1 - CONCEITO DE PROGRAMAÇÃO ORIENTADA A OBJETO

Para compreendermos melhor o novo ambiente de desenvolvimento da *Borland* o *Delphi* é necessário que você, aprenda e, tenha em mente os conceitos de POO (Programação Orientada a Objetos), não confunda os conceitos com POE (Programação Orientada a Eventos), mas ao longo do curso você vão notar as sensíveis diferenças que existem entre esses dois conceitos.

A POO e a POE são facilmente confundidas, mas lembre-se a POO contém a POE, mas a POE não contém a POO, um objeto pode existir mesmo que não exista nenhum evento associado a ele, mas um evento não pode existir se não houver um objeto a ele associado.

Outra característica que pode causar confusão são ambientes Orientados a Objetos e ambientes Baseados em Objetos. Em ambiente Orientado a Objetos consegue-se criar e manipular objetos enquanto que o Baseado em Objetos não é possível a criação de objetos apenas a sua manipulação.

A POO é um conceito desenvolvido para facilitar o uso de códigos de desenvolvimento em interfaces gráficas. Sendo a *Borland*, uma das primeiras a entrar neste novo conceito, possui suas principais linguagens de programação (tais como *Object Pascal* e *C++*), totalmente voltadas para este tipo de programação.

A POO atraiu muitos adeptos principalmente pelo pouco uso de código que o projeto (diferente de sistema) carrega no programa fonte, ao contrário das linguagens mais antigas. O resultado desta “limpeza” no código resulta que a manutenção do projeto torna-se muito mais simples.

2 - CONCEITOS BÁSICOS DA ORIENTAÇÃO A OBJETOS

2.1 Objeto - é qualquer estrutura modular que faz parte de um produto. Uma janela, por exemplo, é um objeto de uma casa, de um carro ou de um software com interface gráfica para o usuário.

2.2 Atributos - São as características do objeto, como cor e tamanho, a janela, por exemplo, tem atributos como o modelo, tamanho, abertura simples ou dupla, entre outros.

2.3 Encapsulação - é um mecanismo interno do objeto "escondido" do usuário. Uma pessoa pode abrir uma janela girando a tranca sem precisar saber o que há dentro dela.

2.4 Ação - é a operação efetuada pelo objeto. Todas as janelas, por exemplo, controlam a iluminação e temperatura ambiente, dependendo do seu design.

2.5 Herança - um objeto novo nem sempre é criado do zero. Ele pode "herdar" atributos e ações de outros já existentes. Um basculante herda atributos das janelas e das persianas.

2.6 Polimorfismo - é a capacidade de objetos diferentes reagirem segundo a sua função a uma ordem padrão. O comando "abre", por exemplo, faz um objeto entrar em ação, seja ele uma janela, uma porta ou uma tampa de garrafa.

2.7 Ligação - é quando um objeto conecta a sua ação a outro. Um sensor de claridade, por exemplo, ativa o acendimento automático da iluminação de rua.

2.8 Embutimento - Permite a um objeto incorporar funções de outros, muito parecido com um liquidificador que mói carne com a mudança do tipo da lâmina.

3 - OBJECT PASCAL (RESUMO) ELEMENTOS DE CONTROLE OPERADORES

Em Object Pascal o operador de atribuição é "**:=**".

Operador	**Descrição**
3.1 Operadores Numéricos ou Aritméticos	
+	Adição ou união de conjuntos
-	Subtração ou diferença de conjuntos
*****	Multiplicação ou interseção de conjuntos
/	Divisão de Real
div	Divisão de Inteiro
mod	Resto de divisão de Inteiros
3.2 Operadores Relacionais e Lógicos	
=	Igual

<>	Diferente
<	Menor
>	Maoir
<=	Menor ou igual
>=	Maior ou igual
not	*Não* booleano ou bit voltado para *não*
and	*E* booleano ou bit voltado para *e*
or	*Ou* booleano ou bit voltado para *ou*
xor	*Ou* exclusivo booleano ou bit voltado para *ou* exclusivo
shl	Deslocamento de bits à esquerda
shr	Deslocamento de bits à direita

4 - CONCATENAÇÃO +

Ex: Edit1.text: =Edit2.text+'Texto Fixo';

5 - TIPOS DE DADOS

O Delphi trata vários tipos de dados padrão, segue uma descrição sucinta desses tipos.

6 - TIPOS INTEIROS

São tipos numéricos exatos, sem casas decimais. O tipo *Integer* é o tipo inteiro padrão.

Tipo	Tamanho em Bytes	Valor Mínimo	Valor Máximo
ShortInt	1	-128	127
SmallInt	2	-32768	32767
Longint	4	-2147483648	2147483647
Byte	1	0	255
Word	2	0	65535
Integer	4	-2147483648	2147483647
Cardinal	4	0	2147483647

7 - TIPOS REAIS

São tipos numéricos com casas decimais. O tipo *Double* é o tipo real padrão.

Tipo	Tamanho em Bytes	Valor Mínimo	Valor Máximo	Dígitos Significativos
Real	6	10^{-39}	10^{38}	11-12
Single	4	10^{-45}	10^{38}	7-8
Double	8	10^{-324}	10^{308}	15-16
Extended	10	10^{-4932}	10^{4932}	19-20
Comp	8	-10^{18}	10^{18}	19-20
Currency	8	-10^{12}	10^{12}	19-20

8 - TIPOS TEXTO

Os tipos texto podem operar com caracteres simples ou grupos de caracteres. O tipo texto padrão é o tipo string.

Tipo	Descrição
Char	Um único caractere ASCII
String	Texto alocado dinamicamente pode ser limitado a 255 caracteres conforme configuração.
PChar	String terminada em nulo (#0), usada geralmente nas funções da API do Windows.

9 - COMENTÁRIOS

Existem 3 estilos de comentário no Delphi, como mostrado abaixo:

```
(*Comentário*)
{ Comentário }
// Comentário - tudo, desde o marcador de comentário, até o término da linha é ignorado.
```

Cuidado com as diretivas de compilação, pois elas são delimitadas por chaves e podem ser confundidas com comentários.

A diretiva de compilação mostrada abaixo é incluída em todas as Units de Forms.
{$R*.DFM}

10 - INSTRUÇÕES

Os programas são compostos por instruções, que são linhas de código executável. Exemplos de instruções simples são atribuições, mensagens entre objetos, chamadas de procedimentos, funções e métodos. As instruções podem ser divididas em várias linhas, o que indica o fim de uma instrução é o ponto e vírgula no final.

Você pode usar várias instruções agrupadas em uma instrução composta, como se fosse uma só instrução. Uma instrução composta delimitada pelas palavras reservadas *begin* e *end*. Toda instrução, simples ou composta, é terminada com um ponto-e-vírgula.

11 - CONVERSÕES DE TIPO

Freqüentemente você vai precisar converter um tipo de dado em outro, como um número em uma string. Para essas conversões você pode as rotinas de conversão de tipos.

12 - ROTINAS DE CONVERSÃO

As principais rotinas de conversão estão listadas na tabela abaixo. Caso você tente usar uma dessas rotinas em uma conversão inválida, pode ser gerada uma exceção.

Rotina	**Descrição**
Chr	Byte em Char
StrToInt	String em Integer
IntToStr	Integer em String
StrToIntDef	String em Integer, com um valor default caso haja erro
IntToHex	Número em String Hexadecimal
Round	Arredonda um número real em um Integer
Trunc	Trunca um número real em um Integer
StrToFloat	String em Real
FloatToStr	Real em string
FormatFloat	Número real em string usando uma string de formato
DateToStr	TDateTime em string de data, de acordo com as opções do Painel de Controle
StrToDate	String de data em TDateTime
TimeToStr	TDateTime em Strind de Hora
StrToTime	String de hora em TDateTime
DateTimeToStr	TDateTime em string de data e hora

StrToDateTime	String de data e hora em TDateTime
FormatDateTime	TDateTime em string usando uma string de formato
VarCast	Qualquer tipo em outro usando argumentos do tipo Variant
VarAsType	Variante em qualquer tipo
Val	String em número, real ou inteiro
Str	Número, real ou inteiro, em String

13 - DECLARAÇÕES

Declarações descrevem ações de um algoritmo a serem executadas.

13.1 Begin... End**;**

.....Prende um conjunto de declarações em um bloco de comandos determinado. A sintaxe do comando é: **BEGIN {comandos} END;**. Ex:

```
begin
   { ... comandos iniciais ... }
   begin
     { ... bloco 1 ... }
   end;
   begin
     { ... bloco 2 ... }
   end;
   { ... comandos finais ... }
end;
```

13.2 If... Then... Else

Esta expressão escolhe entre o resultado de uma condição booleana o caminho verdadeiro (then) ou falso (else). A sintaxe do comando é: **IF {condição} THEN {bloco de comandos} ELSE {bloco de comandos};**. Ex:

```
begin
   { ... comandos iniciais ... }
   if  x > 2 then
     { ... Bloco verdadeiro ... }
   else
     { ... Bloco falso ... };
end;
```

13.3 Goto**... ;**

Transfere a execução de um programa para o ponto determinado pelo **Label**. A sintaxe do comando é: **GOTO {Label};**. Ex:

```
13.4 Label
  primeiro;
begin
   { ... comandos iniciais ... }
```

```
   if x = 2 then
     goto primeiro;
   { ... outros comandos ... }
Primeiro:
   { ... comandos do Primeiro ... }
end;
```

13.5 Case <X> Of... Else... End;

Consiste de uma lista de declarações que satisfaz a condição de um seletor de expressões. Se nenhuma parte da lista satisfizer ao seletor executa os comandos do sub-comando **else**. Para o seletor serão válidos os tipos definidos, tipo Inteiros ou LongInt. A sintaxe do comando é: **CASE {seletor} OF {Expressão 1}: {comando da expressão 1}; {Expressão 2}: {comando da expressão 2}; {Expressão n}: {comando da expressão n} ELSE {comando}; end;**. Ex:

```
begin
   { ... comandos iniciais ... }
   case x of
     1: { ... Bloco para x = 1 ... }
     2, 3: { ... Bloco para x = 2 ou X = 3... }
     4..6: { ... Bloco para  4 <= x <= 6 ... }
   else
     { ... Bloco para  x < 1 ou x > 6 ... };
   end;
end;
```

13.6 Repeat... Until

Repete um determinado bloco de declarações até a condição booleana do subcomando **until** ser satisfeita. A sintaxe do comando é: **REPEAT {comandos}; until {condição};**. Ex:

```
begin
   { ... comandos iniciais ... }
   x := 0;
   repeat
     x := x + 1
   until (x = 2);
end;
```

13.7 For... Do

Incrementa em 1 uma determinada variável inteira, repetindo um bloco de comandos, até que esta atinja o valor final do intervalo, o subcomando **downto** realiza o incremento reverso. A sintaxe do comando é: **FOR {variavel} := {valor inicial} to (downto) {valor final} do {bloco de comandos};**. Ex:

```
begin
   { ... comandos iniciais ... }
   for i := 1 to 10 do          ⇩ Executa o [comandos A] para i = 1,2,3,4,5,6,7,8,9 e 10
     { ... Comandos A ... }
   for s := 10 downto 1 do      ⇩ Executa o [comandos B] para i = 10,9,8,7,6,5,4,3,2 e 1
     { ... Comandos B... }
end;
```

13 .8 While... Do

Repete um bloco de comandos enquanto que determinada condição booleana seja satisfeita. A sintaxe do comando é: **WHILE {condição} DO {bloco de comandos};**. Ex:

```
begin
   { ... comandos iniciais ... }
   while i := 1 do              ⇩ Repete o [Bloco de comandos] enquanto i = 1
     { ... Bloco de comandos ... }
end;
```

14 - BLOCOS DE PROCEDIMENTOS E FUNÇÕES

As procedures e funções são declaradas na seção de tipos de declarações (abaixo do comando type) pertencendo ao objeto, são do tipo public (públicas - executadas por outras unidades) ou private (particulares - restritas a unidade local).

Procedure

procedure {cabeçalho};
var {declaração das variáveis};
{bloco de comandos};
O cabeçalho da procedure é composto pelo nome do procedimento e variáveis que serão recebidas (ou modificadas através da declaração var, ex: procedure teste(var x:string);).

```
procedure soma(a,b: integer);          ⇩ Início enviando as variáveis A e B do tipo inteiro.
var                                    ⇩ Declaração de variáveis locais.
  c: integer;
begin                                  ⇩ Corpo do procedimento.
  c := a + b;
end;
```

Function

function {cabeçalho} : {resultado};
var {declaração das variáveis};
{bloco de comandos};
As funções se diferem dos procedimentos pela obrigatoriedade do retorno de um resultado, podendo este resultado ser retornado pela declação: {nome da função} := valor ou result := valor.
EX:

```
function soma(a,b: integer) : integer; ⇩ Início enviando as variáveis A e B do tipo inteiro.
begin                                  ⇩ Corpo do procedimento.
  soma := a + b;                             ⇩ ou  result := a + b;
end;
```

Assim:
y:=soma(3,5) // y será igual a 8

15 - GERENCIAMENTO DE PROJETOS

Segue uma descrição das mais importantes opções de menu para o gerenciamento de projetos, algumas dessas opções tem um botão correspondente na barra de ferramentas.

15.1 File	
New	Abre um diálogo com novos itens que podem ser adicionados ao projeto
Open	Abrir projetos, pode abrir também Units, Forms e texto no editor de código
Save	Salva o arquivo aberto no editor de código
Save Project As	Salva o projeto com outro nome ou local
Use Unit	Faz com que a Unit atual possa usar outra Unit do projeto
Add to Project	Adiciona uma Unit em disco ao projeto
Remove from Project	Remove uma Unit do projeto
15.2 View	
Project Manager	Mostra o gerenciador de projeto
Project Source	Mostra o código do projeto
Object Inspector	Mostra o Object Inspector
Toggle Form/Unit	Alterna entre o Form e a Unit
Units	Mostra o código fonte de uma Unit ou do Projeto a partir de uma lista
Forms	Seleciona um Form a partir de uma lista
15.3 Project	
Compile	Compila o projeto
Options	Opções do projeto, como ícone do executável, nome da aplicação e opções de compilação
15.4 Run	
Run	Compila e executa o projeto

16 - ARQUIVOS QUE COMPÕEM UMA APLICAÇÃO E ARQUIVOS GERADOS NO DESENVOLVIMENTO

Extensão Arquivo	Definição	Função
.DPR	Arquivo do Projeto	Código fonte em Pascal do arquivo principal do projeto. Lista todos os formulários e units no projeto, e contém código de inicialização da aplicação. Criado quando o projeto é salvo.
.PAS	Código fonte da Unit (Object Pascal)	Um arquivo .PAS é gerado por cada formulário que o projeto contém. Seu projeto pode conter um ou mais arquivos .PAS associados com algum formulário. Contem todas as declarações e procedimentos incluindo eventos de um formulário.
.DFM	Arquivo gráfico do formulário	Arquivo binário que contém as propriedades do desenho de um formulário contido em um projeto. Um .DFM é gerado em companhia de um arquivo .PAS para cada formulário do projeto.
.OPT	Arquivo de opções do projeto	Arquivo texto que contém a situação corrente das opções do projeto. Gerado com o primeiro salvamento e atualizado em subsequentes alterações feitas para as opções do projeto.
.RES	Arquivo de Recursos do Compilador	Arquivo binário que contém o ícone, mensagens da aplicação e outros recursos usados pelo projeto.
.~DP	Arquivo de Backup do Projeto	Gerado quando o projeto é salvo pela segunda vez.
.~PA	Arquivo de Backup da Unit	Se um .PAS é alterado, este arquivo é gerado.
.~DF	Backup do Arquivo gráfico do formulário	Se você abrir um .DFM no editor de código e fizer alguma alteração, este arquivo é gerando quando você salva o arquivo.

17 - ARQUIVOS GERADOS PELA COMPILAÇÃO

Extensão Arquivo	Definição	Função
.EXE	Arquivo compilado executável	Este é um arquivo executável. Este arquivo incorpora todos os arquivos .DCU gerados quando sua aplicação é compilada. O Arquivo .DCU não é necessário em sua aplicação.
.DCU	Código objeto da Unit	A compilação cria um arquivo .DCU para cada .PAS no projeto.

Obs.: Estes arquivos podem ser apagados para economizar espaço em disco.

18 - CÓDIGO FONTE DO ARQUIVO PROJECT (.DPR)

Neste arquivo está escrito o código de criação da aplicação e seus formulários. O arquivo Project tem apenas uma seção.

Esta seção é formada pelo seguinte código:

PROGRAM - Define o Projeto;

USES - Cláusula que inicia uma lista de outras unidades.
Forms = É a unidade do Delphi que define a forma e os componentes do aplicativo
In = A clausula indica ao compilador onde encontrar o arquivo Unit.
Unit1 = A unidade que você criou
{$R *.RES} - Diretiva compiladora que inclui o arquivo de recursos.

Abaixo veja como fica o Project quando você abre um projeto novo:

```
program Project1;

uses
  Forms,
  Unit1 in 'UNIT1.PAS' {Form1};

{$R *.RES}
```

```
begin
  Application.CreateForm(TForm1, Form1);
  Application.Run;
end.
```

19 - CÓDIGO FONTE DO ARQUIVO UNIT (.PAS)

Nesta divisória serão escritos os códigos dos seus respectivos forms (Unit1 = Form1). Aqui serão definidos os códigos de cada procedimento dos componentes que você colocar no form.

Seção Unit

Declara o nome da unit.

Seção Uses

Contém as units acessadas por este arquivo.

Seção Interface

Nesta seção estão as declarações de constantes, tipos de variáveis, funções e procedures gerais da Unit/Form. As declarações desta seção são visíveis por qualquer Unit. Esta seção é formada pelo seguinte código:

INTERFACE - Palavra que inicia a seção;

USES - Cláusula que inicia uma lista de outras unidades compiladas (units) em que se baseia:

SysUtils = utilitários do sistema (strings, data/hora, gerar arquivos)
WinProcs = acesso a GDI, USER e KERNEL do Windows
Wintypes= tipos de dados e valores constantes
Messages=constantes com os números das mensagens do Windows e tipos de dados das Mensagens
Classes=elementos de baixo nível do sistema de componentes
Graphics=elementos gráficos
Controls=elementos de nível médio do sistema de componentes
Forms=componentes de forma e componentes invisíveis de aplicativos
Dialogs=componentes de diálogo comuns

Seção Type

Declara os tipos definidos pelo usuário. Subseções: Private, declarações privativas da Unit. Public declarações publicas da Unit.

Seção Var

Declara as variáveis privadas utilizadas.

19.1 Seção Implementation

Contém os corpos das funções e procedures declaradas nas seções Interface e Type. Nesta seção também estão definidos todos os procedimentos dos componentes que estão incluídos no Form. As declarações desta seção são visíveis apenas por ela mesma. Esta seção é formada pelo seguinte código:

{$R*.DFM} - Diretiva compiladora que inclui toda a interface, propriedades da forma e componentes do arquivo *.DFM

{$S+} - Diretiva compiladora que ativa verificação de pilha.

Seção uses adicional

Serve para declarar Units que ativam esta.

19.2 Inicialization

Nesta seção, que é opcional, pode ser definido um código para proceder as tarefas de inicialização da Unit quando o programa começa. Ela consiste na palavra reservada inicialization seguida por uma ou mais declarações para serem executadas em ordem.

Exemplo:

Abaixo veja como fica a unit quando você abre um projeto novo:

```
unit Unit1;

interface

uses
  SysUtils, WinTypes, WinProcs, Messages, Classes, Graphics, Controls, Forms, Dialogs;

type
  TForm1 = class(TForm)
  private
    { Private declarations }
  public
    { Public declarations }
  end;
```

```
var
  Form1: TForm1;

implementation

{$R *.DFM}
{Uses Adicional}
{Initialization}

end.
```