



# Interface

---

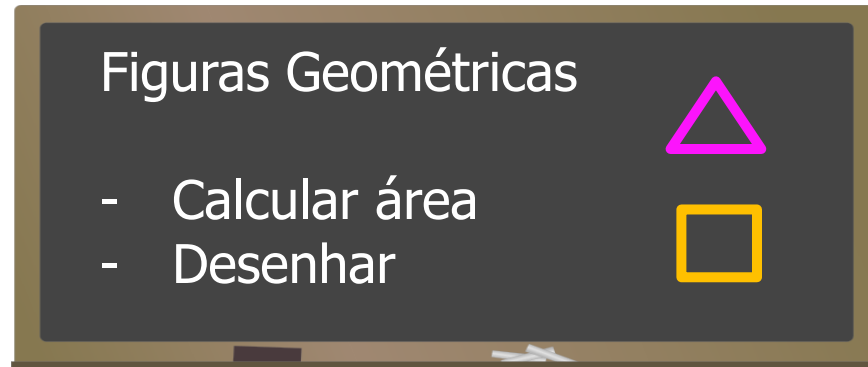
Prof<sup>a</sup>. Rachel Reis  
rachel@inf.ufpr.br

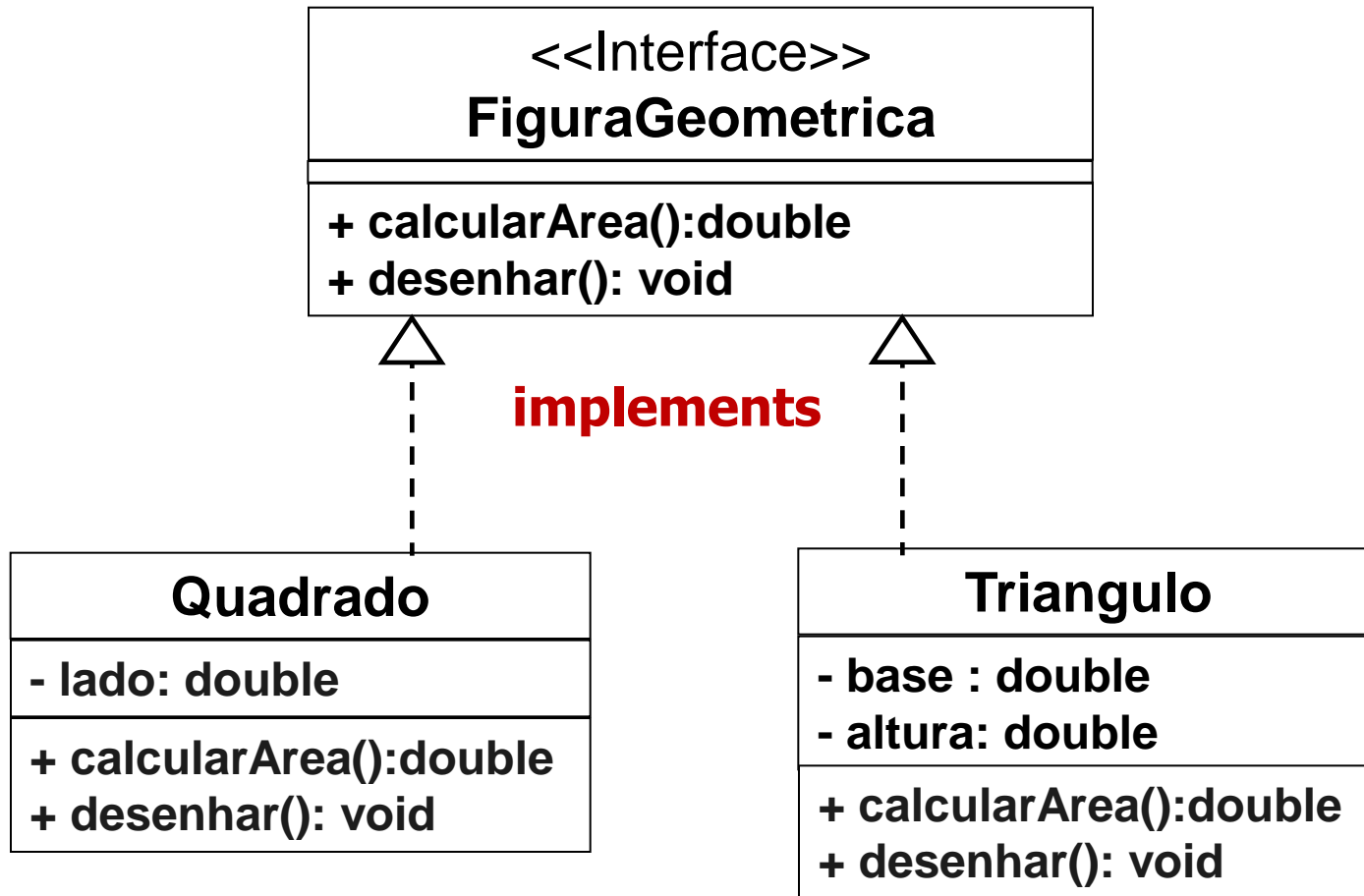


# Problema

---

- Um professor de matemática deseja implementar uma aplicação em Java, para representar figuras geométricas. Para cada figura geométrica, o sistema deverá ser capaz de calcular a área e desenhar.







# Interface – o que é?

---

- Define um conjunto de métodos
  - Uma interface define um conjunto de métodos que uma classe deve implementar, mas não define **como** esses métodos devem ser implementados.
- Exemplos de métodos na interface:

```
public abstract double calcularArea();
```

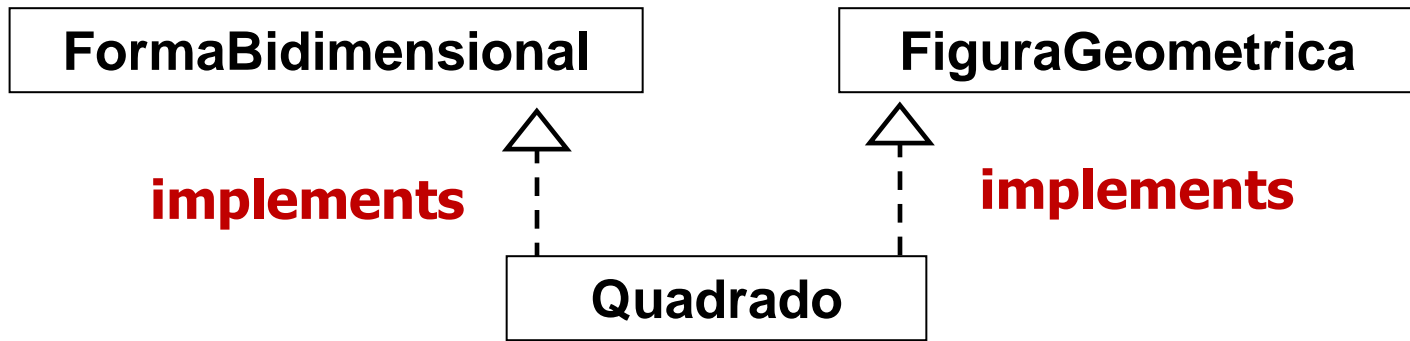
```
public abstract void desenhar();
```



# Herança Múltipla

---

- Como a *herança múltipla* não é permitida em Java, a linguagem oferece o conceito de interface como opção.
- Exemplo:





# Características das Interfaces

---

- Não possuem atributos.
- As constantes são implicitamente definidas como *public*, *static* e *final*.
- Exemplos de declaração da constante PI:

```
double PI = 3.1415;
```

```
public double PI = 3.1415;
```

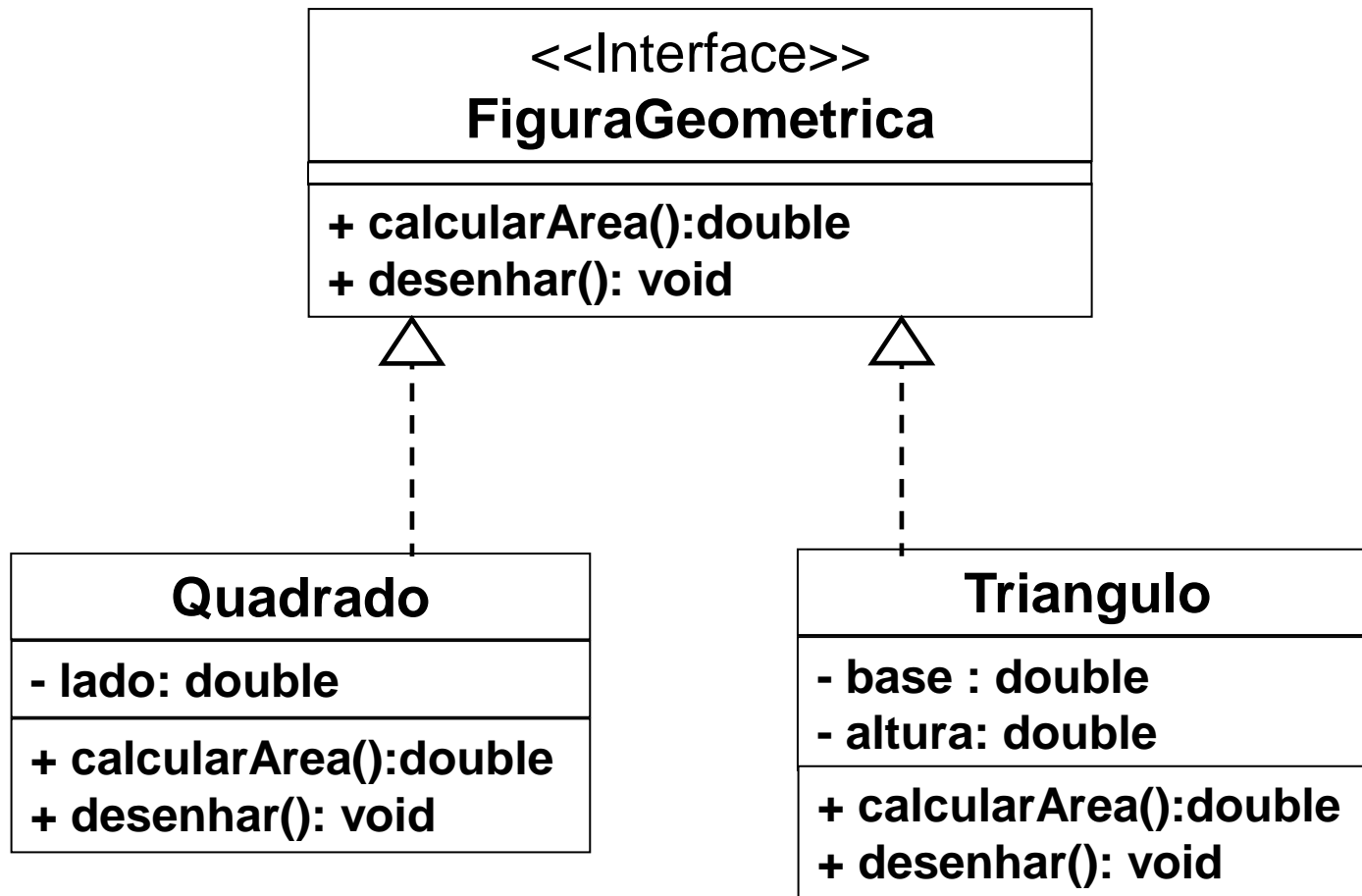
```
public static final double PI = 3.1415;
```



# Características das Interfaces

---

- Não possuem construtor.
- Todos os métodos são implicitamente *public* e *abstract*.
- Não são declaradas como *class*, mas como *interface*



- Vamos ver a implementação...



```
public interface FiguraGeometrica
{

    public abstract double calcularArea() ;
    public abstract void desenhar() ;

}
```

<b>&lt;&lt;Interface&gt;&gt;</b> <b>FiguraGeometrica</b>
<b>+ calcularArea():double</b> <b>+ desenhar(): void</b>

```
public class Quadrado implements FiguraGeometrica
{
    // Atributo
    private double lado;

    // Métodos get e set

    // Outros métodos
    public double calcularArea() {
        return lado * lado;
    }

    public void desenhar() {
        // Código para desenhar quadrado
    }
}
```

Quadrado
- lado: double
+ calcularArea():double + desenhar(): void

```
public class Triangulo implements FiguraGeometrica
{
    // Atributo
    private double base;
    private double altura;


    // Métodos get e set

    // Outros métodos
    public double calcularArea() {
        return base * altura;
    }

    public void desenhar() {
        // Código para desenhar triângulo
    }
}
```

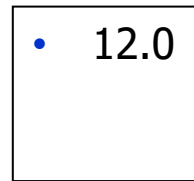
Triangulo
- base : double - altura: double
+ calcularArea():double + desenhar(): void

```
public class Principal{  
    public static void main(String args[]){  
        FiguraGeometrica f1 = new FiguraGeometrica();  
    }  
}
```

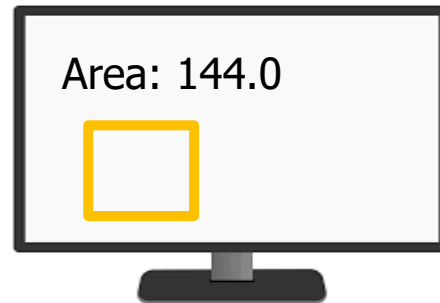


Não é possível instanciar uma interface.

```
public class Principal{  
    public static void main(String args[])  
    {  
        Quadrado f1 = new Quadrado();  
        f1.setLado(12.0);  
        S.o.p("Area: " + f1.calcularArea());  
        f1.desenhar();  
    }  
}
```



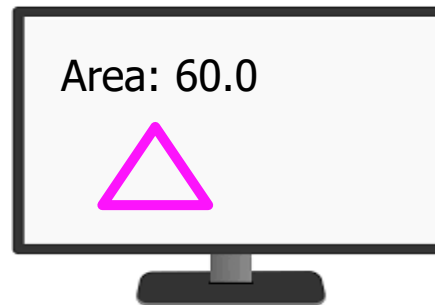
→ f1



```
public class Principal{  
    public static void main(String args[])  
    {  
        Triangulo f2 = new Triangulo();  
        f2.setBase(15.0);  
        f2.setAltura(4.0);  
        S.o.p("Area: " + f2.calcularArea());  
        f2.desenhar();  
    }  
}
```

- 15.0
- 4.0

→ f2





# Interface – revisão

---

- Interfaces são declaradas usando a palavra-chave *interface*

```
public interface FiguraGeometrica{...}
```

- Uma classe se relaciona com uma interface a partir da palavra chave *implements*

```
public class Quadrado implements FiguraGeometrica{...}
```



# Interface – revisão

---

- Em uma interface nenhum método tem corpo e são implicitamente definidos como *public* e *abstract*
- Exemplo:

```
double calcularArea() ;
```

```
public double calcularArea() ;
```

```
public abstract double calcularArea() ;
```

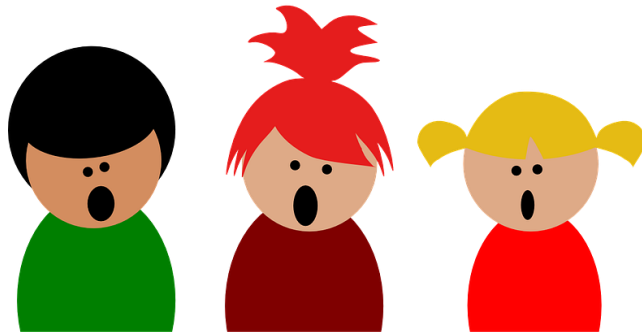




# Problema

---

Suponha que foi desenvolvida uma aplicação Java, para registrar as **principais funcionalidades** dos **instrumentos musicais** de uma orquestra sinfônica. A orquestra é formada por uma equipe de três músicos, sendo cada um especializado em um instrumento: guitarra, bateria e violão.





# Problema

---

Para que cada instrumento funcionasse corretamente, foi necessário registrar na aplicação que eles deveriam ser **afinados** e, além disso, que fosse possível **tocar todas as notas**.





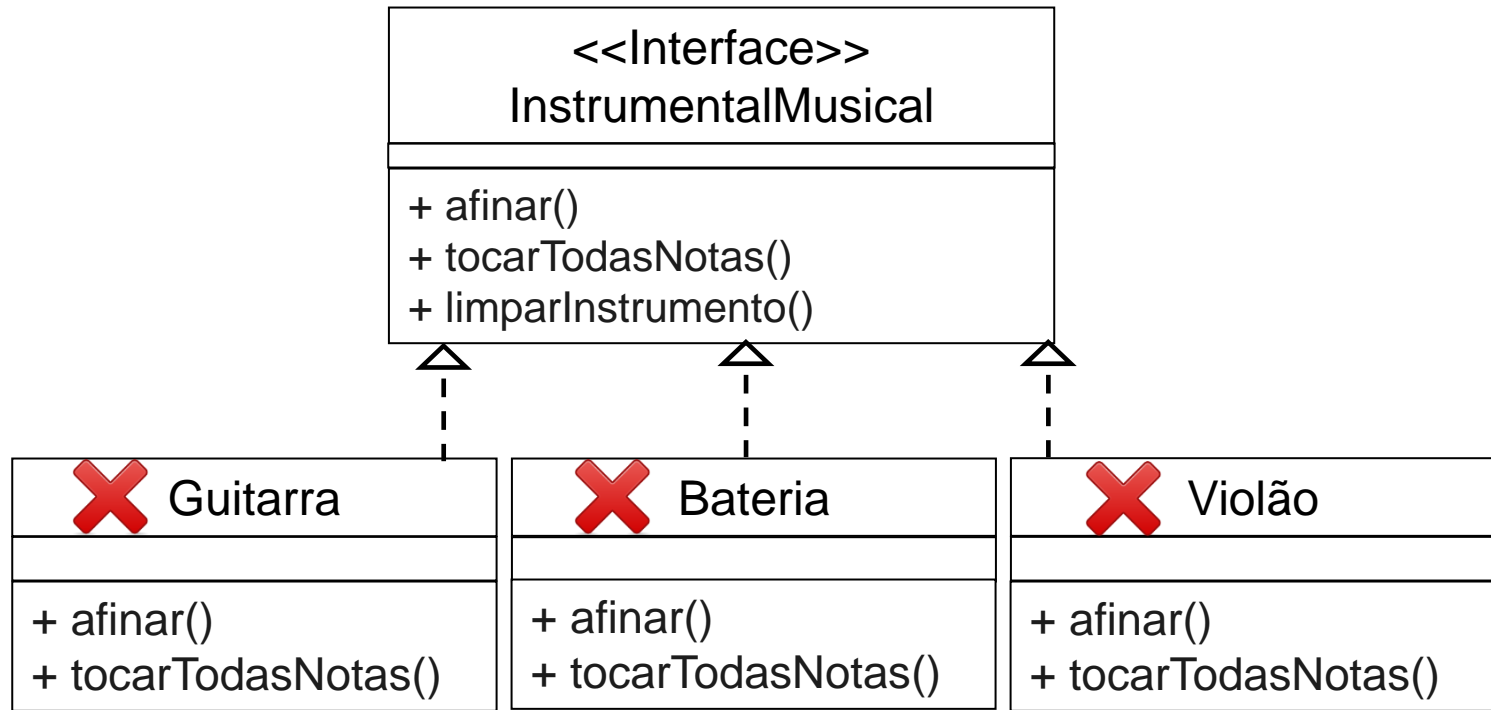
# Problema

---

Um ano depois da aplicação ter sido concluída, foi solicitado pela orquestra adicionar uma nova funcionalidade ao sistema: limpar o instrumento.

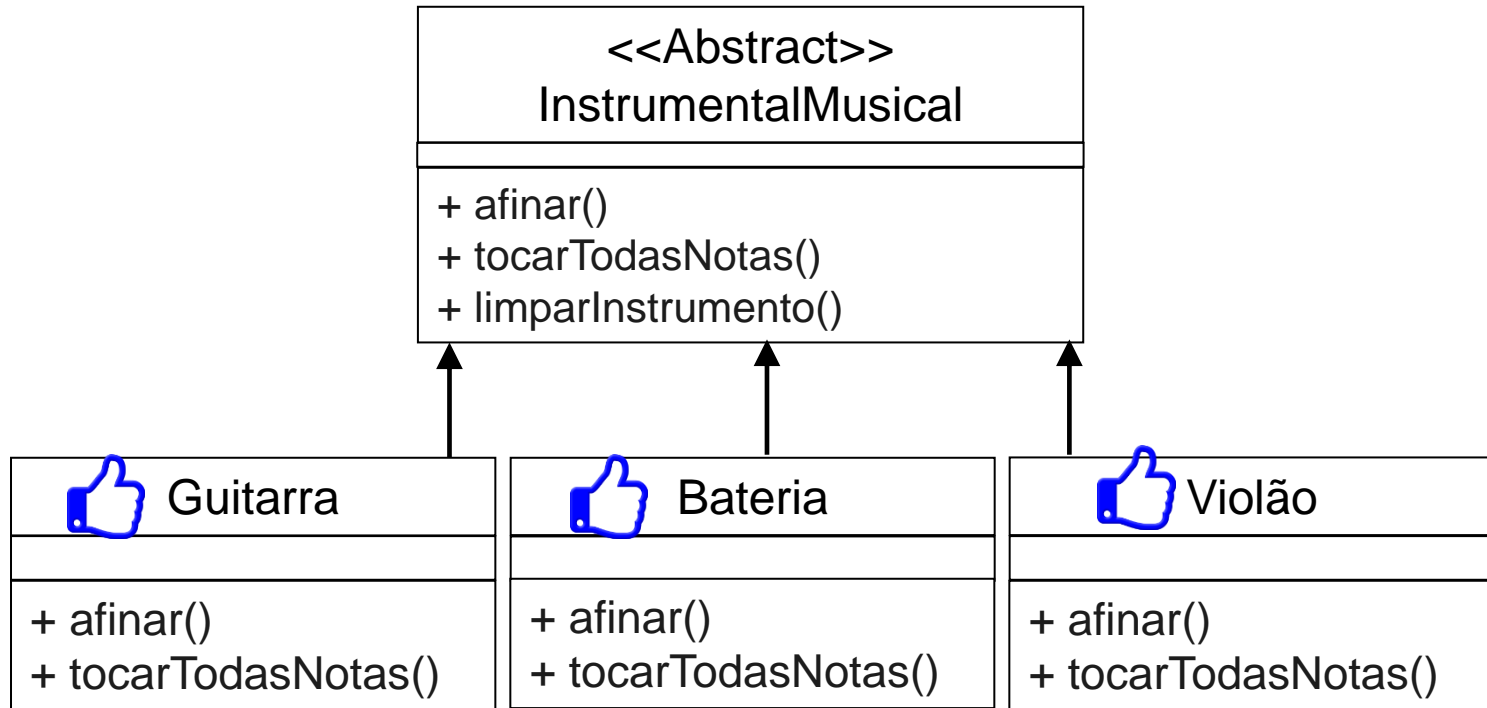
O que seria melhor: usar  
**classe abstrata** ou **interface**?

# Novo Cenário – Interface



- Como as subclasse não implementam o novo método, as classes não irão compilar.

# Novo Cenário – classe Abstrata



- Permite incluir um novo método sem quebrar as subclasses, desde que o método na classe abstrata seja implementado.



# Classe Abstrata ou Interface?

---

- Uma vez que a interface é liberada e está sendo usada amplamente, é quase impossível alterá-la, e por isso é necessário estudar cuidadosamente, logo de início, como ela deve ser implementada.
- Uma interface é a melhor forma de definir um tipo que permite múltiplas implementações. Uma exceção a essa regra é o caso em que a facilidade de evolução é considerada mais importante do que a flexibilidade.



# Referências

---

- Deitel, P. J.; Deitel, H. M. (2017). Java como programar. 10a edição. São Paulo: Pearson Prentice Hall.
- Barnes, D. J. (2009). Programação orientada a objetos com Java: uma introdução prática usando o BlueJ (4. ed.). São Paulo, SP: Prentice Hall.
- Boratti, I. C. (2007). Programação orientada a objetos em Java. Florianópolis, SC: Visual Books.