

Borland Delphi

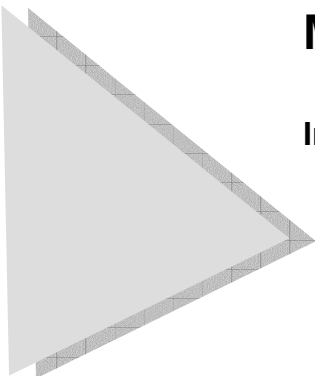
Curso Básico

Módulo 1

Instrutor

Jackson Pires de O. S. Júnior

Jackson_pires@yahoo.com.br



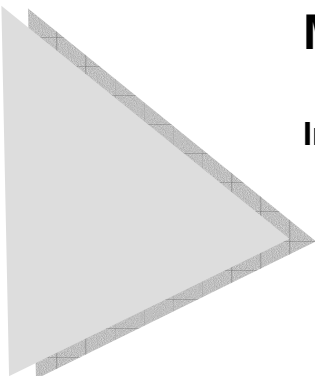


Iniciando

Módulo 1

Instrutor

Jackson Pires de O. S. Júnior
Jackson_pires@yahoo.com.br



1

Introdução

O *Módulo Iniciando* irá demonstrar o caminho certo para o usuário iniciar o desenvolvimento de aplicativos com a ferramenta Delphi. Fique atento aos detalhes e características das ferramentas que iremos apresentar.

O que é o Delphi?

Delphi é um aplicativo para desenvolvimento rápido de aplicações visuais orientadas a objetos o que podemos chamar de ferramenta RAD (Rapid Application Development). Usando o Delphi você poderá criar aplicações altamente eficientes para as plataformas Microsoft Windows 98, NT e 2000 com o mínimo de linhas de código digitadas manualmente. O Delphi também provê soluções cross-platform quando usado conjuntamente com o Borland Kylix, uma ferramenta Borland RAD para Linux. O Delphi dispõe de todas as ferramentas que você precisa para desenvolver, testar e depurar aplicações, incluindo uma grande biblioteca de componentes reutilizáveis, uma suíte de ferramentas para design, exemplos de aplicações e formulários (forms) e ainda assistentes de programação.

Encontrando Informações

Você pode encontrar informações no Delphi das seguintes formas descritas a seguir:

- Help On-line
- Documentação Impressa
- Web Sites e Serviços de Suporte Borland

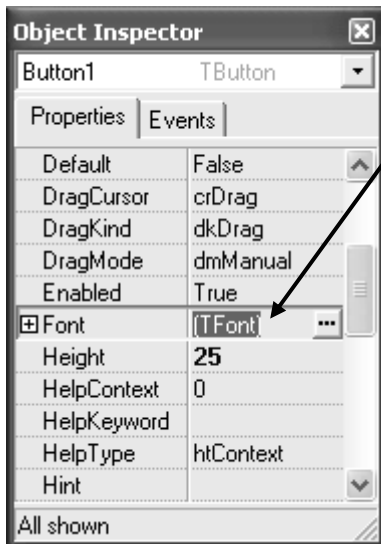
Help On-Line

O Sistema de Help-OnLine possibilita o usuário obter informações detalhadas sobre características, componentes, implementações de linguagem, tarefas de programação, referências sobre a VCL (Visual Component Library) e CLX. Isso inclui todo o material que está disponível no *Developer's Guide* e *Object Pascal Language Guide* que acompanham o pacote Delphi.

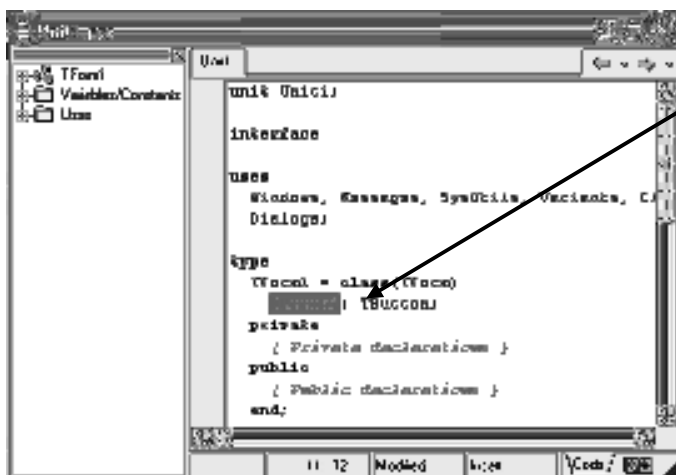
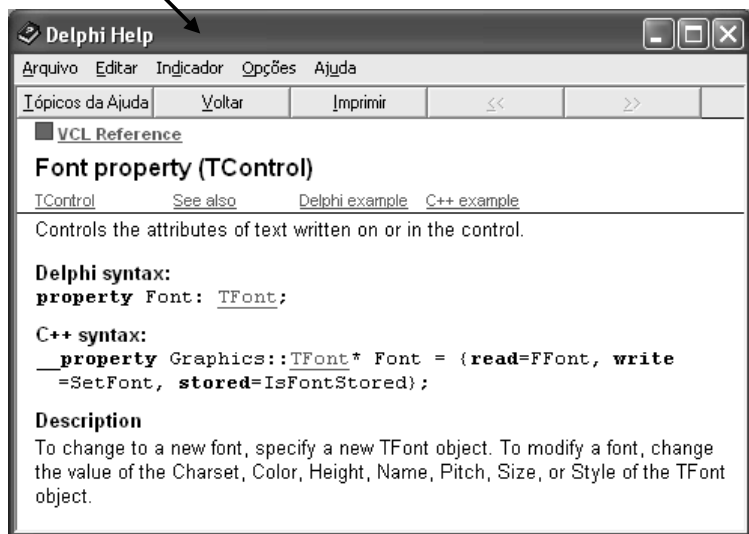
Para ver a tabela de conteúdos, escolha Help | Delphi Help e Help | Delphi Tools, e clique na aba Contents. Para procurar sobre objetos VCL, CLX ou qualquer outro tópico, clique no Índice (Index) ou na aba Procurar (Find).

F1 Help

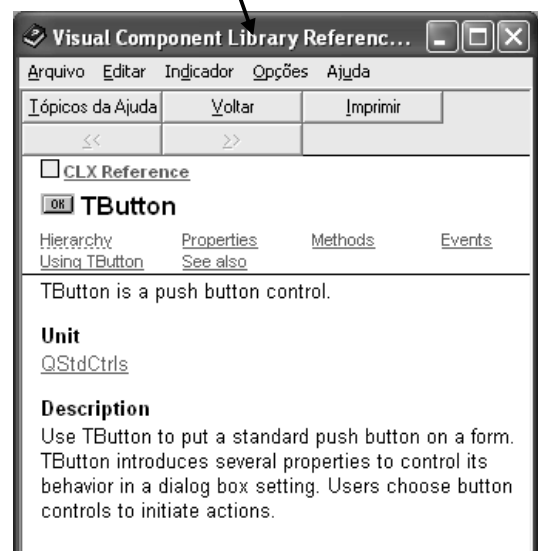
Você pode usar o Help Context-Sensitive na VCL, CLX ou em qualquer parte do ambiente de desenvolvimento, incluindo itens de menu, caixas de diálogo, barra de ferramentas e componentes bastando selecionar o item e pressionar F1.

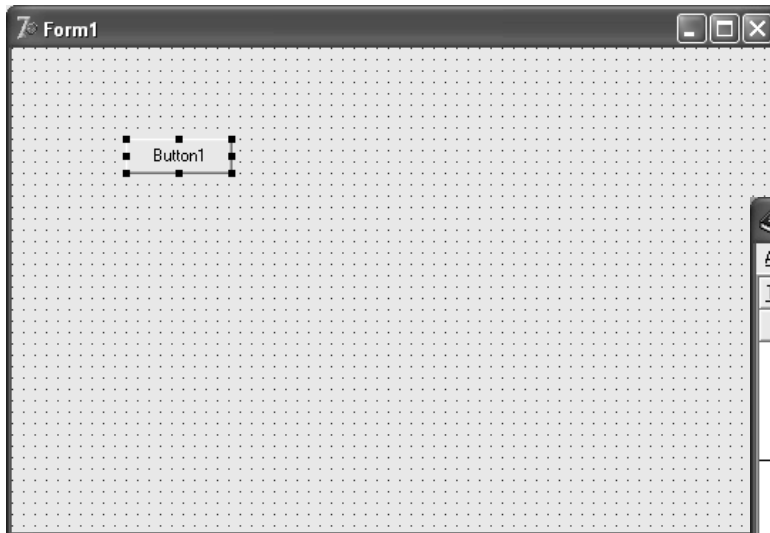


Pressione F1 em qualquer nome de propriedade ou evento no Object Inspector e mostre o Help da VCL.

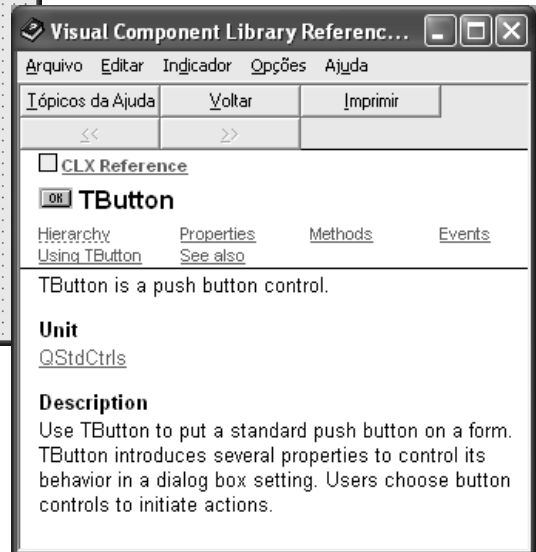


No editor de códigos pressione F1 sobre uma linguagem, VCL ou elemento CLX.





Pressione F1 sobre um Componente em um Form.



Pressionando o botão de ajuda em qualquer menu ou caixa de diálogo também será mostrado a documentação online do contexto-sensitivo.

Mensagens de erro do compilador e linkador aparecerão em uma janela especial abaixo do editor de código. Para pegar a ajuda sobre os erros de compilação, selecione a mensagem na lista e pressione F1.

Documentação Impressa

Este módulo é apenas uma introdução ao delphi, para documentação impressa adicional procure guias de referências em livrarias do ramo ou na Internet.

Web Sites e Serviços de Suporte Borland

A Borland também oferece uma variedade de suportes para encontros de diversos desenvolvedores, para encontrar mais sobre suporte vá até: www.borland.com/devsupport/
Na Internet você também pode procurar sobre Delphi em diversos sites e fóruns, para isto basta procurá-los nos sites de busca. Boa sorte!

2

Um tour pelo desktop

Neste capítulo explanaremos sobre como iniciar o Delphi e faremos um tour sobre as principais partes e ferramentas do desktop, ou Desktop Integrado de Desenvolvimento - IDE (Integrated Desktop Environment).

Iniciando o Delphi

Você pode iniciar o Delphi pelos seguintes caminhos:

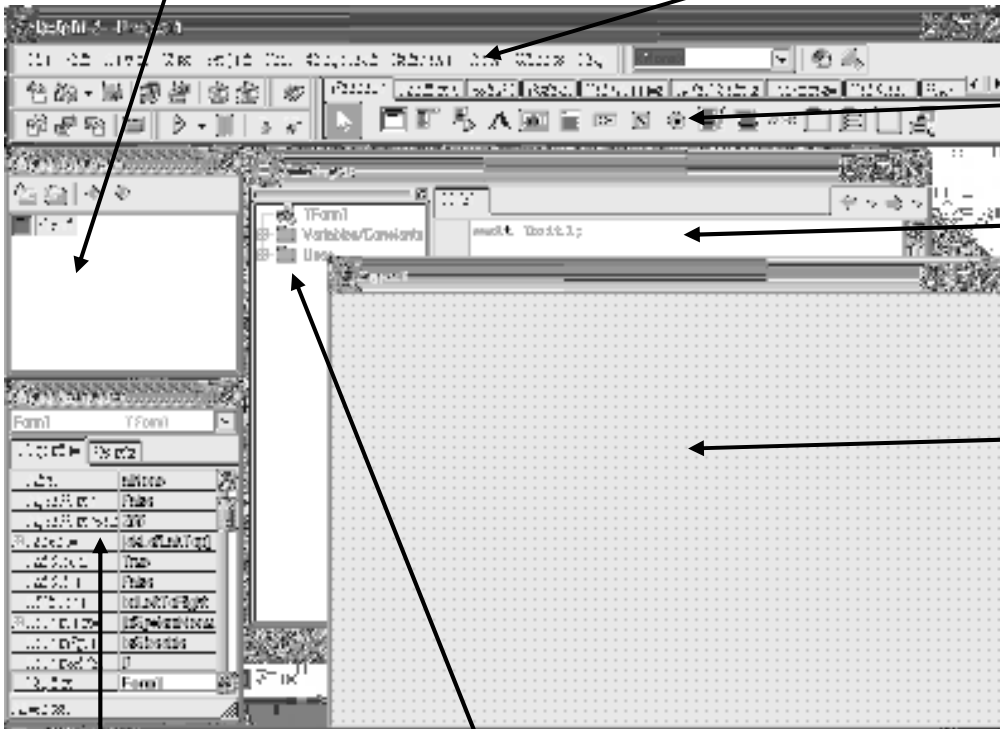
- Duplo clique sobre o ícone do Delphi (Se você tiver criando um atalho);
- Escolha Programas | Boland Delphi X | Delphi X no menu iniciar do seu Windows.
- Escolha Executar no menu iniciar do seu Windows e digite Delphi32 pressionando ENTER.
- Duplo clique em Delphi32.exe na pasta Delphi\Bin.

A IDE

Quando você inicia o Delphi pela primeira vez, você verá que algumas das maiores ferramentas estão na IDE. A IDE do Delphi inclui menus, barra de ferramentas, paleta de componentes, inspetor de objetos (Object Inspector), objeto visualizador de árvore (Object TreeView), editor de código, explorador de código (Code Explorer), gerenciador de projeto (Project Manager), e muitas outras ferramentas. Uma característica particular dos componentes é que existem componentes de terceiros que poderão ser usados, bastando apenas que você compre-os e instale-os corretamente.

O Object TreeView mostra a hierarquia visual de seus componentes parentescos e relacionamentos.

Os Menus e Barras de Ferramentas acessam o anfitrião de características e ferramentas para ajudá-lo a desenvolver suas aplicações.



A paleta de componentes contém os componentes reutilizáveis para você adicionar aos seus projetos.

O editor de códigos mostra o código fonte para você ver e editar.

O Form Designer contém um form em branco que é iniciado juntamente com o Delphi, nele você fará o design de suas aplicações. Uma aplicação poderá conter vários forms.

O Object Inspector é usado para alterar propriedades e eventos de objetos selecionados.

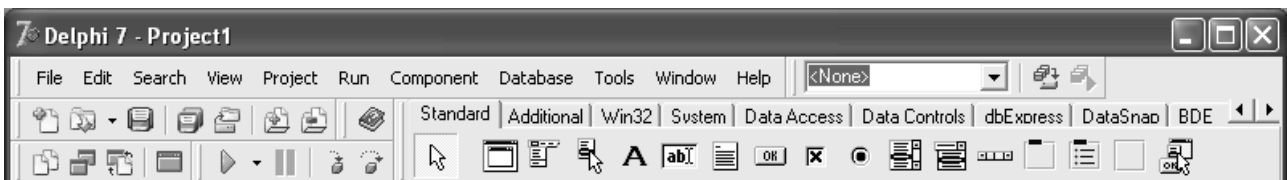
O Code Explorer suas classes variáveis e rotinas da sua unit para que você possa navegar facilmente em seu projeto.

O modelo de desenvolvimento do Delphi é baseado em uma ferramenta *Two-Way* (dois caminhos). Desta maneira é possível mover-se entre o design visual e a parte baseada em códigos de texto. Para um exemplo poderíamos adicionar vários componentes em nosso projeto visual (Form) e logo após nos movermos para a parte baseada em códigos de texto (Unit), poderemos observar que automaticamente o Delphi se encarregou de digitar as novas linhas referente aos novos objetos adicionados ao form. Você também poderá acessar e editar manualmente qualquer código gerado pelo Delphi sem perder o acesso à programação visual.

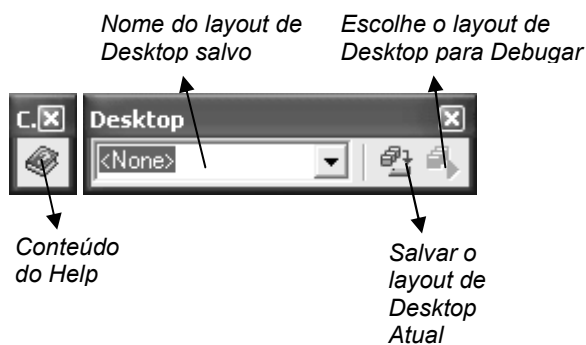
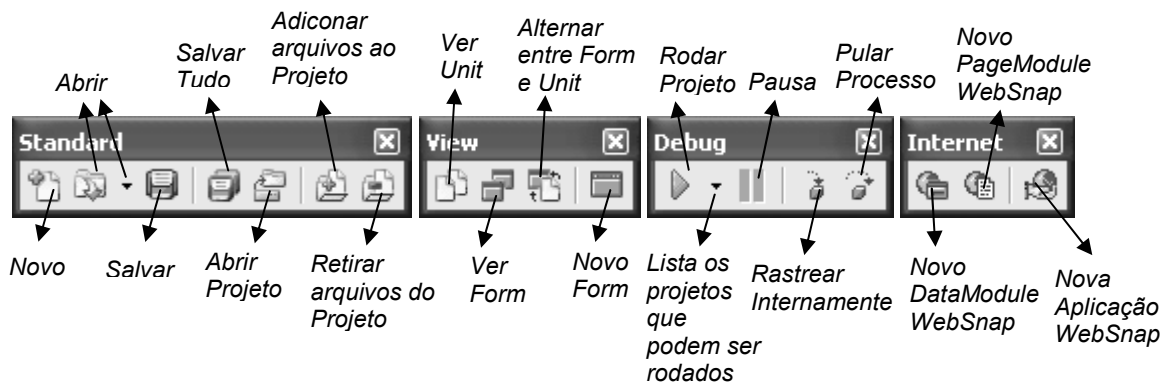
Na IDE, todas as ferramentas são de fácil alcance e programação. Você pode desenhar interfaces gráficas, navegar por bibliotecas de classes, escrever códigos, testar, compilar, debugar e gerenciar projetos, sempre partindo da IDE.

Os Menus e Barras de Ferramentas

A janela principal que ocupa o topo da tela contém o menu principal, barra de ferramentas e paleta de componentes.



A barra de ferramentas do Delphi provê um acesso instantâneo às operações e comandos usados com mais freqüência .
 Lambramos que todos os atalhos disponíveis na barra de ferramentas estão disponíveis nos menus de acesso.

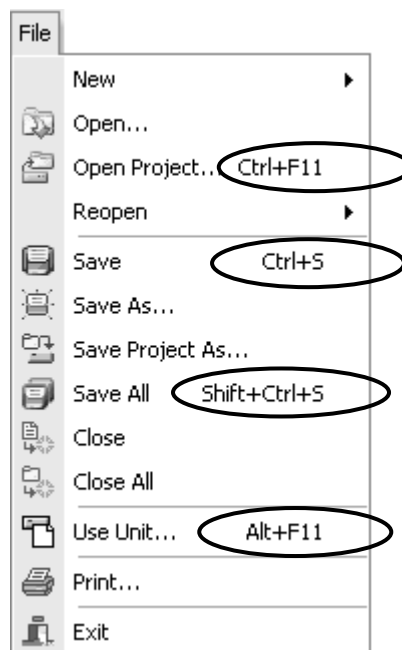


Para saber para que serve cada botão, basta apontar com o mouse para o mesmo e esperar alguns segundos para que a dica apareça.

Você pode usar o botão direito do mouse para mostrar/ocultar qualquer barra de ferramentas.

Para você mostrar/ocultar as barras de ferramentas você também pode escolher View | Toolbars, daí é só selecionar ou não as barras de ferramentas desejadas.

Muitas operações podem ser feitas através de atalhos no teclado bem como nos botões da barra de ferramentas. Quando um atalho está disponível no teclado ele sempre é mostrado no menu ao lado da opção/comando a ser executado. Veja o exemplo na figura abaixo:



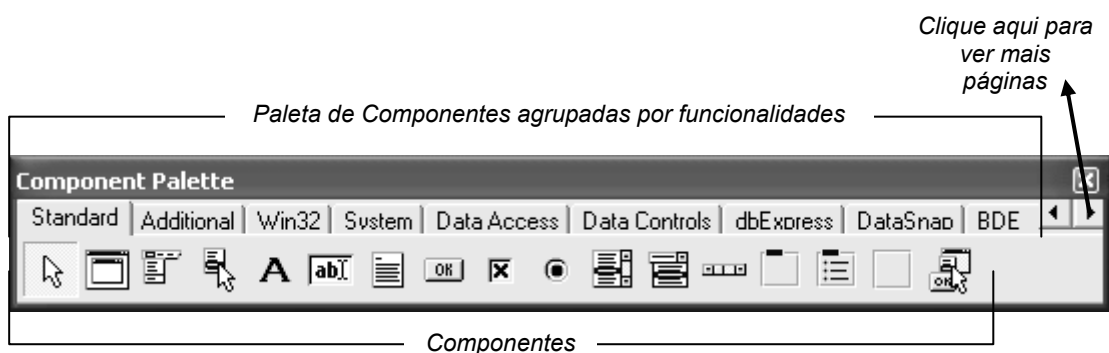
Você pode clicar com o botão direito em várias ferramentas e ícones para mostrar o menu de comandos apropriados dos objetos que você estiver trabalhando. Isso é chamado de *Menus Contextos*.

As barras de ferramentas são personalizáveis. Você pode adicionar comandos, se desejar pode mover para locais diferentes. Para maiores informações procure o help.

A Paleta de Componentes, o Form Designer, Object Inspector e o Object TreeView.

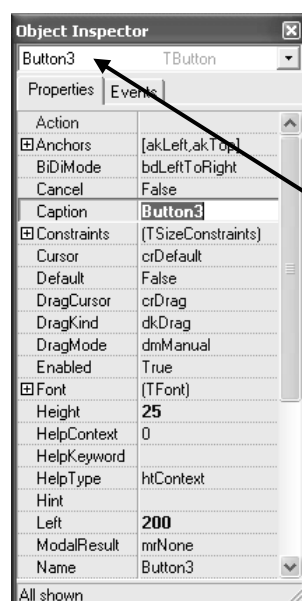
A Paleta de Componentes, o Form Designer, o Object Inspector e o Object TreeView trabalham juntos para ajudá-lo a construir a interface de suas aplicações.

A Paleta de Componentes inclui páginas tabuladas por grupos de ícones que representam componentes VCL e CLX visuais e não visuais. As páginas se dividem em grupos de componentes que tem as mesmas funcionalidades. Um exemplo, a paleta Standard, Additional e Win32 incluem controles do Windows como um edit box, e um menu dropdown. Já a paleta Dialogs inclui chamada a caixas de diálogo para fazer operações com arquivos, abrindo-os e salvando.

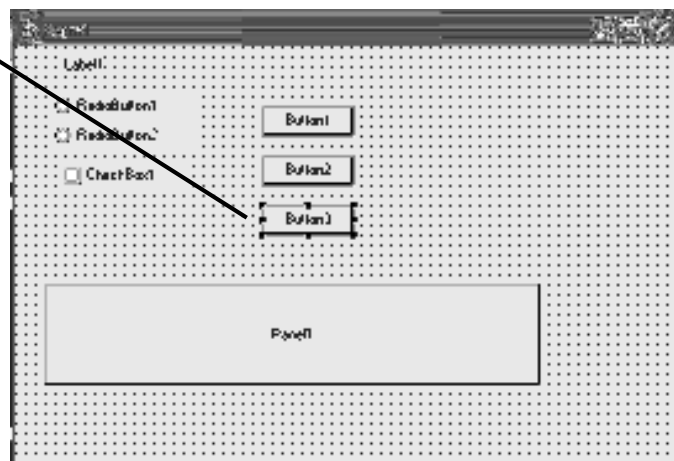


Cada componente tem suas especificações como atributos/propriedades, eventos e métodos. Estas especificações serão controladas pela sua aplicação.

Depois que você colocar os componentes no Form ou *Form Designer*, você organizará os componentes da maneira que seus usuários irão visualizar a interface. Além de localizar um componente no seu Form o Object Inspector pode setar propriedades criar chamadas à eventos e filtrar propriedades e eventos, fazendo assim uma conexão entre a aparência visual do seu programa e os códigos fontes que fazem o mesmo funcionar.



Depois de colocar os componentes em um Form o Object Inspector altera dinamicamente as propriedades mostradas. baseando-se no componente selecionado.

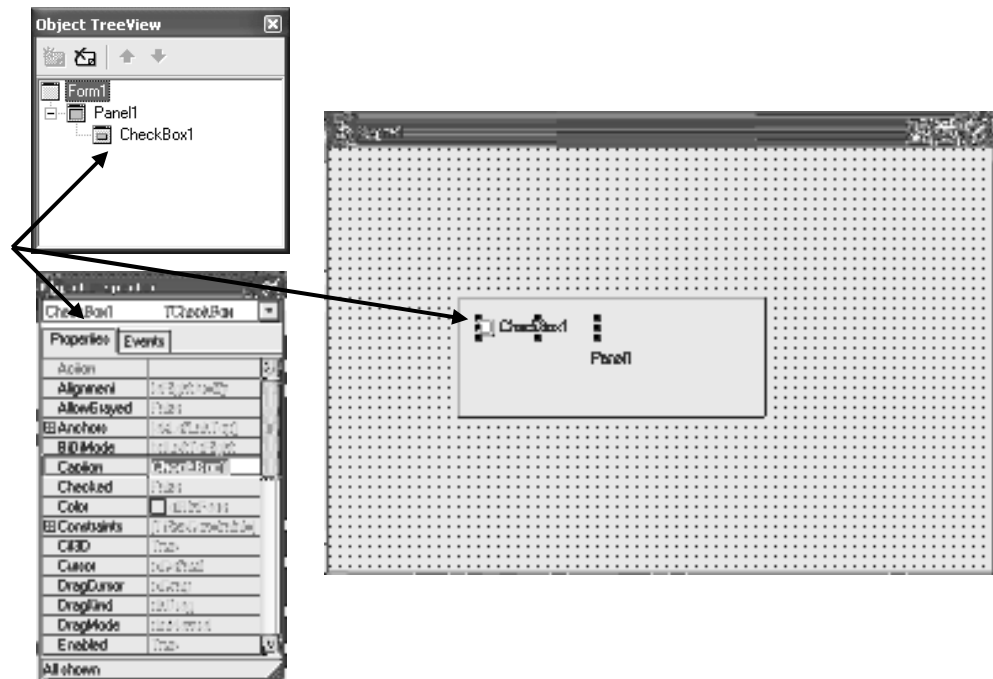


O Object TreeView

O Object TreeView mostra a simbologia e as relações de parentesco dos componentes em um diagrama hierárquico de árvore. O diagrama de árvore é sincronizado com o Object Inspector e o Form Designer, assim quando mudamos o foco no Object TreeView, ambos (o Object Inspector e o Form Designer) mudam o foco.

Você pode usar o Object TreeView para mudar a ordem e a relação entre os componentes. Para um exemplo, podemos adicionar um componente *CheckBox* e um componente *Panel* no Form. Será mostrados os dois símbolos que representam os dois componentes no Object TreeView. Note que os dois componentes estão na raiz do diagrama, então basta clicar e arrastar o *CheckBox* para cima do *Panel*. Podemos verificar que agora o *CheckBox* é um filho do *Panel*.

O Object TreeView, o Object Inspector e o FormDesigner Trabalham juntos. Quando Você clica em um objeto no seu Form, isso muda o foco do Object TreeView e do Object Inspector e vice-versa.



Se uma propriedade necessária não estiver completada, o Object TreeView mostrará um símbolo de uma interrogação vermelha pra que o usuário perceba o problema.

Você também pode dar um duplo clique em qualquer objeto do diagrama de árvore para abrir o editor de códigos e localizar onde você pode escrever uma chamada a um evento.

Se o Object TreeView Não estiver sendo mostrado selecione View | Object TreeView.

3

Object Pascal – Conceitos Básicos

Neste capítulo iremos expor os conceitos básicos do Object Pascal que é a linguagem base das ferramentas Borland. Leia atentamente o que iremos apresentar.

Descrição Básica da Linguagem

O Object Pascal é uma linguagem de alto nível que suporta estruturação e design de orientação a objetos. Entre os benefícios da linguagem podemos citar a fácil interpretação dos códigos fonte, compilação rápida e o uso de múltiplos arquivos (Unit) para programação modular.

Uma característica especial do Object Pascal é o suporte a componentes e frameworks Borland e ambiente RAD.

Organização do Programa

Os programas são usualmente divididos dentro de módulos de códigos-fonte chamados de Units. Cada programa é iniciado com um cabeçalho onde é especificado o nome do programa. O cabeçalho é seguido de uma cláusula **Uses** e então um bloco de declarações. A cláusula **Uses** lista as units que serão linkadas dentro do programa. Estas units podem ser compartilhadas por diferentes programas, bastando estarem na cláusula **Uses** das mesmas.

A cláusula **Uses** uma compilação com informações sobre as dependências entre os módulos. Por causa destas informações que são gravadas dentro destes módulos, os programas desenvolvidos em Object Pascal não precisam de arquivos externos, cabeçalho de arquivos ou diretivas "INCLUDE".

Códigos-Fonte Pascal

Ao compilar os códigos-fonte pascal são encontrados três arquivos, são eles:

- Arquivos de códigos-fonte **Unit** (com extensão de arquivo .pas)
- Arquivos de **Projeto** (com extensão de arquivo .dpr)
- Arquivos de **Pacotes** (com extensão de arquivo .dpk)

Arquivos de códigos-fonte Unit contém a maioria dos códigos de uma aplicação. Cada aplicação tem um simples arquivo de projeto e alguns arquivos de unit. O arquivo de projeto (que corresponde ao arquivo "inicial" de uma aplicação desenvolvida em Pascal) organiza os

arquivos de unit dentro de uma aplicação. As ferramentas Borland se encarregam de manter os arquivos de projeto em cada aplicação.

Se você estiver compilando um programa em comando de linha, você poderá colocar todos os códigos-fonte dentro de arquivos **.pas*. Mas se você estiver usando a IDE do Delphi para desenvolver os seus aplicativos, você terá que ter os arquivos de projeto **.dpr*.

Arquivos fontes de pacotes (packages) são similares a arquivos de projeto, mas são usados para construir bibliotecas especiais de linkagem dinâmica chamadas *packages*.

Palavras Chaves

Palavras-chave ou palavras reservadas são aquelas usadas pela sintaxe da linguagem com significado específico e, portanto, não podem ser usadas com outra finalidade. As palavras-chave aparecem em negrito no *Code Editor*.

Em nosso módulo de código *forma*, encontramos as seguintes palavras-chave, exibidas em negrito :

- **unit** : esta palavra-chave define o nome da unidade de código. Este nome será inserido na cláusula *uses* de outras unidades que precisem fazer referência a esta unidade.

- **interface** : define o início de um trecho de código que termina antes da palavra-chave *implementation*. Neste trecho se insere o código que poderá ser acessado por outras unidades. É aqui que se declaram as constantes, as variáveis, os tipos de dados, funções e procedimentos a serem utilizados por outras unidades de código.

- **uses** : esta palavra especifica as unidades de código que serão acessadas por esta unidade.

- **type** : com esta palavra-chave o Delphi define o início do trecho de código em que são definidos os tipos de variáveis e de classes criados pelo programa. O aluno pode observar que o Delphi já utilizou esta parte para declarar a classe *TFormaPrincipal*.

- **private** : define os elementos de uma classe que não poderão ser acessados de fora da classe;

- **public** : define os elementos de uma classe que poderão ser acessados de fora da classe.

- **var** : define o início do trecho de código em que são declaradas as variáveis e objetos.

- **implementation** : define o início do trecho de código em que são implementadas as funções e procedimentos declaradas no trecho iniciado pela palavra chave **interface**.

- **end** : palavra-chave usada para encerrar um bloco de código. São blocos de código :

- um bloco de comandos iniciado pela palavra chave **begin**, caso em que **end** deve ser seguida de um ponto-e-vírgula (;).

- a definição de uma **unit**; neste caso a palavra **end** deve ser seguida de um ponto (.) e este será o final do arquivo.

Exceção : antes da cláusula **else** em instruções condicionais **if-then-else** compostas, não se insere ponto (.) ou ponto-e-vírgula (;), utilizando-se apenas a palavra **end**.

Variáveis

Nossos dados são armazenados na memória do computador. Para que nós não tenhamos que nos referir a estes dados de forma direta, através de um endereço numérico difícil de memorizar, o compilador nos permite utilizar variáveis com esta finalidade. Escolhendo nomes sugestivos (mnemônicos) para nossas variáveis (tais como *nome*, *funcao*, *idade*, *salario*) facilitamos bastante a compreensão de nosso código.

Para que o Delphi possa usar nossas variáveis, devemos primeiro declará-las, isto é, informar o nome e o tipo desejados. Por exemplo : o comando a seguir declara *idade* como sendo uma variável do tipo inteiro (*integer*) : `idade : integer;`

As variáveis inteiras podem assumir valores entre -32768 e +32767. Elas ocupam 2 bytes na memória. Assim sendo, a declaração acima faz com que o Delphi reserve 2 bytes para a nossa variável *idade*. Note que a declaração do tipo de uma variável, em princípio não lhe atribui valores. Um erro comum em programação é tentarmos *ler* valores de variáveis não inicializadas, ou às quais ainda não se atribuiu valores...

Damos a seguir uma lista dos tipos de variáveis mais comuns do Object Pascal com suas faixas de valores e o espaço ocupado em memória:

BOOLEAN: Tipo lógico que pode assumir somente os valores TRUE ou FALSE e ocupa 1 byte de memória.

BYTE: Tipo numérico inteiro, pode assumir valores numa faixa de 0 a 255, ocupa 1 byte.

CHAR: Tipo alfa-numérico, pode armazenar um caractere ASCII, ocupa 1 byte.

COMP: Tipo numérico real, pode assumir valores na faixa de $-9.2.10^{-18}$ a $9.2.10^{+18}$, ocupa 8 bytes, pode ter entre 19 e 20 algarismos significativos.

EXTENDED: Tipo numérico real, pode assumir valores na faixa de $-3,4.10^{-4932}$ a $+1,1.10^{+4932}$, ocupa 10 bytes de memória e tem entre 19 e 20 algarismos significativos.

INTEGER: Tipo numérico inteiro, pode assumir valores numa faixa de -32768 a +32767, ocupa 2 byte de memória.

LONGINT: Tipo numérico inteiro, pode assumir valores numa faixa de -2147483648 a +2147483647, ocupa 4 bytes de memória.

REAL: Tipo numérico real, pode assumir valores na faixa de $-2,9.10^{-39}$ a $+1,7.10^{+38}$, ocupa 6 bytes de memória e tem entre 11 e 12 algarismos significativos.

SHORTINT: Tipo numérico inteiro, pode assumir valores numa faixa de -128 a +127, ocupa 1byte de memória.

SINGLE: Tipo numérico real, pode assumir valores numa faixa de $-1,5.10^{-45}$ a $+3,4.10^{+38}$, ocupa 4 bytes de memória, e tem de 7 a 8 algarismos significativos.

WORD: Tipo numérico inteiro, pode assumir valores numa faixa de 0 a 65535, ocupa 2bytes de memória.

STRING: Tipo alfanumérico, possuindo como conteúdo uma cadeia de caracteres. O número de bytes ocupados na memória varia de 2 a 256, dependendo da quantidade máxima de caracteres definidos para a string. O primeiro byte contém a quantidade rela de caracteres da cadeia.

Os nomes de variáveis devem começar com uma letra ou o caractere sublinhado (_) seguido por uma sequência de letras, dígitos ou caractere sublinhado (_) e não podem conter espaço em branco nem quaisquer tipos de acentos. Os nomes de variáveis podem ter qualquer tamanho mas somente os 63 primeiros caracteres serão considerados.

Exemplos : Para definir uma variável *Nome* do tipo *string* e uma variável *Salario* do tipo *double*, podemos inserir as seguintes linhas de código na cláusula *var* da unidade de código correspondente.

```
Nome : string;  
Salario : double;
```

Pode-se declarar mais de uma variável do mesmo tipo na mesma linha, separando-as por vírgula.

```
nome, funcao, endereco : string;
```

Arrays (Vetores)

Arrays são conjuntos de variáveis com o mesmo nome e diferenciadas entre si por um índice. Eles são úteis para manipularmos grandes quantidades de dados de um mesmo tipo pois evitam a declaração de diversas variáveis.

Considere o caso de um programa de Folha de Pagamento que precise armazenar os seguintes dados referentes a 100 funcionários : nome, funcao, salário, etc... Seríamos obrigados a declarar 100 variáveis nome, 100 variáveis funcao, etc... O array nos permite declarar uma única variável com um índice para apontar para as diferentes ocorrências.

Declara-se um array da seguinte forma :

```
nome_da_variável : array[i1..i2] of tipo_de_variável; onde i1 e i2 representam os valores  
mínimo e máximo, respectivamente, do índice.
```

O Object Pascal permite que i1 e i2 possuam qualquer valor desde que i1 seja menor ou igual a i2. Assim, poderíamos declarar um array de 100 variáveis inteira *idade* de várias formas diferentes :

```
idade : array [1..100] of integer; ou  
idade : array [-100..-1] of integer; ou  
idade : array [0..99] of integer, etc...
```

Pode-se definir arrays multidimensionais (com vários índices) como, por exemplo :
espaco3d:array[1..10,-5..20,0..30] of double; que pode armazenar 10x26x31=8060 variáveis do tipo double.

Um dos casos mais comuns é a matriz com m linhas e n colunas : matriz : array[1..m,1..n] of qqr_tipo.

Os elementos dos arrays podem ser quaisquer tipos de variáveis ou objetos.

Records (Registros)

O Object Pascal permite definir tipos compostos de variáveis denominados registros. Define-se da seguinte forma :

```
nome_do_tipo : Record  
    variavel1 : primeiro_tipo;  
    variavel2 : segundo_tipo;  
    .....  
    variaveln : n-ésimo-tipo;  
end;
```

variavel1,variavel2.. variaveln são chamadas de campos do registro.

Declaramos uma variável deste tipo da mesma forma que procedemos para declarar variáveis de qualquer tipo pré-definido

```
variavel : nome_do_tipo;
```

Usamos a notação de ponto para acessar um campo de uma variável composta :

```
nome_da_variavel.nome_do_campo;
```

Exemplo :

```
funcionario = Record
    nome : string;
    funcao : string;
    salario : double;
end;
```

Assim, no exemplo citado anteriormente, ao invés de declararmos um array nome de 100 elementos, um array funcao de 100 elementos, um array salario de 100 elementos, podemos declarar uma única variável chamada empregado, por exemplo, como sendo um array de 100 elementos do tipo funcionário.

```
empregado : array[1..100] of funcionario;
```

Para obter os dados do décimo funcionário, basta fazer :

```
empregado[10].nome;
empregado[10].funcao;
empregado[10].salario;
```

Classes e Objetos

Definição de classe :

```
nome_da_classe_derivada = class(nome_da_classe_base)
    private
    {propriedades, campos e métodos privados}
    public
    {propriedades, campos e métodos públicos}
end;
```

A definição de uma classe é bastante similar a de um registro. As classes, diferentemente dos registros podem conter funções ou procedimentos, chamados métodos, além de variáveis. Quando uma classe é derivada de outra (chamada classe base, classe pai ou classe mãe), ela herda os campos, propriedades e métodos da classe base.

O Delphi possui uma classe chamada TObject da qual todas as demais classes se derivam, ou seja, todas as classes são derivadas de TObject ou de classes derivadas de TObject. Quando você deriva uma classe de TObject, não é preciso declarar explicitamente a classe base pois o Delphi assume TObject como default.

Um objeto é uma instância de uma classe, isto é, depois de definirmos uma classe podemos declarar e utilizar objetos desta classe, da mesma forma que podemos declarar e usar uma variável composta (um registro, por exemplo) depois que o tivermos definido. De forma análoga, também, nos referimos aos elementos de um objeto utilizando a notação de ponto : nome_do_objeto.nome_do_elemento;

Componentes, Controles e Propriedades

O Delphi já possui muitas classes predefinidas para utilização no ambiente Windows. Algumas das classes do Delphi, chamadas genericamente de *componentes* podem ser acessadas através da paleta de componentes conforme já mostramos anteriormente.

Outra classe bastante utilizada é a classe Tform que define um formulário, mas que não está disponível na paleta de componentes. Mas, se olharmos na listagem do exercício Ex01 verificamos que o Delphi define automaticamente uma classe derivada de Tform (TFormaPrincipal no nosso exemplo) e declara um objeto desta classe (FormaPrincipal no nosso exemplo).

As propriedades são um tipo especial de campo em um objeto, cujo valor pode, em alguns casos, ser alterado usando-se o Object Inspector, como fizemos, por exemplo, com a propriedade Caption de FormaPrincipal no exercício Ex01. Outras propriedades só podem ser alteradas por programa (run-time). O sistema de Help do Delphi fornece, para cada classe, uma listagem completa de suas propriedades e seus métodos.

Funções

Uma função se define em Pascal de forma semelhante a definição matemática. Ela recebe valores como parâmetros e retorna um outro valor como resultado. A definição de função obedece a seguinte sintaxe :

```
function nome_da_função(parâmetro_1:tipo_1,,,,,parâmetro_n:tipo_n) : tipo de retorno
var
{declaração de variáveis locais á função}
begin
{corpo da função}
end;
```

O trecho de código a seguir define a função PRODXY, que retorna como resultado o produto de dois números reais X e Y:

```
function PRODXY(X,Y:Real):Real
begin
PRODXY := X*Y;
end;
```

Observe que o sinal de atribuição em Object Pascal é := e, não o sinal de igual simplesmente.

Em Object Pascal podem-se usar os seguintes operadores aritméticos :

```
+ : Soma;
- : Subtração;
* : Multiplicação;
/ : Divisão;
div : Divisão inteira;
mod : Resto da divisão inteira.
```

O Object Pascal tem várias funções predefinidas, parte das quais listamos a seguir :

```
Abs(x)   Retorna o valor absoluto de x.
ArcTan(x) Retorna o valor do arco tangente de x (em radianos).
Cos(x)   Retorna o cosseno de x (x em radianos).
Dec(x)   Decrementa (subtrai 1) uma variável inteira x.
Exp(x)   Retorna o valor de e elevado a x , onde e é a base dos logaritmos neperianos.
```

Frac(x)	Retorna a parte fracionária do real x.
Inc(x)	Incrementa (soma 1) uma variavel inteira x.
Int(x)	Retorna a parte inteira do real x.
Ln(x)	Retorna o logaritmo neperiano de x.
ODD(x)	Retorna True se x for impar.
Sqr(x)	Retorna o quadrado de x.
Sqrt(x)	Retorna a raiz quadrada de x.

Procedimentos

Um procedimento é semelhante a uma função, mas não retorna um valor. A chamada a um procedimento não pode ser colocada do lado direito de um comando de atribuição.

A definição de um procedimento obedece a seguinte sintaxe :

```
procedure nome_do_procedimento(parâmetro_1:tipo_1,...,parâmetro_n : tipo_n)
var
{declaração de variáveis locais ao procedimento}
begin
{corpo do procedimento}
end;
```

Passagem de Parâmetros

Podemos passar parâmetros para uma função ou um procedimento *por valor* e *por referência*. Quando se passa um parâmetro *por valor* estamos realmente passando um valor ou a cópia de um valor armazenado numa variável. Quando a passagem se faz por referência, estamos passando o endereço da variável na memória e não o seu valor. Quando se altera, dentro da função, o valor de uma variável passada *por referência* esta alteração surte efeito em todo o programa (fora da função).

Para passarmos um parâmetro *por referência* devemos precedê-lo da palavra reservada **var**.

Métodos e Eventos

Um método é um tipo especial de procedimento ou função, definido dentro de uma classe. Por ser definido dentro da classe, ele pode acessar diretamente os campos dela sem a necessidade de passar estes campos como parâmetros.

Para executar um método de uma determinada classe basta usar a mesma notação de ponto usada para acessar um campo.

Um evento é muito semelhante a um método pois ambos são definidos internamente a uma classe. A diferença é que os eventos são executados em resposta a alguma ação do usuário ou em resposta a uma mensagem enviada pelo Windows. Cite-se como exemplo o evento OnClick de um formulário, um procedimento que é executado toda vez que o usuário dá um clique com o mouse sobre o formulário. Isto permite que o nosso programa execute alguma ação quando o usuário clica com o mouse sobre o controle.

Estruturas de Controle em Object Pascal

Nos limitaremos a apresentar uma breve descrição da sintaxe dessas estruturas; os próximos exemplos utilizam-nas e servirão de exemplos de aplicação.

Estrutura Condicional (If-Then-Else)

Sintaxe :

```
if (condição)
then
  begin
    {Bloco de comandos executados se a função for verdadeira}
  end
else
  begin
    {Bloco de comandos executados se a função for falsa}
  end;
```

Caso você não queira executar qualquer comando se a condição for falsa, suprima toda a cláusula **else** :

```
if(condição)
then
  begin
    {Bloco de comandos executados se a função for verdadeira}
  end;
```

Note que o **end** que precede a cláusula **else** não é seguido de ;.

Note que as cláusulas **begin** e **end** só são necessárias para blocos com mais de uma linha.

Estrutura Condicional (Case-Of)

Sintaxe :

```
case expressão of
  constante_1_1,...constante_1_n : bloco_de_comando_1;
  constante_2_1,...constante_2_n : bloco_de_comando_2;
  ...
  constante_n_1,...constante_n_n : bloco_de_comando_n;
else
  bloco_de_comando;
end;
```

O comando **case** é um substituto mais elegante e mais legível para **if-then-else** múltiplos. A expressão (ou seletor) deverá ser de tipo com o tamanho máximo de 2 bytes (**Byte**, **Char**, **Word** ou **Integer**).

Segue-se um exemplo :

```
case Ch of
  'A'..'Z', 'a'..'z' : WriteLn('Letra');
  '0'..'9' : WriteLn('Digito');
  '+', '-', '*', '/' : WriteLn('Operador');
else
  WriteLn('Caracter Especial');
end;
```

Estrutura De Repetição (For-To, For-Downto)

Sintaxe :

```
for contador:=valor_inicial to valor_final do
  begin
    {bloco_de_comandos}
  end;
```

onde :

contador é uma variável inteira;

valor_inicial é o valor inicial do contador, que deve ser um número inteiro;

valor_final é o valor final (para o qual o laço se encerrará) a ser assumido pelo contador, deve ser inteiro;

Se se desejar que o contador assuma valores decrescentes deve-se usar a seguinte sintaxe :

```
for contador:=valor_inicial downto valor_final do
  begin
    {bloco_de_comandos}
  end;
```

Note que as cláusulas **begin** e **end** só são necessárias para blocos com mais de uma linha.

Estrutura De Repetição (While-Do)

Sintaxe:

```
while expressão_booleana do
  begin
    bloco_de_comando;
  end;
```

Note que o bloco_de_comando é executado enquanto a expressão_booleana for verdadeira.

Note que como a expressão_booleana é avaliada antes da execução do bloco_de_comando, se a expressão_booleana for falsa o bloco_de_comando não será executado nenhuma vez.

Note que as cláusulas **begin** e **end** só são necessárias para blocos com mais de uma linha.

Estrutura (Repeat-Untill)

Sintaxe :

```
repeat bloco_de_comando until expressão_booleana;
```

Note que o bloco de comando é executado em sequência, enquanto a expressão_booleanda for verdadeira.

Note que o bloco de comando é executado pelo menos uma vez, já que a expressão_booleana é avaliada depois dele.

Note que bloco_de_comando pode exigir as cláusula **begin** e **end** se contiver mais de uma linha.

Estrutura (With)

Sintaxe

```
with identificador_de_registro(objeto) do bloco_de_comando
```

Note que **with** é uma forma abreviada de referenciar os campos de um registro ou os campos e métodos de um objeto. Dentro do bloco_de_comando (com **with**) uma variável ou método fica completamente identificada usando-se apenas seu identificador de campo.

Estrutura Try-Except-End

Sintaxe :

```
try  
sequência_de_comandos  
except  
bloco_a_ser_executado_no_caso_de_erros  
end;
```

Esta estrutura nos permite tratar de forma conveniente os erros ocorridos no programa. Sem ela, ocorrendo um erro, o Delphi dispara o seu mecanismo de tratamento de exceções, normalmente emitindo uma mensagem de erro em inglês e não nos dando chance de recuperação. Com esta estrutura podemos detectar o erro, fornecer uma mensagem inteligível e permitir o retorno à rotina onde ocorreu o erro e permitir, por exemplo, a possibilidade de digitação.