

Nesta seção você encontra artigos intermediários sobre Delphi Win32 e Delphi .NET

P00 e Client/Server

Saiba como colocar em prática o uso dos principais recursos da P00 em aplicações reais



Maikel Marcelo Scheid

(maikelscheid@gmail.com)

é técnico em Informática com ênfase em Análise e Programação de Sistemas. Atua na área de Desenvolvimento de Softwares em Delphi para plataforma Win32 e .NET com banco de dados Firebird e MS SQL há 3 anos.

Veremos neste artigo uma aplicação simplificada do uso da Programação Orientada a Objetos (POO) em classes de negócios através de uma aplicação *Client/Server*. Estudaremos algumas estruturas de programação utilizadas e, dentre elas, uma das principais características da POO: a capacidade de reutilização de códigos.

Criaremos uma *Unit* de funções com códigos genéricos, que não serão utilizados em apenas uma aplicação, mas que se empregarão perfeitamente na grande maioria dos nossos sistemas *Client/Server*, seja com funções de manipulação de formulários, atualização de registros em componentes de acessos a dados ou qualquer outra funcionalidade.

Para usar os recursos da POO precisamos entender e planejar muito bem nossas aplicações, estudar bem as regras de negócios e assim construir estruturas de códigos que poderão simplificar em muito nosso trabalho em casos de alterações de

estrutura e manutenções do sistema.

Tomaremos como exemplo no decorrer deste artigo uma pequena aplicação desenvolvida em BDS 2006 com o acesso a uma estrutura de dados do Firebird 1.5 simulando um cadastro de produtos.

Utilizaremos a classe de uma forma simples em um exemplo prático onde criaremos métodos encapsulados para realizar cálculos referentes aos valores da simulação de itens de uma venda e também do fechamento da venda em relação à devolução de troco se assim necessário.

Conhecendo a estrutura de uma classe (*class*)

A estrutura de uma classe é definida por um conjunto de membros que definem o estado e o comportamento de um objeto geralmente implementando métodos e atributos. Os métodos de uma classe são geralmente divididos em públicos ("Public") e privados ("Private"). Os métodos públicos são de livre acesso durante seu

uso. Já os privados são acessados somente internamente pela classe, ou seja, ficam encapsulados.

Nas definições desse conjunto de peculiaridades que formam uma classe, podemos observar a seguinte estrutura:

Unit: informa que o arquivo é referente a uma unidade seguida do nome da mesma.

Interface: estabelece a fronteira de comunicação entre a classe e o ambiente externo. É aqui também que declaramos nossos novos tipos e métodos ("Procedimentos ou Funções").

Uses: esta cláusula permite listar e utilizar objetos da nossa pasta de trabalho. Quando declaramos essa cláusula numa determinada parte da unidade estamos dizendo ao Delphi que poderemos utilizar os métodos e/ou propriedades que o objeto da lista proporciona acesso.

Type: com esta cláusula podemos declarar novos tipos e/ou classes.

Implementation: é a partir desta cláusula que codificamos nossos métodos, funções e procedimentos. Nesta seção também podemos declarar novos tipos e constantes.

Dentro desta mesma estrutura ainda poderemos ter a herança de classes, onde uma classe pode ser criada a partir de outra já existente e herdar os atributos e comportamentos da estrutura.

Criando o banco de dados

Para iniciarmos nosso exemplo prático, criaremos uma estrutura simplificada de apenas uma tabela, *Produtos* (Figura 1).

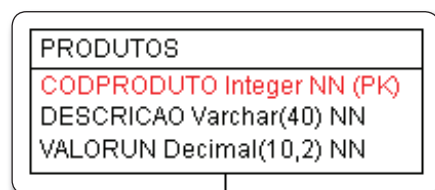


Figura 1. Diagrama ER da tabela

Listagem 1. Script de criação da base de dados

```

SET SQL DIALECT 3;
SET NAMES Win1252;

CREATE DATABASE 'C:\SISVendas\dbvendas.fdb'
USER 'SYSDBA' PASSWORD 'masterkey'
PAGE_SIZE 4096
DEFAULT CHARACTER SET Win1252;

CREATE TABLE PRODUTOS (
  CODPRODUTO Integer NOT NULL,
  DESCRICAO Varchar(40) NOT NULL,
  VALORUN Decimal(10,2) NOT NULL,
  Primary Key (CODPRODUTO)
);
  
```

A partir desta estrutura de dados iremos utilizar os códigos orientados a objetos.

Definida a estrutura no programa de modelagem de dados, realize a exportação do script da **Listagem 1** e utilize o gerenciador de banco de dados para então fazer a importação e criação da base de dados. Crie sua base com o nome de "dbvendas.fdb".

Com o banco de dados já criado, acesse a tabela de "Produtos" e crie um auto-incremento na chave primária "COD-PRODUTO". Depois de realizada esta configuração, nossa base de dados estará

apta a ser utilizada e passaremos agora à criação da nossa aplicação de cadastros e consultas. Utilizarei neste artigo a BDS 2006, mas fique à vontade para escolher outra versão.

Criando a aplicação

Crie uma nova VCL Forms Application – Delphi for Win32 no menu File do BDS 2006 e salve o projeto no caminho "C:\SISVendas\" com o nome de "prjVendas.bdsproj". Salve a **Unit** principal como "uPrincipal.pas". Altere as seguintes propriedades do formulário principal:

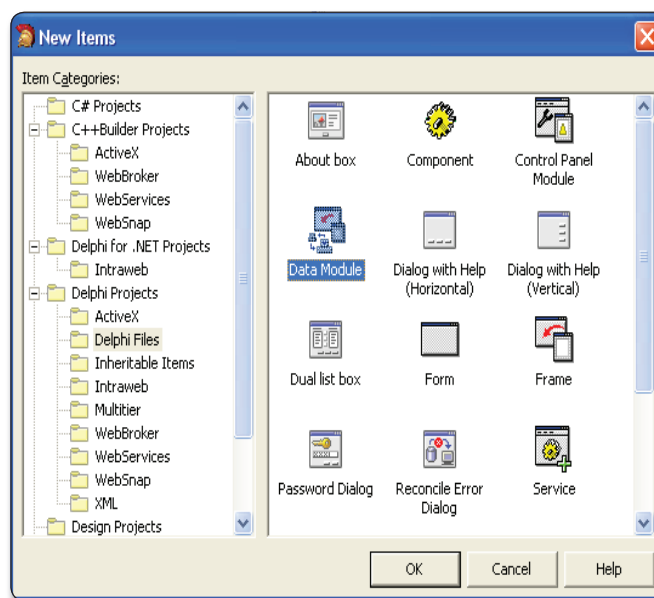


Figura 2. Adicionando o DataModule ao projeto

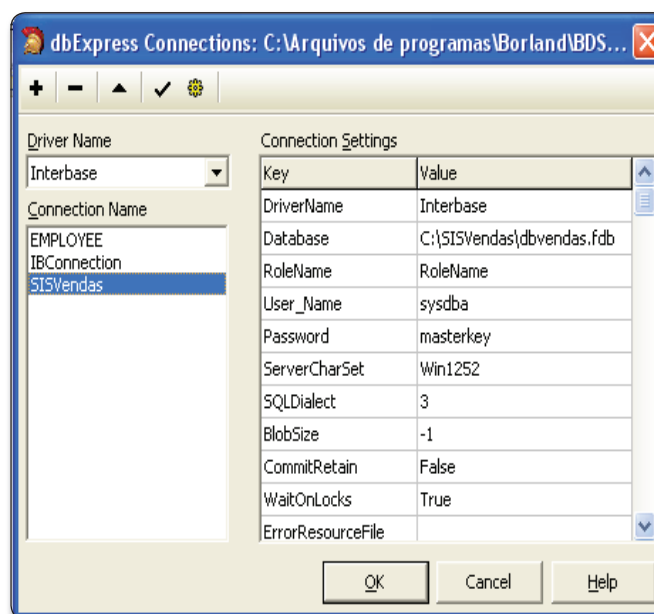


Figura 3. Configurando o acesso à fonte de dados

No formulário principal altere o seu *Caption* para “Sistema de Vendas” e marque a opção *biMaximize* como *False* na propriedade *BorderIcons*. Em seguida altere a propriedade *Name* para “frmPrincipal” e o *KeyPreview* para *True*.

Adicione agora a sua aplicação um *Data Module* o qual utilizaremos para concentrar a conexão com o banco de dados e também os componentes de consultas e manipulação dos registros. Acesse o menu *File|New>Other>Delphi Files* e selecione *Data Module* e confirme (Figura 2).

Com o *Data Module* já criado, altere seu nome para “DM” e salve a *Unit* como “uDM.pas”. Arraste para o mesmo um componente *SQLConnection* da paleta *dbExpress*, altere o nome para “sqlCONEXAO”. Com um duplo clique sobre o mesmo faça a configuração de maneira que seja possível conectar-se ao banco “dbvendas.fdb” criado anteriormente como na Figura 3.

Realizada a configuração anterior faremos a configuração dos componentes de cadastros de produtos. Adicione um componente *SQLDataSet* da paleta *dbExpress* (“sdsProdutos”) apontando-o para o *sqlConexao* usando a propriedade *SQLConnectoin*. Em seguida digite a seguinte instrução *SQL* na propriedade *CommandText*:

```
select CODPRODUTO, DESCRICAO, VALORUN
from PRODUTOS where CODPRODUTO = :CODPRODUTO
```

Note que estamos usando uma instrução *SQL* simples com a passagem de um parâmetro, “CODPRODUTO”. Essa estrutura de programação em aplicações *Client/Server* é muito importante, pois evita de trazermos muitos registros em um resultado quando queremos apenas um registro. Qual seria a necessidade de carregar esses 20 mil registros de uma única vez na memória de um *ClientDataSet* se precisaremos de apenas 1 registro por vez? Nenhuma. Criaremos um formulário de pesquisa onde o usuário irá buscar o dado que deseja alterar, e tão somente este será carregado na memória do *ClientDataSet*. São estas as técnicas de programação que dão performance a sua aplicação.

Precisamos agora configurar o parâmetro adicionado, por isso acesse a propriedade *Params* do *sdsProdutos* e selecione o parâmetro CODPRODUTO alterando sua propriedade *DataType* para *ftInteger*. Com um duplo clique sobre o componente abra o *Fields Editor* e adicione todos os campos da consulta.

Selecione o campo CODPRODUTO e altere sua propriedade *ProviderFlags* marcando como *True* a opção *pflnKey*. Aproveite e modifique também a propriedade *Required* para *False*.

Agora configure o campo DESCRICAO alterando sua opção *pflnWhere* em *ProviderFlags* para *False*. Repita a última

configuração para o campo VALORUN e por último marque a propriedade *Currency* para *True*.

Feche o *Fields Editor* e adicione agora ao DM um componente *DataSetProvider* (“dspProdutos”). Relacione sua propriedade *DataSet* ao *sdsProdutos*. Adicione também um *ClientDataSet* (“cdsProdutos”) e relacione a propriedade *ProviderName* ao *dspProdutos*.

Uma configuração importante é trazer os parâmetros do *SQLDataSet* para que possam ser configurados diretamente no *ClientDataSet*, para isso clique com o botão direito do mouse sobre o *ClientDataSet* e selecione a opção *Fetch Params*. Observe na propriedade *Params* que agora o parâmetro CODPRODUTO está configurado no *ClientDataSet*. Abra o *Fields Editor* e adicione todos os campos da seleção assim como foi feito no *sdsProdutos*.

Crie agora um novo formulário usando o menu *File|New>Form*. Salve-o como “uProdutos.pas” e altere o nome do formulário para “frmProdutos”. Adicione um *Button* ao formulário e altere sua propriedade *Caption* para “Pesquisar”. Volte ao DM, abra o *Fields Editor* do *cdsProdutos* selecionando todos os campos e arrastando-os para o *frmProdutos*. Inclua um componente *DBNavigator* conectando sua propriedade *DataSource* ao *DataSource1* automaticamente adicionado pelo Delphi. Selecione o *DBEdit1* e marque *True* na sua propriedade *ReadOnly*. Arranje os componentes no formulário conforme Figura 4.

Nota: Não esqueça de criar o auto-incremento na tabela de PRODUTOS no banco de dados.

No evento *OnShow* do formulário digite o código a seguir, que será responsável por abrir a conexão com o banco quando o formulário for chamado:

```
DataSource1.DataSet.Open;
```

Da mesma forma é necessário fechar o *DataSet* no evento *OnClose* do formulário, para isso digite o código a seguir no evento citado:

```
DataSource1.DataSet.Close;
```

Usando Units

Acabamos de criar nosso formulário, mas que ainda não está sendo chamado

Listagem 2. Implementação da procedure de criação de formulários

```
procedure CriarFormulario(FormClasse: TComponentClass;
  NomeForm : TForm);
begin
  Application.CreateForm(FormClasse, NomeForm);
  try
    NomeForm.ShowModal;
  finally
    NomeForm.Free;
  end
end
```

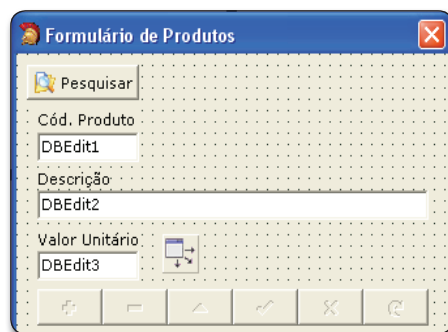


Figura 4. Formulário de Produtos

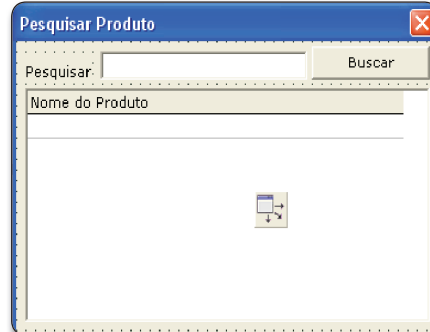


Figura 5. Formulário de pesquisa de produtos

em nenhuma outra *Unit*. Para realizar esta chamada iremos criar uma *Procedure* que irá criar o formulário e dar o *ShowModal* ao mesmo. Você poderá usar esta *Procedure* não somente com a janela de produtos, mas sim com todos os demais formulários do seu sistema.

Para começar adicione uma nova *Unit* ao projeto usando a opção *File|New>Other>Delphi Files>Unit*. Salve-a como "uFuncoes.pas" e declare as *Units Forms* e *Classes a Uses da Unit criada* logo abaixo a declaração *Interface*. Em seguida adicione a seguinte *Procedure* logo abaixo de *Uses*:

```
procedure CriarFormulario(FormClasse:
TComponentClass; NomeForm : TForm);
```

Na seção *Implementation* realize a codificação da *Procedure*, adicionando o código da **Listagem 2**.

Volte agora até o formulário principal do sistema, adicione ao topo um *Button*, altere sua propriedade *Caption* para "Produtos" e adicione a seguinte linha de código no evento *OnClick* do *Button*:

```
CriarFormulario(TfrmProdutos,frmProdutos);
```

Perceba que nesta linha de código estamos passando a classe e o nome do formulário que queremos criar. Ao executar a aplicação e clicar sobre o botão de produtos veja que o formulário é criado em *runtime*.

Nota: Acesse o menu *Project>Options* e retire o formulário de produtos da lista de criação automática do projeto e mova o *Data Module* para cima de maneira que seja o primeiro a ser carregado pelo sistema.

Provavelmente você deve ter tentado realizar o cadastro de um produto, que aparentemente tenha funcionado, mas o registro não consta no banco de dados, pois o componente *DBNavigator* apenas realiza a ação de *Post*, mas precisamos aplicar estes registros na base de dados. Para isso volte a *Unit* de funções e crie uma nova *Procedure* que receba como parâmetro um objeto do tipo *ClientDataSet* e codifique-a como a seguir. Declare *DBClient* ao *Uses da Unit* de funções.

```
procedure AplicarRegistro(NomeCDS :
TClientDataSet);
begin
  NomeCDS.ApplyUpdates(-1);
end;
```

A *Procedure* anterior recebe como parâmetro o nome de um *ClientDataSet*, que recebe um *ApplyUpdates* fazendo a gravação dos registros no banco de dados. Volte ao *DM* e adicione no evento *AfterPost* e *AfterDelete* do *cdsProdutos* a seguinte linha de código:

```
AplicarRegistro(TClientDataSet(DataSet));
```

Nesse momento ao cadastrar um produto verá que o mesmo foi corretamente adicionado ao banco de dados. Faremos agora um novo formulário o qual usaremos para pesquisar os produtos para a alteração. Adicione um novo formulário, salve-o como "uSearchProduto.pas" e altere sua propriedade *Caption* para "Pesquisa de Produtos".

Vá até o *Data Module* e inclua um novo trio de componentes sendo: um novo *SQLDataSet* ("sdsPesqProduto") com a propriedade *SQLConnection* igual a *sqlConexao* e *CommandText* com a instrução *SQL* a seguir:

```
select CODPRODUTO, DESCRICAO from PRODUTOS
where Upper(descricao) like :descricao
```

Configure na propriedade *Params* o parâmetro *DESCRICAO* como *DataType = ftString*. Insira um *DataSetProvider* ("dspPesqProduto") modificando sua propriedade *DataSet* para *sdsPesqProduto*. E por último inclua um novo *ClientDataSet* ("cdsPesqProduto") modificando sua propriedade *ProviderName* para *dspPesqProduto*. No *ClientDataSet* clique com

PENSE...

QUANTO TEMPO
VOCÊ GASTARIA
PARA DESENVOLVER
COBRANÇA COM BOLETOS
BANCÁRIOS PARA
APENAS UM BANCO
NO SEU SOFTWARE

COBREBEMX



56 BANCOS E MAIS DE 430 CARTEIRAS DE COBRANÇA PARA IMPRESSÃO E/OU ENVIO DE BOLETO BANCÁRIO POR EMAIL;



GERAÇÃO DE BOLETOS ON LINE;



GERAÇÃO E LEITURA DE ARQUIVOS (REMESSA/RETORNO) NOS PADRÕES FEBRABAN E CNAB;



MAIS DE 40 EXEMPLOS EM DIVERSAS LINGUAGENS DE PROGRAMAÇÃO



Tecnologia

DOWNLOADS E INFORMAÇÕES EM WWW.COBBREBEM.COM

Listagem 3. Implementação dos códigos da criação da classe

```
unit uClassPVendas;
interface
uses
  SysUtils;
type
  TVendas = class
  { Cria a classe para calcular os itens da Venda }
  private
    FQuantidade : Real;
    FValorItem : Real;
    FTotalItem : Real;
    procedure SetQuantidade(const Value: Real);
    procedure SetTotalItem(const Value: Real);
    procedure SetValorItem(const Value: Real);
  public
    procedure SomaValorTotalItem;
    property Quantidade : Real read fQuantidade write setQuantidade;
    property ValorItem : Real read fValorItem write setValorItem;
    property TotalItem : Real read fTotalItem write setTotalItem;
  end;
  TValores = class
  { Cria a classe para calcular devolução de troco }
  private
    FTotalVenda : Real;
    FDinheiroRecebido : Real;
    FTroco : Real;
    FDesconto : Real;
    procedure SetDesconto(const Value: Real);
    procedure SetDinheiroRecebido(const Value: Real);
    procedure SetTotalVenda(const Value: Real);
    procedure SetTroco(const Value: Real);
  public
    procedure CalcularTroco;
    property TotalVenda : Real read fTotalVenda write setTotalVenda;
    property DinheiroRecebido : Real read fDinheiroRecebido write setDinheiroRecebido;
    property Troco : Real read fTroco write setTroco;
    property Desconto : Real read fDesconto write setDesconto;
  end;
implementation
{ TVendas }
procedure TVendas.SetQuantidade(const Value: Real);
begin
  if Value > 0 then
    fQuantidade := Value
  else
    raise
      Exception.Create('Quantidade deve ser maior que 0');
end;
procedure TVendas.SetTotalItem(const Value: Real);
begin
  FTotalItem := Value;
end;
procedure TVendas.SetValorItem(const Value: Real);
begin
  if Value > 0 then
    fValorItem := Value
  else
    raise
      Exception.Create('Valor deve ser maior que R$0,00');
end;
procedure TVendas.SomaValorTotalItem;
begin
  try
    FTotalItem := FValorItem * fQuantidade;
  except
    on E:Exception do
      raise
        Exception.Create('Erro durante o cálculo');
  end;
end;
{ TValores }
procedure TValores.CalcularTroco;
begin
  Troco := DinheiroRecebido - (TotalVenda - Desconto);
end;
procedure TValores.setDesconto(const Value: Real);
begin
  FDesconto := Value;
end;
procedure TValores.setDinheiroRecebido(const Value: Real);
begin
  FDinheiroRecebido := Value;
end;
procedure TValores.setTotalVenda(const Value: Real);
begin
  FTotalVenda := Value;
end;
procedure TValores.setTroco(const Value: Real);
begin
  FTroco := Value;
end;
end.
```

o botão direito do mouse e selecione a propriedade *Fetch Params*.

Volte agora ao formulário, adicione um *Label* com o *Caption* “Pesquisar” e ao lado coloque um *Edit* com a propriedade *Text* em branco. Adicione um *Button* com o *Caption* “Buscar”. Insira um componente *DBGrid* da paleta *Data Controls* e em seguida um *DataSource*.

Pressione **ALT + F11** e inclua a *Unit* do *Data Module* ao *Uses* do formulário de pesquisa desta forma poderá ligar o *DataSource* inserido diretamente ao *cdsPesqProdutos* pela propriedade *DataSet*.

Relacione também a propriedade *DataSource* do *DBGrid* ao *DataSource1* recém adicionado. Organize os componentes no formulário conforme **Figura 5**.

No evento *OnClick* do *Button* faça a seguinte codificação, que será responsável por passar o valor digitado ao parâmetro do componente anteriormente configurado e realizar a pesquisa:

```
DM.cdsPesqProdutos.Close;
DM.cdsPesqProdutos.Params[0].AsString :=
  UpperCase(Edit1.Text)+'%';
DM.cdsPesqProdutos.Open;
```

Agora no evento *OnDblClick* do *DBGrid* codifique conforme a seguir. O código será responsável por passar o código do produto selecionado ao parâmetro do primeiro *ClientDataSet* que configuramos habilitado para trazer somente um registro por vez:

```
DM.cdsProdutos.Close;
DM.cdsProdutos.Params[0].AsInteger :=
  DM.cdsPesqProdutos.CODPRODUTO.AsInteger;
DM.cdsProdutos.Open;
Close;
```

Figura 6. Formulário para simulação de venda

Antes de testar novamente a aplicação faça a chamada ao formulário de pesquisa usando o código a seguir no evento *OnClick* do botão de pesquisa.

```
CriarFormulario(
  TfrmSearchProduto, frmSearchProduto);
```

Veja que novamente estamos utilizando a mesma função que utilizamos para chamar antes o formulário de produtos, porém agora passando como parâmetro as definições para criação do formulário de pesquisa de produtos.

Da mesma forma que criamos essas *Procedures* você poderá implantar vários outros métodos orientados a objeto durante sua programação e assim reutilizar os códigos nesta aplicação ou ainda reutilizar toda a *Unit* de funções para demais projetos.

Usando classes de negócios

Para mostrar os benefícios da utilização de classes de negócios em sua programação, simularemos uma venda com itens, onde iremos nos beneficiar de regras concentradas na nossa classe para o cálculo dos valores dos itens e também valores do fechamento da compra.

No seu projeto adicione uma nova *Unit* e salve-a como “uClassPVendas.pas”. Adicione a *Unit* o código da **Listagem 3**, onde na mesma estrutura estamos declarando duas classes (“TVendas” e “TValores”) e cada uma delas com suas devidas variáveis e métodos encapsulados com somente as propriedades visíveis ao ambiente exterior da classe.

Note que em algumas *procedures* estamos verificando os valores recebidos e se não atenderem a regra estabelecida, uma exceção será criada. Você poderá fazer inúmeros testes com os valores recebidos, o que irá dar performance às suas regras de negócios.

Para implantar a utilização da classe no projeto, criaremos um formulário onde iremos digitar os valores, mas que nas suas aplicações certamente serão valores resultantes da consulta de itens de uma base de dados.

Criado um novo formulário, salve-o como “uCalcValores.pas”, altere seu nome para “frmCalcValores” e o *Caption* para “Calculo de Valores”. Coloque seis componentes *Label* ao formulário, seis componentes *Edit* e dois *Buttons*. Altere o

Listagem 4. Implementação dos códigos para cálculo de valores do item

```
procedure TfrmCalcValores.Button1Click(Sender: TObject);
var
  Item : TVendas;
begin
  Item := TVendas.Create;
  Item.Quantidade := StrToFloat(edtQuantidade.Text);
  Item.ValorItem := StrToFloat(edtValorUnitario.Text);
  Item.SomaValorTotalItem;
  edtTotalItem.Text := FormatFloat('#0.00', Item.TotalItem);
end;
```

Listagem 5. Implementação dos códigos para cálculo de devolução de troco

```
procedure TfrmCalcValores.Button2Click(Sender: TObject);
var
  Valor : TValores;
begin
  Valor := TValores.Create;
  Valor.TotalVenda := StrToFloat(edtTotalItem.Text);
  Valor.DinheiroRecebido := StrToFloat(edtDinheiroRecebido.Text);
  Valor.Desconto := StrToFloat(edtDesconto.Text);
  Valor.CalcularTroco;
  edtTroco.Text := FormatFloat('#0.00', Valor.Troco);
end;
```

nome dos componentes de acordo com a nomenclatura de cada *Label* e organize-os conforme **Figura 6**.

Nota: Pressione *ALT + F11* para incluir a *Unit* de funções de cálculo ao *Uses* do formulário de cálculo.

Vamos iniciar a codificação pelo botão *Calcular Total do Item* incluindo em seu evento *OnClick* o código da **Listagem 4**.

Observe que estamos criando uma variável com o tipo da classe (“TVendas”), e na implementação da *Procedure* estamos dando um *Create* na mesma. Após criada conseguimos usar todas as propriedades e métodos que definimos como públicos na classe.

Você poderá notar que nenhum dos métodos que estão encapsulados são listados ao usarmos a classe. No código anterior, estamos passando os valores para as propriedades da classe, convertemos o texto digitado nos *Edits* para *Float* e passamos para cada propriedade.

Antes de jogar o resultado de volta ao *Edit*, executamos o método responsável pelo cálculo criado na classe. Ao testar sua aplicação, faça um teste digitando uma quantidade de produtos e informe o valor zero e veja a forma com que a exceção será tratada.

Da mesma forma faremos uso da outra classe criada na estrutura, atribuindo o código da **Listagem 5** ao evento *OnClick* do botão de cálculo do troco, onde também criaremos uma variável instanciada ao objeto da classe,

dando um *Create* na implementação do método e então passando a usar suas propriedades passando-lhes os devidos valores.

Em seguida chamamos o método para realizar o cálculo, retornando assim através de outra propriedade da classe a quantidade de troco que deverá ser devolvida.

Você poderá ainda fazer outras verificações na classe, como exemplo se a quantidade de dinheiro recebida for suficiente para pagar o valor total da venda, se o valor do desconto estiver em determinada escala de porcentagem, entre outras várias funções que irão definir as regras do negócio para a venda de itens.

Perceba que essa classe poderá ser utilizada de várias formas diferentes, e que o principal benefício é que os códigos ficam centralizados em um único espaço, diminuindo o tempo de manutenção do sistema.

Conclusão

Vimos nesse artigo de forma simples exemplos práticos que irão trazer mais credibilidade e rapidez no processo de programação. Vimos uma forma de como reutilizar códigos de métodos orientados a objetos, o que irá evitar o seu re-trabalho em fazer um sistema com funções internas semelhantes, faz-se uma única função e utiliza-se em todo o resto da aplicação.

No uso de classes, vimos uma forma simples de centralizar as regras de negócio, onde em casos de manutenção de software você terá bons ganhos em relação ao uso dessa técnica de desenvolvimento. ●