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 06

1. As classes `Carro`, `Veiculo` e `CarroEletrico` são bem semelhantes. Refatore as classes para que os atributos duplicados não sejam mais necessários.

public class Veiculo { String placa; int ano; }	public class Carro { String placa; int ano; String modelo; }
public class CarroEletrico { String placa; int ano; String modelo; int autonomiaBateria }	```class Veiculo { placa: string = '0000000'; ano: number = 0; } class Foo extends Veiculo { modelo: string ; autonomiaBateria: number; constructor(modelo: string, autonomia: number) { super(); this.modelo = modelo; this.autonomiaBateria = autonomia; } }```

2. Crie uma classe `Calculadora` com:

a. Dois tributos privados chamados `_op1` e `_op2`;

b. Crie um construtor que inicializa os atributos;

c. Crie um método chamado adicionar que retorna a soma dos dois atributos;

d. Teste a classe.

```
class Calcuradora {
    private _operando1: number
    private _operando2: number

    constructor(x: number, y: number) {
        this._operando1 = x;
        this._operando2 = y;
    }

    adicionar(): number {
        return this._operando1 + this._operando2;
    }
}

//Teste
let operacao = new Calcuradora(1989, 65);

console.log(`Retorno da soma => ${operacao.adicionar()}`);
```

3. Crie uma classe chamada `CalculadoraCientifica` que herda da classe `Calculadora` do exercício passado e:

a. Implemente um método chamado `exponenciar` que retorne o `_op1` elevado ao `_op2`;

b. Teste a classe;

c. Foi necessária alguma modificação em `Calculadora` para o acesso aos atributos?

```
class Calcuradora {
    private _operando1: number
    private _operando2: number

    constructor(x: number, y: number) {
```

```
        this._operando1 = x;
        this._operando2 = y;
    }

    get getOperando1() {
        return this._operando1;
    }

    get getOperando2() {
        return this._operando2;
    }

    adicionar(): number {
        return this._operando1 + this._operando2;
    }
}

class CalcuradoraCientifica extends Calcuradora {
    constructor(x: number, y: number) {
        super(x, y);
    }

    exponenciar(): number {
        return Math.pow(
            this.getOperando1, this.getOperando2
        );
    }
}

//Teste
let op = new Calcuradora(3, 2); //primeira classe
let opCientifica = new CalcuradoraCientifica(3, 2); //segunda classe

console.log(`Retorno da soma => ${op.adicionar()}`);
console.log(`Retorno da exponenciacao => ${opCientifica.exponenciar()}`);
```

Sim. Visando realizar o acesso aos atributos privados da primeira classe (Calculadora), foi preciso alterar esta classe com dois métodos get para que fosse preciso realizar a operação de exponenciação na segunda classe (Calculadora Científica).

4. Implemente na classe Banco o método `renderJuros(numero: String): number`, onde:

a. É passado por parâmetro o número de uma poupança e feita uma consulta para ver se a conta existe. Note que a consulta não se altera sendo Conta ou Poupança;

b. Caso a poupança seja encontrada, teste se realmente se trata de uma poupança com o operador instanceof, desconsidere a operação caso contrário;

c. Caso seja, faça um cast e invoque o método renderJuros da própria instância encontrada;

d. Teste o método da classe Banco passando tanto um número de poupança como de conta passados inseridos anteriormente.

```
class Conta {
    private _numero: String;
    private _saldo: number;

    constructor(numero: String, saldoInicial: number) {
        this._numero = numero;
        this._saldo = saldoInicial;
    }

    sacar(valor: number): void {
        if (this._saldo >= valor) {
            this._saldo = this._saldo - valor;
        }
    }

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

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

    get getNumero(): String {
        return this._numero;
    }

    get getSaldo(): number {
        return this._saldo;
```

```
    }
}

class Poupanca extends Conta {
    private _taxaJuros: number;

    constructor(numero: String, saldo: number, taxaJuros: number ) {
        super(numero, saldo);
        this._taxaJuros = taxaJuros;
    }

    public renderJuros(): void {
        this.depositar(this.getSaldo * this._taxaJuros/100);
    }

    get taxaJuros(): number {
        return this._taxaJuros
    }
}

let objConta: Conta = new Conta("1", 100);
let objPoupanca1: Poupanca = new Poupanca("2", 100, 0.5);
let objPoupanca2: Conta = new Poupanca("3", 100, 1); //polimorfismo

objPoupanca1.renderJuros();
console.log(objPoupanca1.getSaldo);

objPoupanca2.depositar(100);
//let objCast = objPoupanca2 as Poupanca;
//(<Poupanca> poupanca2).renderJuros();
(objPoupanca2 as Poupanca).renderJuros();
console.log(objPoupanca2.getSaldo);

class Banco {
    private _contas: Conta[] = [];

    inserir(conta: Conta): any {
        let contaConsultada = this.consultar(conta.getNumero);

        if (contaConsultada == null) {
```

```
            this._contas.push(conta);
        } else {
            console.log("Impossivel adicionar uma mesma conta");
            console.log(`Operacao => ${conta.getNumero}`);
        }
    }

    consultar(numero: String): Conta {
        let contaConsultada: Conta;
        for (let conta of this._contas) {
            if (conta.getNumero == numero) {
                contaConsultada = conta;
                break;
            }
        }
        return contaConsultada;
    }

    private consultarPorIndice(numero: String): number {
        let indice: number = -1;
        for (let i: number = 0; i < this._contas.length; i++) {
            if (this._contas[i].getNumero == numero) {
                indice = i;
                break;
            }
        }
        return indice;
    }

    alterar(conta: Conta): void {
        let indice: number = this.consultarPorIndice(conta.getNumero);
        if (indice != -1) {
            this._contas[indice] = conta;
        }
    }

    excluir(numero: string): void {
        let indice: number = this.consultarPorIndice(numero);

        if (indice != -1) {
```

```
            for (let i: number = indice; i < this._contas.length; i++) {
                this._contas[i] = this._contas[i+1];
            }

            this._contas.pop();
        }
    }

    depositar(numero: String, valor: number): void {
        let contaConsultada = this.consultar(numero);

        if (contaConsultada != null) {
            contaConsultada.depositar(valor);
        }
    }

    sacar(numero: String, valor: number): void {
        let contaConsultada = this.consultar(numero);

        if (contaConsultada != null) {
            contaConsultada.sacar(valor);
        }
    }

    transferir(numeroCredito: string, numeroDebito: string, valor: number){
        let contaCredito: Conta = this.consultar(numeroCredito);
        let contaDebito: Conta = this.consultar(numeroDebito);

        if (contaCredito != null && contaDebito != null) {
            contaDebito.transferir(contaCredito, valor);
        }
    }

    calcularQuantidadeContas(): number {
        return this._contas.length;
    }

    calcularTotalSaldos(): number {
        let totalSaldo: number = 0;
        for (let conta of this._contas) {
```

```
            totalSaldo += conta.getSaldo;
        }

        return totalSaldo;
    }

    calcularMediaSaldos() {
        return this.calcularTotalSaldos()/this.calcularQuantidadeContas();
    }

    renderJuros(numero: String): any {
        for (let i: number = 0; i < this._contas.length; i++) {
            if (this._contas[i].getNumero === numero) {
                let contaAtual: Conta = this._contas[i];

                if (contaAtual instanceof Poupanca) {
                    (contaAtual as Poupanca).renderJuros();
                }
            }
        }
    }
}

let objetoConta: Conta = new Conta("1", 100)
let objetoBanco: Banco = new Banco();

objetoBanco.inserir(objetoConta);
console.log(`Saldo da primeira conta  => ${objetoConta.getSaldo}`)

// teste
// objetoBanco.inserir(new Conta("1", 150));
/*
objetoBanco.sacar("1", 20);
console.log(objetoConta.getSaldo);

objetoBanco.inserir(new Conta("2", 350));
objetoBanco.inserir(new Conta("3", 1090));
objetoBanco.inserir(new Conta("4", 100));

console.log(objetoBanco.consultar("2").getSaldo);
```

```
objetoBanco.transferir("4", "1", 10);

console.log(objetoBanco.calcularQuantidadeContas());
console.log(objetoBanco.calcularTotalSaldos());
console.log(objetoBanco.calcularMediaSaldos());
*/

// test com a classe Poupanca
// let objPoupanca1: Poupanca = new Poupanca("2", 100, 0.5);
objetoBanco.inserir(new Poupanca("5", 1500, 0.25));
objetoBanco.renderJuros("5");
console.log(objetoBanco.consultar("5").getSaldo);
```

5. Suponha duas classes `Produto` e `ProdutoPerecivel`. Produto tem atributos privados `_id`, `_descricao`, `_quantidade` e `_valor`. Já `ProdutoPerecivel` tem as mesmas características de `Produto`, porém possui a mais um atributo chamado `_dataValidade` (https://www.javatpoint.com/typescript-date-object).

`Produto` possui dois métodos: `repor` e `darBaixa`, onde ambos somam e subtraem uma quantidade passada por parâmetro do atributo `quantidade`. Além disso, um produto perecível possui um método que diz se um produto está válido ou não comparando sua data de validade com a data atual.

Dessa forma implemente:

a. Usando herança, as duas classes `Produto` e `ProdutoPerecivel`;

b. Uma classe chamada `Estoque` que possui um atributo privado que é um array de produtos (`Produto` ou `ProdutoPerecivel`);

c. Métodos para `inserir`, `consultar`, `excluir` produtos na classe estoque;

d. Crie validações para não deixar serem incluídos produtos com mesmo id ou mesmo nome;

e. Os métodos `repor` e `darBaixa`, onde após uma consulta são chamados os métodos da classe produto para finalmente alterar a quantidade;

f. Um método que lista todos os produtos perecíveis vencidos.

```
class Produto {
    private _id: string;
    private _descricao: string;
    private _quantidade: number;
    private _valor: number;

    constructor(id: string, descricao: string, quantidade: number, valor:
number) {
        this._id = id;
```

```
        this._descricao = descricao;
        this._quantidade = quantidade;
        this._valor = valor;
    }

    repor(): void {
        this._quantidade++;
    }

    darBaixa(): void {
        this._quantidade--;
    }

    get getId() {
        return this._id;
    }

    get getQuantidade() {
        return this._quantidade;
    }
}

class ProdutoPerevicel extends Produto {
    private _dataValidade: string;

    constructor(id: string, descricao: string, quantidade: number, valor:
number, dataValidade: string) {
        super(id, descricao, quantidade, valor);
        this._dataValidade = dataValidade;
    }
}

class Estoque {
    private _produtos: Produto[] = [];

    consultar(id: string): Produto {
        let consulta: Produto;

        for (let produto of this._produtos) {
            if (produto.getId === id) {
```

```
            consulta = produto;
            break;
        }
    }
    return consulta;
}

inserir(produto: Produto): void {
    if (this.consultar(produto.getId) == null) {
        this._produtos .push(produto);
    }
}

excluir(produto: Produto): void {
    if  (this.consultar(produto.getId) != null) {
        this._produtos.pop();
    }
}

repor(id: string): void {
    let consulta: Produto;

    for (let produto of this._produtos) {
        if (produto.getId === id) {
            consulta = produto;
            consulta.repor();
        }
    }
}

darBaixa(id: string): void {
    let consulta: Produto;

    for (let produto of this._produtos) {
        if (produto.getId === id) {
            consulta = produto;

            if (consulta.getQuantidade <= 0) {
                consulta.darBaixa();
            }
```

```
            }
        }
    }
}
```