void
3
+ validarAltura(altura:float): void
4
+ controleCalcularArea(base:float,altura:float): never
+ controleCalcularPerimetro(base:float,altura:float): never
+ controleCalcularDiagonal(base:float,altura:float): never
+ validateJsonBody(requestBody string): stdClass
Retangulo
- base: float
- altura: float
+ __construct(float base, float altura)
+ jsonSerialize(): array
+ calcularArea(): float
7
+ calcularPerimetro(): float
+ calcularDiagonal(): float
+ getBase(): float
+ setBase(novaBase:float): self
5
+ getAltura(): float
+ setAltura(novaAltura:float): self
6
Response
- success: bool
- message: string
- data: array
- error: array
- httpcode: int
+ __construct(success:bool,message:string, data:array,error:array,httpcode: int)
+ jsonSerialize(): array
+ send(): never
8

Observe que agora o arquivo .htaccess agora redireciona para o arquivo index.php

```
# Ativa o mecanismo de reescrita de URL
RewriteEngine on
# Se a requisição não for um arquivo existente...
RewriteCond %{REQUEST_FILENAME} !-f
# ... E se a requisição não for um diretório existente...
RewriteCond %{REQUEST_FILENAME} !-d
# Redireciona todas as requisições para index.php
RewriteRule ^(.*)$ /index.php
```

Response.php

```php
<?php
declare(strict_types=1);
class Response implements JsonSerializable{
    public function __construct(
        private  bool $success,
        private  string $message,
        private  array $data = [],
        private array $error = [],
        private int $httpcode = 200
    ) {}
    public function jsonSerialize(): array
    {
        $response = [
            'success' => $this->success,
            'message' => $this->message,
        ];

           //se o campo não estiver vazio, adiciona o campo error ao array de resposta
        if (empty($this->data) == false) {
            $response['data'] = $this->data;
        }
        //se o campo não estiver vazio, adiciona o campo error ao array de resposta
        if (empty($this->error) == false) {
            $response['error'] = $this->error;
        }

           return $response;
    }
    public function send(): never
    {
        http_response_code($this->httpcode);
        echo json_encode($this);
        exit;
    }
}
```

```php
<?php
require_once "http/Response.php";

(new Response(
    success: false,
    message: 'Erro de validação',
    data: [],
    error: [
        'errorCode' => 'invalid_data',
        'message' => 'A (base) não pode ser menor ou igual a
zero',
        'details' => 'teste'
    ],
    httpcode: 400 // Bad Request
))->send();
```

```
{
    "success": false,
    "message": "Erro de validação",
        "error": {
            "errorCode": "invalid_data",
            "message": "A (base) não pode ser menor ou igual a zero",
            "details": "teste"
          }
}
```

Index.php

```php
<?php
require_once "Roteador.php";
// Cria uma instância da classe Roteador
$roteador = new Roteador();

  // Inicia o roteamento chamando o método público
$roteador->start();
```

Roteador.php

```php
class Roteador {
    private $router;
    public function __construct() {
        header('Access-Control-Allow-Origin: *’); // responde requisições de qualquer domínio
        header('Access-Control-Allow-Methods: GET, POST,OPTIONS’); //permite apenas get, post e options
        header('Access-Control-Allow-Headers: Content-Type'); //necessário para requisições que enviam dados JSON.
        header("Content-Type: application/json");  //define que a resposta será um json
        $this->router = new Router();
        $this->router->get('/retangulos/areas/([0-9.]+)/([0-9.]+)', function (float $base, float $altura): never {
            $controle = new ControleRetangulo();            // Cria uma instância do controlador
            $controle->validarBase(base:$base);            // Valida a base do retângulo
            $controle->validarAltura(altura: $altura);      // Valida a altura do retângulo
            $stdRetangulo = new stdClass();
            $stdRetangulo->retangulo = new stdClass();
            $stdRetangulo->retangulo->base = $base;        // Define a base do retângulo
            $stdRetangulo->retangulo->altura = $altura;    // Define a altura do retângulo
            $controle->controleCalcularArea(stdRetangulo: $stdRetangulo);
        });
}
 public function start(): never   {
        $this->router->run();
   }
}
```

Controle - Validação

```
    public function validarBase(float $base): void {
        if (isset($base) == false) {
            (new Response(
                success: false,
                message: 'Erro de validação',
                data: [],
                error: [
                    'errorCode' => 'invalid_data',
                    'message' => 'base não enviada',
                ],
                httpcode: 400 // Bad Request
            ))->send(); // imprime a resposta e encerra a execução
        }
        //Outras validações
}
```

controleCalcularArea(stdClass $stdRetangulo)

```php
// Função responsável por controlar o cálculo da área
    public function controleCalcularArea(stdClass $stdRetangulo): never  {
        // Cria o objeto da classe Retangulo e define os valores
        $retangulo = new Retangulo();

          $area = $retangulo
            ->setBase(novaBase: $stdRetangulo->retangulo->base)
            ->setAltura(novaAltura: $stdRetangulo->retangulo->altura)
            ->calcularArea();

          (new Response(
            success: true,
            message: 'Calculado com sucesso',
            data: [
                'retangulo' => [
                    'area' => $area
                ],
            ],
            error: [],
            httpcode: 200
        ))->send();
    }
```

O método recebe um objeto stdClass que representa os atributos de um retângulo

```
{
   "retangulo":{
                "base":5.3,
                "altura":10
   }
}
```

É realizada a instancia da classe Retangulo, os atributos base e algura são passados pelo método get, é realizo o calculo da área. É montada uma instancia da classe response com uma resposta padrão para a requisição. Observe que no construtor da classe Reponse são passados

POST

- Quando é enviado um post para:
  http://localhost:8080/retangulos/areas/
- É enviado um json no seguinte padrão:

```
{
   "retangulo": {
                  "base":5.3,
                  "altura":10
   }
}
```

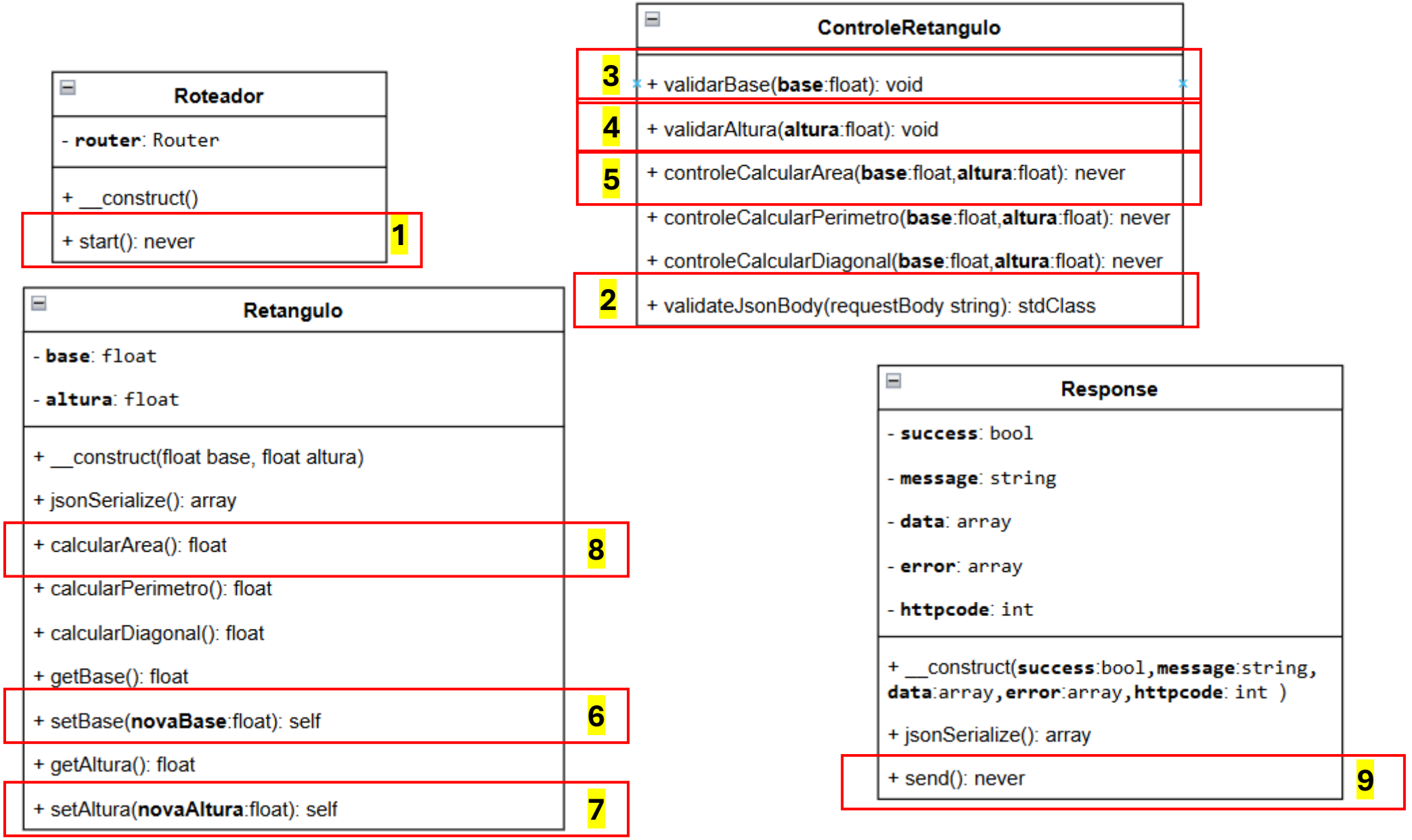
- Esse json deve ser valido, é necessário verificar:
  1. Se o json é válido.
  2. Se o json possui “retangulo”
  3. Se “retângulo” possui “base” e “altura”

Rota: POST: /retangulos/areas

```
// Define uma rota POST que calcula a área de um retângulo a partir de dados JSON
      $this->router->post('/retangulos/areas/', function (): never {
            // Lê o conteúdo do corpo da requisição
          $requestBody = file_get_contents('php://input');
            // Cria uma instância do controlador responsável pelas validações e cálculos
          $controle = new ControleRetangulo();
            // Valida o JSON recebido e retorna um objeto contendo os dados
          $stdRetangulo = $controle->validateJsonBody(requestBody: $requestBody);
            // Valida se o valor da base é válido (ex: maior que zero)
          $controle->validarBase(base: $stdRetangulo->retangulo->base);
            // Valida se o valor da altura é válido (ex: maior que zero)
          $controle->validarAltura(altura: $stdRetangulo->retangulo->altura);
            // Realiza o cálculo da área com base e altura válidas
          $controle->controleCalcularArea(stdRetangulo: $stdRetangulo);
      });
```

POST

Sequência de processamento:



Validações são responsabilidade do controle.

- Foi criado um método: **`validateJsonBody(requestBody: $requestBody)`**
- Esse método recebe o corpo da requisição, ou seja o json enviado pelo cliente com os dados referentes ao retângulo.

```
{
   "retangulo": {
                  "base":5.3,
                  "altura":10
   }
}
```

`validateJsonBody(string $requestBody): stdClass`

- Etapas de processamento
  1. Verifica se o Json é Válido
  2. Verifica se foi enviado um retângulo
  3. Verifica se foi enviada a base do retângulo
  4. Verifica se enviada a altura do retângulo.

Etapa 1 + Tentar converter o texto da requisição para um objeto stdClass

```php
public function validateJsonBody(string $requestBody): stdClass  {
        $stdRetangulo = json_decode($requestBody);
        // Verifica se houve erro na decodificação (JSON inválido)
        if (json_last_error() !== JSON_ERROR_NONE) {
            (new Response(
                success: false,
                message: 'Erro de validação',
                data: [],
                error: [
                    'errorCode' => 'invalid_data',
                    'message' => 'Dados enviados em formato inválido.',
                ],
                httpcode: 400
            ))->send();
        }
```

Etapa 2 – Verifica se o objeto possui o tributo “retângulo”

```
else if (!isset($stdRetangulo->retangulo)) {
            (new Response(
                success: false,
                message: 'Erro de validação',
                data: [],
                error: [
                    'errorCode' => 'invalid_data',
                    'message' => 'Atributo (base) ausente dentro de "retangulo',
                ],
                httpcode: 400
            ))->send();
        }
```

Etapa 3 – Verifica se existe os atributos “retângulo->base”

```
else if (!isset($stdRetangulo->retangulo->base)) {
            (new Response(
                success: false,
                message: 'Erro de validação',
                data: [],
                error: [
                    'errorCode' => 'invalid_data',
                    'message' => 'Atributo (base) ausente dentro de "retangulo',
                ],
                httpcode: 400
            ))->send();
        }
```

Etapa 3 – Verifica se existe os atributos “retângulo->altura”

```
else if (!isset($stdRetangulo->retangulo->altura)) {
            (new Response(
                success: false,
                message: 'Erro de validação',
                data: [],
                error: [
                    'errorCode' => 'invalid_data',
                    'message' => 'Atributo (altura) ausente dentro de "retangulo',
                ],
                httpcode: 400
            ))->send();
        }
```

```php
<?php
declare(strict_types=1); //implementa a tipagem estrita
class Retangulo implements JsonSerializable {
    public function __construct(
        private float $base = 0,
        private float $altura = 0
    ) {}
    public function jsonSerialize(): array {
        return [
            'base'      => $this->base,
            'altura'    => $this->altura,
            'area'      => $this->calcularArea(),
            'perimetro' => $this->calcularPerimetro(),
            'diagonal'  => $this->calcularDiagonal()
        ];
    }
    public function calcularArea(): float | int {
        return $this->base * $this->altura;
    }
    public function calcularPerimetro(): float {
        return 2 * ($this->base + $this->altura);
    }
    public function calcularDiagonal(): float {
        return sqrt($this->base ** 2 + $this->altura ** 2);
    }
```

```php
public function getBase(): float {
        return $this->base;
    }

public function getAltura(): float     {
        return $this->altura;
}
public function setBase(float $novaBase): self {
        $this->base = $novaBase;
        return $this;
}
public function setAltura(float $novaAltura): self {
        $this->altura = $novaAltura;
        return $this;
    }
}
```

Rota padrão para 404

```php
$this->router->set404(function () {
        (new Response(
            success: false,
            message: 'Rota não encontrada',
            data: [],
            error: [],
            httpcode: 404
        ))->send();
    });
```


```
{
        "success": false,
        "message": "Rota não encontrada",
        "data":{
                "route": "/retangulos/perimetros2/"
        }
}
```