PROGRAMAÇÃO ORIENTADA A OBJETOS COM DELPHI

HERANÇA VISUAL

Com o Delphi, desde sua primeira versão, é possível criar formulários básicos, padrão, para que sejam herdados, tanto quanto suas funcionalidades como seu visual, e a partir daí, adicionar novas funcionalidades tanto em código quanto no visual. Isso graças ao paradigma da Programação Orientada a Objetos – POO.

Para ilustrar este conceito, vamos desenvolver um exemplo de formulário padrão para cadastros básicos, utilizando MS SQL Server e uma aplicação VLC Forms. Considerando que já exista um projeto, abra o projeto e siga os seguintes passos:

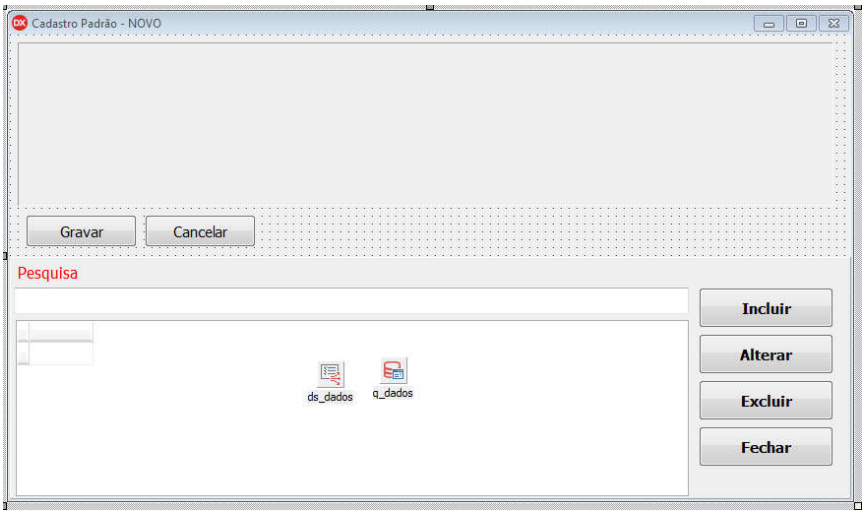
CRIANDO UM FORMULÁRIO BASE PADRÃO PARA SER HERDADO

Clique no menu File -> New -> VCL Form – Delphi

Salve como u_CadastroPadrao.pas, retire-o do auto-create e altere as seguintes propriedades:

- Name: f_CadastroPadrao
- Position: poScreenCenter
- FormStyle: fsMDIChild
- BorderStyle: bsSingle

Adicione os seguintes componentes, e deixe o layout da tela semelhante ao que segue:



Componentes e respectivas propriedades:

- TButton :
  - Name: B_gravar
  - Name: B_cancelar
  - Name: B_incluir
  - Name: B_alterar
  - Name: B_excluir
  - Name: B_fechar
- TEdit :
  - Name: Edit_pesq
- TFDQuery
  - Name: q_dados

Neste momento, vá ao código deste form, procure a cláusula ***uses*** e adicione a unit do datamodule ***u_dm***, save a unit. Neste momento, altere a seguinte propriedade de q_dados:

- ConnectionName: ?????????

- DataSource:
  - Name: ds_dados
  - DataSet: q_dados
- DBGrid :
  - Name: Grid_dados
  - DataSource: ds_dados

Funções, Procedimentos e Eventos a Programar

Coloque os seguintes protótipos de procedimentos, funções e propriedades na declaração da classe do formulário:

```
protected

  procedure controle_visual; virtual;
  function pode_excluir : string; virtual;
  function validar : boolean; virtual;

public
  { Public declarations }

  tabela, campo_chave, campo_pesquisa : string;

  procedure carregar_dados(p:string); virtual;

end;
```

Note que, por padrão, não há uma área protegida na classe, temos que adiciona-la manualmente. Isso é necessário porque essa classe será herdada e, portanto, precisamos que esses métodos sejam visíveis também para as classes filhas. Se fossem privadas, isso não seria possível. Não são públicos porque implementam tarefas específicas dessa classe, e então utilizamos o recurso do ***encapsulamento*** para protegê-los.

A palavra ***virtual*** é necessária porque esses métodos poderão ser reescritos nas classes filhas, conceituando o ***polimorfismo***.

As propriedades públicas, *tabela, campo_chave* e *campo_pesquisa* deverão ser preenchidos pelos objetos dessa classe ou das classes filhas.

Faça a seguinte implementação dos métodos:

```
//---------------------------------------------------------------------------
procedure TF_CadPadrao.carregar_dados(p:string);
begin
     q_dados.SQL.Clear;
     q_dados.SQL.add(' select * ');
     q_dados.SQL.add(' from ' + tabela +'  ');
     q_dados.SQL.add(' where '+campo_pesquisa+' like ' + QuotedStr('%'+p+'%') + ' ');
     q_dados.SQL.add(' order by '+campo_pesquisa+' ');
     q_dados.Open;
end; // carregar_dados;
//---------------------------------------------------------------------------
procedure TF_CadPadrao.controle_visual;
begin
     b_gravar.Enabled:= q_dados.State in [dsEdit, dsInsert];
     b_cancelar.Enabled:= b_gravar.Enabled;
     grid_Dados.Enabled:= not b_gravar.Enabled;
     b_incluir.Enabled:= not b_gravar.Enabled;
     b_alterar.Enabled:= not b_gravar.Enabled;
     b_excluir.Enabled:= not b_gravar.Enabled;
     b_fechar.Enabled:= not b_gravar.Enabled;
     edit_p.Enabled:= not b_gravar.Enabled;
     if edit_p.Enabled then
     begin
          edit_p.Color:= clWhite;
     end
     else begin
          edit_p.Color:= clSilver;
     end;
end;
//---------------------------------------------------------------------------
function TF_CadPadrao.validar : boolean;
begin
     result:= true;
end;

function TF_CadPadrao.pode_excluir : string;
begin
     result:= '';
end;
//---------------------------------------------------------------------------
```

Observações:

- Não há a necessidade e não é correto colocar a palavra ***virtual*** na implementação, quando definidas assim em seus protótipos na classe
- O método validar tem o objetivo de validar o formulário antes de gravar, e neste caso retorna sempre ***true*** pois não conhecemos ainda o formulário. As classes filhas deverão reescrever este método
- O método pode_excluir tem o objetivo de verificar se um registro pode ser excluído, e neste caso retorna sempre ***vazio ‘’*** , pois não conhecemos ainda as regras do negócio para exclusão. As classes filhas deverão reescrever este método

Eventos a Programar:

Edit_pesq OnChangeTracking

carregar_dados(edit_pesq.Text);

q_dados : AfterCancel, AfterClose, AfterDelete, AfterEdit, AfterOpen, AfterInsert, AfterPost

controle_visual;

b_incluir : OnClick

q_dados.Insert;

b_alterar : OnClick

```
procedure TF_CadPadrao.b_alterarClick(Sender: TObject);
begin
     if q_dados.IsEmpty then
     begin
          ShowMessage('Não há um registro selecionado para alterar !');
          exit;
     end;

     q_dados.Edit;
end;
```

b_excluir : OnClick

```
procedure TF_CadPadrao.b_excluirClick(Sender: TObject);
var
   erro : string;
begin
     if q_dados.IsEmpty then
     begin
          ShowMessage('Não há um registro selecionado para excluir !');
          exit;
     end;

     erro:= pode_excluir;

     if erro = '' then
     begin
          if MessageDlg('Deseja realmente excluir este registro !',mtConfirmation,[mbYes,mbNo],0) = mrYes then
          begin
               q_dados.Delete;
               carregar_dados('');
          end;
     end
     else begin
          ShowMessage( erro );
     end;
end;
```

b_fechar: OnClick

close;

b_gravar: OnClick

```
if validar then
begin
     q_dados.Post;
end;
```

b_cancelar: OnClick

```
q_dados.Cancel;
```

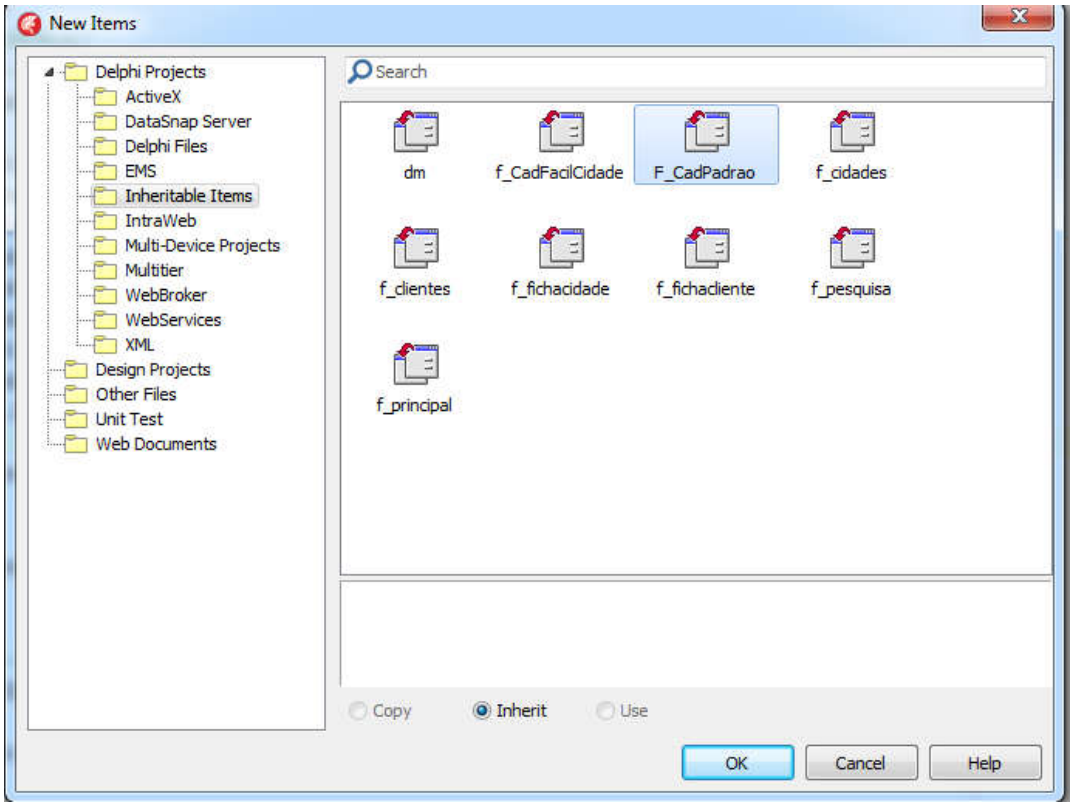
CRIANDO UM CADASTRO BÁSICO HERDADANDO DO FORMULÁRIO PADRÃO

Considerando a tabela unidades, neste tópico será implementado um cadastro para a manipulação dos dados e pesquisa, herdando do formulário de cadastro padrão criado no tópico anterior.

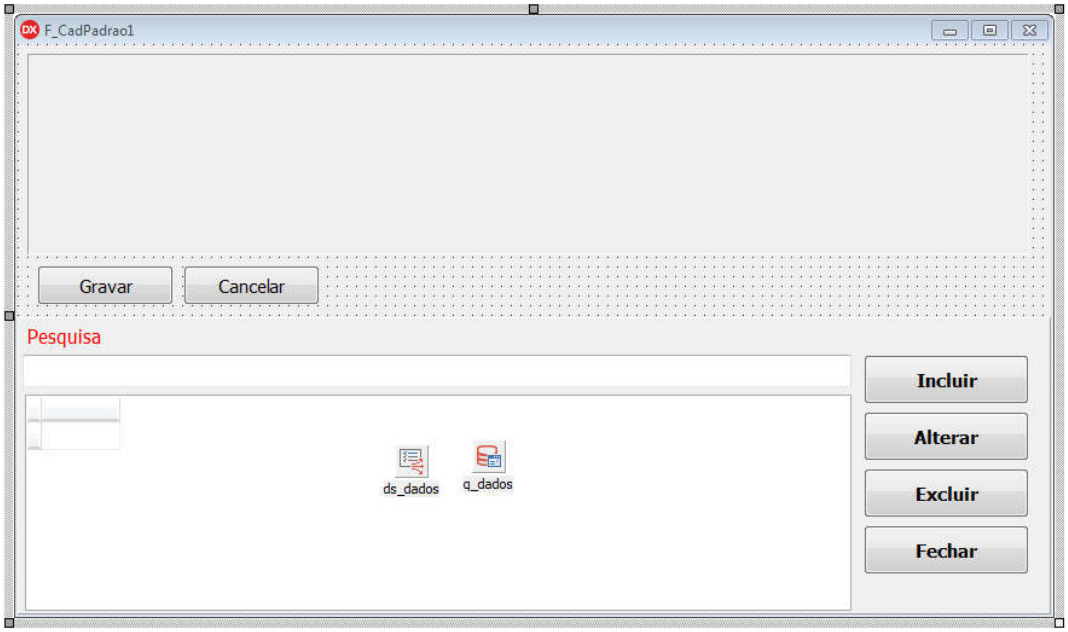
Clique no menu:

File -> New -> Other

Selecione a opção ***Inheritable Items*** no menu esquerdo, o que significa tabela de itens a herdar, e nos itens à direita selecione o formulário ***f_CadastroPadrao***. Clique em ***Ok***, como ilustra a figura abaixo:



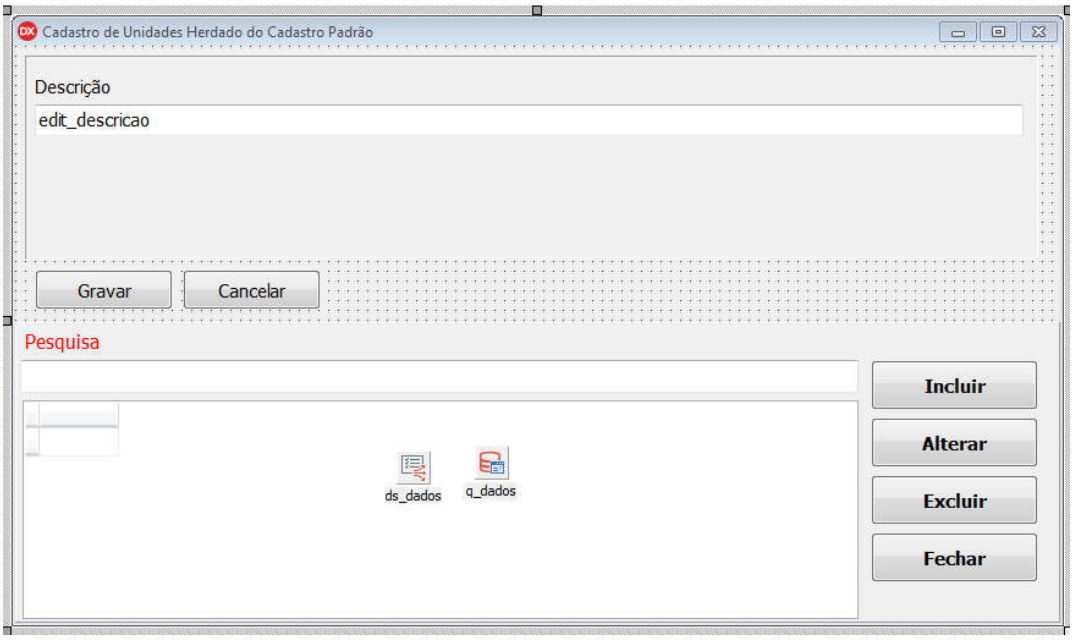
O seguinte formulário irá aparecer:



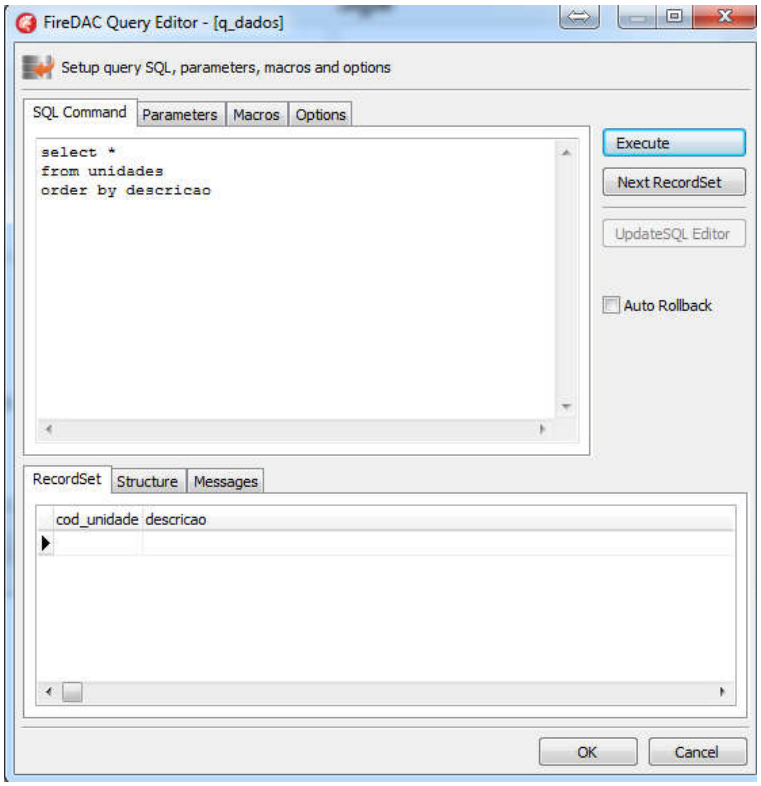
Note que é idêntico ao formulário f_cadastroPadrao, com exceção das propriedades *name* e *caption*. Verique que o código está totalmente limpo, entretanto, todas as funcionalidades implementadas no anterior foram herdadas e funcionam perfeitamente.

Altere sua propriedade *name* para f_unidades_herdado, *caption* para "Cadastro de Unidades – Herdado do Cadastro Padrão", salve como *u_unidades_herdado.pas* e retire-o do auto-create.

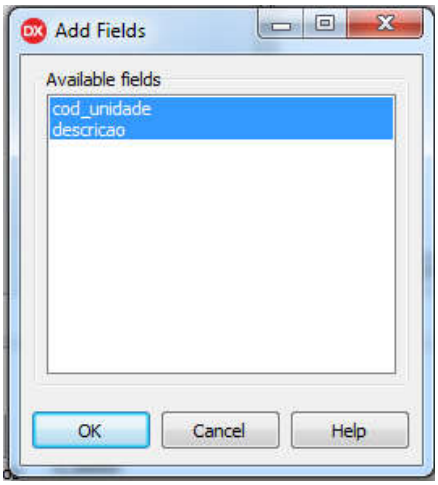
Coloque os seguintes labels e DBEdit (edit_descricao), referente ao campo da tabela unidades, e relacione o DBEdit ao DataSource ds_dados e seu respectivo campos



Abra o DataModulo, volte ao f_unidades_herdado, dê um duplo clique na query q_dados, preenche o seguinte código Sql, clique em *Execute* para testar e se funcionar clique em OK:



Para adicionar os campos na query, clique com o botão direito do mouse sobre q_dados e selecione a opção *Fields Editor*. Com o botão direito do mouse, clique na área branca do editor de campos da query, selecione a opção *Add Fields*, marque todos os campos e clique em Ok.



No formulário principal *f_principal* crie no menu de "Cadastros Herdados" a opção para chamar o cadastro de unidades, e programe seu evento onClick com o seguinte código:

```
// se o formulário não estiver instanciado
if not Assigned(f_unidades_herdado) then
begin
     // instanciando o formulário
     f_unidades_herdado:= Tf_unidades_herdado.create(application);
     f_unidades_herdado.tabela:= 'unidades';
     f_unidades_herdado.campo_chave:= 'cod_unidade';
     f_unidades_herdado.campo_pesquisa:= 'descricao';
     f_unidades_herdado.carregar_dados('');

end
else begin
     f_unidades_herdado.Show;
end;
```

Note que f_unidades_herdado está sublinhado em vermelho porque a unit deste formulário não foi declarada na cláusula *uses* de f_principal.

Retire o f_unidades_herdado do auto-create, compile a aplicação, e o Delphi irá solicitar a inclusão da *unit* u_unidades_herdado em u_principal. Confirme, salve e compile novamente.

PRONTO !!!!!!

O cadastro de unidades está pronto e funcionando, basta agora fazer as validações e incluir especificidades deste cadastro.

FAZENDO ALTERAÇÕES NO FORMULÁRIO FILHO

Para ilustrar a reescrita de métodos em POO com Delphi, vamos reescrever o método validar para que não permita a gravação dos dados quando algum campo esteja em branco.

Na definição da classe *f_unidades_herdado*, na área protegida, adicione o seguinte protótipo do método *validar*:

```
protected

   function  validar : boolean; override;
```

Note a palavra ***override***, onde na classe pai era ***virtual***. Isso informa ao Delphi que este método pertencente à classe pai e será reescrito.

Faça a implementação da função *validar*, como segue:

```
  private
    { Private declarations }

  public
    { Public declarations }

  protected
    { Protected declarations }

    function validar : boolean; override;

  end;

var
  f_unidades_herdado: Tf_unidades_herdado;

implementation

{$R *.dfm}

function Tf_unidades_herdado.validar : boolean;
begin
     if trim(edit_descricao.Text) = '' then
     begin
          ShowMessage('A descrição deve ser informada !');
          result:= false;
          exit;
     end;

     // tudo ok para gravar
     result:= true;
end;
```