Как создать потомок от **TCustomPanel**, которая имеет свойство градиентной заливки и умеет отображать изображения.

```
unit LWGradientPanel; interface uses Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs, ExtCtrls; {$R LWGradientPanel.dcr} type TLWFillDirection = (fdTopToBottom, fdBottomToTop, fdLeftToRight, fdRightToLeft); TLWGradientPanel = class(TCustomPanel) private { Private declarations } FGradientStartColor : TColor; FGradientEndColor : TColor; FGradient : boolean; FGradientFillDir : TLWFillDirection; FTextFillsPanel : boolean; FIcon : TPicture; procedure SetGradientStartColor(value : TColor); procedure SetGradientEndColor(value : TColor); procedure SetGradient(value : boolean); procedure SetGradientFillDir(value : TLWFillDirection); procedure SetTextFillsPanel(value : boolean); procedure SetIcon(value : TPicture); protected { Protected declarations } procedure paint; override; constructor create(AOwner : TComponent); override; destructor destroy; override; procedure loaded; override; public { Public declarations } published { Published declarations } property GradientStartColor : TColor read FGradientStartColor write SetGradientStartColor; property GradientEndColor : TColor read FGradientEndColor write SetGradientEndColor; property Gradient : boolean read FGradient write SetGradient; property GradientFillDir : TLWFillDirection read FGradientFillDir write SetGradientFillDir; property TextFillsPanel : boolean read FTextFillsPanel write SetTextFillsPanel; property Icon : TPicture read FIcon write SetIcon; property Align; property Alignment; property BevelInner; property BevelOuter; property BevelWidth; property BorderWidth; property BorderStyle; property DragCursor; property DragMode; property Enabled; property FullRepaint; property Caption; property Color; property Ctl3D; property Font; property Locked; property ParentColor; property ParentCtl3D; property ParentFont; property ParentShowHint; property PopupMenu; property ShowHint; property TabOrder; property TabStop; property Visible; property OnClick; property OnDblClick; property OnDragDrop; property OnDragOver; property OnEndDrag; property OnEnter; property OnExit; property OnMouseDown; property OnMouseMove; property OnMouseUp; property OnResize; property OnStartDrag; end; procedure Register; implementation function Min(A, B: Longint): Longint; begin if A  B then Result := A else Result := B; end; function WidthOf(R: TRect): Integer; begin Result := R.Right - R.Left; end; function HeightOf(R: TRect): Integer; begin Result := R.Bottom - R.Top; end; procedure GradientFillRect(Canvas: TCanvas; ARect: TRect; StartColor, EndColor: TColor; Direction: TLWFillDirection; Colors: Byte); var StartRGB: array[0..2] of Byte; { Значения RGB } RGBDelta: array[0..2] of Integer; ColorBand: TRect; I, Delta: Integer; Brush: HBrush; begin if IsRectEmpty(ARect) then Exit; if Colors
0 then begin for I := 0 to Colors do begin case Direction of { Calculate the color band's top and bottom coordinates } fdTopToBottom, fdBottomToTop: begin ColorBand.Top := ARect.Top + I * Delta; ColorBand.Bottom := ColorBand.Top + Delta; end; { Calculate the color band's left and right coordinates } fdLeftToRight, fdRightToLeft: begin ColorBand.Left := ARect.Left + I * Delta; ColorBand.Right := ColorBand.Left + Delta; end; end; {case} { Calculate the color band's color } Brush := CreateSolidBrush(RGB( StartRGB[0] + MulDiv(I, RGBDelta[0], Colors - 1),
```

StartRGB[1] + MulDiv(I, RGBDelta[1], Colors - 1), StartRGB[2] + MulDiv(I, RGBDelta[2], Colors - 1))); FillRect(Canvas.Handle, ColorBand, Brush); DeleteObject(Brush); end; end; if Direction in [fdTopToBottom, fdBottomToTop] then Delta := HeightOf(ARect) mod Colors else Delta := WidthOf(ARect) mod Colors; if Delta > 0 then begin case Direction of { Calculate the color band's top and bottom coordinates } fdTopToBottom, fdBottomToTop: begin ColorBand.Top := ARect.Bottom - Delta; ColorBand.Bottom := ColorBand.Top + Delta; end; { Calculate the color band's left and right coordinates } fdLeftToRight, fdRightToLeft: begin ColorBand.Left := ARect.Right - Delta; ColorBand.Right := ColorBand.Left + Delta; end; end; {case} case Direction of fdTopToBottom, fdLeftToRight: Brush := CreateSolidBrush(EndColor); else {fdBottomToTop, fdRightToLeft } Brush := CreateSolidBrush(StartColor); end; FillRect(Canvas.Handle, ColorBand, Brush); DeleteObject(Brush); end; end; procedure TLWGradientPanel.Loaded; begin inherited loaded; end; procedure TLWGradientPanel.paint; var r : Trect; x,y : integer; begin r := ClientRect; if BevelOuter bvNone then if BevelOuter = bvRaised then Frame3D(Canvas, r, clBtnHighlight, clBtnShadow, BevelWidth) else Frame3D(Canvas, r, clBtnShadow, clBtnHighlight, BevelWidth); if BevelInner bvNone then if BevelInner = bvRaised then Frame3D(Canvas, r, clBtnHighlight, clBtnShadow, BevelWidth) else Frame3D(Canvas, r, clBtnShadow, clBtnHighlight, BevelWidth); if FGradient then begin GradientFillRect(canvas, r, FGradientStartColor, FGradientEndColor, FGradientFillDir, 255) end; if not FGradient then begin canvas.brush.color := color; canvas.fillrect(r); end; canvas.brush.style := bsClear; canvas.Font.assign(font); if TextFillsPanel and (caption '') then begin if (canvas.TextWidth(caption)

По материалам http://delphi.3000.com