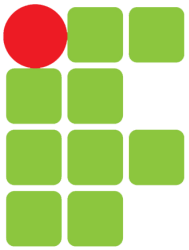

INSTITUTO FEDERAL DE
EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
PIAUÍ

INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DO PIAUÍ
TECNOLOGIA EM ANÁLISE E DESENVOLVIMENTO DE SISTEMAS
MÓDULO II - PROGRAMAÇÃO ORIENTADA A OBJETOS
PROF.: Ely Miranda
ALUNO: Lucas Gomes de Oliveira (20191ADS0185)

Exercício 03

1. Suponha uma classe Hotel que sirva apenas para guardar a quantidade de solicitações de reservas feitas conforme abaixo:

```
class Hotel {
 quantReservas : number;
 adicionarReserva() : void {
 quantReservas++;
 }
}
```

Podemos afirmar que haverá um problema de compilação, pois a variável inteira não foi inicializada previamente? Justifique.

Não é possível realizar a operação de incremento do atributo justamente por não ter sido inicializado ou por não possuir um construtor. Ademais, mesmo que fosse inicializada, seria necessário chamar o atributo dentro do método da seguinte forma: this.`quantReservas`.

2. Ainda sobre a classe do exemplo anterior, considere o método main abaixo:

```
let hotel : Hotel = new Hotel(2);
console.log(hotel.quantReservas);
```

Adicione o construtor que aceite um parâmetro inteiro e faça a inicialização do atributo quantReservas.

```
class Hotel {

    quantReservas: number;
```

```
    constructor(n: number) {
        this.quantReservas = n;
    }

    adcionarReserva(): void {
        this.quantReservas++;
    }
}

let hotel : Hotel = new Hotel(2);
console.log(hotel.quantReservas);
```

3. Considere a classe Radio e as instruções fazem seu uso abaixo::

```
class Radio {
 volume : number;
 constructor(volume : number) {
 this.volume = volume;
 }
}
let r : Radio = new Radio();
r.volume = 10;
```

Justifique o erro de compilação e proponha uma solução.

Descrição do erro: Expected 1 arguments, but got 0. Significa que nenhum argumento não foi providenciado para a classe durante a sua instância. Para o tratamento há duas soluções possíveis.

Primeira solução

```
class Radio {

    volume : number = 0;

}



let r : Radio = new Radio();

r.volume = 10;
```

Segunda solução

```
class Radio {

    volume : number;



    constructor(volume : number) {

        this.volume = volume;

    }

}



let r : Radio = new Radio(3);

r.volume = 10;
```

4. Considerando o uso da classe Conta apresentada em aula e seu uso abaixo:

```
let c1 : Conta = new Conta("1",100);
 let c2 : Conta = new Conta("2",100);
 c1 = c2;
 c1.sacar(10);
 c1.tranferir(c2,50);
```

```
  console.log(c1.saldo);
  console.log(c2.saldo);
```

a. Qual o resultado dos dois "prints"? Justifique sua resposta.

O resultado de ambos os prints é 90.

b. O que acontece com o objeto para o qual a referência c1 aponta?

O que acontece é que, pelo fato de c1 estar apontando para o mesmo endereço de memória de c2, ambos apresentaram as mesmas características. Em suma, o que ocorrerá em um objeto irá influenciar no outro.

```
class Conta {
    numero: String;
    saldo: number;
    //construtor
    constructor(num: String, saldoInicial: number) {
        this.numero = num;
        this.saldo = saldoInicial;
    }

    sacar(valor: number): void {
        this.saldo = this.saldo - valor;
    }

    depositar(valor: number): void {
        this.saldo = this.saldo + valor;
    }

    consultarSaldo(): number {
        return this.saldo;
    }

    transferir(contaDestino: Conta, valor: number): void {
        this.sacar(valor);
        contaDestino.depositar(valor);
    }
}

let c1 : Conta = new Conta("1",100);
let c2 : Conta = new Conta("2",100);
```

```
c1 = c2;
c1.sacar(10);
c1.transferir(c2, 50);

console.log(c1.saldo);
console.log(c2.saldo);
```

5. Crie uma classe chamada Jogador e nela:
    a. Crie 3 atributos inteiros representando força, nível e pontos atuais;
    b. Crie um construtor no qual os 3 parâmetros são passados e inicialize os respectivos atributos;
    c. Crie um método que calcule os pontos relativos a um ataque que são calculados pela multiplicação de força pelo nível;
    d. Crie um método chamado atacar em que é passado um outro jogador como parâmetro e é feita a subtração de pontos de tal jogador baseado na quantidade de pontos do jogador atual (“this”).
    e. Avalie em com testes dois jogadores instanciados e inicializados através do construtor. Utilize o método de ataque de cada jogador e ao final, verifique qual jogador tem mais pontos.

```
class Jogador {

    forca: number;

    nivel: number;

    pontos: number;


    constructor(f: number, n: number, p: number) {

        this.forca = f;

        this.nivel = n;

        this.pontos = p;

    }


    calculaPontos(): number {

        return this.forca * this.nivel;

    }
```

```
    atacar(objeto: Jogador): any {
        if(this.pontos > 0) {
            return this.pontos--;
        }
    }
}

let j1: Jogador = new Jogador(12, 5, 10);
let j2: Jogador = new Jogador(6, 2, 20);

// calculo de pontos relativos
console.log(`Primeiro jogador: ${j1.calculaPontos()}`);
console.log(`Segundo jogador: ${j2.calculaPontos()}`);

// ataque
console.log(`Primeiro pontos atuais: ${j1.atacar(j2)}`);
console.log(`Segundo pontos atuais: ${j2.atacar(j1)}`);
```

6. Altere a classe conta dos slides conforme as instruções abaixo:
    a. Altere o método sacar de forma que ele retorne verdadeiro ou falso. Caso o saque deixe saldo negativo, o mesmo não será realizado, retornando falso;
    b. Altere o método transferir() para que o mesmo use os métodos sacar() e depositar(). Visto pelo prisma da "proteção do saldo", chamar outros métodos em vez de acessar o saldo diretamente é mais seguro?
    c. Altere o método transferir() para que retorne também um valor lógico e que não seja feita a transferência caso o sacar() na conta origem não seja satisfeito;
    d. Verifique as diferentes operações implementadas.

A chamada de outros métodos ajuda a regulamentar a manutenção do código.

```
class Conta {
    numero: String;
```

```
    saldo: number;

    //construtor
    constructor(num: String, saldoInicial: number) {
        this.numero = num;
        this.saldo = saldoInicial;
    }

    sacar(valor: number): any {
        this.saldo = this.saldo - valor;

        if (this.saldo <= 0) {
            console.log('Impossivel realizar a operacao! Saldo muito
baixo.');
        }
    }

    depositar(valor: number): void {
        this.saldo = this.saldo + valor;
    }

    consultarSaldo(): number {
        return this.saldo;
    }

    transferir(contaDestino: Conta, valor: number): any {
        if (this.saldo <= 0) {
            console.log('Impossivel realizar a operacao! Saldo muito
baixo.')
        } else {
```

```
            this.sacar(valor);
            contaDestino.depositar(valor);
        }
    }
}

let c1 : Conta = new Conta("1", 9);
let c2 : Conta = new Conta("2", 100);

c1.sacar(10);
c1.transferir(c2, 50);

console.log(c1.saldo);
console.log(c2.saldo);
```

7. Crie uma classe chamada Produto e nela:
    a. Crie os atributos codigo, descricao, valor e um construtor que os inicialize;
    b. Crie os métodos baixar(quantidade : number) e repor(quantidade : number) que reduzem e incrementam a quantidade disponível do produto;
    c. Crie um atributo quantidadeMinima e reescreva o método baixar para que não seja possível realizar a baixa caso a operação deixe a quantidade abaixo da quantidade mínima;
    d. Crie um método da classe Produto chamado reajusta(taxa : number) que reajusta em x% o valor do produto.
    e. Crie um método chamado toString() que retorna a representação textual do produto concatenando todos os atributos.
    f. Crie um método equals(Produto produto) que retorna true ou false se o produto passado como parâmetro possui o mesmo código;
    g. Verifique as diferentes operações implementadas com testes.

```
class Produto {
    codigo: number;
```

```
    descricao: String;
    valor: number;

    constructor(c: number, d: String, v: number) {
        this.codigo = c;
        this.descricao = d;
        this.valor = v;
    }

    baixar(quantidade: number): number {
        return quantidade--;
    }

    repor(quantidade: number): number {
        return quantidade++;
    }

    quantidadeMinima(): void {
        if (this.baixar() <== 0) {
            console.log('Alerta! Quantidade do produto
indisponivel')
        }
    }

    reajustar(taxa: number) {
        return this.valor * taxa;
    }

    toString(): String {
```

```
        return `${this.codigo} - ${this.descricao} - ${this.valor}`
    }


    equals(produto: Produto): boolean {
        return true;
    }
}
```