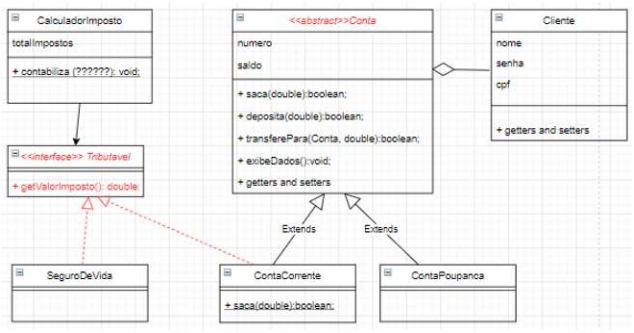
Aula 10 – parte 2
Exercício para ser feito em classe.

1) Importe o projeto fornecido pelo professor.
2) Vamos modificar nosso sistema para ficar parecido com o diagrama de classes abaixo.



3) Nunca haverá um objeto do tipo Conta em memória. Essa classe existe apenas para nos dar polimorfismo. Logo, transforme Conta em classe abstrata.
4) Em seguida, crie a classe SeguroDeVida que deve possuir os atributos numero, valor e titular que é do tipo Cliente. Só deve ser possível instanciar um objeto do tipo SeguroDeVida fornecendo número, valor e titular. Veja como deve ficar essa classe.

```
3  public class SeguroDeVida {
4      private int numero; //Imutável
5      private Cliente titular; //Imutável
6      private double valor;//get e set
7
8      public SeguroDeVida(int numero, double valor, Cliente titular) {
9          this.numero = numero;
10         this.setValor(valor);
11         this.titular = titular;
12     }
13     public double getValor() {
14         return valor;
15     }
16     public void setValor(double valor) {
17         this.valor = valor;
18     }
19     public int getNumero() {
20         return numero;
21     }
22     public Cliente getTitular() {
23         return titular;
24     }
25 }
```

5) Em nosso sistema desejamos contabilizar os impostos sobre ContaCorrente e Seguro de Vida. Para isso precisamos criar uma classe ContabilizadorDeImpostos com um método contabiliza. Esse método deve poder receber os objetos citados acima e acumular em um atributo totalDeImpostos os valores fornecidos por um método getValorImposto contido em ContaCorrente e em SeguroDeVida. O imposto sobre uma ContaCorrente é de 5% do saldo e o imposto sobre SeguroDeVida é de 7% sobre seu valor. Temos 2 problemas a resolver. Precisamos fornecer um mesmo status (polimorfismo) a ContaCorrente e SeguroDeVida e precisamos garantir que ambos possuam um método getValorImposto. Isso te lembra algo? Precisamos de polimorfismo obtido por meio de interface!
6) Crie a interface Tributável conforme código abaixo:

```
3  public interface Tributavel {
4      double getValorDoImposto();
5  }
```

7) Em seguida faça com que ContaCorrente e SguroDeVida implementem esta interface. Assim ambas passarão a poder ser referenciadas como Tributavel. Você também precisará implementar getValorDoImposto em ambas as classes, porém com formas diferentes de calcular esse valor. Veja com devem ficar essas classes.

SeguroDeVida

```java
public class SeguroDeVida implements Tributavel{
    private int numero; //Imutável
    private Cliente titular; //Imutável
    private double valor;//get e set

    public SeguroDeVida(int numero, double valor, Cliente titular) {
        this.numero = numero;
        this.setValor(valor);
        this.titular = titular;
    }
    //Método exigido pela interface
    @Override
    public double getValorDoImposto() {
        return this.valor * 0.07;
    }

    public double getValor() {
        return valor;
    }
    public void setValor(double valor) {
        this.valor = valor;
    }
    public int getNumero() {
        return numero;
    }
    public Cliente getTitular() {
        return titular;
    }
}
```

ContaCorrente

```java
public class ContaCorrente extends Conta implements Tributavel{
    public ContaCorrente(int numero) {
        // Invocando o construtor da superclasse. Construtores NÃO são herdados
        super(numero);
    }
    //Método exigido pela interface
    @Override
    public double getValorDoImposto() {
        return super.saldo * 0.05;
    }
    @Override
    public boolean saca(double valor) {
        if(valor<=0)
            return false;
        return super.saca(valor+0.50);
    }
}
```

8) Em seguida, crie a classe ContabilizadorDeImpostos com o método contabiliza de modo que ele possa receber tanto ContaCorrente quanto SeguroDeVida. Para isso, faça uso do polimorfismo obtido pela implementação da interface Tributavel. Veja.

```java
public class ContabilizadorDeImpostos {
    private double totalDeImpostos;

    public void contabiliza(Tributavel t) {
        this.totalDeImpostos += t.getValorDoImposto();
    }

    public double getTotalDeImpostos() {
        return this.totalDeImpostos;
    }
}
```

9) Agora é hora de testar nosso ContabilizadorDeImpostos. Crie a classe TestaContabilizadorDeImpostos conforme código abaixo.

```java
public class TestaContabilizadorDeImpostos {
    public static void main(String[] args) {
        ContabilizadorDeImpostos contabilizador = new ContabilizadorDeImpostos();

        ContaCorrente cc = new ContaCorrente(1);
        cc.getTitular().setNome("Maria");
        cc.deposita(1000);
        ContaCorrente cc2 = new ContaCorrente(2);
        cc2.getTitular().setNome("Pedro");
        cc2.deposita(2000);
        SeguroDeVida seguro = new SeguroDeVida(5, 1000, new Cliente("Fulano","12345678910"));

        contabilizador.contabiliza(cc);//75,00
        contabilizador.contabiliza(cc2);//125,00
        contabilizador.contabiliza(seguro);//70,00

        System.out.println("Total: "+contabilizador.getTotalDeImpostos());//270
    }
}
```

10) Nosso sistema está pronto e suscetível a modificações futuras. Veja os possíveis cenários e faça os testes que julgar adequados.

a. Nosso sistema agora também precisa contabilizar o imposto de uma ContaPoupança que também é de 5% sobre o saldo. Solução: ContaCorrente não implementa mais Tributável. Conta implementa Tributavel e o método getValorDoImposto (mesma regra).
b. Nosso sistema agora também precisa contabilizar o imposto de uma ContaPoupança que é de 6% sobre o saldo. Solução: Conta implementa Tributavel e cada subclasse fornece sua implementação do método getValorDoImposto (regras distintas).
c. Nosso sistema agora precisa contabilizar o imposto de uma ContaPoupança que é de 6% sobre o saldo, mas não deve mais contabilizar o imposto de uma ContaCorrente. Solução: Conta não implementa mais Tributável. ContaPoupança implementa Tributavel e o método getValorDoImposto.

Em todos os cenários não precisaremos alterar absolutamente nada em ContabilizadorDeImpostos.

Precebam que nosso Sistema ficou altamente flexível e escalável. Cso surja um novo tipo de Conta que seja tributável, não teremos problema nenhum com isso. Não precisaremos alterar absolutamente nada em ContabilizadorDeImpostos.

Esse é o poder do polimorfismo obtido por interface!

Mais exercícios sobre interfaces:

1) Crie um projeto chamado interfaces e crie a interface AreaCalculavel:

```
public interface AreaCalculavel {
        double calculaArea();
}
```

2) Crie algumas classes que são AreaCalculavel:

```
public class Quadrado implements AreaCalculavel {
        private int lado;
        public Quadrado(int lado) {
                this.lado = lado;
        }
        public double calculaArea() {
                return this.lado * this.lado;
        }
}
public class Retangulo implements AreaCalculavel {
        private int largura, altura;

        public Retangulo(int largura, int altura) {
                this.largura = largura;
                this.altura = altura;
        }
        public double calculaArea() {
                return this.altura * this.largura;
        }
}
```

Se você tivesse usado herança, não ia ganhar muito, já que cada implementação é totalmente diferente da outra: Quadrado, Retângulo e Círculo são Figuras Geométricas e poderiam ter uma superclasse com esse nome. O problema é que elas possuem atributos e métodos bem diferentes. O que é comum aqui **é o que elas fazem**. Elas calculam sua Área.

Mesmo que elas tivessem atributos em comum, usar interfaces é uma madeira mais elegante de modelar suas classes. A grande vantagem é o desacoplamento. Herança traz muito acoplamento, o que pode quebrar o encapsulamento, lembra?

3) Crie a classe Circulo:

```
public class Circulo implements AreaCalculavel {
	private double raio;

	public Circulo(double raio) {
		this.raio = raio;
	}

	public double calculaArea() {
		return Math.PI * this.raio;
	}
}
```

4) Crie uma classe Teste. Repare no polimorfismo. Poderíamos estar passando esses objetos como argumento para alguém que aceitasse AreaCalculavel como argumento:

```
public class Teste {

	public static void main(String[] args) {

		AreaCalculavel a = new Retangulo(3, 2);
		System.out.println(a.calculaArea());
	}

}
```

5) Adicione um método imprimirArea à interface AreaCalculavel.
6) Crie uma classe chamada MostradorDeArea. Esse método deve poder receber qualquer das figuras acima e imprimir sua área.
7) Crie uma classe de testes para instanciar e utilizar os objetos criados acima.