

## REVISÃO TYPESCRIPT LÓGICA/POO

- 1) Criando a pasta do projeto:
  - Escolha uma pasta no seu computador para armazenar os códigos.
  - Crie uma subpasta chamada “src”, que irá conter o código fonte.
- 2) Comandos para iniciar o TypeScript no projeto:
  - a. `npm init -y` (criar package.json)
  - b. `npm i -D ts-node typescript` (instalar pacotes ts-node e typescript)
  - c. `tsc --init` (criar tsconfig.json)
- 3) Codificando:
  - a. Crie o arquivo `index.ts` na pasta `src`;

- b. Altere o arquivo package.json conforme abaixo:

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

- c. Dentro do `index.ts` crie a seguinte classe:

Pessoa
+ nome: string
+ email: string
+ constructor(nome: string, email:string)

```
class Pessoa {
  nome:string;
  email:string;

  constructor(nome:string, email:string){
    this.nome = nome;
    this.email = email;
  }
}
```

- d. Vamos testar a classe criada. Transforme o código anterior conforme mostrado abaixo:

```
class Pessoa {  
    nome:string;  
    email:string;  
  
    constructor(nome:string, email:string){  
        this.nome = nome;  
        this.email = email;  
    }  
}
```

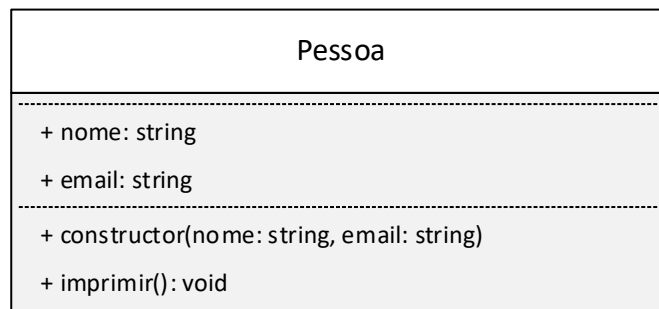
```
const cliente = new Pessoa("Seu nome","Seu email");
```

```
console.log(cliente.nome);  
console.log(cliente.email);
```

- e. Para executar o código digite:

```
npm run index
```

- f. Altere a classe conforme o diagrama de Classes UML a seguir:



O método imprimir() deverá mostrar no console os dados da instância da classe pessoa.

- g. Altere o código conforme mostrado a seguir:

```
class Pessoa {  
    nome:string;  
    email:string;  
  
    constructor(nome:string,email:string){  
        this.nome = nome;
```

```
        this.email = email;
    }

    imprimir():void{
        console.log("Nome: "+this.nome);
        console.log("e-Mail: "+this.email);
    }
}

const cliente = new Pessoa("Seu nome","Seu email");

cliente.imprimir();
```

h. Execute o código novamente.

i. Problema: PO solicitou que fosse calculada a idade de uma pessoa, tendo a data de nascimento informada.

j. Vamos incrementar um pouquinho nossa classe:

Pessoa
+ nome: string
+ email: string
+ nasc: string
+ constructor(nome:string, email:string, nasc:string)
+ imprimir(): void

```
class Pessoa {
    nome:string;
    email:string;
    nasc:string;

    constructor(nome:string,email:string,nasc:string){
        this.nome = nome;
        this.email = email;
        this.nasc = nasc;
    }

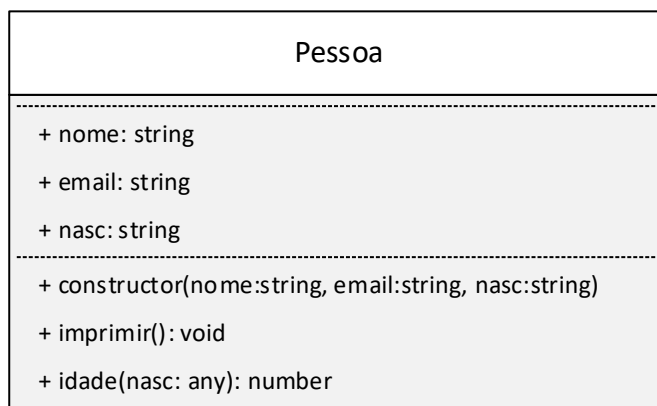
    imprimir():void{
        console.log("Nome: "+this.nome);
        console.log("e-Mail: "+this.email);
        console.log("Data Nasc.: "+this.nasc);
    }
}
```

```
}  
}
```

```
const cliente = new Pessoa("Seu nome","Seu email","Sua data de  
nascimento");
```

```
cliente.imprimir();
```

- k. Execute o código novamente para verificar se está tudo certo.
- l. Agora criaremos um método chamado “idade” que fará o cálculo da idade da pessoa a partir da data de nascimento informada. Altere a classe pessoa de acordo com o diagrama UML a seguir:



Altere o método “imprimir”, inclua o método “idade” dentro da classe e altere o instanciamento da classe Pessoa , conforme código abaixo:

```
imprimir():void {  
    console.log("Nome: "+this.nome);  
    console.log("e-Mail: "+this.email);  
    console.log("Data Nasc.: "+this.nasc);  
    console.log("Idade: "+this.idade(this.nasc)+" anos");  
}  
  
public idade(nasc: any): number {  
    const hoje = new Date();  
    const ano:number = parseInt(nasc.substring(6,10));  
    const mes:number = parseInt(nasc.substring(3,5))-1;  
    const dia:number = parseInt(nasc.substring(0,2));  
    const datan = new Date(ano,mes,dia);  
    let idade:number = hoje.getFullYear() - datan.getFullYear();  
    const m:number = hoje.getMonth() - datan.getMonth();
```

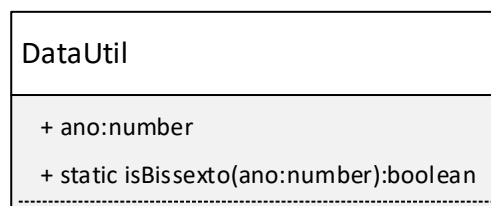
```
        if (m < 0 || (m === 0 && hoje.getDate() < datan.getDate())) {  
            idade--;  
        }  
  
        return idade;  
    }  
}
```

```
const cliente = new Pessoa("Seu nome","Seu email","DD/MM/AAAA");
```

m. Execute o código alterado acima.

n. Problema: PO solicitou que fosse uma rotina que retorne quantos anos bissextos uma Pessoa já viveu, incluindo ano que nasceu e ano atual, caso ainda não tenha feito aniversário.

o. Crie a classe DataUtil de acordo com o diagrama UML a seguir:



### Como calcular um ano bissexto

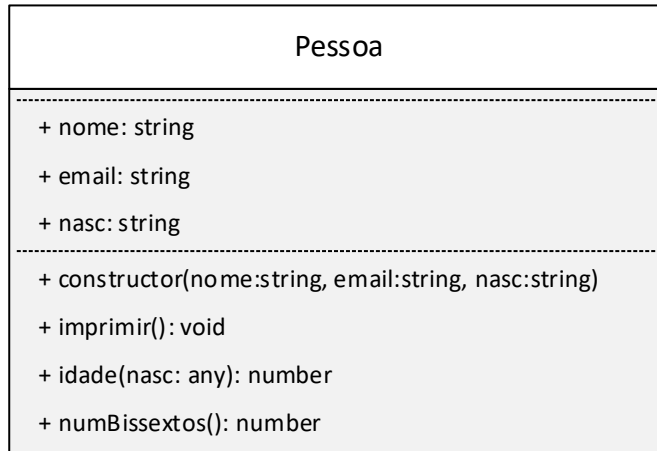
- É múltiplo de 400. Exemplos: 1200, 1600, 2000, 2400, 2800...

- É múltiplo de 4 mas não é de 100. Exemplos: 1996, 2000, 2004, 2008, 2012, 2016...

p. Crie a classe DataUtil com o método estático “isBissexto”:

```
class DataUtil {  
    static isBissexto(ano: number) {  
        if (ano % 400 == 0){  
            return true;  
        } else if (ano % 4 == 0 && ano % 100 != 0){  
            return true;  
        }  
        return false;  
    }  
}
```

- q. Crie o método “numBissextos” conforme diagrama UML a seguir:



```
public numBissextos(){
    const ano: number = parseInt(this.nasc.substring(6, 10));
    const hoje = new Date();
    const anoatual = hoje.getFullYear();
    let quant:number = 0;

    for (let x=ano;x<=anoatual;x++){
        if (DataUtil.isBissexto(x)){
//            console.log(x);
            quant++
        }
    }
    return quant;
}
```

- r. Altere o método “imprimir” da classe Pessoa e inclua a seguinte linha de código:

```
console.log("Anos Bissextos: "+this.numBissextos());
```

- s. Execute o código para verificar se está tudo certo.
- t. Caso queira verificar quais anos que a pessoa viveu e que sejam bissextos, retire os comentários da seguinte linha, dentro do laço “for”:

```
console.log(x);
```

u. Agora o PO quer que a classe contenha uma rotina que além da idade informe também a faixa etária da pessoa, baseada na tabela a seguir:

- Menor de 13 anos: Criança
- Menor de 18 anos: Adolescente
- Menor de 60 anos: Adulto
- Menor de 100 anos: Idoso
- Maior de 100: Matusalém

v. Mudemos a UML para conter um método faixaEtaria que retorne a faixa etária de uma pessoa, conforme tabela acima.

Pessoa
+ nome: string + email: string + nasc: string
+ constructor(nome:string, email:string, nasc:string)  + imprimir():void + idade(nasc:any):number + numBissextos():number + faixaEtaria():string

w. Vejamos a seguir o código da implementação do método faixaEtaria:

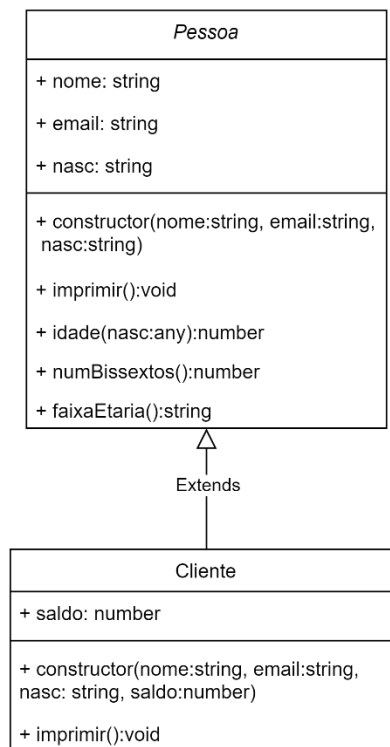
```
public faixaEtaria(): string{  
    if (this.idade()<13){  
        return "Criança";  
    } else if (this.idade()<18){  
        return "Adolescente";  
    } else if (this.idade()<60){  
        return "Adulto";  
    } else if (this.idade()<100){  
        return "Idoso";  
    } else {  
        return "Matusalem";  
    }  
}
```

x. Alteremos o código do método imprimir para incluir a Faixa Etária:

```
imprimir(): void {  
    console.log("Nome: ", this.nome);
```

```
console.log("e-Mail: ", this.email);
console.log("Data Nasc.: ", this.nasc.toLocaleString('pt-BR'));
console.log("Idade: ", this.idade() + " anos");
console.log("Faixa Etária: ", this.faixaEtaria());
console.log("Anos Bissextos: ", this.numBissextos());
}
```

y. O PO solicitou agora que implementemos uma classe filha Cliente conforme UML a seguir:



z. Vejamos a seguir a implementação da classe Cliente:

```
import Pessoa from "./Pessoa";
```

```
class Cliente extends Pessoa{
```

```
    saldo: number;
```

```
    constructor(nome: string, email: string, nasc: Date, saldo: number) {
        super(nome, email, nasc);
        this.saldo = saldo;
    }
```

```
    public imprimir():void{
        super.imprimir();
    }
```



```
        console.log("Saldo: ", this.saldo);  
    }  
}
```

```
export default Cliente;
```

aa. Por fim, iremos instanciar um objeto da classe Cliente e rodar o método imprimir() resultante:

```
import Cliente from "./Cliente";
```

```
const nasc = new Date("11 01 1965");;
```

```
const cliente = new Cliente("Fulano", "fulano@gmail.com", nasc, 500);
```

```
cliente.imprimir();
```