INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA BAHIA Campus Irecê	**Instituto Federal de Educação, Ciência e Tecnologia da Bahia – *Campus* Irecê** **Disciplina: Linguagem Técnica I** **Prof° Jonatas Bastos**

Nome: _____

LISTA DE EXERCÍCIO 7 – Interfaces

1. A sintaxe do uso de interfaces pode parecer muito estranha, à primeira vista.Vamos começar com um exercício para praticar a sintaxe. Crie um projeto interfaces e crie a interface AreaCalculavel:

```
interface AreaCalculavel {

    double calculaArea();
}
```

Queremos criar algumas classes que são `AreaCalculavel`:

```
class Quadrado implements AreaCalculavel {

    private int lado;


    public Quadrado(int lado) {
        this.lado = lado;
    }

    public double calculaArea() {
        return this.lado * this.lado;
    }
}


class Retangulo implements AreaCalculavel {
    private int largura;
    private int altura;

    public Retangulo(int largura, int altura) {
        this.largura = largura;
        this.altura = altura;
    }

    public double calculaArea() {
        return this.largura * this.altura;
    }
}
```

```
class Teste {

    public static void main(String[] args) {
        AreaCalculavel a = new Retangulo(3,2);
        System.out.println(a.calculaArea());
    }
}
```

Opcionalmente, crie a classe `Circulo`:

```
class Circulo implements AreaCalculavel {

    // ... atributos (raio) e métodos (calculaArea)
}
```

Utilize `Math.PI * raio * raio` para calcular a área.

2. Nosso banco precisa tributar dinheiro de alguns bens que nossos clientes possuem. Para isso, vamos criar uma interface no nosso projeto já existente:

```
interface Tributavel {

    double calculaTributos();
}
```

Lemos essa interface da seguinte maneira: “todos que quiserem ser *tributável* precisam saber *calcular tributos*, devolvendo um double”.

Alguns bens são tributáveis e outros não, ContaPoupanca não é tributável, já para ContaCorrente você precisa pagar 1% da conta e o SeguroDeVida tem uma taxa fixa de 42 reais.

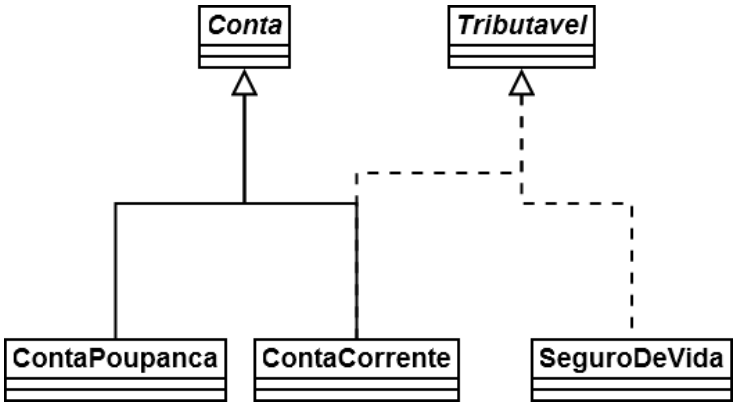
```
class ContaCorrente extends Conta implements Tributavel {


    // outros atributos e metodos

    public double calculaTributos() {
        return this.getSaldo() * 0.01;
    }
}
```

```
class SeguroDeVida implements Tributavel {

    public double calculaTributos() {
        return 42;
    }
}
```



Vamos criar uma classe TestaTributavel com um método main para testar o nosso exemplo:

```
class TestaTributavel {


   public static void main(String[] args) {
       ContaCorrente cc = new ContaCorrente();
       cc.deposita(100);


         System.out.println(cc.calculaTributos());

         // testando polimorfismo:
         Tributavel t = cc;
         System.out.println(t.calculaTributos());
    }
}
```

Tente chamar o método getSaldo através da referência t, o que ocorre? Por quê?

3. Crie um GerenciadorDeImpostoDeRenda, que recebe todos os tributáveis de uma pessoa e soma seus valores, e inclua nele um método para devolver seu total:

```
class GerenciadorDeImpostoDeRenda {

    private double total;

    void adiciona(Tributavel t) {
        System.out.println("Adicionando tributavel: " + t);

        this.total +=  t.calculaTributos();
    }

    public double getTotal() {
        return this.total;
    }
}
```

Crie um main para instanciar diversas classes que implementam Tributavel e passar como argumento para um GerenciadorDeImpostoDeRenda. Repare que você não pode passar qualquer tipo de conta para o método adiciona, apenas a que implementa Tributavel. Além disso, pode passar o SeguroDeVida.

```
public class TestaGerenciadorDeImpostoDeRenda {

    public static void main(String[] args) {

        GerenciadorDeImpostoDeRenda gerenciador = new GerenciadorDeImpostoDeRenda();

        SeguroDeVida sv = new SeguroDeVida();
        gerenciador.adiciona(sv);

        ContaCorrente cc = new ContaCorrente();
        cc.deposita(1000);
        gerenciador.adiciona(cc);

        System.out.println(gerenciador.getTotal());
    }
}
```

Repare que, de dentro do GerenciadorDeImpostoDeRenda, você não pode acessar o método getSaldo, por exemplo, pois você não tem a garantia de que o Tributavel que vai ser passado como argumento tem esse método. A única certeza que você tem é de que esse objeto tem os métodos declarados na interface Tributavel.

É interessante enxergar que as interfaces (como aqui, no caso, Tributavel) costumam ligar classes muito distintas, unindo-as por uma característica que elas tem em comum. No nosso exemplo, SeguroDeVida e ContaCorrente são entidades completamente distintas, porém ambas possuem a característica de serem tributáveis.

Se amanhã o governo começar a tributar até mesmo PlanoDeCapitalizacao, basta que essa classe implemente a interface Tributavel! Repare no grau de desacoplamento que temos: a classe GerenciadorDeImpostoDeRenda nem imagina que vai trabalhar como PlanoDeCapitalizacao. Para ela, o único fato que importa é que o objeto respeite o contrato de um tributável, isso é, a interface Tributavel. Novamente: programe voltado a interface, não a implementação.

4) Use o método printf para imprimir o saldo com exatamente duas casas decimais: System.out.printf("O saldo é: %.2f", conta.getSaldo());

Atenção: os proximos exercícios precisam ser feitos emum projeto à parte conta-interface pois usaremos a Conta como classe em exercícios futuros.

5) Transforme a classe Conta em uma interface.

```
interface Conta {

    double getSaldo();
    void deposita(double valor);
    void retira(double valor);
    void atualiza(double taxaSelic);
}
```

Adapte `ContaCorrente` e `ContaPoupanca` para essa modificação:

```
class ContaCorrente implements Conta {

    // ...
}


class ContaPoupanca implements Conta {
    // ...
}
```

6) Subinterfaces:Às vezes, é interessante criarmos uma interface que herda de outras interfaces.

```
interface ContaTributavel extends Conta, Tributavel {

}
```

Dessa maneira, quem for implementar essa nova interface precisa implementar todos os métodos herdados das suas superinterfaces (e talvez ainda novos métodos declarados dentro dela):

```
 class ContaCorrente implements ContaTributavel {

  // metodos
}

Conta c = new ContaCorrente();
Tributavel t = new ContaCorrente();
```

Repare que o código pode parecer estranho, pois a interface não declara método algum, só herda os métodos

abstratos declarados nas outras interfaces.

Ao mesmo tempo que uma interface pode herdar de mais de uma outra interface, classes só podem possuir uma classe mãe (herança simples).