Universidade Federal de Ouro Preto

CSI032 - Programação de Computadores II

Polimorfismo: exemplo

Professor: Dr. Rafael Frederico Alexandre
Contato: rfalexandre@decea.ufop.br
Colaboradores: Eduardo Matias Rodrigues
Contato: eduardo.matias@aluno.ufop.edu.br

Polimorfismo: exemplo

Collections

Polimorfismo
Definição

1. Polimorfismo é a capacidade de um objeto poder ser referenciado de várias formas;
2. Em uma das aplicações o polimorfismo permite que diferentes classes tenham métodos com a mesma assinatura, mas com comportamento diferente;

Formas geométricas

Exemplo

1. Queremos escrever um programa que recebe uma forma geométrica de duas dimensões realize o cálculo de sua área e perímetro e exiba no monitor. Para fins de exemplo, temos as classes: Circulo, Triangulo e Losango;

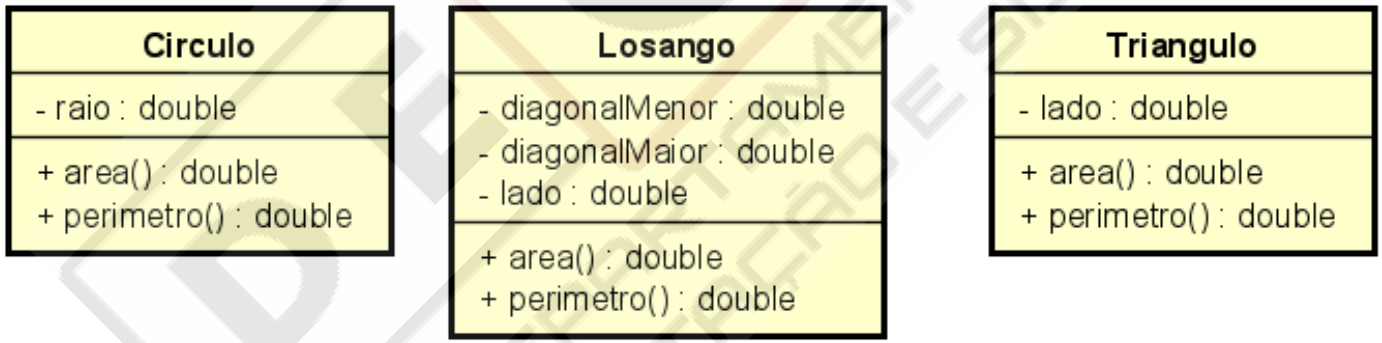


Figura: Formas geométricas bidimensionais

Formas geométricas
Exemplo

1. Agora queremos fazer uma classe para testar os métodos das formas geométricas, para isso, iremos criar instâncias de cada classe e armazenar em vetores de seus respectivos tipos;

Formas geométricas

Círculo

```
public class Circulo {

    private double raio;

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

    public double area() {
        return Math.PI*Math.pow(this.raio, 2);
    }

    public double perimetro() {
        return 2*Math.PI*this.raio;
    }
}
```

Formas geométricas
Triângulo

```
public class Triangulo {

    private double lado;

    public Triangulo(double lado) {
        this.lado = lado;
    }

    public double area() {
        return Math.pow(this.lado, 2)*Math.sqrt(3)/4;
    }

    public double perimetro() {
        return this.lado*3;
    }
}
```

Formas geométricas
Losango

```
public class Losango {

    private double diagonalMaior;
    private double diagonalMenor;
    private double lado;

    public Losango(double diagonalMaior, double diagonalMenor,
            double lado) {
        this.diagonalMaior = diagonalMaior;
        this.diagonalMenor = diagonalMenor;
        this.lado = lado;
    }

    public double area() {
        return (this.diagonalMaior*this.diagonalMenor)/2;
    }

    public double perimetro() {
        return this.lado*4;
    }
}
```

Classe de Teste

Parte 1

```
public static void main(String[] args) {

    Losango[] ls = new Losango[2];
    ls[0] = new Losango(32, 25.5, 18);
    ls[1] = new Losango(10, 7, 5);

    Circulo[] cs = new Circulo[3];
    cs[0] = new Circulo(10);
    cs[1] = new Circulo(5);
    cs[2] = new Circulo(8);

    Triangulo[] ts = new Triangulo[2];
    ts[0] = new Triangulo(25);
    ts[1] = new Triangulo(10);
```

Classe de Teste

Parte 2

```
for (int i = 0; i < ls.length; i++) {
    System.out.println("Forma: Losango");
    System.out.println("Área: " + ls[i].area());
    System.out.println("Perímetro: " + ls[i].perimetro() + "\n");
}

for (int i = 0; i < cs.length; i++) {
    System.out.println("Forma: Círculo");
    System.out.println("Área: " + cs[i].area());
    System.out.println("Perímetro: " + cs[i].perimetro() + "\n");
}

for (int i = 0; i < ts.length; i++) {
    System.out.println("Forma: Triângulo");
    System.out.println("Área: " + ts[i].area());
    System.out.println("Perímetro: " + ts[i].perimetro() + "\n");
}
```

Saída do programa
Parte 1

```
Forma: Losango
Área: 408.0
Perímetro: 72.0

Forma: Losango
Área: 35.0
Perímetro: 20.0

Forma: Círculo
Área: 314.1592653589793
Perímetro: 62.83185307179586
```

Saída do programa
Parte 2

```
Forma: Círculo
Área: 78.53981633974483
Perímetro: 31.41592653589793

Forma: Círculo
Área: 201.06192982974676
Perímetro: 50.26548245743669

Forma: Triângulo
Área: 270.6329386826371
Perímetro: 75.0

Forma: Triângulo
Área: 43.30127018922193
Perímetro: 30.0
```

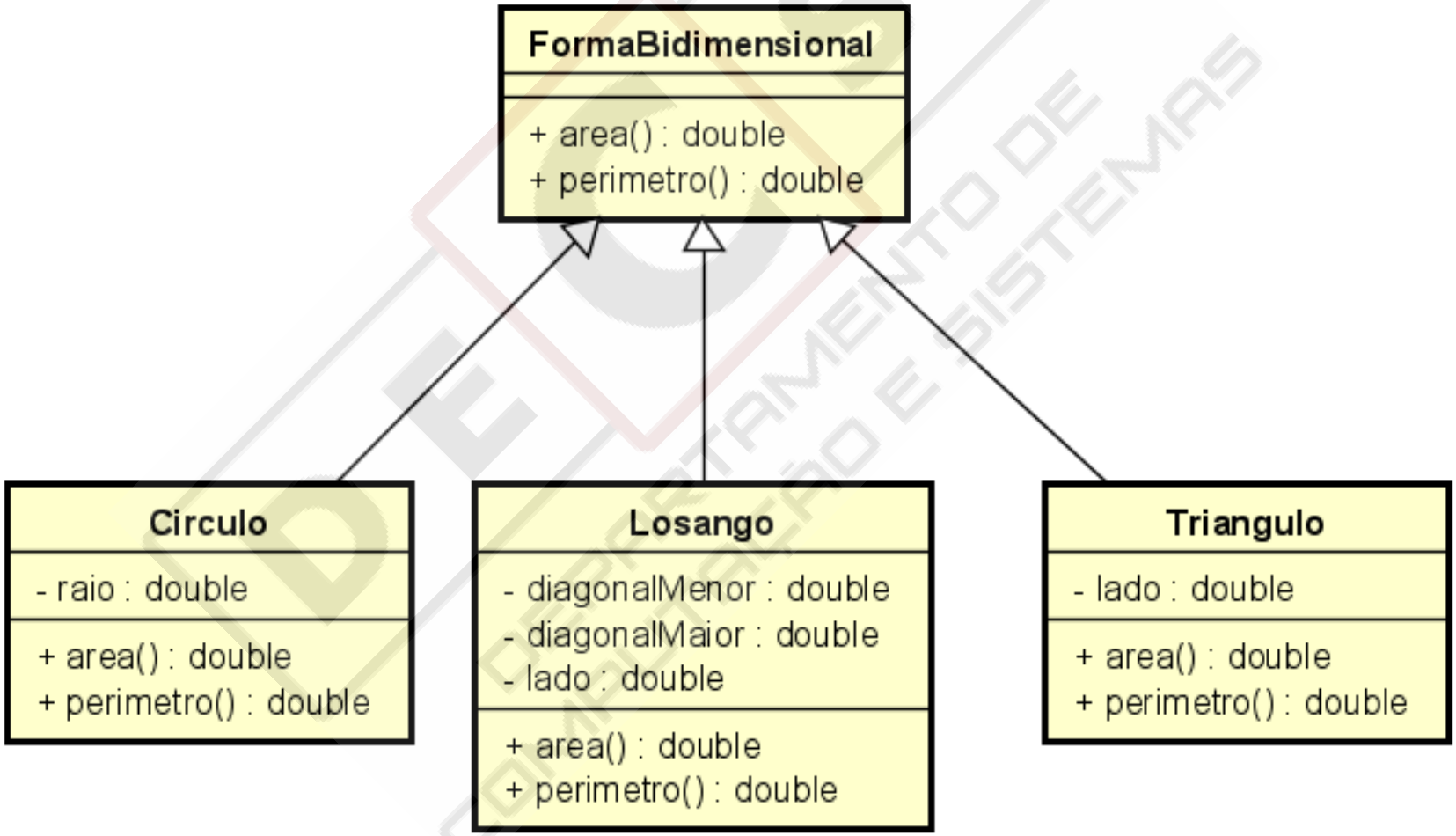
Problema
Código repetido

1. Na classe de teste repetimos o "mesmo"código para cada tipo de forma geométrica criada, se tivessemos mais formas, necessitaria de mais repetição;
2. Todas as nossas classes possuem comportamentos em comum (calculam área e perímetro);
3. e se todas as formas geométricas pudessem ser processadas da mesma maneira?

Problema
Código repetido

1. Podemos utilizar herança, se Circulo, Losango e Triangulo herdam da mesma classe (FormaBidimensional por exemplo) podemos referênciar cada instância delas como sendo uma FormaBidimensional!
    1. Isso é polimorfismo, um objeto da classe Circulo pode ser referênciado como Circulo ou como FormaBidimensional.

Herança
Diagrama de classes



Herança
Classe FormaBidimensional

```
public class FormaBidimensional {

    public FormaBidimensional() {/* */}

    public double area() {
        return 0;
    }

    public double perimetro() {
        return 0;
    }
}
```

Herança
Círculo

```
public class Circulo extends FormaBidimensional {

    private double raio;

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

    @Override
    public double area() {
        return Math.PI*Math.pow(this.raio, 2);
    }

    @Override
    public double perimetro() {
        return 2*Math.PI*this.raio;
    }
}
```

Herança
Triângulo

```
public class Triangulo extends FormaBidimensional {

    private double lado;

    public Triangulo(double lado) {
        this.lado = lado;
    }

    @Override
    public double area() {
        return Math.pow(this.lado, 2)*Math.sqrt(3)/4;
    }

    @Override
    public double perimetro() {
        return this.lado*3;
    }
}
```

Herança
Losango

```
public class Losango extends FormaBidimensional {

    private double diagonalMaior;
    private double diagonalMenor;
    private double lado;

    public Losango(double diagonalMaior, double diagonalMenor,
            double lado) {
        this.diagonalMaior = diagonalMaior;
        this.diagonalMenor = diagonalMenor;
        this.lado = lado;
    }

    @Override
    public double area() {
        return (this.diagonalMaior*this.diagonalMenor)/2;
    }
```

Herança
Formas bidimensionais

1. Ao fazer as classes Circulo, Triangulo e Losango herdarem de FormaBidimensional, adicionamos a notação **@override** em cima dos métodos *area* e *perimetro*;
    1. O *@override* indica que estamos reescrevendo o método da classe FormaBidimensional;
    2. Quando chamarmos o método *area* por exemplo, o método invocado será o implementado na classe filha.

Polimorfismo

Classe de Teste

```
FormaBidimensional[] formas = new FormaBidimensional[7];
formas[0] = new Losango(32, 25.5, 18);
formas[1] = new Losango(10, 7, 5);
formas[2] = new Circulo(10);
formas[3] = new Circulo(5);
formas[4] = new Circulo(8);
formas[5] = new Triangulo(25);
formas[6] = new Triangulo(10);

for (int i = 0; i < formas.length; i++) {
    System.out.println("Forma: " + formas[i].getClass().getSimpleName());
    System.out.println("Área: " + formas[i].area());
    System.out.println("Perímetro: " + formas[i].perimetro() + "\n");
}
```

Polimorfismo

Saída parte 1

```
Forma: Losango
Área: 408.0
Perímetro: 72.0

Forma: Losango
Área: 35.0
Perímetro: 20.0

Forma: Circulo
Área: 314.1592653589793
Perímetro: 62.83185307179586
```

Polimorfismo

Saída parte 2

```
Forma: Circulo
Área: 78.53981633974483
Perímetro: 31.41592653589793

Forma: Circulo
Área: 201.06192982974676
Perímetro: 50.26548245743669

Forma: Triangulo
Área: 270.6329386826371
Perímetro: 75.0

Forma: Triangulo
Área: 43.30127018922193
Perímetro: 30.0
```

Formas Bidimensionais

Polimorfismo

1. Uma vez que Triangulo, Circulo e Losango são herdam de *FormaBidimensional*, podemos referênciar seus objetos como sendo formas bidimensionais!
    1. Se precisarmos acrescentar mais classes de formas bidimensionais, como Quadrado, basta fazê-la herdar de *FormaBidimensional* e podemos processá-la como sendo qualquer outra forma bidimensional do nosso programa!

Herança
Problema

1. Observe que criamos a classe *FormaBidimensional* apenas para podermos tratar suas filhas de forma polimorfica;
2. Seus métodos *area* e *volume* são métodos concretos (possuem implementação) que devem ser reescritos por cada classe filha, já que cada forma geométrica tem seus próprios cálculos de área e volume;
3. É uma má prática criar métodos concretos apenas para podermos utilizar o polimorfismo, um solução melhor é tornar os métodos *area* e *volume* na classe *FormaBidimensional* **abstratos**;
    1. Um método abstrato não possui implementação e deve ser reescrito pelas classes filhas

Collections

Herança
Classe Abstrata

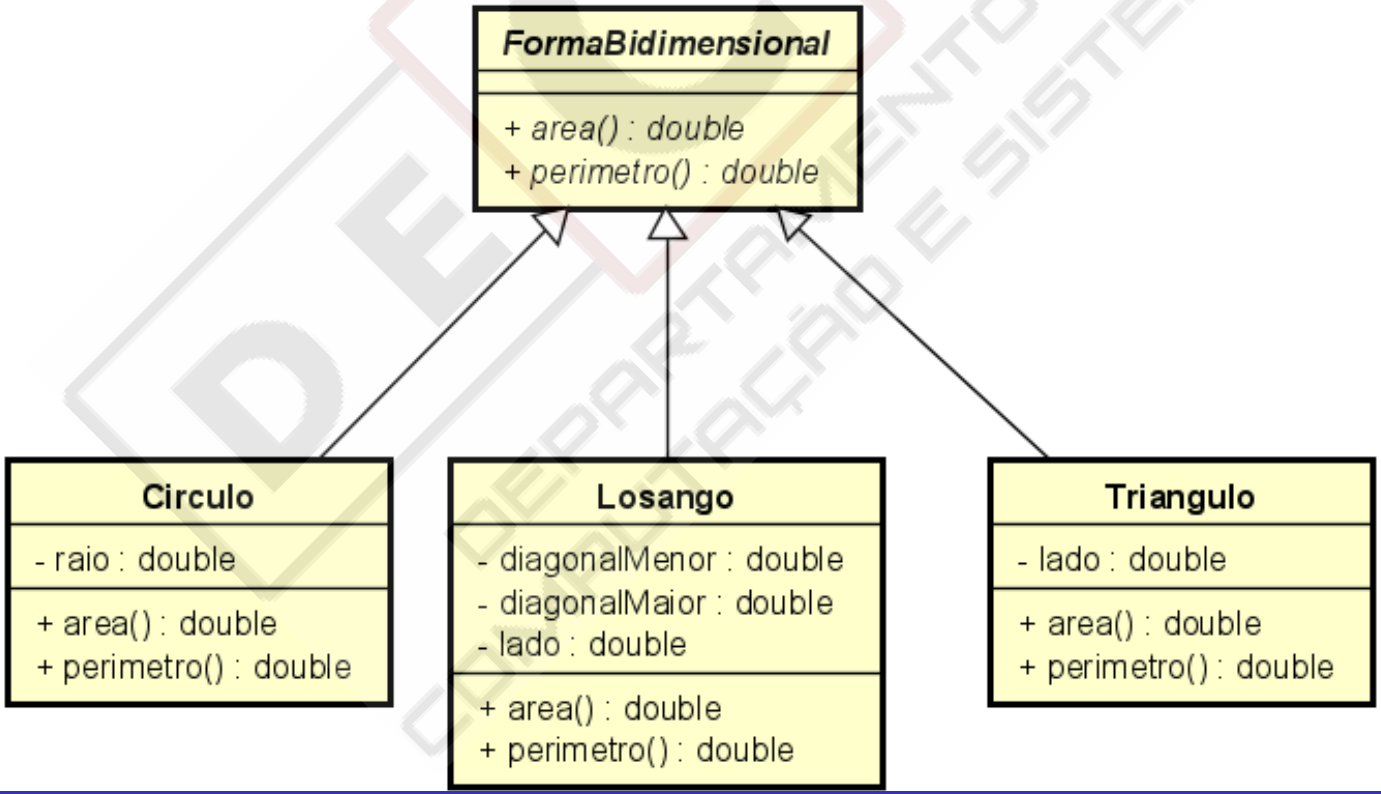
1. Toda classe que possui método abstrato deve ser abstrata (uma classe abstrata não pode ser instânciada);
2. A implementação nas outras classes permanece a mesma.

```
public abstract class FormaBidimensional {

    public abstract double area();

    public abstract double perimetro();
}
```

Classe Abstrata

Diagrama de Classes

1. Uma classe abstrata pode ser representa no diagrama de classes como uma classe com o nome em itálico;
2. Um método abstrato também pode ser representado colocando sua assinatura em itálico.



Collections

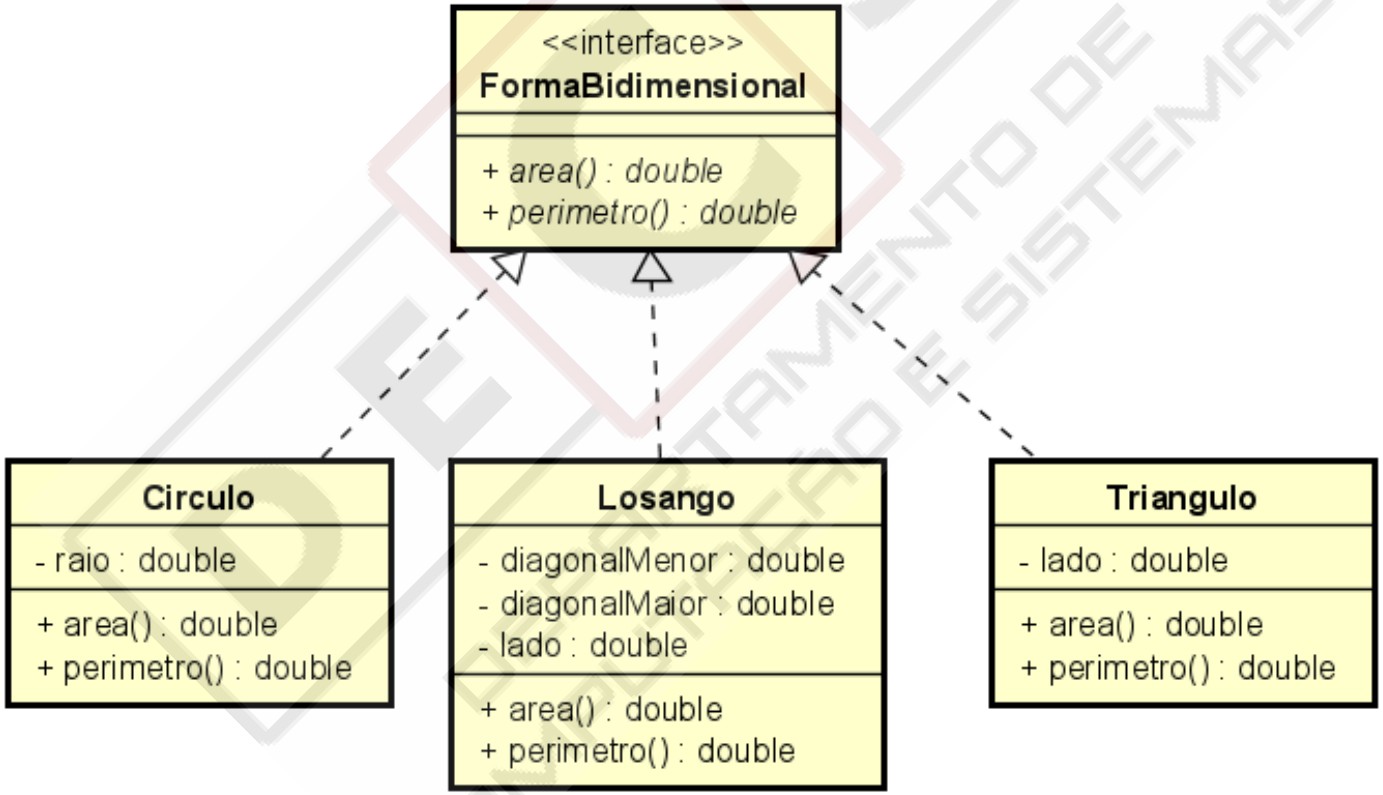
Formas Bidimensionais

Polimorfismo

1. Se uma classe possui apenas métodos abstratos, ela pode ser substituída por uma **Interface**;
    1. Em uma interface seus métodos são implicitamente **public abstract** e seus atributos são implicitamente **public static final**;
    2. Ao implementar uma interface, uma classe concreta assina um contrato (obrigada a implementar todos os métodos definidos na interface).

Formas Bidimensionais
Interface

1. Faremos nossa *FormaBidimensional* ser uma *Interface*;



Contrato
Área e perímetro

1. Ao fazer as classes concretas Circulo, Triangulo e Losango implementar a interface FormaBidimensional, estamos assinando um contrato no qual cada forma geométrica assume a responsabilidade de calcular sua área e perímetro.

FormaBidimensional
interface

```
public interface FormaBidimensional {

    double area();
    double perimetro();
}
```

Círculo

Implementa FormaBidimensional

```
public class Circulo implements FormaBidimensional {

    private double raio;

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

    @Override
    public double area() {
        return Math.PI*Math.pow(this.raio, 2);
    }

    @Override
    public double perimetro() {
        return 2*Math.PI*this.raio;
    }
}
```

Triangulo

Implementa FormaBidimensional

```
public class Triangulo implements FormaBidimensional {

    private double lado;

    public Triangulo(double lado) {
        this.lado = lado;
    }

    @Override
    public double area() {
        return Math.pow(this.lado, 2)*Math.sqrt(3)/4;
    }

    @Override
    public double perimetro() {
        return this.lado*3;
    }
}
```

Losango
Implementa FormaBidimensional

```
public class Losango implements FormaBidimensional {

    private double diagonalMaior;
    private double diagonalMenor;
    private double lado;

    public Losango(double diagonalMaior, double diagonalMenor,
            double lado) {
        this.diagonalMaior = diagonalMaior;
        this.diagonalMenor = diagonalMenor;
        this.lado = lado;
    }

    @Override
    public double area() {
        return (this.diagonalMaior*this.diagonalMenor)/2;
    }

    @Override
    public double perimetro() {
        return this.lado*4;
    }
```

Interface
Polimorfismo

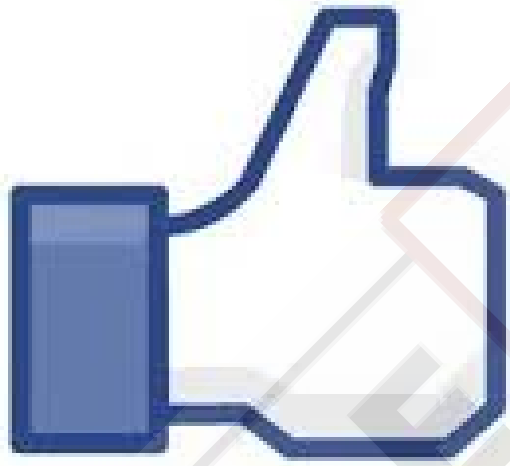
1. Assim como em classes, Interfaces nos permite programar de forma polimorfica;
2. Podemos trabalhar com as classes Circulo, Losango e Triangulo como sendo uma *FormaGeometrica*;

Polimorfismo
Classe de Teste

```
public static void main(String[] args) {

    FormaBidimensional[] formas = new FormaBidimensional[7];
    formas[0] = new Losango(32, 25.5, 18);
    formas[1] = new Losango(10, 7, 5);
    formas[2] = new Circulo(10);
    formas[3] = new Circulo(5);
    formas[4] = new Circulo(8);
    formas[5] = new Triangulo(25);
    formas[6] = new Triangulo(10);

    printFormas(formas);
}

static void printFormas(FormaBidimensional[] formas) {
    for (int i = 0; i < formas.length; i++) {
        System.out.println("Forma: " + formas[i].getClass().getSimpleName());
        System.out.println("Área: " + formas[i].area());
        System.out.println("Perímetro: " + formas[i].perimetro() + "\n");
    }
}
```

Polimorfismo
Classe de Teste

1. Caso não estivessemos programando de forma polimorfica, teríamos que ter além dos 3 vetores (um para cada tipo de objeto), 3 métodos *printFormas*, um que recebe cada tipo de forma geométrica;
2. Sempre opte por trabalhar da forma mais genérica possível, deixa o código escalável e melhora a sua manutenibilidade.



MUITO OBRIGADO!!!