Interfaces e polimorfismo

em Java

Vamos considerar um exemplo de uma classe chamada Calculos que possa receber números e nos retornar a média dos valores recebidos e o maior (máximo) valor recebido. O código de tal classe poderia ser o seguinte:

```
class Calculos
{ private int num;
  private double soma;
  private double max;
  public ConjuntoNumerico() {
    num = 0;
    soma = 0;
    max = 0;
  }
  public void add(double x)
  { soma += x();
    if (num==0 || max < x)
      max = x;
    num++;
  }
  public double media()
  { return (num>0?soma/num:0);
  }
  public double getMaximo()
  {
    return max;
  }
}
```

E um programa para testá-la seria:

```
class TesteCalculos
{
  public static void main(String args[])
  {
    Calculos c = new Calculos();
    c.add(2.5);
    c.add(1.5);
    System.out.println("Dados do conjunto...");
    System.out.println("media: " + c.media());
    double max = c.getMaximo();
    System.out.println("maximo: "+max);
  }
}
```

Vejamos agora uma situação mais abrangente. Suponhamos que na verdade queremos realizar cálculos com um conjunto não apenas de tipos números, mas sim um conjunto de objetos quaisquer que possuam alguma característica de sirva de “medida” para que sejam comparados com outros objetos nas mesmas condições.

Poderíamos usar como medida o valor de número, a altura de uma pessoa, sua idade, a área de uma figura geométrica etc. O que importa é que tais objetos tenham um método que nos forneça uma “medida” que possa ser por nós utilizada.

Observe então que nesta situação, os objetos tem algo em comum, um método, e, portanto nos vem logo em mente o princípio de herança, em que subclasses herdam métodos de superclasses. Neste nosso caso é bastante claro

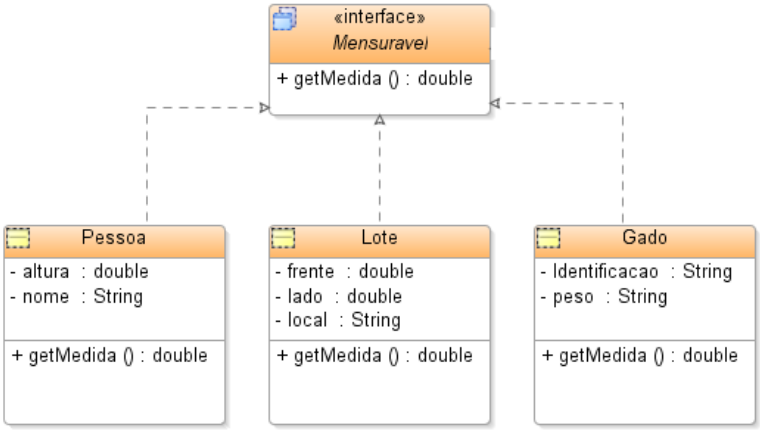
que cada classe deve ter sua própria implementação do método que devolve sua medida. Um objeto “pessoa” retorna sua altura ou idade, um objeto “triângulo” retorna sua área etc.

Vemos assim que não haveria sentido em implementar esse método na superclasse, mas sim que cada classe tenha sua implementação.

Java nos oferece o conceito de “***interface***”. Trata-se de uma entidade semelhante a classes, mas cujos métodos não possuem implementação. Outras classes que herdam esses métodos os implementam de acordo com seu próprio contexto.

Vamos supor então que para nós os objetos a serem considerados em nosso conjunto mais geral devem ter então um método que devolva sua medida, todos são então “mensurável”. O que importa é que apenas eles sabem como devolver essa medida.

Esquematicamente poderíamos ter o seguinte:



Numa interface todos os métodos são ditos “abstratos”, têm nome, parâmetros e tipo de retorno, mas não têm implementação. Todos os métodos são automaticamente públicos, e uma interface não tem atributos (propriedades), apenas constantes que são herdadas junto com os métodos.

Em nosso exemplo teríamos algo bem simples:

```
interface Mensuravel
{
  double getMedida();
}
```

Classes poderiam então “implementar” a interface com a sintaxe dos exemplos abaixo:

```
class Pessoa implements Mensuravel
{  private String nome;
   private double altura;
   public Pessoa(String n, double a)
   {  nome = n;
      altura = a;
   }
   public double medida()
   {  return altura;
   }
}
class Lote implements Mensuravel
{  private String local;
   private double frente, lado;
   public Lote(String l, double f, double l)
   {  local = l;
      frente = f;
      lado = l;
   }
   public double medida()
   {  return frente x lado;
   }
}
```

Enquanto uma classe pode estender uma única superclasse, uma mesma classe pode implementar mais de uma interface.

Observemos que nos exemplos as classes implementam o método descrito na interface, cada uma à sua maneira. Uma classe que implementa uma interface deve implementar todos os métodos da interface.

Nossa classe **Calculos** pode então ser adaptada para receber qualquer tipo de objeto que implemente a interface e assim possua um método que devolva uma medida para ser usada no cálculo.

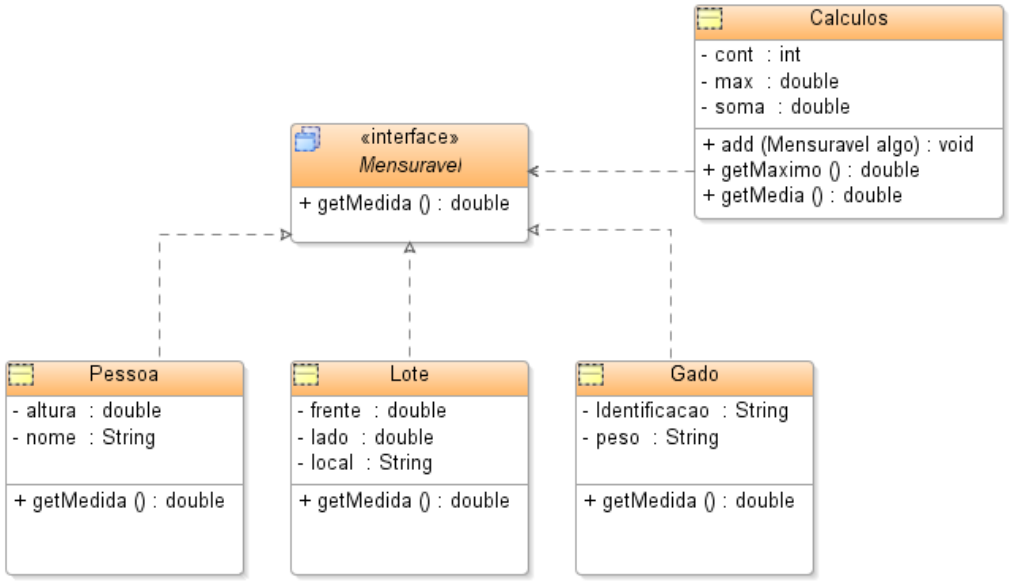
Do mesmo modo que uma referência a uma superclasse pode indicar um objeto de qualquer de suas subclasses, uma referência a uma interface pode indicar qualquer objeto de classes que implementam a interface.

```
class Calculos
{ private int num;
  private double soma;
  private Mensuravel maximo;
  public ConjuntoNumerico() {
    num = 0;
    soma = 0;
    maximo = null;
  }
  public void add(Mensuravel x)
  { soma += x.getMedida();
    if (num==0 || maximo.getMedida < x.getMedida)
      maximo = x;
    num++;
  }
  public double getMedia()
  { return (num>0?soma/num:0);
  }
  public Mensuravel getMaximo()
  {
    return maximo;
  }
}
```

E assim a classe de calculos pode receber qualquer objeto que implemente a interface Mensuravel. Veja como fica o programa de teste:

```
class TesteImplements
{
  public static void main(String args[])
  {
    Calculos c = new Calculos();
    Pessoa p = new Pessoa("joao", 1.80);
    Lote l = new Lote(“Vila Cachaça”,10,22);
    c.add(p);
    c.add(l);
    System.out.println("media: " + c.getMedia());
    Mensuravel max = c.getMaximo();
    System.out.println("maximo: "+max.getMedida());
  }
}
```

Em UML, temos o seguinte diagrama:



Observemos no exemplo que dentro da classe Calculos a utilização do método getMedida() é feita de acordo com o objeto sendo adicionado. Se o objeto é uma pessoa, a implementação da classe Pessoa é utilizada, mas se o objeto é um lote, a implementação da classe Lote é utilizada. Este é o conceito de “polimorfismo” sendo aplicado. A forma correta do método é determinada pela instância do objeto sendo referenciado através da interface.