**Objetivo:**

I. Estrutura de uma classe no TS;

II. Construtor;

III. Diferença entre tipos primitivos e objetos;

IV. Representação de uma classe no diagrama UML.

**Observação:** antes de começar, crie um projeto para reproduzir os exemplos:

1. Crie a pasta exemplo (ou qualquer outro nome sem caracteres especiais) no local de sua preferência do computador;
2. Abra a pasta no VS Code e acesse o terminal do VS Code;
3. No terminal, execute o comando `npm init -y` para criar o arquivo fundamental de um projeto Node (arquivo package.json):

```
TERMINAL   PROBLEMS   OUTPUT

PS C:\Desktop\exemplo> npm init -y
```

4. No terminal, execute o comando `npm i -D ts-node typescript` para instalar os pacotes ts-node e typescript como dependências de desenvolvimento;

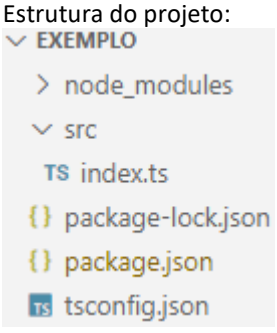
```
TERMINAL   PROBLEMS   OUTPUT

PS C:\Desktop\exemplo> npm i -D ts-node typescript
```

5. No terminal, execute o comando `tsc --init` para criar o arquivo de opções e configurações para o compilador TS (arquivo tsconfig.json):

```
TERMINAL   PROBLEMS   OUTPUT

PS C:\Desktop\exemplo> tsc --init
```

6. Crie a pasta src na raiz do projeto;
7. Crie o arquivo index.ts na pasta src;

Estrutura do projeto:




8. Adicione na propriedade `scripts`, do package.json, o comando para executar o arquivo index.ts. Ao final o arquivo package.json terá o seguinte conteúdo:

```
{
  "name": "exemplo",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "start": "ts-node ./src/index"
  },
```

```
  "keywords": [],
  "author": "",
  "license": "ISC",
  "devDependencies": {
    "ts-node": "^10.9.1",
    "typescript": "^5.1.6"
  }
}
```

Observação: se o comando `ts-node` não funcionar, então use o programa `npx` para executar o comando `ts-node`:

```
"scripts": {
  "start": "npx ts-node ./src/index"
},
```

## I. Estrutura de uma classe no TS

As linguagens de programação orientada a objetos (POO) permitem ao programador definir seus tipos de dados através da composição de outros tipos de dados e operações (funções/métodos). O TS suporta a definição de classes, herança, interfaces, polimorfismo, encapsulamento e outros conceitos fundamentais da POO.

A classe é uma estrutura que define um tipo de dado composto. No TS uma classe é definida usando a palavra-chave `class` seguida pelo nome da classe e par de chaves:

```
class Pessoa {
    //corpo da classe
}
```

Dentro das chaves fica o corpo da classe e pode conter propriedades, métodos e construtores:

- Propriedade: é uma variável definida no corpo da classe. No exemplo a seguir todos os objetos (instâncias da classe) do tipo de dado `Pessoa` possuirão as propriedades `nome` e `idade`:

```
class Pessoa {
    nome:string = "Ana";
    idade:number = 18;
}

// constrói uma instância (objeto) do tipo Pessoa
// e coloca na variável x
const x = new Pessoa();
// acessa a propriedade nome do objeto que está na
variável x
console.log("Nome:", x.nome);
// acessa a propriedade idade do objeto que está na
variável x
console.log("Idade:", x.idade);
```

```
PS C:\Desktop\exemplo> npm start
> exemplo@1.0.0 start
> ts-node ./src/index
Nome: Ana
Idade: 18
```

- Método: é uma função definida no corpo da classe. Os métodos são chamados de operações no contexto da POO. No exemplo a seguir todos os objetos do tipo de dado `Pessoa` possuirão as propriedades `nome` e `idade` e o método `imprimir`:

```
class Pessoa {
    nome:string = "Ana";
    idade:number = 18;

    imprimir(){
        console.log(`${this.nome} possui ${this.idade} anos`);
    }
}

const x = new Pessoa();
// chama o método imprimir do objeto que está na variável x
x.imprimir();
```

```
PS C:\Desktop\exemplo> npm start
> exemplo@1.0.0 start
> ts-node ./src/index

Ana possui 18 anos
```

- Construtor: é um "método especial" dentro de uma classe que é responsável por criar e inicializar objetos da classe. Ele é executado automaticamente quando um objeto é criado a partir da classe. No exemplo a seguir observe que o construtor foi executado 2 vezes, ou seja, ele foi executado para criar o objeto que colocamos na variável `x` e depois foi executado para criar o objeto que colocamos na variável `y`.

  Para criar uma instância (objeto) da classe, usamos a palavra-chave `new` seguida pelo nome da classe. Como exemplo, `new Pessoa()` invocará o construtor da classe, e este por sua vez criará uma instância da classe e retornará essa instância para colocarmos na variável x. Se quisermos outra instância, teremos de invocar o construtor novamente e colocar a instância retornada em outra variável, assim como fizemos na variável y.

```
class Pessoa {
    constructor(){
        console.log("Construiu");
    }
}

// cria uma instância (objeto) do tipo Pessoa
// e coloca na variável x
const x = new Pessoa();
// cria uma instância (objeto) do tipo Pessoa
// e coloca na variável y
const y = new Pessoa();
```

```
PS C:\Desktop\exemplo> npm start
> exemplo@1.0.0 start
> ts-node ./src/index
Construiu <-- instrução new Pessoa()
Construiu <-- instrução new Pessoa()
```

Instância da classe e objeto da classe são termos sinônimos no contexto da POO.

A classe é um template para a construção de objetos/instâncias da classe. Um objeto/instância é uma "cópia da classe".

Dentro do corpo de uma classe:

- Uma variável recebe o nome de propriedade ou atributo;
- Uma função recebe o nome de método.

O TS oferece suporte total para a palavra reservada `class` desde a ES2015. Para mais detalhes https://www.typescriptlang.org/docs/handbook/2/classes.html.

## II. Construtor

O construtor é um bloco de código que **só** será chamado durante a construção (instanciação) do objeto. Na linguagem TS o construtor é definido pelo bloco de nome `constructor`.

Todas as classes possuem um construtor padrão, no exemplo a seguir a instrução `new Pessoa()` invocará o construtor padrão da classe `Pessoa`:

```
class Pessoa { }

const x = new Pessoa();
```

Porém, o construtor padrão deixa de existir se definirmos um construtor no corpo da classe, assim como fizemos no exemplo a seguir:

```
class Pessoa {
    constructor(){
        console.log("Construiu");
    }
}
```

Geralmente, usamos o construtor para inicializar os atributos. No exemplo a seguir o construtor recebe uma `string` e um `number` como parâmetro e esses valores são usados para inicializar, respectivamente, as propriedades `nome` e `idade`.

O termo `this` refere-se à instância atual da classe. Ele é uma palavra-reservada que permite acessar as propriedades e métodos da instância da classe dentro dos métodos e construtor da classe. No exemplo, a seguir a instrução `this.nome` acessará a propriedade `nome` do objeto que está sendo criado.

```
class Pessoa {
    nome:string;
    idade:number;

    constructor(a:string, b:number){
        this.nome = a;
        this.idade = b;
    }

    imprimir(){
        console.log(`${this.nome} possui ${this.idade} anos`);
    }
}

const x = new Pessoa("Ana",18);
const y = new Pessoa("Pedro",20);
x.imprimir();
y.imprimir();
```

```
PS C:\Desktop\exemplo> npm start
> exemplo@1.0.0 start
> ts-node ./src/index
Ana possui 18 anos
Pedro possui 20 anos
```

Constitui erro chamar o construtor:

- Passando os parâmetros fora de ordem:

```
const w = new Pessoa(21,"Maria");
```

- Passando a quantidade incorreta de parâmetros:

```
const r = new Pessoa();
const s = new Pessoa("João");
const t = new Pessoa("Lucas",25,"Souza");
```

Obrigatoriamente, as propriedades precisam ser inicializadas ao construir o objeto. No exemplo a seguir, como a classe `Cliente` não possui construtor explícito, então as propriedades precisaram ser inicializadas na definição da classe e, desta forma, os objetos que estão nas variáveis `um` e `dois` serão inicializados com `"Ana"` e `18`. Já na classe `Individuo` as propriedades são inicializadas no construtor, desta forma, os objetos que estão nas variáveis `tres` e `quatro` serão inicializados com os valores recebidos como parâmetro pelo construtor.

```
class Cliente {
    nome:string = "Ana";
    idade:number = 18;
}

const um = new Cliente();
const dois = new Cliente();
```

```
class Individuo {
    nome:string;
    idade:number;

    constructor(a:string, b:number){
        this.nome = a;
        this.idade = b;
    }
}

const tres = new Individuo("Ana",18);
const quatro = new Individuo("Pedro",20);
```

Observações:

- Obrigatoriamente, as propriedades precisam ser inicializadas ao construir o objeto. Porém, podemos desabilitar essa checagem atribuindo `false` na propriedade `"strictPropertyInitialization":false`;
- A seguir tem-se erros cometidos ao criar o construtor:

Errado: o construtor não pode ter anotação de tipo de retorno. Neste exemplo a anotação em amarelo está errada.

```
class Individuo {
    nome:string;
    idade:number;

    constructor(a:string, b:number):void {
        this.nome = a;
        this.idade = b;
    }
}
```

Errado: o construtor não pode ter a instrução return com um valor diferente de `this`. Um construtor retorna por padrão o objeto criado e neste exemplo está retornando uma string.

```
class Individuo {
    nome:string;
    idade:number;

    constructor(a:string, b:number){
        this.nome = a;
        this.idade = b;
        return "oi";
    }
}
```

**Tipo condicional:** no TS, as propriedades precisam ser inicializadas na declaração ou na instanciação (dentro do construtor). Porém, podemos usar o tipo condicional, à direita da propriedade, para indicar que ela não precisa ser inicializada. No exemplo a seguir, as propriedades x e y só serão inicializadas ao chamar os métodos `setX` e `setY`.

```
class Operacao {
    x?: number;
    y?: number;

    setX(x: number) {
        this.x = x;
    }

    setY(y: number) {
        this.y = y;
    }

    somar(): number | undefined {
        if (this.x !== undefined && this.y !== undefined) {
            return this.x + this.y;
        }
        return undefined;
    }
}

const op = new Operacao();
// as propriedades ainda não foram inicializadas
console.log("x + y:", op.somar());
// atribui valores às propriedades x e y
op.setX(5);
op.setY(4);
console.log("x + y:", op.somar());
```

```
PS C:\Desktop\exemplo> npm start
> exemplo@1.0.0 start
> ts-node ./src/index
x + y: undefined
x + y: 9
```

O construtor pode receber parâmetros condicionais quando a propriedade que ele inicializa também é condicional. No exemplo a seguir o construtor pode ser invocado passando zero (`new Operacao()`), um (`new Operacao(5)`) ou dois (`new Operacao(5,3)`) parâmetros.

```
class Operacao {
    x?: number;
    y?: number;

    constructor(x?:number, y?:number){
        this.x = x;
        this.y = y;
    }

    somar(): number | undefined {
        if (this.x !== undefined && this.y !== undefined) {
            return this.x + this.y;
        }
```

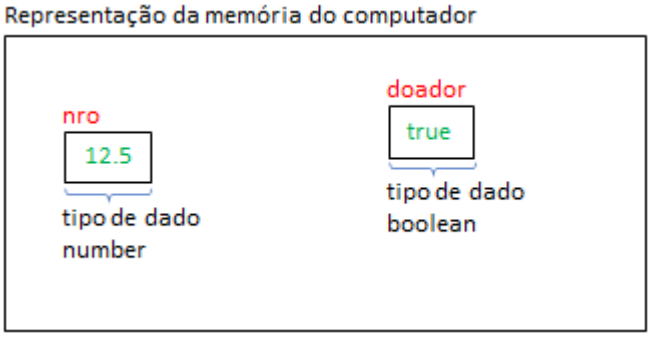
```
PS C:\Desktop\exemplo> npm start
> exemplo@1.0.0 start
> ts-node ./src/index
x + y: undefined
x + y: undefined
x + y: 8
```

```
        return undefined;
    }
}

const a = new Operacao();
console.log("x + y:", a.somar());
const b = new Operacao(5);
console.log("x + y:", b.somar());
const c = new Operacao(5,3);
console.log("x + y:", c.somar());
```

### III. Diferença entre tipos primitivos e objetos

Os tipos de dados primitivos são os tipos básicos e fundamentais disponíveis em uma linguagem de programação. Eles representam valores simples e imutáveis, como exemplo, as variáveis `nro` e `doador` receberam conteúdos simples, constituídos por uma única parte. O bloco ocupado pela variável na memória do computador é constituído apenas pelo valor:

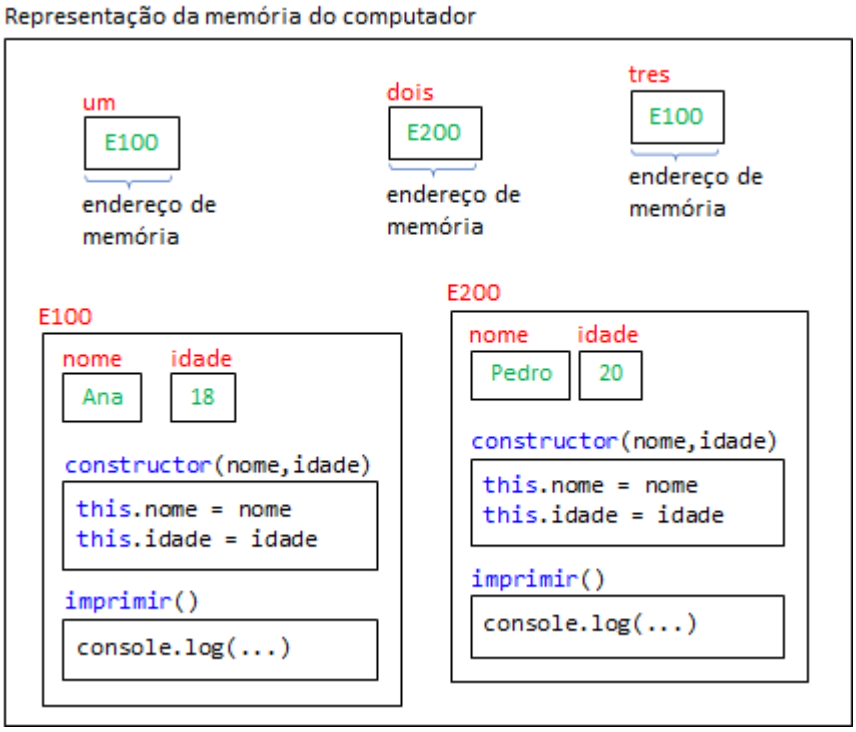
```
let nro = 12.5;
let doador = true;
```




Em contraste, os tipos de dados não primitivos no TS são chamados de tipos de referência. Isso inclui objetos, arrays, funções e outros tipos compostos, que são atribuídos por referência e têm comportamentos mais complexos. Na prática isso tem o seguinte impacto na memória do computador. As variáveis `um` e `dois` recebem o endereço dos objetos na memória do computador. Na ilustração a seguir, `E100` e `E200` são os endereços dos objetos na memória do computador. A variável `tres` recebeu o conteúdo da variável `um`, ou seja, o endereço `E100`. O endereço é uma referência (apontador) para o objeto na memória.

```
class Pessoa {
    nome:string;
    idade:number;

    constructor(nome:string, idade:number){
        this.nome = nome;
        this.idade = idade;
    }

    imprimir():void{
        console.log(`${this.nome} possui ${this.idade} anos`);
    }
}

const um = new Pessoa("Ana",18);
const dois = new Pessoa("Pedro",20);
```

```
const tres = um;
um.imprimir();
dois.imprimir();
tres.imprimir();
```



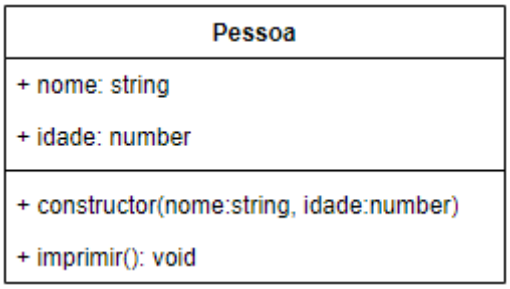

Observe que o objeto é uma cópia da classe na memória do computador.

Embora a string seja um tipo de dado primitivo, o TS fornece recursos adicionais para trabalhar com strings, como operações e métodos específicos disponíveis para manipulação e transformação de strings. Esses recursos facilitam o trabalho com texto e permitem realizar várias operações com eficiência.

## IV. Representação de uma classe no diagrama UML

A UML (Unified Modeling Language - Linguagem de Modelagem Unificada) possui um conjunto de diagramas para representar visões de um software.

O diagrama UML de classes é uma ferramenta visual que mostra a estrutura da classe, suas propriedades/atributos, métodos e relacionamentos com outras classes. A seguir tem-se o diagrama da classe Pessoa (mostrada no último exemplo):




A classe é representada no diagrama UML por retângulo dividido em três partes:

1ª parte: fica o nome da classe;

2ª parte: ficam as propriedades da classe;

3ª parte: ficam os construtores e métodos da classe.

Observação: esse diagrama foi feito usando a ferramenta online draio.io (https://www.drawio.com). Porém, existem outras opções disponíveis numa busca rápida na internet.

### Exercícios

Instruções para criar o projeto e fazer os exercícios:

1. Crie a pasta `aula2` (ou qualquer outro nome sem caracteres especiais) no local de sua preferência do computador;
2. Abra a pasta no VS Code e acesse o terminal do VS Code;
3. No terminal, execute o comando `npm init -y` para criar o arquivo package.json:
4. No terminal, execute o comando `npm i -D ts-node typescript` para instalar os pacotes `ts-node` e `typescript` como dependências de desenvolvimento;
5. Crie uma pasta de nome `src` na raiz do projeto e crie os arquivos dos exercícios nela – assim como é mostrado na figura ao lado;


6. Adicione na propriedade `scripts` do `package.json` os comandos para executar cada exercício. Como exemplo, aqui estão os comandos para executar os dois primeiros exercícios:

```
{
  "name": "aula2",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "um": "ts-node ./src/exercicio1",
    "dois": "ts-node ./src/exercicio2"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "devDependencies": {
    "ts-node": "^10.9.1",
    "typescript": "^5.1.6"
  }
}
```

Observação: se o comando `ts-node` não funcionar, então use o programa `npx` para executar o comando `ts-node`:

```
"scripts": {
  "um": "npx ts-node ./src/exercicio1",
  "dois": "npx ts-node ./src/exercicio2"
},
```

**Exercício 1:** Criar duas instâncias da classe Filme usando os seguintes valores e, na sequência, chamar o método print para exibir o resultado no terminal.

`Exemplo de saída:`

| Título | Duração |
|---|---|
| De volta para o futuro | 116 |
| Matrix | 136 |

```
PS D:\aula2> npm run um
> aula2@1.0.0 um
> ts-node ./src/exercicio1
O filme De volta para o futuro possui 116 min.
O filme Matrix possui 136 min.
```

```
class Filme {
    titulo: string;
    duracao: number;

    constructor(titulo: string, duracao: number) {
        this.titulo = titulo;
        this.duracao = duracao;
    }

    print(): void {
        console.log(`O filme ${this.titulo} possui ${this.duracao} min.`);
    }
}
```

**Exercício 2:** Criar uma instância da classe Retangulo e, na sequência, imprimir no terminal o valor da área e perímetro usando os métodos correspondentes.

Exemplo de saída:

```
PS D:\aula2> npm run dois
> aula2@1.0.0 dois
> ts-node ./src/exercicio2
Área: 6
Perímetro: 10
```

```
class Retangulo {
    base: number;
    altura: number;

    constructor(base: number, altura: number) {
        this.base = base;
        this.altura = altura;
    }

    area(): number {
        return this.base * this.altura;
    }

    perimetro(): number {
        return 2 * this.base + 2 * this.altura;
    }
}
```

**Exercício 3:** Utilize o método get da classe Aleatorio para imprimir 5 números no terminal. Será necessário criar apenas uma instância da classe Aleatorio.

Exemplo de saída:

```
class Aleatorio {
    get(): number {
        return Math.floor(Math.random() * 100 + 1);
    }
}
```

```
PS D:\aula2> npm run tres
> aula2@1.0.0 tres
> ts-node ./src/exercicio3
20
83
24
100
72
```

**Exercício 4:** Criar duas instâncias da classe Carro usando os seguintes valores e, na sequência, chamar o método print para exibir o resultado no terminal.

| Modelo | Marca |
|---|---|
| Gol | VW |
| Uno | Fiat |

Exemplo de saída:

```
PS D:\aula2> npm run quatro
> aula2@1.0.0 quatro
> ts-node ./src/exercicio4
VW Gol
Fiat Uno
```

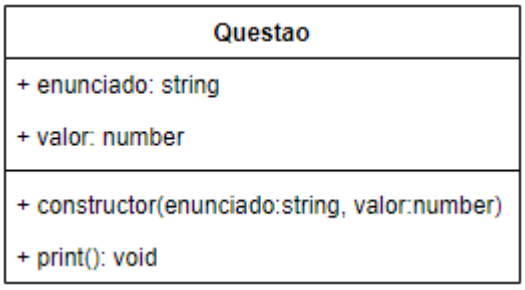
```
class Carro {
    marca: string = "";
    modelo: string = "";

    setMarca(marca: string): void {
        this.marca = marca;
    }

    setModelo(modelo: string): void {
        this.modelo = modelo;
    }

    print(): void {
        console.log(`${this.marca} ${this.modelo}`);
    }
}
```

**Exercício 5:** Codificar a classe Questao representada no diagrama UML a seguir. Para testar a classe, crie o seguinte objeto:

```
const questao = new Questao("O que é um array?", 0.5);
questao.print();
```

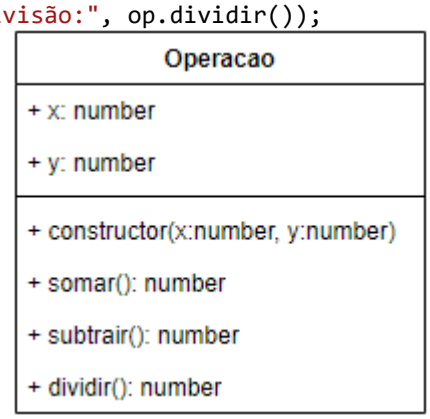
Exemplo de saída:

```
PS D:\aula2> npm run cinco
> aula2@1.0.0 cinco
> ts-node ./src/exercicio5
O que é um array? (0.5 pts.)
```



**Exercício 6:** Codificar a classe Operacao representada no diagrama UML a seguir. Para testar a classe, crie o seguinte objeto:

```
const op = new Operacao(3,5);
console.log("Soma:", op.somar());
console.log("Diferença:", op.subtrair());
console.log("Divisão:", op.dividir());
```

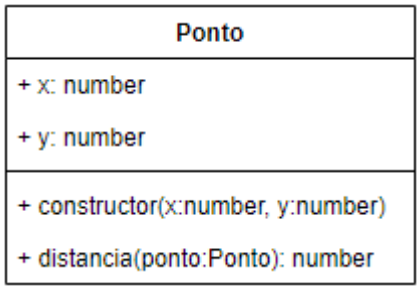



Exemplo de saída:

```
PS D:\aula2> npm run seis
> aula2@1.0.0 seis
> ts-node ./src/exercicio6
Soma: 8
Diferença: -2
Divisão: 0.6
```

**Exercício 7:** Codificar a classe Ponto representada no diagrama UML a seguir. Para testar a classe, crie o seguinte objeto:

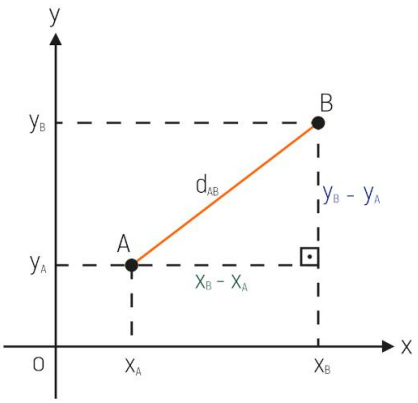
```
const a = new Ponto(3,5);
const b = new Ponto(1,2);
//observe que passamos como parâmetro um objeto do tipo Ponto
console.log("Distância:", a.distancia(b));
```




Exemplo de saída:

```
PS D:\aula2> npm run sete
> aula2@1.0.0 sete
> ts-node ./src/exercicio7
Distância: 3.605551275463989
```

O cálculo da distância entre os pontos A e B é dado pela equação:




$$d_{AB} = \sqrt{(x_B - x_A)^2 + (y_B - y_A)^2}$$

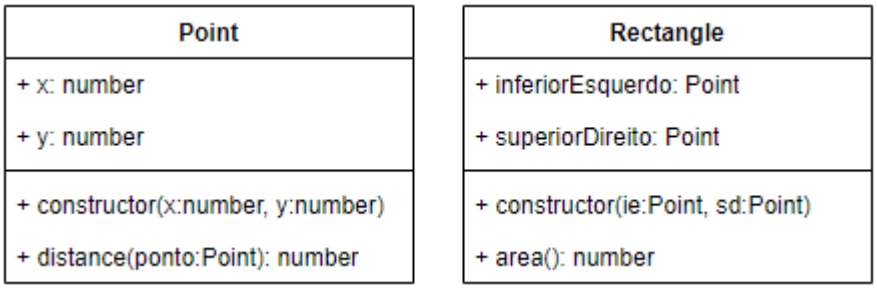
**Exercício 8:** Codificar as classes Point e Rectangle representadas no diagrama UML a seguir. Para testar a classe, crie o seguinte objeto Rectangle:

```
const sd = new Point(3,5);
const ie = new Point(1,2);
```

Exemplo de saída:

```
//observe que o construtor da classe Rectangle
//recebe dois objetos do tipo Point como parâmetro
const r = new Rectangle(ie,sd);
console.log("Área:", r.area());
```

```
PS D:\aula2> npm run oito
> aula2@1.0.0 oito
> ts-node ./src/exercicio8
Área: 6
```




**Exercício 9:** Criar um objeto do tipo Numero e coloque nele 5 números, na sequência, imprima no terminal o somatório e o maior valor.

```
class Numero {
    nros: number[] = [];

    add(nro: number): void {
        //adiciona o nro no final do array
        this.nros.push(nro);
    }

    sum(): number {
        let s = 0;
        for (let i = 0; i < this.nros.length; i++) {
            s += this.nros[i];
        }
        return s;
    }

    max() {
        let maior = this.nros[0];
        for (let i = 0; i < this.nros.length; i++) {
            if (this.nros[i] > maior) {
                maior = this.nros[i];
            }
        }
        return maior;
    }
}
```

Exemplo de saída:

Aqui foram fornecidos os valores: 8, 2, 9, 4 e 5

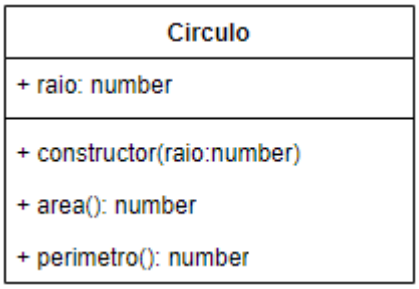
```
PS D:\aula2> npm run nove
> aula2@1.0.0 nove
> ts-node ./src/exercicio9
Somatório: 28
Maior: 9
```

**Exercício 10:** Codificar a classe Circulo representada no diagrama UML a seguir.

Para testar a classe, crie o seguinte objeto:

Exemplo de saída:

```
const circulo = new Circulo(5);
console.log("Área:", circulo.area());
console.log("Perímetro:", circulo.perimetro());
```




```
PS D:\aula2> npm run dez
> aula2@1.0.0 dez
> ts-node ./src/exercicio10

Área: 78.53981633974483
Perímetro: 31.41592653589793
```