Sobre Polimorfismo

Mário Leite

...

Polimorfismo, etimologicamente, quer dizer “várias formas”; entretanto, no universo da OOP é definido como sendo um código que “possui vários comportamentos” ou que “produz vários comportamentos”; em outras palavras, é um código que pode ser aplicado a várias classes de objetos, provocando “comportamentos diferentes”. De maneira prática isto significa que a operação em questão mantém seu comportamento transparente para quaisquer objetos de classes diferentes; isto é, a mesma mensagem é enviada a objetos de classes distintas e eles poderão reagir de maneiras diferentes. Entre os conceitos básicos da tecnologia de orientação a objetos, muitos autores concordam que o polimorfismo é o mais complicado de ser entendido, pois se trata de considerar objetos diferentes que implementam métodos de mesmo nome, porém com operações diferentes (!). É o caso dos objetos **Circulo1**, **Quadrado1** e **Triangulo1** - instâncias das classes - **TCirculo**, **TQuadrado** e **TTriangulo**, respectivamente, mostrados da **figura 1**, onde cada um executa seus próprios métodos de “desenhar”, “calcular área” e “calcular perímetro”, mas, o método é o mesmo, herdado da classe ancestral **TFiguraPlana**. O mecanismo pelo qual é calculada a área de um quadrado é bem diferente do mecanismo de cálculo da área de um triângulo; o mesmo acontecendo com a ação de calcular o perímetro. Do ponto de vista da codificação, o polimorfismo evita que o programador escreva várias linhas de instruções do tipo *if..else* ou *case’s* nas linguagens de programação. Tudo isto ajuda a reutilização de código, uma vez que pode-se especificar novas classes de objetos; por exemplo, *elipses*, *retângulos*, *trapézios* e *losangos*, sem a necessidade de “mexer” na interface.

A **figura 2** mostra a interface da aplicação, onde o usuário pode escolher o tipo de figura plana na *ComboBox* da esquerda, e o que deve ser feito, na *ComboBox* da direita. No caso em questão foi escolhido o *triângulo*, para calcular o seu *perímetro*.

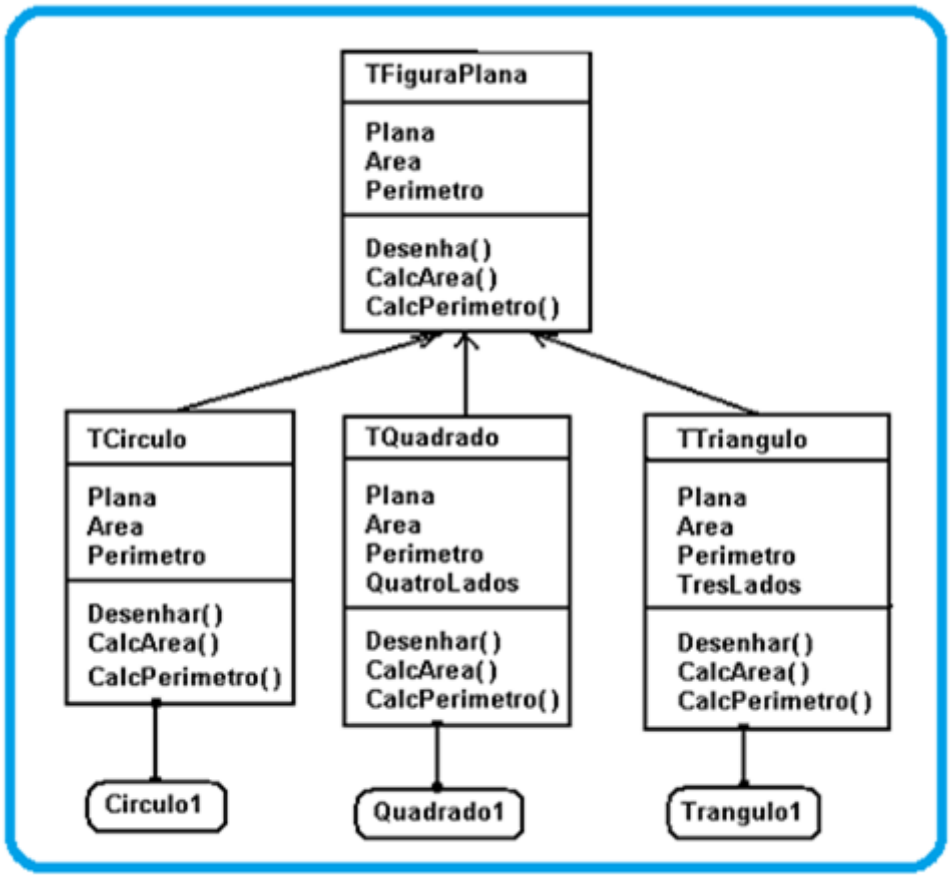

Figura 1 - Esquema de Especialçização e Generalização das classes

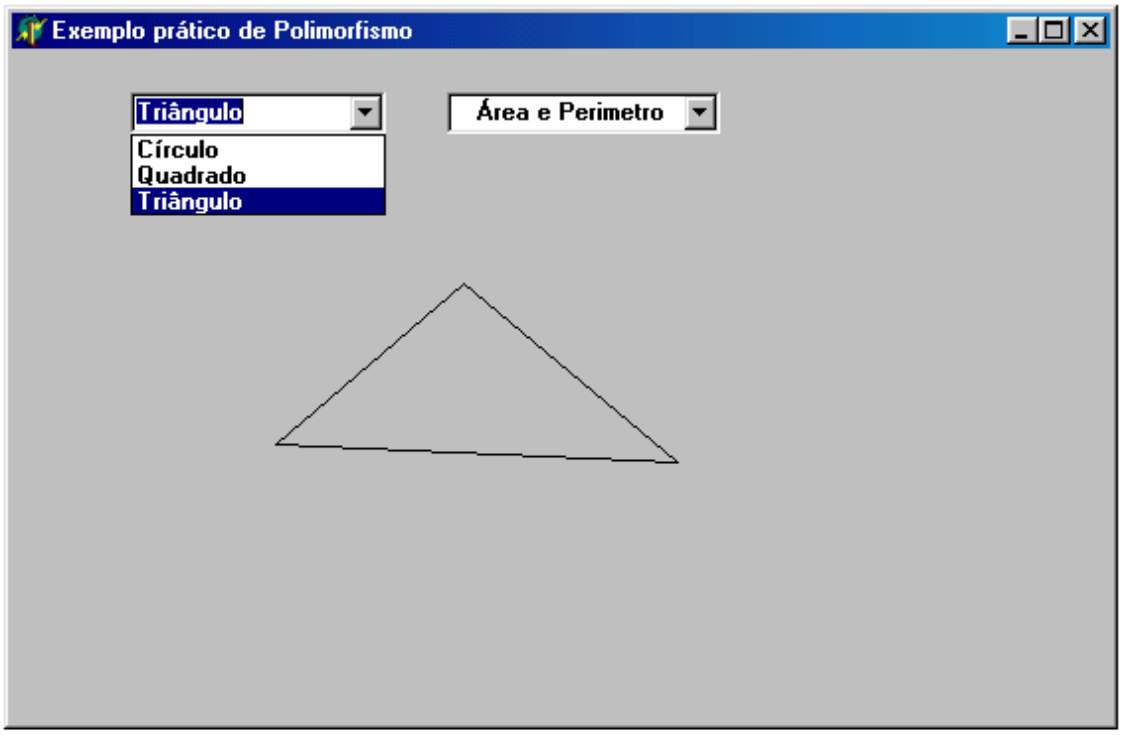

Figura 2 - Selecionando a figura e sua funcionalidade

```
{* Criação das classes e de seus métodos *}
type
  TFiguraPlana = class  //Classe abstrata
    constructor Create(const Pos: TPoint);
    procedure Desenhar(param:integer); virtual; abstract;
    function  CalcArea(param:integer): real; virtual; abstract;
    function  CalcPerimetro(param:integer): real; virtual; abstract;
  end;

  TCirculo = class(TFiguraPlana)
    procedure  Desenhar(param:integer); override;
    function  CalcArea(diametro:integer): real; override;
    function  CalcPerimetro(diametro:Integer): real; override;
  end;

  TQuadrado = class(TFiguraPlana)
    procedure Desenhar(Lado:integer); override;
    function  CalcArea(Lado:integer) : real; override;
    function  CalcPerimetro(Lado:integer): real; override;
  end;

  TTriangulo = class(TFiguraPlana)
    procedure  Desenhar(Lado:integer); override;
    function  CalcArea(Lado:integer) : real; override;
    function  CalcPerimetro(Lado:integer): real; override;
    function  CalcVerifTriang(L1, L2, L3:real): boolean;
  end;
 {* Fim da criação das classes e dos métodos *}
//---------------------------------------------------------------------------
...
{* Implementação dos métodos da classe "TTriangulo"*}
procedure TTriangulo.Desenhar(Lado:integer);
begin
  Form1.Canvas.Polygon([Point(xP1, yP1), Point(xP2, yP2), Point(xP3, yP3)]);
end;

function TTriangulo.CalcArea(Lado:integer): Real;
var
  Lado1, Lado2, Lado3: real;
  D12, D22, D32: real;
  Prod1, Prod2, Dif: real;
  P, S: real;
begin
  // (xP1,yP1) ==> Coordenadas do Ponto1
  // (xP2,yP2) ==> Coordenadas do Ponto2
  // (xP3,yP3) ==> Coordenadas do Ponto3
  // Lado1 ==> Distância entre o Ponto1 e o Ponto2
  // Lado2 ==> Distância entre o Ponto2 e o Ponto3
  // Lado3 ==> Distância entre o Ponto3 e o Ponto1
  // Dk2 ==>  Soma dos quadrados das diferenças entre as coordenadas (k=1,3)
  // p   ==>  Semiperímetro do triângulo
  // S   ==>  Área do triângulo
 // Area do triângulo: S = p.r (p=Semiperímetro,r=Raio da circunferência inscrita);
  // Ou ainda:  S = Sqrt(p*(p-Lado1)*(p-Lado2)*(p-Lado3))
  D12 := Sqr(xP2-xP1) + Sqr(yP2-yP1);
  D22 := Sqr(xP3-xP2) + Sqr(yP3-yP2);
  D32 := Sqr(xP3-xP1) + Sqr(yP3-yP1);
  Lado1 := Int(Sqrt(D12));
  Lado2 := Int(Sqrt(D22));
  Lado3 := Int(Sqrt(D32));
```

```
  //Invoca função para verificar se os pontos escolhidos pelo usuário definem um triângulo
   If(VerifTriang(Lado1, Lado2, Lado3)) then
     begin
       P := (Lado1 + Lado2 + Lado3)/2;
       S := Sqrt(Abs((P*(P-Lado1)*(P-Lado2)*(P-Lado3))));
       Result:= S;
     end
  Else
     begin
       MessageDlg('Os pontos marcados não definem um triângulo', mtError, [mbOk], 0);
       Result:= 0.00;
     end;
end;

function TTriangulo.CalcPerimetro(Lado:integer): real;
var
  Lado1, Lado2, Lado3: real;
  D12, D22, D32: real;
begin
  D12 := Sqr(xP2-xP1) + Sqr(yP2-yP1);
  D22 := Sqr(xP3-xP2) + Sqr(yP3-yP2);
  D32 := Sqr(xP3-xP1) + Sqr(yP3-yP1);
  Lado1 := Int(Sqrt(D12));
  Lado2 := Int(Sqrt(D22));
  Lado3 := Int(Sqrt(D32));

  If(VerifTriang(Lado1,Lado2,Lado3))then //Verifica se pontos definem triângulo
     Result :=  (Lado1 + Lado2 + Lado3)
  Else
     begin
      MessageDlg('Pontos marcados não definem um triângulo', mtError,[mbOk], 0);
       Result:= 0.00;
     end;
end;
{* Fim da implementação dos métodos da classe derivada "TTriangulo" *}
//---------------------------------------------------------------------


unit PoliUni1;
   interface
uses
   Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms,
   Dialogs,Math, StdCtrls, Buttons;

type
  TForm1 = class(TForm)
    CmbDesenhar: TComboBox;
    CmbCalcular: TComboBox;
    Label1: TLabel;
    procedure FormCreate(Sender: TObject);
    procedure CmbDesenharClick(Sender: TObject);
    procedure CmbCalcularClick(Sender: TObject);
    procedure FormMouseDown(Sender: TObject; Button: TMouseButton;
              Shift: TShiftState; X, Y: Integer);
  private
    { Private declarations }

  public
    { Public declarations }
 end;

var  Form1: TForm1;
const PI:single = 3.14159265;
//-------------------------------------------------------------------------------------
```

```
implementation
var
 Figuras: array [1..3] of TFiguraPlana; //Cria array para conter os objetos
 xP1,yP1: integer;
 xP2,yP2: integer;
 xP3,yP3: integer;
 nVezes : integer;
 OpCalcular: string;

{$R *.DFM}

{*Construtor da classe abstrata *}
constructor TFiguraPlana.Create(const Pos: TPoint);
begin
  inherited Create;
end;

{Os métodos da classe "TFgiuraPlana" não podem ser implementados, devido ser esta uma
classe abstrata (a palavra-chave Abstract na definição dos métodos impõe esta restrição.
Se for tentado implementar esses métodos na classe abstrata será gerado um erro em tempo
de compilação}
//---------------------------------------------------------------------------------------


function TTriangulo.VerifTriang(L1, L2, L3:real): boolean;
//Verifica se os pontos escolhidos pelo usuário definem um triângulo
var Prod1, Prod2, Dif: real;
    Perim, Are: real;
begin
  Prod1 := (xP3-xP1)*(yP2-yP1);
  Prod2 := (yP3-yP1)*(xP2-xP1);
  Dif := Prod1-Prod2;
  If(Dif <> 0) then
     Result := True
  Else
    begin
      //Os três pontos estão sobre uma mesma reta
      Result := False;
   end;
end;
(* Fim do código da seção implementation *}

//---------------------------------------------------------------------------------------
// Implementação dos métodos da classe derivada "TCirculo"
procedure TCirculo.Desenhar(param:integer);
var Rect: TRect;
begin
  Rect.Left   := Trunc(Form1.Width/2) - Trunc(param/2);
  Rect.Right  := Rect.Left + param;
  Rect.Top    := Trunc(Form1.Height/2) - Trunc(param/2);
  Rect.Bottom := Rect.Top + param;
  Form1.Canvas.Ellipse(Rect);
end;

function TCirculo.CalcArea(diametro:integer): real;
var
  Raio: integer;
begin
  Raio := Trunc(diametro/2);
  Result := PI*sqr(Raio);
end;
```

```
function TCirculo.CalcPerimetro(diametro:integer): real;
var Raio: real;
begin
  Raio := diametro/2;
  Result := 2*PI*Raio;
end;
//Fim da implementação dos métodos da classe derivada "TCirculo"
//---------------------------------------------------------------------------------------

{* Implementação dos métodos da classe "TQuadrado" *}
procedure TQuadrado.Desenhar(Lado:integer);
var
   Rect: TRect;
begin
  Rect.Left   := Trunc(Form1.Width/2) - Trunc(Lado/2);;
  Rect.Right  := Rect.Left + Lado;
  Rect.Top    := Trunc(Form1.Height/2) - Trunc(Lado/2);
  Rect.Bottom := Rect.Top + Lado;
  Form1.Canvas.Rectangle(Rect);
end;

function TQuadrado.CalcArea(Lado:integer): real;
begin
  Result := Power(Lado, 2);
end;

function TQuadrado.CalcPerimetro(Lado:integer): real;
begin
  Result := 4*Lado;
end;
{* Fim da implementação dos métodos da classe derivada "TQuadrado" *}
//---------------------------------------------------------------------------------------

{* Implementação dos métodos da classe "TTriangulo"*}
procedure TTriangulo.Desenhar(Lado:integer);
begin
  Form1.Canvas.Polygon([Point(xP1, yP1), Point(xP2, yP2), Point(xP3, yP3)]);
end;

function TTriangulo.CalcArea(Lado:integer): Real;
var Lado1, Lado2, Lado3: real;
    D12, D22, D32: real;
    Prod1, Prod2, Dif: real;
    P, S: real;
begin
  // (xP1,yP1) ==> Coordenadas do Ponto1
  // (xP2,yP2) ==> Coordenadas do Ponto2
  // (xP3,yP3) ==> Coordenadas do Ponto3
  // Lado1 ==> Distância entre o Ponto1 e o Ponto2
  // Lado2 ==> Distância entre o Ponto2 e o Ponto3
  // Lado3 ==> Distância entre o Ponto3 e o Ponto1
  // Dk2 ==>  Soma dos quadrados das diferenças entre as coordenadas (k=1,3)
  // p ==> Semiperímetro do triângulo
  // S ==> Área do triângulo
 // Area do triângulo: S = p.r (p=Semiperímetro,r=Raio da circunferência inscrita);
  // Ou ainda:  S = Sqrt(p*(p-Lado1)*(p-Lado2)*(p-Lado3))
  D12 := Sqr(xP2-xP1) + Sqr(yP2-yP1);
  D22 := Sqr(xP3-xP2) + Sqr(yP3-yP2);
  D32 := Sqr(xP3-xP1) + Sqr(yP3-yP1);
  Lado1 := Int(Sqrt(D12));
  Lado2 := Int(Sqrt(D22));
  Lado3 := Int(Sqrt(D32));
```

```
  //Invoca função para verificar se os pontos escolhidos pelo usuário definem um triângulo
   If(VerifTriang(Lado1, Lado2, Lado3)) then
     begin
       P := (Lado1 + Lado2 + Lado3)/2;
       S := Sqrt(Abs((P*(P-Lado1)*(P-Lado2)*(P-Lado3))));
       Result:= S;
     end
  Else
     begin
       MessageDlg('Os pontos marcados não definem um triângulo', mtError, [mbOk], 0);
       Result:= 0.00;
     end;
end;

function TTriangulo.CalcPerimetro(Lado:integer): real;
var Lado1, Lado2, Lado3: real;
    D12, D22, D32: real;
begin
  D12 := Sqr(xP2-xP1) + Sqr(yP2-yP1);
  D22 := Sqr(xP3-xP2) + Sqr(yP3-yP2);
  D32 := Sqr(xP3-xP1) + Sqr(yP3-yP1);
  Lado1 := Int(Sqrt(D12));
  Lado2 := Int(Sqrt(D22));
  Lado3 := Int(Sqrt(D32));
  If(VerifTriang(Lado1,Lado2,Lado3))then //Verifica se pontos definem triângulo
     Result :=  (Lado1 + Lado2 + Lado3)
  Else
    begin
       MessageDlg('Pontos marcados não definem um triângulo', mtError,[mbOk], 0);
       Result:= 0.00;
    end;
end;
{* Fim da implementação dos métodos da classe derivada "TTriangulo" *}

//---------------------------------------------------------------------------------------
{* Implementação da camada de código da interface usuário-aplicação *}
procedure TForm1.FormCreate(Sender: TObject);
begin
  nVezes := 0;
end;

procedure TForm1.CmbDesenharClick(Sender: TObject);
var NovaPos: TPoint;
    SL,msg1,msg2:string;
    NL:integer;
    Perc: real;
begin
  NovaPos.X := Random(Self.Width);
  Label1.Visible := False;
  If(CmbDesenhar.Text = Trim('Círculo')) then begin
     msg1 := 'Entre com diâmetro do círculo';
     msg2 := 'Diâmetro do círculo muito grande';
    SL := InputBox('Desenho de uma figura geométrica plana', msg1,'100');
     NL := Abs(StrToInt(SL));
     Perc := NL /(Self.Height);
     If(Perc > 0.80) then begin
        MessageDlg(msg2, mtWarning, [mbOk], 0);
        Exit;
     end;
     Figuras[1] := TCirculo.Create(NovaPos);
     Figuras[1].Desenhar(NL); //Invoca o método "Desenhar", especializado em círculo
   end;
```

```
    If(CmbDesenhar.Text = Trim('Quadrado')) then begin
       msg1 := 'Entre com o lado do quadrado';
       msg2 := 'Lado do quadrado muito grande';
       SL := InputBox('Desenho de um quadrado', msg1,'100');
       NL := Abs(StrToInt(SL));
       Perc := NL /(Self.Height);
       If(Perc>0.80) then //Verifica se quadro excederá a área reservada para desenho
         begin
            MessageDlg(msg2, mtWarning, [mbOk], 0);
            Exit;
         end;
       Figuras[2] := TQuadrado.Create(NovaPos);
       Figuras[2].Desenhar(NL); //Invoca método "Desenhar", especializado em quadrado
    end;
   If(CmbDesenhar.Text = Trim('Triângulo')) then begin
     Figuras[3] := TTriangulo.Create(NovaPos);
     MessageDlg('Clique em três pontos diferentes', mtInformation, [mbOK], 0);
   end;
   nVezes := 0;
end;
//---------------------------------------------------------------------------------------
procedure TForm1.CmbCalcularClick(Sender: TObject);
var NovaPos : TPoint;
    AreaCirc, AreaQuad: real;
    PerimCirc, PerimQuad: real;
    NL: integer;
    SL: string;
begin
    NovaPos.X := Random(Self.Width);
    Label1.Visible := True;
    Label1.Caption := '';
    Case CmbCalcular.ItemIndex  of
      0:   //Calcula área do círculo
       begin
         Figuras[1] := TCirculo.Create(NovaPos);
         SL := InputBox('Desenho de uma figura', 'Entre com o diâmetro
                        do círculo','100');
         NL := Abs(StrToInt(SL));
         AreaCirc := Figuras[1].CalcArea(NL);
         Label1.Caption:='Área do círculo: ' +
         Trim(FloatToStr(AreaCirc));
       end;
      1:  //Calcula área do quadrado
       begin
         Figuras[2] := TQuadrado.Create(NovaPos);
         SL := InputBox('Desenho de uma figura', 'Entre com o lado do
                       quadrado','100');
         NL := Abs(StrToInt(SL));
         AreaQuad := Figuras[2].CalcArea(NL);
         Label1.Caption := 'Área do quadrado: ' + Trim(FloatToStr(AreaQuad));

      2:  //Prepara para calcular área do triângulo
       begin
         MessageDlg('Marque três pontos do triângulo',mtInformation,[mbOK], 0);
         OpCalcular := 'Área do triângulo';
       end;
      3:  //Calcula comprimento da circunferência do círculo
       begin
         Figuras[1] := TCirculo.Create(NovaPos);
         SL := InputBox('Desenho de uma figura', 'Entre com o diâmetro
                       do círculo','100');
         NL := Abs(StrToInt(SL));
```

```
        PerimCirc  := Figuras[1].CalcPerimetro(NL);
        Label1.Caption := 'Perímetro do círculo ' + Trim(FloatToStr(PerimCirc));
       end;
      4:  //Calcula perímetro do quadrado
       begin
        Figuras[2] := TQuadrado.Create(NovaPos);
        SL := InputBox('Desenho de uma figura', 'Entre com o lado do
                       quadrado','100');
        NL := Abs(StrToInt(SL));
        PerimQuad := Figuras[2].CalcPerimetro(NL);
        Label1.Caption := 'Perímetro do quadrado ' + Trim(FloatToStr(PerimQuad));
       end;
      5:   //Prepara para calcular perímetro do triângulo
       begin
        MessageDlg('Marque três pontos do triângulo',mtInformation,[mbOK], 0);
        OpCalcular := 'Perímetro do triângulo';
       end;
     end;
end;

//---------------------------------------------------------------------------
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton;
  Shift: TShiftState; X, Y: integer);
var
  NovaPos :Tpoint;
begin
  Label1.Caption := '';
  Inc(nVezes); //Incrementa a variável modal para garantir os três clicks de mouse
  Case nVezes of
    1:  //Define as coordenadas do primeiro ponto do triângulo
     begin
       xP1 := X;
       yP1 := Y;
     end;
    2:  //Define as coordenadas do segundo ponto do triângulo
     begin
       xP2 := X;
       yP2 := Y;
     end;
    3:  //Define as coordenadas do terceiro ponto do triângulo
     begin
       xP3 := X;
       yP3 := Y;
       If(OpCalcular = 'Perímetro do triângulo') then begin
          NovaPos.X  := Random(Self.Width);
          Figuras[3] := TTriangulo.Create(NovaPos);
          Figuras[3].Desenhar(0);
          Label1.Caption:= 'Perímetro do triângulo: ' +
                           FloatToStr(Figuras[3].CalcPerimetro(0));
           nVezes := 0;
        end;
        If(OpCalcular = 'Área do triângulo') then
          begin
            NovaPos.X  := Random(Self.Width);
            Figuras[3] := TTriangulo.Create(NovaPos);
            Figuras[3].Desenhar(0);
            Label1.Caption := 'Área do triângulo: ' +
                           FloatToStr(Figuras[3].CalcArea(0));
            nVezes := 0;
          end;
          If(OpCalcular='') then
            begin  //Se não selecionado "Calcular" na ComboBox...
```

```
                NovaPos.X  := Random(Self.Width);
                Figuras[3].Desenhar(0);//Invoca método "Desenhar", especializa triângulo
                nVezes := 0;
              end;
            end
            Else
              Exit
        end;
    end;  //Fim da camada de código da interface usuário-aplicação

End.  //Fim da aplicação
```