Polimorfismo: interfaces e classes abstratas

Prof. Tiago Sombra

Interfaces

- Corresponde a parte **pública** de uma classe de objetos ou de um componente de software
-
- Em geral **especificações ou funcionalidades** esperadas
-
- Também utilizadas para definir o **comportamento padrão,** apresentado por todas as classes que implementam a interface

Interfaces

- Em Java um tipo ***interface*** declara um conjunto de métodos e suas assinaturas
  - Permite que uma classe que implementa uma interface torne-se mais reutilizável

- Exemplo: Classe DataSet ...

Interfaces

- O exemplo demonstrou que:
  - Todas as classes poderiam ter um método que fornece a medida a ser utilizada no cálculo.
- A classe Coin retornaria o atributo value
- A classe Bank retornaria o atributo balance
-
- Por exemplo ambas classes poderiam implementar o método getMeasure() - o qual retorna tais valores.

Interfaces

- A classe *DataSet* original poderia ser reutilizada para qualquer tipo de objeto desde que esse implemente o método getMeasure(), seu método add seria adaptado para:
-

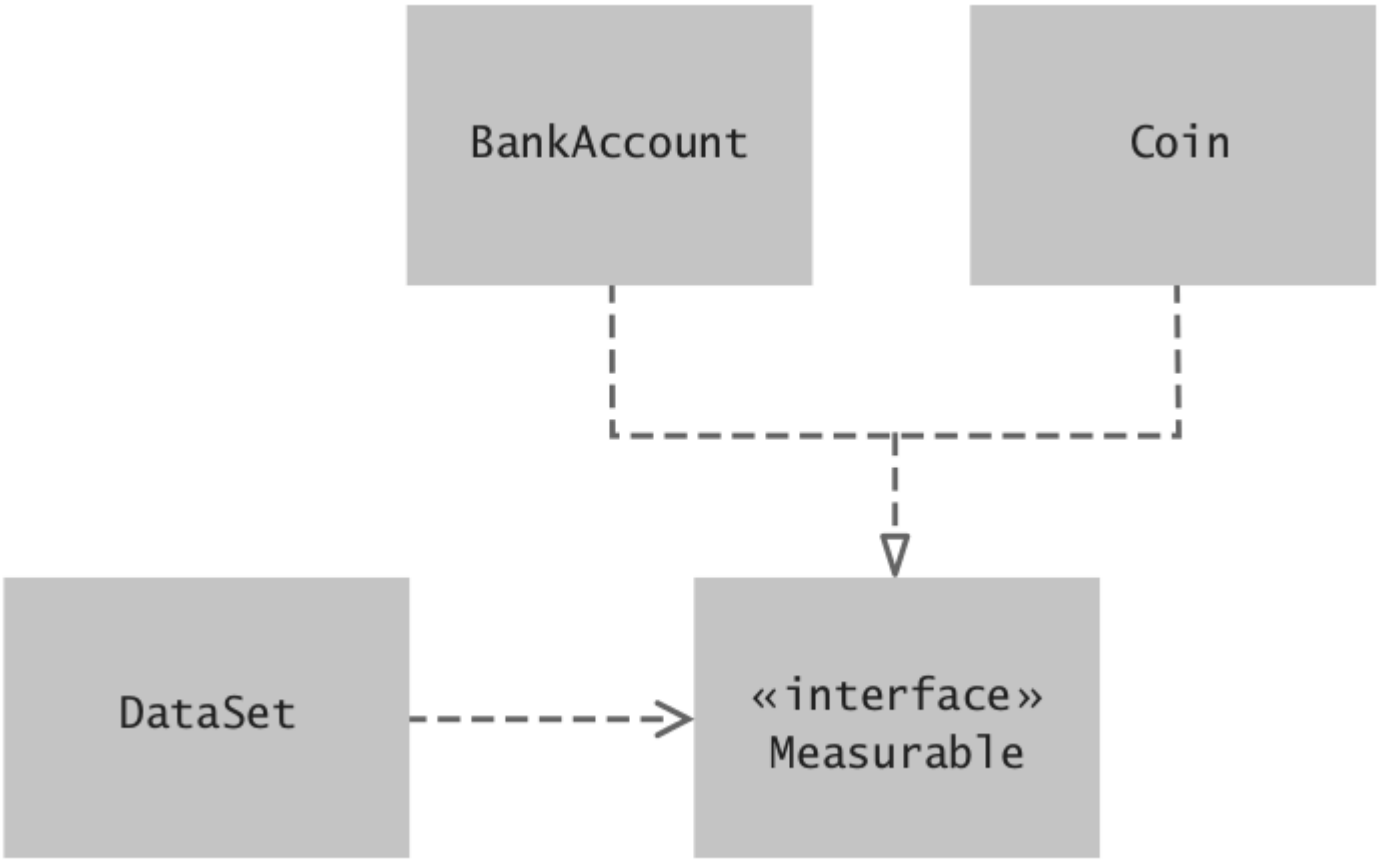
sum = sum + **x.getMeasure();**

if (count == 0 || maximum**.getMeasure()** < x**.getMeasure()**)

maximum = x;

count++;

Interfaces



Interfaces vs. Classes

- Interfaces e classes são similares, mas:
  - Todos os métodos de uma interface são abstratos eles não tem uma implementação
  - Todos os métodos de uma interface são automaticamente públicos
  - Uma interface não tem variáveis

Classes Abstratas

- Outra forma de implementar um **comportamento padrão** para toda uma hierarquia de classes é através de **classes abstratas**
-
- Trata-se de uma classe que **não pode ser instanciada**:
  - Nenhum objeto pode ser criado a partir dela
-

Classes Abstratas

```
public abstract class Figura {
   public double calculaArea(){ return 0; }
}

Figura f = new Figura; //erro de compilação
```

Métodos Abstratos

- Java permite que métodos sejam definidos como abstratos:
  - Sem nenhuma implementação
  -

```
public abstract class Figura {
   public abstract double calculaArea();
}
```

Regras para classes abstratas

- Toda classe derivada de uma classe abstrata deve implementar os métodos abstratos;
- Uma classe com um ou mais métodos abstratos deve ser definida como abstrata
- Classes abstratas podem conter métodos não abstratos
  - Caso não ocorra sobreposição o comportamento será herdado por todas as subclasses

Interfaces vs Classes Abstratas

- Se existirem atributos na interface, serão implicitamente considerados ***public, static e final:***
  - Deverão ser inicializados na sua declaração
- Se uma classe abstrata contém apenas métodos abstratos, então, ela pode ser criada como uma interface:
  - Comportamento padrão apresentado por todas as classes que a implementam

Interfaces vs Classes Abstratas

- Métodos com implementação:
  - Classes abstratas permitem esse tipo de método: todas as **subclasses irão herdar seus comportamentos**
  - Interfaces não permitem: todos os métodos são implicitamente ***abstract e public***
  –

- Dado que Java não permite herança múltipla, o uso de interfaces cria um mecanismo semelhante, possibilitando que uma classe implemente mais de uma interface

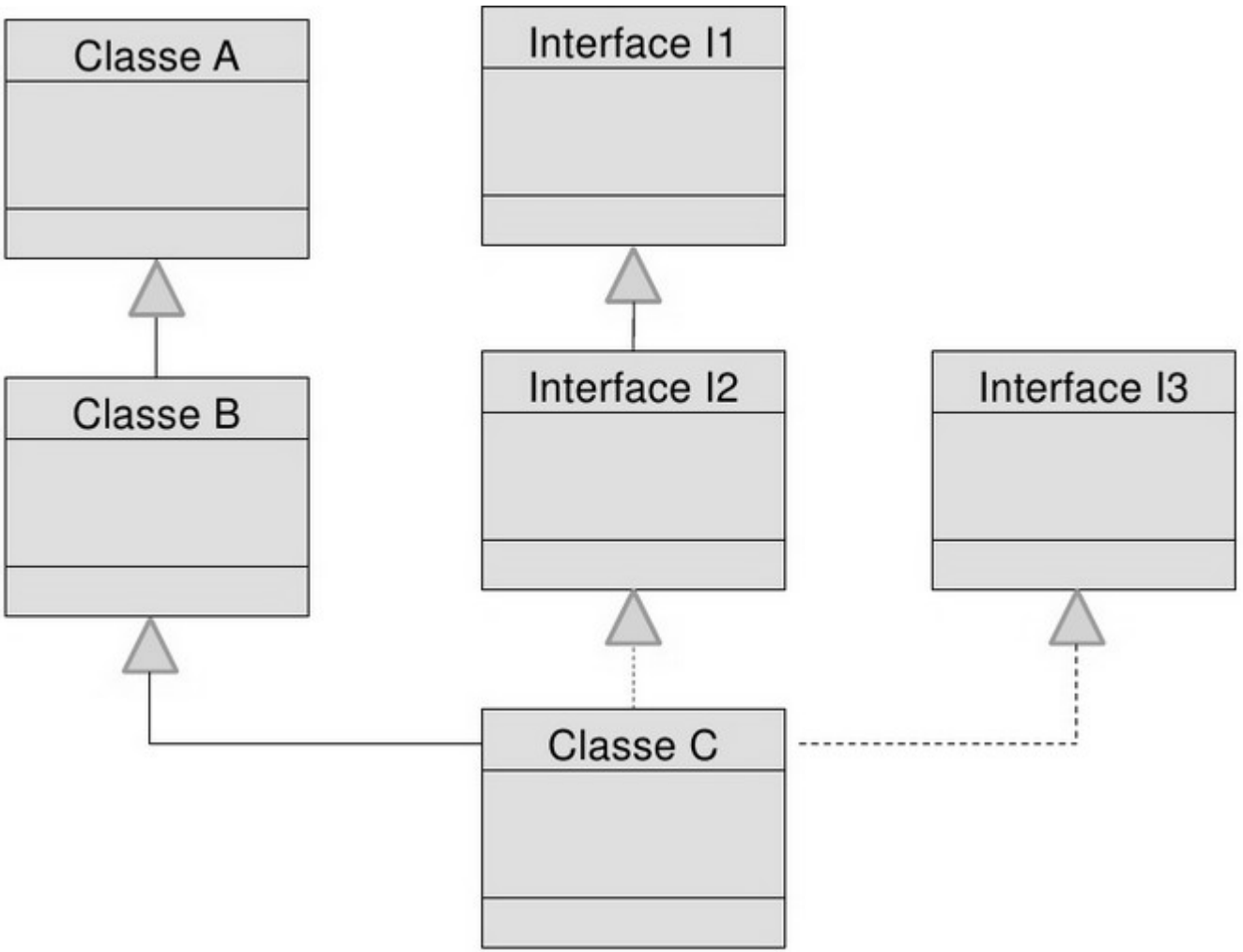
Múltiplas Interfaces

```
public interface Impressora {
   public void imprime(Documento d);
}
public interface Fax {
   public void transmite(Documento d);
}
public class FaxImpressora implements Impressora,Fax{
   public void imprime(){
      ...
   }
   public void transmite(){
      ...
   }
}
```

Interfaces e subinterfaces

- Da mesma forma que uma classe **B** pode estender outra classe **A**, uma interface ***I2*** pode estender outra interface **I1**.
  - Assim, quando uma classe C implementar I2, obrigatoriamente terá de implementar os métodos de I1 também
  –
- Exemplo. Interface java.util.List, que estende a interface Collection do mesmo pacote java.util.

Interfaces e subinterfaces



Polimorfismo

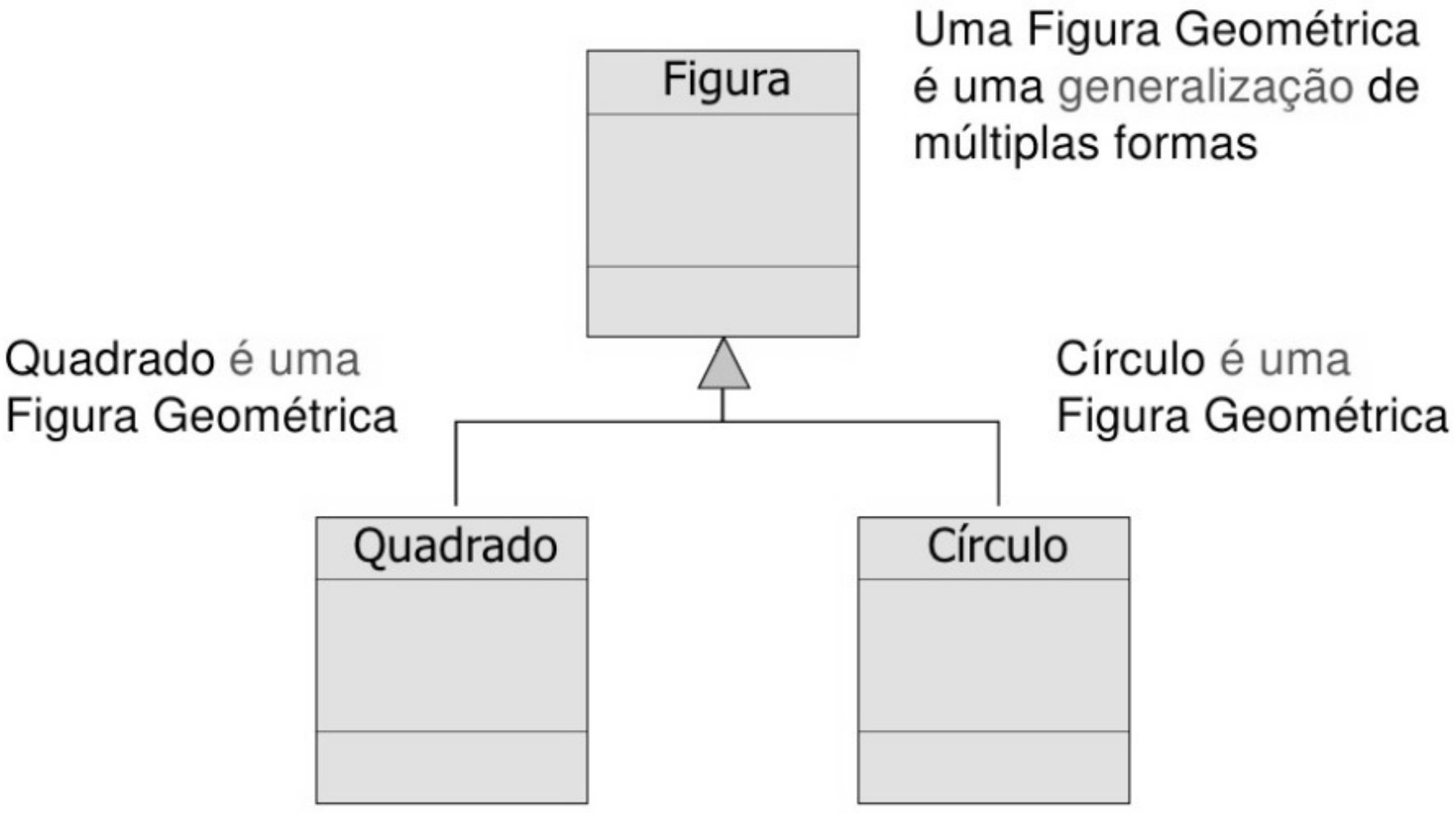
- O conceito de **herança** permite a criação de classes a partir de outras já existentes desde que exista a **relação “é um”** entre a subclasse e a superclasse;
-
- Dessa forma é possível criar classes mais **especializadas** a partir de uma classe mais genérica
-
- Essa relação permite a existência de outra característica fundamental em OO, o **polimorfismo**

Polimorfismo

Polimorfismo ou “múltiplas formas” – permite a manipulação de instâncias de classes que herdam de uma mesma classe “pai” de forma unificada:

Ao escrever métodos que recebam instâncias de uma classe “pai” A, os mesmos métodos serão capazes de processar instâncias de qualquer classe que herde dessa mesma classe “pai”, uma vez que, qualquer classe que herde de A **“é um”** A.

Polimorfismo



Polimorfismo

```
public class Circulo extends Figura {   → Herança

    double raio;

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

    public double calcularArea( ) {      → Sobreposição do método da
        double area = 0;                   superclasse.
        area = 3.14 * raio * raio;
        return area;
    }
}
```

Polimorfismo

```
public class Principal {

    public static void main(String[ ] args)  {

        Figura f1 = new Quadrado(4);

        Figura f2 = new Circulo(2);

        System.out.println("Área da Figura 1 é: "
                           + f1.calcularArea( ) + "\n"
                           + "Área da Figura 2 é: "
                           + f2.calcularArea( ));

    }
}
```

→ Comportamento polimórfico.

Um mesmo tipo base Figura, por meio das variáveis f1 e f2, é utilizado para enviar uma mesma mensagem calcularArea para objetos de tipos diferentes Quadrado e Circulo e o comportamento executado será distinto.

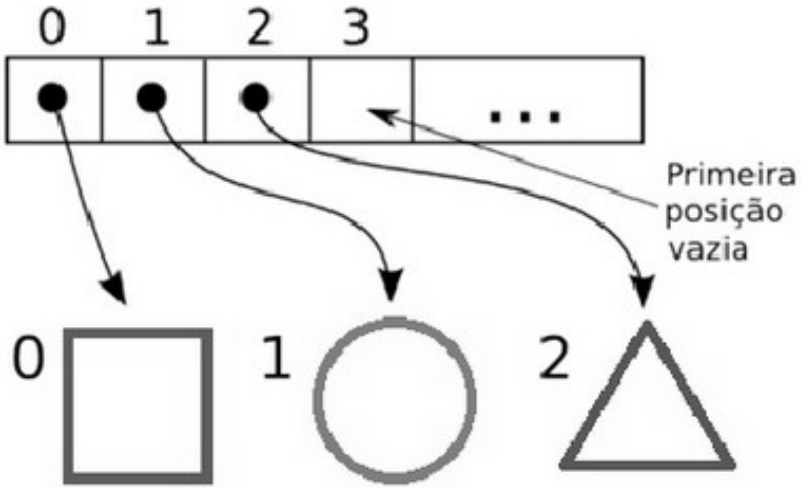
Polimorfismo

```
public class VetorFiguras {

    private Figura[] figuras = new Figura[10];
    private int totalDeElementos;

    public double calcularAreaTotal() {
        double areaTotal = 0;
        for (int i = 0; i < figuras.length; i++) {
            if (figuras[i] != null) {
                areaTotal = areaTotal + figuras[i].calcularArea(); → Comportamento
            }                                                        polimórfico.
        }
        return areaTotal;
    }
}
```

Polimorfismo




```
Quadrado q = new Quadrado(2.0);
Circulo c = new Circulo(2.0);

if(vetor.contem(q))
    ...
if(vetor.contem(c))
    ...
```

```
public class VetorFiguras{

    private Figura[] figuras = new Figura[10];
    private int totalDeElementos;

    public boolean contem(Figura fig) {
        boolean resultado = false;
        for(int i = 0; i < this.totalDeElementos; i++){
            if(fig.equals(this.figuras[i])){
                resultado = true;
                break;
            }
        }
        return resultado;
    }
}
```