



# Índice Analítico

<b>Introdução</b>	<b>9</b>
<b>Agradecimentos</b>	<b>9</b>
<b>Objetivos</b>	<b>9</b>
<b>Convenções</b>	<b>9</b>
<b>Parte I - Programação Visual e Linguagem</b>	<b>10</b>
<b>O que é o Delphi ?</b>	<b>10</b>
<b>Terminologia do Delphi</b>	<b>11</b>
<b>1 Aplicativos e Projetos</b>	<b>12</b>
<b>1.1 Criando um Projeto</b>	<b>12</b>
<b>1.2 Abrindo um Projeto</b>	<b>12</b>
<b>1.3 Salvando um Projeto</b>	<b>12</b>
<b>1.4 Modificando um Projeto</b>	<b>12</b>
<b>1.5 Rodando Aplicativos</b>	<b>13</b>
1.5.1 Compilando um Projeto	13
1.5.2 Depurando um Projeto	13
<b>2 Formulários</b>	<b>14</b>
<b>2.1 Criando um Formulário</b>	<b>14</b>
<b>2.2 Adicionando um Formulário ao Projeto</b>	<b>14</b>
<b>2.3 Removendo um Formulário do Projeto</b>	<b>14</b>
<b>2.4 Manipulando Formulários</b>	<b>14</b>
<b>3 Units</b>	<b>15</b>
<b>3.1 Criando uma Unit</b>	<b>15</b>
<b>3.2 Adicionando uma Unit ao Projeto</b>	<b>15</b>
<b>3.3 Removendo uma Unit do Projeto</b>	<b>15</b>
<b>3.4 Manipulando Units</b>	<b>16</b>
<b>4 Componentes</b>	<b>17</b>
<b>4.1 Adicionando Componentes a um Formulário</b>	<b>17</b>
<b>4.2 Removendo Componentes de um Formulário</b>	<b>17</b>
<b>4.3 Manipulando Componentes de um Formulário</b>	<b>17</b>
<b>4.4 Manipulando Propriedades e Eventos - Object Inspector</b>	<b>19</b>
<b>5 O Repositório de Objetos e os Experts</b>	<b>20</b>
<b>5.1 Personalizando o Repositório de Objetos</b>	<b>21</b>
5.1.1 Adicionando Modelos ao Repositório	21
5.1.2 Opções do Repositório	21
<b>6 A Linguagem Object Pascal</b>	<b>22</b>

<b>6.1 Tipos de Dados Pré-definidos</b>	<b>22</b>
6.1.1 Tipos Ordiniais	22
6.1.2 Tipos Reais	22
6.1.3 Tipos String	22
6.1.4 Tipo Variant	22
<b>6.2 TypeCasting e Conversão de Tipos</b>	<b>23</b>
<b>6.3 Tipos de Dados Definidos pelo Usuário</b>	<b>23</b>
<b>6.4 Estilos de Codificação</b>	<b>24</b>
6.4.1 Comentários	24
6.4.2 Letras Maiúsculas e Minúsculas	24
6.4.3 Espaços em Branco	24
6.4.4 Destaque da Sintaxe	24
<b>6.5 Instruções Pascal</b>	<b>25</b>
6.5.1 Operadores	25
6.5.2 Instruções Simples e Compostas	26
6.5.3 Instruções Condicionais	26
6.5.4 Loops	26
<b>6.6 Procedimentos e Funções</b>	<b>27</b>
6.6.1 Parâmetros	28
6.6.2 Convenções de Chamadas	29
6.6.3 Declarações	29
6.6.4 Tipos Procedurais	29
<b>6.7 Classes e Objetos</b>	<b>30</b>
6.7.1 Construtores e Destrutores	30
6.7.2 Encapsulamento	31
6.7.3 Classes e Units	32
6.7.4 Escopo	32
6.7.5 A Palavra-Chave Self	32
6.7.6 Métodos Classe e Dados Classe	32
6.7.7 Ponteiro de Método	33
6.7.8 Referências a Classes	33
6.7.9 Herdando de Tipos Existentes	33
6.7.10 Métodos Virtuais e Dinâmicos	34
6.7.11 Manipuladores de Mensagens	35
6.7.12 Métodos Abstratos	35
6.7.13 Informações de Tipo em Tempo de Execução (RTTI)	35
6.7.14 Manipulando Exceções	36
<b>7 A Biblioteca de Componentes Visuais</b>	<b>38</b>
<b>7.1 A Hierarquia da VCL</b>	<b>38</b>
7.1.1 Componentes	38
7.1.2 Objetos	39
<b>7.2 Usando Componentes e Objetos</b>	<b>39</b>
<b>7.3 Propriedades</b>	<b>39</b>
7.3.1 Propriedades Mais Comuns	41
<b>7.4 Métodos de Componentes</b>	<b>43</b>
<b>7.5 Eventos de Componentes</b>	<b>44</b>
<b>7.6 Usando Coleções Delphi</b>	<b>46</b>
<b>Parte II - Usando Componentes</b>	<b>48</b>
<b>8 Uma Viagem Pelos Componentes Básicos</b>	<b>48</b>
<b>8.1 Seta do Mouse</b>	<b>48</b>
<b>8.2 Botão (Button)</b>	<b>48</b>

<b>8.3 Rótulo (Label)</b>	<b>49</b>
<b>8.4 Caixa de Diálogo Color (DialogColor)</b>	<b>49</b>
<b>8.5 Caixa de Edição (Edit)</b>	<b>50</b>
<b>8.6 Caixa de Edição com Máscara</b>	<b>50</b>
<b>8.7 Memo</b>	<b>51</b>
<b>8.8 Caixa de Diálogo Font (FontDialog)</b>	<b>52</b>
<b>8.9 RichEdit</b>	<b>53</b>
<b>8.10 Caixa de Verificação (CheckBox)</b>	<b>54</b>
<b>8.11 Botão de Radio (RadioButton)</b>	<b>54</b>
<b>8.12 Caixa de Listagem (ListBox)</b>	<b>54</b>
<b>8.13 Caixa Combinada (ComboBox)</b>	<b>55</b>
<b>8.14 Barra de Rolagem (ScrollBar)</b>	<b>56</b>
<b>9 Criando e Manipulando Menus</b>	<b>58</b>
<b>9.1 A Estrutura de um Menu</b>	<b>58</b>
9.1.1 Diferentes Papeis dos Itens do Menu	58
<b>9.2 Menu Principal (MainMenu)</b>	<b>58</b>
9.2.1 Editando um Menu com o Menu Designer	58
9.2.2 Observações	59
9.2.3 Respondendo a Comando de Menu	59
9.2.4 Mudando Menus em Tempo de Execução	59
<b>9.3 Menu Pop-Up (PopupMenu)</b>	<b>60</b>
9.3.1 Criando Menus Pop-Up	60
9.3.2 Manipulando Menus Pop-Up Automaticamente	60
9.3.3 Manipulando Menus Pop-Up Manualmente	61
<b>9.4 Alterando o Menu de Sistema</b>	<b>61</b>
<b>10 De Volta ao Formulário</b>	<b>63</b>
<b>10.1 Formulários Versus Janelas</b>	<b>63</b>
<b>10.2 Janelas Sobrepostas, Pop-Up e Filhas</b>	<b>63</b>
<b>10.3 O Aplicativo é uma Janela</b>	<b>63</b>
<b>10.4 Definindo Estilos e Comportamentos de Formulários</b>	<b>64</b>
10.4.1 O Estilo do Formulário	64
10.4.2 O Estilo da Borda	64
<b>10.5 Os Ícones da Borda</b>	<b>64</b>
<b>10.6 Configurando Mais Estilos de Janelas</b>	<b>65</b>
<b>10.7 Formulários em Diferentes Resoluções de Tela</b>	<b>65</b>
10.7.1 Graduação Manual do Formulário	66
10.7.2 Graduação Automática do Formulário	66
<b>10.8 Definindo a Posição e o Tamanho do Formulário</b>	<b>66</b>
10.8.1 Propriedade Position	66
10.8.2 Propriedade WindowState	66
<b>10.9 O Tamanho de um Formulário e sua Área Cliente</b>	<b>67</b>
<b>10.10 O Tamanho Máximo e Mínimo de um Formulário</b>	<b>67</b>
<b>10.11 Criação Automática dos Formulários</b>	<b>68</b>
<b>10.12 Fechando um Formulário</b>	<b>69</b>

<b>10.13 Recebendo Entrada do Mouse</b>	<b>69</b>
10.13.1 O Papel dos Botões do Mouse	69
10.13.2 Usando o Windows sem um Mouse	69
<b>10.14 Desenhando no Formulário</b>	<b>70</b>
10.14.1 As Ferramentas de Desenho	70
10.14.2 Desenhando Formas	70
<b>10.15 Desenhando e Pintando no Windows</b>	<b>71</b>
10.15.1 Pintando uma Única Forma	71
10.15.2 Pintando uma Série de Formas	72
<b>11 Componentes Gráficos</b>	<b>74</b>
<b>11.1 O Botão de Bitmap (BitBtn)</b>	<b>74</b>
11.1.1 Muitas Imagens em um Bitmap	74
<b>11.2 O Image</b>	<b>75</b>
<b>11.3 Desenhando em um Bitmap</b>	<b>75</b>
<b>11.4 Outline</b>	<b>75</b>
<b>11.5 ImageList</b>	<b>76</b>
<b>11.6 TreeView</b>	<b>77</b>
<b>11.7 ListView</b>	<b>77</b>
<b>11.8 Construindo Grades (Tabelas)</b>	<b>79</b>
11.8.1 StringGrid e DrawGrid	79
11.8.2 ColorGrid	79
<b>12 Uma Barra de Ferramentas e uma Barra de Status</b>	<b>80</b>
<b>12.1 Agrupando Controles com o Painel</b>	<b>80</b>
<b>12.2 SpeedButton</b>	<b>80</b>
<b>12.3 Criando uma Barra de Ferramentas</b>	<b>81</b>
12.3.1 Adicionando Dicas à Barra de Ferramentas (Hint)	81
<b>12.4 Criando uma Barra de Status</b>	<b>81</b>
12.4.1 Adicionando Dicas à Barra de Status	82
<b>13 Formulários Múltiplos e Caixas de Diálogo</b>	<b>83</b>
<b>13.1 Caixas de Diálogo Versus Formulários</b>	<b>83</b>
<b>13.2 Adicionando um Segundo Formulário a um Programa</b>	<b>83</b>
<b>13.3 Formulários Modais e Não-Modais</b>	<b>84</b>
<b>13.4 Mesclando Menus de Formulários</b>	<b>84</b>
<b>13.5 Criando uma Caixa de Diálogo</b>	<b>85</b>
<b>13.6 Usando Caixas de Diálogo Pré-definidas</b>	<b>85</b>
13.6.1 Caixas de Diálogo Comuns do Windows	85
13.6.2 Caixas de Mensagem e de Entrada	86
<b>13.7 Herança Visual de Formulários</b>	<b>86</b>
13.7.1 Herdando de um Formulário-Base	86
<b>14 Rolagem e Formulários Multipáginas</b>	<b>89</b>
<b>14.1 Rolando um Formulário</b>	<b>89</b>
<b>14.2 Criando Fichários</b>	<b>89</b>
14.2.1 TabControl, PageControl e TabSheet	89
14.2.2 Abas sem Páginas e Páginas sem Abas	90

<b>15</b>	<b><i>Dividindo Janelas</i></b>	<b>91</b>
15.1	Técnicas de Divisão de Formulários	91
15.2	Dividindo com um Cabeçalho (HeaderControl)	91
<b>16</b>	<b><i>Criando Aplicativos MDI</i></b>	<b>93</b>
16.1	Janelas-Filhas e Janelas de Moldura	93
16.2	Criando um Menu Janela Completo	94
16.3	Montando uma Janela Filha	94
<b>17</b>	<b><i>Utilizando Controles OLE</i></b>	<b>95</b>
17.1	Controles OLE Versus Componentes Delphi	95
17.2	Instalando um Novo Controle OLE	95
17.3	Utilizando Controles OLE	96
<b>18</b>	<b><i>Criando Aplicativos de Bancos de Dados</i></b>	<b>98</b>
18.1	Acesso a Bancos de Dados	98
18.2	Componentes de Bancos de Dados	99
18.2.1	DataSource	99
18.2.2	Data Sets	100
18.2.3	Outros Componentes de Acesso a Dados	100
18.2.4	Componentes Relativos a Dados	100
18.3	Acessando os Campos de uma Tabela ou Query	101
18.4	Usando Campos para Manipular uma Tabela	102
18.4.1	Procurando Registros em uma Tabela	102
18.4.2	Atualizando Registros em uma Tabela	102
18.5	Criando um Formulário Mestre-Detalhe com o Form Expert	102
18.6	Caixas Combinadas Relativas a Dados	103



# Introdução

## **Agradecimentos**

Meus agradecimentos são destinados ao meu amigo de graduação Bruno de P. Ribeiro, por meio de quem eu tive conhecimento desta atividade de iniciação científica, à minha orientadora Vera M. B. Werneck por ter me dado a oportunidade de adquirir novos conhecimentos ao participar de atividades de pesquisa, aos outros participantes do grupo pela cooperação e complementação das minhas tarefas e, finalmente, àqueles que de alguma forma apoiaram nosso trabalho, dentre os quais, o professor Gustavo J. M. Hajdu.

## **Objetivos**

Esta apostila é baseada no Livro Dominando o Delphi 2 para Windows 95 / NT “A Bíblia”, do escritor Marco Cantù e, de um modo geral, segue a mesma ordem de apresentação dos assuntos. O Help on-line do aplicativo foi um dos recursos secundários utilizados para um maior detalhamento de certos tópicos.

O objetivo deste material é fornecer o conhecimento necessário a respeito da ferramenta Delphi 2.0 ao grupo de iniciação científica “Engenharia de Software para Sistemas Baseados em Conhecimento” da Universidade do Estado do Rio de Janeiro, orientado pela professora Vera M. B. Werneck e do qual faço parte deste agosto de 1997 na condição de bolsista.

São requisitos para um bom aproveitamento da apostila o conhecimento por parte do leitor da linguagem Pascal e de conceitos de Orientação a Objetos, já que o texto se concentra na implementação desse recurso e não na sua teoria.

## **Convenções**

- Identificadores usados em linguagem de programação e exemplos de programas aparecem em uma fonte diferente da usada no resto do texto.
- Ao citar um comando efetuado através de um menu, é utilizada a notação NomeDoMenu / ÍtemDoMenu. Significa que o menu NomeDoMenu deve ser aberto e a opção ÍtemDoMenu escolhida. O mesmo vale para menus com maior número de níveis (NomeDoMenu / ÍtemDoMenu / Sub-ítemDoMenu / ...).

Rio de Janeiro, 4 de fevereiro de 1998

Autor: Luiz Reuter Silva Torro  
e-mail: luiz\_t@hotmail.com

## Parte I - Programação Visual e Linguagem

Nesta parte será apresentado o ambiente, e dada uma visão geral da linguagem Object Pascal e da Biblioteca de Componentes Visuais.

### O que é o Delphi ?

O Delphi é um ambiente de desenvolvimento de softwares com as seguintes características:

1. Visual: A definição da interface e até mesmo de parte da estrutura de um aplicativo Delphi pode ser realizada com o auxílio de ferramentas visuais. Por exemplo, uma tela é criada com um simples clique e um botão, selecionando esta imagem em uma barra de ferramentas e clicando sobre a tela onde ele deve aparecer.
2. Orientada a Objeto: Os conceitos de classe, herança e polimorfismo são abarcados pela linguagem de programação do Delphi, o Object Pascal. Esta não é, no entanto, uma linguagem puramente orientada a objeto como Smalltalk e Eiffel.
3. Orientada a Eventos: Cada elemento de uma interface de aplicativo é capaz de capturar e associar ações a uma série de eventos (caracter teclado, clique do mouse, ...).
4. Compilada: A geração de código em linguagem de máquina acelera a execução dos aplicativos.

## Terminologia do Delphi

Alguns termos utilizados no ambiente Delphi:

- **Expert**: Gerador de código baseado em perguntas feitas ao desenvolvedor. Os ícones que invocam Experts costumam apresentar uma lâmpada para diferenciá-los



**Figura 1-A**

Expert .

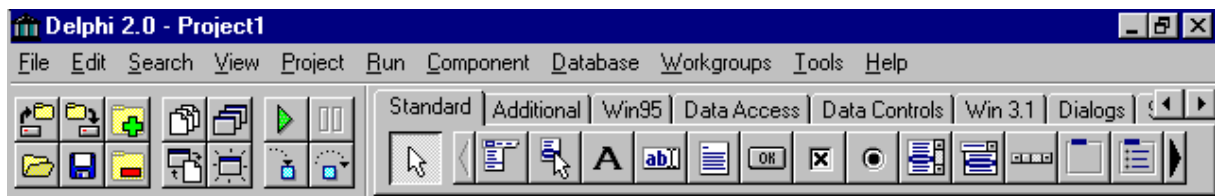
- **Hint**: Uma dica, que aparece dentro de um retângulo, a respeito do objeto sobre o qual o mouse é posicionado.



**Figura 1-B:**

Hint.

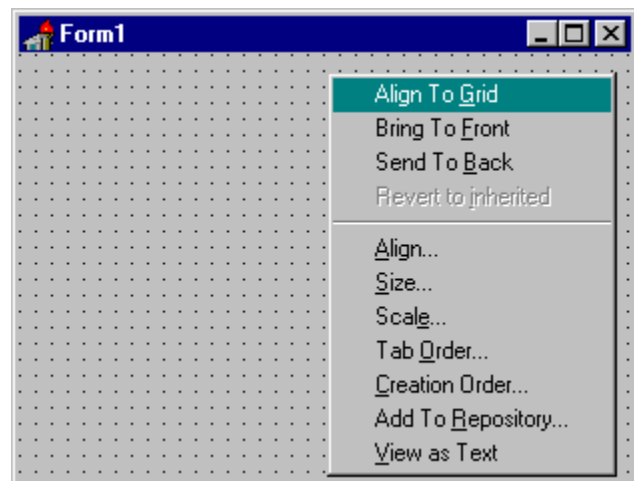
- **SpeedBar** (ou barra de ferramentas): Conjunto de ícones posicionados no topo da tela com funções equivalentes às de alguns itens dos menus.



**Figura 1-C**

Janela do Delphi 2.0 (barra de título, barra de menus e speedbar).

- **SpeedMenu** (ou menu de atalho): Menu reduzido que surge quando se clica um objeto com o botão secundário do mouse.



**Figura 1-D**

Speedmenu de um formulário.

# 1 Aplicativos e Projetos

Um aplicativo Delphi é criado a partir de um projeto, que engloba:

- um arquivo de projeto (.dpr) responsável pela unidade do projeto - lista todos os elementos de um projeto e possui o código de inicialização;
- arquivo(s) de formulário (.dfm) com as informações gráficas do aplicativo. Cada arquivo de formulário possui uma unit (descrita abaixo) associada;
- unit(s) - arquivo(s) de código Object Pascal (.pas) com as ações do aplicativo. Podem estar associadas ou não a formulários.

Outros tipos de arquivos utilizados no ambiente Delphi têm as extensões:

- bmp, ico: arquivos gráficos;
- dcu: unit Delphi compilada;
- ~df: backup de formulário gráfico;
- dof: arquivo de opções do Delphi;
- ~dp: backup do projeto;
- dsk: configurações de desktop (posicionamento de janelas, ...);
- dsm: dados do object browser;
- exe: arquivo executável compilado;
- ~pa: backup de unit;
- res: arquivo de recursos compilados (ícone do projeto, ...).

## 1.1 Criando um Projeto

Ao iniciar o Delphi, é fornecido um projeto em branco. Se outro projeto estiver aberto, um novo poderá ser criado selecionando o menu File / New Application:

## 1.2 Abrindo um Projeto

Selecione o menu File / Open e localize o arquivo do projeto desejado (.dpr).

## 1.3 Salvando um Projeto

Selecione o menu File / Save Project As. Será pedido um nome para cada unit e para o arquivo de projeto.

## 1.4 Modificando um Projeto

Selecione o menu View / Project Manager. Esta janela permite a alteração das propriedades do projeto, como a inclusão e remoção de units, alteração do nome do aplicativo e definição dos formulários exibidos quando o programa é iniciado. A alteração de algumas das opções iniciais gera um arquivo com a extensão dof contendo as novas definições.

O código do arquivo de projeto (.dpr), que pode ser visto com o comando View / Project Source, costuma servir apenas para iniciar o programa que executa a janela principal do aplicativo.

## **1.5 Rodando Aplicativos**

### **1.5.1 Compilando um Projeto**

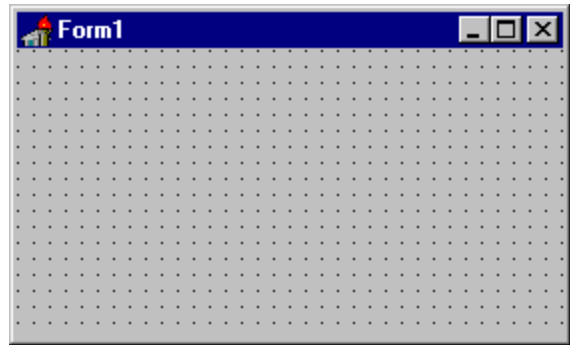
Para gerar o executável de um projeto basta selecionar o menu Project / Compile. Após a compilação de cada uma das units em arquivos de código objeto (.dcu), elas são associadas com códigos da biblioteca visual do Delphi (VCL) para formar o aplicativo.

### **1.5.2 Depurando um Projeto**

Um projeto será executado no depurador integrado sempre que ele for rodado a partir do Delphi com o comando Run / Run.

## 2 Formulários

Dentro do Delphi, as janelas são referenciadas como formulários e contêm objetos de interface com o usuário.



**Figura 1-A**

Formulário recém-criado.

### 2.1 Criando um Formulário

Quando um projeto é criado, ele já fornece um formulário em branco. Para criar outros formulários, selecione o menu File / New Form.

Cada formulário criado no projeto virá acompanhado de uma unit associada.

### 2.2 Adicionando um Formulário ao Projeto

Pode-se reaproveitar formulários prontos em um projeto sem a necessidade de criá-los novamente. Para adicionar um formulário existente a um projeto, selecione o menu File / Add to Project e localize a unit associada ao formulário desejado.

Quando se adiciona um formulário a um projeto, não é feita uma cópia, e sim uma referência ao original. Portanto, quaisquer alterações realizadas se refletirão em todos os projetos que utilizem o formulário.

### 2.3 Removendo um Formulário do Projeto

Selecione o menu File / Remove from Project e escolha o formulário (e sua unit associada) a ser removido na lista dos componentes do projeto que é exibida.

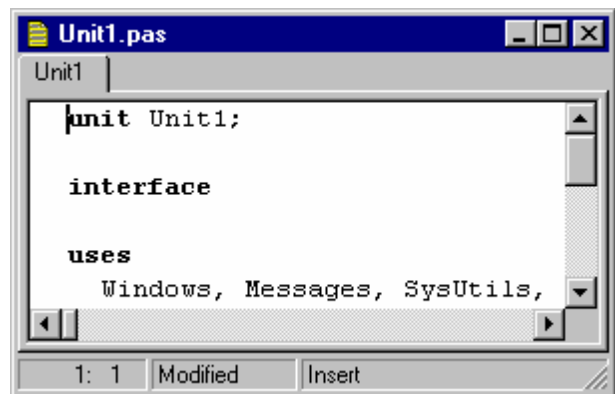
A remoção do componente de um projeto não acarreta a sua exclusão, os arquivos apenas deixam de ser considerados pelo projeto.

### 2.4 Manipulando Formulários

Use a lista de formulários do menu View / Forms para localizar e visualizar os formulários. Já a unit associada ao formulário pode ser vista com o comando View / Toggle Form/Unit, e vice-versa.

### 3 Units

As units são a base da modularidade da linguagem e servem para definir classes, rotinas e elementos visuais. São compostas pelas seções:



**Figura 2-A**

Código padrão de uma unit.

1) Interface: Declara os elementos que são visíveis para outras units: rotinas, variáveis globais, tipos de dados, etc. A cláusula `uses` no início da seção `interface` indica quais outras units precisamos acessar dentro desta seção.

2) Implementation: Contém o código real e outras declarações. A cláusula `uses` no início da seção `implementation` indica quais outras units precisamos acessar dentro desta seção.

3) Initialization (opcional): Contém o código a ser executado quando o programa que usa a unit é carregado para a memória.

4) Finalization: (opcional): Contém o código a ser executado quando o programa que usa a unit é descarregado da memória.

#### 3.1 Criando uma Unit

Cada formulário inserido no projeto virá acompanhado de uma unit; o inverso, porém, não é verdadeiro. Para criar uma unit desvinculada de um formulário, selecione o menu `File / New`; na página `New` da caixa de diálogo apresentada, escolha o ícone `Unit`.

#### 3.2 Adicionando uma Unit ao Projeto

Pode-se reaproveitar units prontas em um projeto sem a necessidade de criá-las novamente. Para adicionar uma unit existente a um projeto, selecione o menu `File / Add to Project` e localize a unit desejada.

Quando se adiciona uma unit a um projeto, não é feita uma cópia, e sim uma referência à original. Portanto, quaisquer alterações realizadas se refletirão em todos os projetos que utilizem a unit.

#### 3.3 Removendo uma Unit do Projeto

Selecione o menu `File / Remove from Project` e escolha a unit a ser removida na lista dos componentes do projeto que é exibida.

A remoção do componente de um projeto não acarreta a sua exclusão, os arquivos apenas deixam de ser considerados pelo projeto.

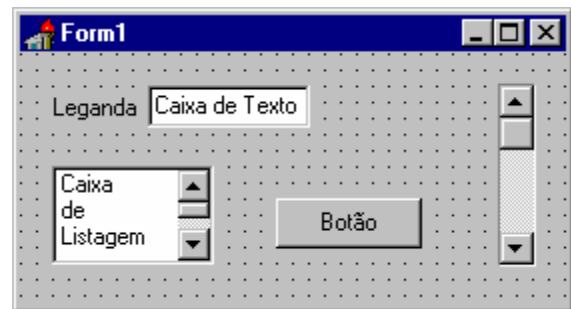
### **3.4 Manipulando Units**

Use a lista de units do menu View / Units para localizar e visualizar as units.

## 4 Componentes

Os componentes são as classes básicas sobre as quais são construídos aplicativos Delphi. Alguns exemplos de componentes são:

- Botões
- Barras de Rolagem
- Caixas de Listagem
- Caixas de Texto
- Legendas



**Figura 4-A**

Formulário com alguns componentes.

Um formulário pode conter uma série de componentes de forma a criar uma interface para o usuário com botões, menus, etc.

Os componentes são divididos em categorias como Standard, Data Access, ..., de acordo com suas funções e ficam na palheta de componentes posicionada no alto tela. Para ver os componentes de uma categoria selecione a página correspondente na palheta.



**Figura 4-B**

Palheta de componentes com a página Standard selecionada.

### 4.1 Adicionando Componentes a um Formulário

Selecione o componente desejado na palheta de componentes e clique no local do formulário onde o componente deve ser posicionado.

### 4.2 Removendo Componentes de um Formulário

Selecione o componente a ser removido e selecione o menu Edit / Delete.

### 4.3 Manipulando Componentes de um Formulário

Em tempo de projeto um componente pode ser reposicionado arrastando-o para uma nova posição, ou pode ter seu tamanho alterado selecionando-o com um clique e arrastando os controladores de tamanho que surgem ao seu redor. A Paleta de Alinhamento (menu View / Alignment Palette) auxilia no posicionamento relativo dos componentes.

A ordem de tabulação<sup>1</sup> pode ser alterada através do menu Edit / Tab Order.

---

<sup>1</sup> A ordem em que os componentes são acessados quando o usuário tecla <Tab>.

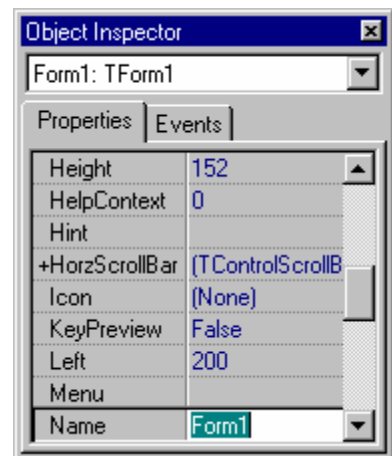
Um formulário pode ser reaproveitado adicionando-o ao repositório com o SpeedMenu Add to Repository.

#### 4.4 Manipulando Propriedades e Eventos - Object Inspector

Selecione o menu View / Object Inspector para abrir a janela do Object Inspector. Ele permite a visualização e a alteração das propriedades e das respostas a eventos do componente selecionado. A seleção de um componente é feita clicando-se sobre o mesmo ou selecionando seu nome na caixa de opções do Object Inspector.

Para alterar uma propriedade, basta selecioná-la e digitar um novo valor. Em alguns casos o valor deve ser escolhido em uma lista drop-down<sup>2</sup> que aparece ao lado da propriedade selecionada. Em outros, aparece um sinal de adição à frente da propriedade, indicando a existência de sub-propriedades. Você pode dar um duplo-clique sobre o sinal para exibi-las e alterá-las. Há, ainda, um outro caso, em que surge um pequeno botão com reticências. Você deve, então, dar um duplo-clique na área branca da propriedade para abrir uma caixa de diálogo.

No caso dos eventos, após selecioná-lo, pode-se escolher um procedimento existente na lista drop-down ao lado do evento, ou dar um duplo-clique na área branca para criar o procedimento padrão do evento.



**Figura 4-C**

Object Inspector exibindo as propriedades do formulário Form1

---

<sup>2</sup> Lista de opções que surge quando se clica uma caixa combinada ou um menu.

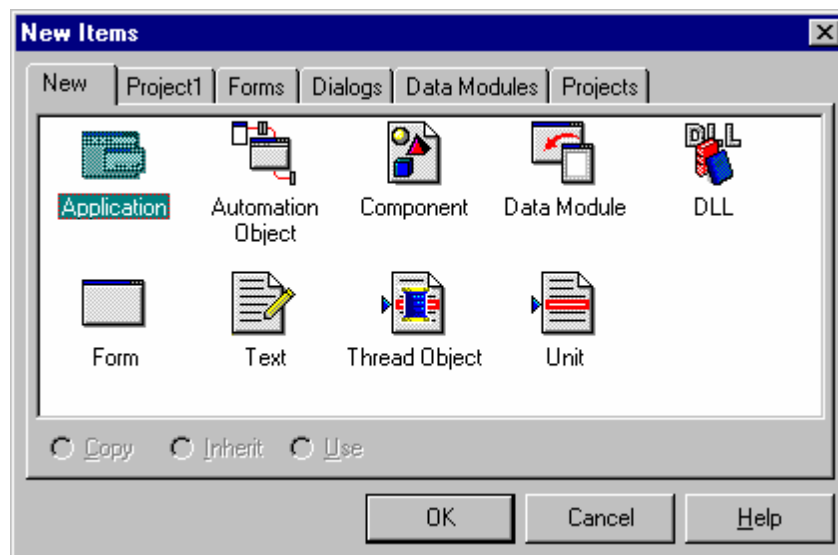
## 5 O Repositório de Objetos e os Experts

Um outro modo de se criar objetos é através do Repositório de Objetos, composto por modelos de objetos pré-definidos ou definidos pelo usuário, e dos Experts, que guiam o processo de criação através de janelas de diálogo.

O Repositório de Objetos permite a reutilização de objetos utilizando modelos prontos como ponto de partida. A reutilização compreende 3 possibilidades:

1. **Copiar:** Duplica o objeto original - alterações na cópia não alteram o original.
2. **Herdar:** Herda as características do objeto original e mantém um vínculo - alterações no original são refletidas no novo objeto.
3. **Usar:** Referencia o objeto original - alterações (em tempo de projeto) no objeto inserido são refletidas no objeto original .

Para criar um objeto com o Repositório de Objetos, selecione o menu File / New. Surge a caixa de diálogo mostrada abaixo, com as seguintes páginas:



**Figura 5-A**

Página New do Repositório de Objetos.

6. **New:** Cria objetos, em alguns casos com a participação de um Expert. Exemplos:
  - Application:** Projeto em branco.
  - Automation Object:** Cria um objeto OLE através do Automation Object Expert.
  - Component:** Cria um componente através do Component Expert.
  - Data Module:** Módulo de dados em branco.
  - DLL:** DLL em branco
  - Form:** Formulário em branco.
  - Text:** Arquivo texto ASCII em branco.
7. **“Projeto Atual”:** Herda um dos formulários ou uma das units do projeto atual.
8. **Forms:** Insere um tipo pré-definido de formulário no projeto. Exemplos:
  - About Box:** “Caixa Sobre ...”.

**Database Form:** Cria um formulário de interface para banco de dados através do Database form Expert.

**Dual list box:** Formulário dotado de duas caixas de listas e de botões que permitem a transferência de elementos de uma caixa para a outra.

**QuickReport Labels:** Formulário de relatório.

**QuickReport List:** Formulário de relatório.

**QuickReport Master/Detail:** Formulário de relatório de estrutura mais complexa.

**Tabbed pages:** Formulário baseado em páginas.

9. **Dialogs:** Insere caixas de diálogo, que são formulários com alguns atributos especiais, como o tipo da borda. Exemplos:

**Dialog Expert:** Expert gerador de caixas de diálogos com uma ou múltiplas páginas.

**Dialog with help (1):** Caixa de diálogo com botão de ajuda na parte inferior.

**Dialog with help (2):** Caixa de diálogo com botão de ajuda no lado direito.

**Password dialog:** Caixa de diálogo com uma caixa de edição simples para a digitação de senha.

**Standard dialog (1):** Caixa de diálogo com botões na parte inferior.

**Standard dialog (2):** Caixa de diálogo com botões no lado direito.

10. **Data Modules:** Insere um modulo de dados, que pode ser utilizado para o armazenamento de componentes não visuais.

11. **Projects:** Inicia um projeto. Exemplos:

**Application Expert:** Expert que permite opções sobre a estrutura do arquivo e alguns outros elementos da aplicação.

**MDI Application:** Define os elementos principais de uma aplicação MDI (Multiple Document Interface) - formulário principal com menu, barra de status e de ferramentas e um componente para a criação de janelas filhas.

**SDI Application:** Define os elementos principais de uma aplicação SDI (Single Document Interface).

**Win95 Logo Application:** Basicamente uma aplicação SDI com um componente RichEdit e código para a manipulação de correio eletrônico.

## **5.1 Personalizando o Repositório de Objetos**

### **5.1.1 Adicionando Modelos ao Repositório**

As páginas do Repositório podem ser personalizadas adicionando Experts e modelos.

Para adicionar um projeto ao Repositório, ative o comando Add to Repository do Menu Project; para adicionar um formulário, ative o comando Add to Repository do seu SpeedMenu.

Em ambos os casos, surgirá uma caixa de diálogo com espaços para o título, a descrição, a página do Repositório e o ícone do novo modelo.

### **5.1.2 Opções do Repositório**

O comando Tools / Repository dá acesso à tela de opções do Repositório. Nela é possível incluir, excluir, renomear e alterar o posicionamento das páginas onde os modelos e Experts são armazenados com a auxílio dos botões e setas ao redor da lista de páginas. A lista de modelos oferece as opções de edição e exclusão. Também é possível tornar um modelo o padrão de criação marcando a caixa de verificação New Form (ou New Project). No caso de formulários, pode-se tornar um modelo o formulário padrão de novos projetos marcando a caixa de verificação Main Form.

## 6 A Linguagem Object Pascal

Como esta apostila pressupõe por parte do leitor uma certa familiaridade com a Linguagem Pascal, que é um subconjunto da Object Pascal, suas características principais serão citadas sucintamente. Uma referência mais minuciosa pode ser obtida no Guia de Referência e no Help on-line da linguagem.

### 6.1 Tipos de Dados Pré-definidos

Há três grupos principais: ordinais, reais e strings.

#### 6.1.1 Tipos Ordinais

Tipo	Propósito
Integer, ShortInt, SmallInt, Longint	Inteiros com sinal
Byte, Word, Cardinal	Inteiros sem sinal
Boolean, ByteBool, WordBool, LongBool	Booleanos
Char	Caractere de 8 bits
ANSIChar	Caractere de 8 bits
WideChar	Caractere de 16 bits (Unicode)

Tabela 6.1-A

#### 6.1.2 Tipos Reais

Tipo	Propósito
Single, Real, Double, Extended	Números reais
Comp	Números inteiros muito longos
Currency	Números reais com 4 dígitos decimais

Tabela 6.1-B

#### 6.1.3 Tipos String

Tipo	Propósito
ShortString	Tamanho máximo de 255 caracteres
String, ANSIString	Tamanho "ilimitado"

Tabela 6.1-C

#### 6.1.4 Tipo Variant

O tipo de dado variant tem checagem de tipo em tempo de execução, e foi introduzido na linguagem para suportar automação OLE (Object Linking and Embedding). No entanto, uma variável deste tipo pode ser usada para armazenar qualquer tipo de dado.

## 6.2 TypeCasting e Conversão de Tipos

A notação de typecasting se assemelha à chamada de uma função, cujo argumento é o valor original.

```
var
  n: integer;
  c: char;
  b: boolean;
  ...

begin
  n := integer ('x');
  c := char (n);
  b := boolean (0);
end;
```

As rotinas de conversão de tipo estão listadas abaixo:

Rotina	Converte
chr	Número ordinal em caracter
ord	Valor ordinal no número que indica sua ordem
round	Valor real em inteiro, arredondando
trunc	Valor real em inteiro, truncando
inttostr	Número em string
inttohex	Número em string com representação hexadecimal
strtoint	String em número, suscitando uma exceção se a string não estiver correta
strtointdef	String em número, usando um valor padrão se a string não estiver correta
val	String em número
str	Número em string, usando parâmetros de formatação
strpas	String terminada em nulo em string estilo Pascal
strpcopy	String estilo Pascal em string terminada em nulo
strplcopy	Parte de uma string estilo Pascal em string terminada em nulo

## 6.3 Tipos de Dados Definidos pelo Usuário

Novos tipos de dados são definidos na seção Type do programa empregando os construtores de tipos:

**Faixas Secundárias:** Faixa de valores dentro da faixa de outro tipo.

Exemplo: `Type Maiúsculas = 'A'..'Z';`

**Enumerações:** Lista de valores na ordem desejada.

Exemplo: `Type Cores = (Vermelho, Verde, Azul);`

**Conjuntos:** Conjunto de valores do tipo definido.

Exemplo: `Type Letras = Set Of Maiúsculas;`

**Matrizes:** Número fixo de elementos de um tipo específico.

Exemplo: `Type TemperaturasDia = Array [1..24] of integer;`

**Registros:** Coleção fixa de elementos de diferentes tipos, os campos.

Exemplo: `Type Data = Record  
 Ano: integer;  
 Mês: Byte;`

```
    Dia: Byte;
end;
```

**Indicadores ou Ponteiros:** Define uma variável que mantém o endereço de memória de outra variável de certo tipo de dados.

Exemplo: `Type PonteiroInt = ^integer;`

**Arquivos:** Relacionados à entrada / saída.

Exemplo: `Type IntFile = File Of integer;`

## 6.4 Estilos de Codificação

### 6.4.1 Comentários

Tipo	Propósito
// <comentário>	Comentário “até o fim da linha”
{ <comentário> }	Comentário entre as chaves
(* <comentário> *)	Comentário entre o grupo “parênteses / asterisco”

**Tabela 6.4-A**

É possível aninhar comentários de tipos diferentes.

Os dois últimos esquemas de comentários também são empregados na construção de diretivas de compilação, tais como {`$X+`}.

### 6.4.2 Letras Maiúsculas e Minúsculas

O compilador Pascal não distingue entre letras maiúsculas e minúsculas nos identificadores. Desta forma, `MeuIdentificador` e `meuidentificador` são equivalentes.

### 6.4.3 Espaços em Branco

Os espaços, tabulações e caracteres de novas linhas, coletivamente conhecidos como espaços em branco, são ignorados pelo compilador, mesmo dentro de um comando já que o separador utilizado pelo compilador é o ponto-e-vírgula.

### 6.4.4 Destaque da Sintaxe

O Destaque em Cores da Sintaxe presente no editor Delphi utiliza formatos e cores diferentes para exibir as palavras digitadas. Por padrão, as palavras chaves estão em negrito, as strings e os comentários estão em cores e assim por diante. Você pode personalizar o destaque da sintaxe usando a página `Colors` do comando `Tools / Options`.

## 6.5 Instruções Pascal

### 6.5.1 Operadores

Estes são os grupos de operadores em ordem decrescente de precedência:

#### Unários

Operador	Propósito
@	Endereço de (retorna um ponteiro)
not	“Não” booleano ou bit-a-bit

Tabela 6.5-A

#### Operadores Multiplicativos e de type casting

Operador	Propósito
*	Multiplicação aritmética ou interseção de conjuntos
/	Divisão real
div	Divisão inteira
mod	Resto da divisão de números inteiros
as	Typecast seguro
and	“E” booleano e bit-a-bit
shl	Deslocamento de bits à esquerda
shr	Deslocamento de bits à direita

Tabela 6.5-B

#### Operadores Aditivos

Operador	Propósito
+	Adição, união de conjuntos, concatenação de strings, valor positivo, ou adição de compensação (offset)
-	Subtração, diferença de conjuntos, valor negativo, ou subtração de compensação (offset)
or	“Ou” booleano ou bit-a-bit
xor	“Ou exclusivo” booleano ou bit-a-bit

Tabela 6.5-C

#### Operadores Relacionais, de Pertinência de Conjunto e de Comparação de tipo

Operador	Propósito
=	Testar se é igual
<>	Testar se é diferente
<	Testar se é menor que
>	Testar se é maior que
<=	Testar se é menor ou igual que, ou se é subconjunto de um conjunto
>=	Testar se é maior ou igual que, ou se é superconjunto de um conjunto
in	Testar se é membro de um conjunto
is	Testar se o tipo é compatível

Tabela 6.5-D

## 6.5.2 Instruções Simples e Compostas

As instruções simples são separadas por ponto-e-vírgula.

```
x := y + z;  
Randomize;
```

As instruções compostas são instruções delimitadas pelo par `begin` e `end`.

```
begin  
  x := y + z;  
  Randomize;  
end;
```

## 6.5.3 Instruções Condicionais

1) Instruções If: Podem ser usadas as sintaxes `if-then` e `if-then-else`.

```
if (a < b) then  
  a := a + 1;
```

```
if (a < b) then  
  a := a + 100  
else  
  c := a;
```

2) Instruções Case: Podem ser usadas com e sem o `else`.

```
case Numero of  
  1: Texto := 'Um';  
  2: Texto := 'Dois';  
  3: Texto := 'Três';  
end;
```

```
case Numero of  
  1..3: Texto := 'Um';  
  4..6: Texto := 'Dois';  
  else Texto := 'Maior que 6';  
end;
```

Por padrão, a linguagem Object Pascal realiza a avaliação de curto-circuito de expressões booleanas. Isto significa que assim que o resultado da expressão puder ser determinado, para-se de avaliar a expressão. Este comportamento pode ser alterado com a opção `Complete boolean eval` na página `Compiler` do menu `Project / Options`.

## 6.5.4 Loops

1) Instruções For: Podem incrementar ou decrementar a variável.

```
for i := 1 To 10 do  
  k := k + 1;
```

```
for i := 10 DownTo 5 do  
  k := k + 2;
```

2) Instruções While e Repeat: A instrução Repeat é sempre executada pelo menos uma vez, o que nem sempre ocorre com a instrução While.

```
while i < 100 do
  i := i * 2;

repeat
  i := i * 2;
until i >= 100;
```

Dentro de um loop, o procedimento Break interrompe as iterações e segue para a próxima instrução do programa; já o procedimento Continue desvia a execução para o início da próxima iteração.

3) Instruções With: Simplifica a referência a subitens de uma estrutura.

Ao invés de acessar propriedades desta forma

```
Form1.canvas.pen.width := 2;
Form1.canvas.pen.color := clRead;
```

Podemos escrever

```
with Form1.Canvas.Pen do
begin
  width := 2;
  color := clRead;
end;
```

## 6.6 Procedimentos e Funções

Ao contrário dos procedimentos, as funções retornam um valor, definido através da sua atribuição ao identificador da função ou, alternativamente, à palavra-chave result. Esta atribuição pode aparecer um número qualquer de vezes dentro da rotina (dentro de if e else, etc.).

Definição:

```
procedure Alo (s : String);
begin
  ShowMessage ('Alo ' + s);
end;
```

Chamada:

```
Alo ('você !');
```

Resultado:

```
Alo você !
```

Definição:

```
function Dobro (i : integer) : integer;
begin
```

```
Dobro := i * 2;           //ou Result := i * 2;
end;
```

Chamada:

```
Result := Dobro (250);
```

Resultado:

```
Result := 500;
```

### 6.6.1 Parâmetros

O compilador assume que os parâmetros são passados por valor, a menos que seja usada a palavra-chave `var`.

Definição:

```
procedure Dobro (var i : integer);
begin
    i := i * 2;
end;
```

Chamada:

```
Dobro (Valor);
```

A palavra-chave `Const` impede a alteração de um parâmetro dentro de uma rotina gerando um erro caso você tente compilar um código que altere o valor do parâmetro constante.

```
function Dobro (Const i : integer) : integer;
begin
    Dobro := i * 2;
end;
```

O recurso da matriz aberta é a forma de passar uma quantidade variável de parâmetros. Basta definir uma matriz sem índices. As funções `Low`, `High` e `SizeOf` fornecem os valores necessários à manipulação da matriz.

Definição:

```
procedure Clear (var A: Array of Double);
var
    i: integer;
begin
    for i := 0 to High (A) do A [i] := 0;
end;
```

Chamada:

```
var List1 : Array [0..3] of Double;
begin
    Clear (List1);
end;
```

## 6.6.2 Convenções de Chamadas

A convenção padrão na passagem de parâmetros do Delphi 2 é a fastcall, que não é compatível com a API do Windows; as funções da API do Windows devem ser declaradas solicitando-se a utilizando da convenção padrão do Pascal com a palavra-chave `stdcall` antes da declaração da função.

## 6.6.3 Declarações

1) Declarações Forward: Disponibiliza o protótipo de uma rotina antes da sua definição completa.

```
procedure Ola; Forward;

procedure OlaDuplo;
begin
  Ola;
  Ola;
end;

Procedure Ola;
begin
  ShowMessage ('Ola mundo !');
end;
```

2) Declarações External: Possibilitam a vinculação de rotinas externas escritas em assembly ou localizadas em DLLs.

```
function GetMode: Word; external;
function LineTo; external; 'gdi32' name 'LineTo';
```

## 6.6.4 Tipos Procedurais

Você pode declarar variáveis de tipos procedurais e passá-las como argumentos para rotinas. A declaração de um tipo procedural indica a lista dos parâmetros (ou apenas os tipos dos parâmetros).

```
Type
  IntProc = procedure (var Num: integer);

procedure DobraValor (var Valor: integer);
begin
  Valor := Valor * 2;
end;

var
  IP: IntProc;
  X: integer;

begin
  IP := DobraValor;
  X := 5;
  IP (X);
end;
```

## 6.7 Classes e Objetos

O Delphi, assim como a linguagem Eiffel, trabalha com o modelo de referência de objetos, no qual uma variável objeto é um ponteiro para o objeto real. É preciso chamar um método construtor (`create` é o construtor padrão) para gerar uma instância real do objeto na memória. A definição de uma classe é feita com a palavra-chave `class`.

Definição da classe `Data`:

```
type
  Data = class          //isto equivale a Data = class (TObject)
    Dia, Mes, Ano: integer;          //campos
    procedure DefVal (m, d, a: integer);          //métodos
    function AnoBis: Boolean;
  end;
```

Para indicar que um método pertence a uma classe, usa-se a notação `[Nome da Classe].[Método]`:

```
procedure Data.DefVal (m, d, a: integer);
begin
  Mes := m;
  Dia := d;
  Ano := a;
end;
```

```
function Data.AnoBis: Boolean;
begin
  if (Ano mod 4 <> 0) then
    AnoBis := False
  else
    if (Ano mod 100 <> 0) then
      AnoBis := True
    else
      if (Ano mod 400 <> 0) then
        AnoBis := False
      else
        AnoBis := True;
  end;
```

Utilização de um objeto da classe `Data`:

```
var
  Dia: Data;
  Bis: Boolean;

begin
  ...
  Dia := Data.Create;          //cria uma instância de objeto
  Dia.DefVal (10, 10, 1994);  //chama métodos do objeto
  Bis := Dia.AnoBis;
  ...
  Dia.Free                    //destrói o objeto
end;
```

### 6.7.1 Construtores e Destruutores

Construtores e destrutores são rotinas especiais que alocam e liberam memória para objetos automaticamente. Elas são declaradas com as palavras-chave `constructor` e `destructor`

respectivamente; Todas as classes que criamos herdamos o método construtor `create` e o destrutor `destroy`, porém podemos definir versões personalizadas de construtores e destrutores que realizem outras tarefas além de criar e destruir objetos, como inicializar o objeto e liberar recursos.

A destruição de um objeto que não teve seu destrutor redefinido deve ser feita indiretamente com o método `free` e não com o método `destroy`.

Definição da classe `Data` com o construtor personalizado `Init`:

```
type
  Data = class
    Dia, Mes, Ano: integer;           //campos
    constructor Init (d, m, a, integer); //construtor
    procedure DefVal (m, d, a: integer); //métodos
    function AnoBis: Boolean;
  end;

constructor Data.Init (m, d, a: integer);
begin
  Mes := m;
  Dia := d;
  Ano := a;
end;
```

Com o novo construtor, a criação e inicialização do objeto ocorrem de uma só vez:

```
var
  Dia: Data;
  Bis: Boolean;

begin
  ...
  Dia := Data.Init (10, 10, 1994); //cria e inicializa
  ...
  Dia.Free;                       //destrói o objeto
end;
```

## 6.7.2 Encapsulamento

Há duas construções envolvidas no encapsulamento: classes e units.

Os especificadores de acesso associadas às classes são:

- 1) Private: Denota componentes acessíveis somente dentro da unit onde é definida a classe.
- 2) Public: Denota componentes acessíveis a todo o programa.
- 3) Protected: Denota componentes acessíveis somente pela classe que os define e por suas classes descendentes.
- 4) Published: Denota componentes acessíveis em tempo de projeto inclusive. Esta é a especificação padrão.
- 5) Automated: Denota componentes públicos para automação OLE.

### 6.7.3 Classes e Units

Quando você define uma nova classe em uma unit, escreve na seção `interface` a declaração da classe, que é composta pela declaração de seus dados e pela declaração (implicitamente) `forward` de seus métodos. O código dos métodos usualmente é colocado na seção `implementation`.

### 6.7.4 Escopo

Um identificador declarado na parte de implementação de uma unit só poderá ser usado dentro da própria unit (identificadores locais). Já os identificadores declarados na parte de interface também serão visíveis às units que usarem a unit que os declara (identificadores globais).

### 6.7.5 A Palavra-Chave Self

A palavra-chave `self` é um parâmetro implícito passado a qualquer método e representa a instância de objeto que o chamou. Na maior parte dos casos é desnecessário utilizar este parâmetro pois ele fica subentendido.

```
procedure Data.DefVal (d, m, a: integer);
begin
  Dia := d;           //equivale a Self^.Dia := d;
  Mês := m;          //equivale a Self^.Mês := m;
  Ano := a;          //equivale a Self^.Ano := a;
end;
```

### 6.7.6 Métodos Classe e Dados Classe

Normalmente os dados dos campos de uma classe são independentes para cada instância de objeto desta classe. Existe uma forma de adicionarmos campos compartilhados: se declararmos um campo na parte de implementação da unit, ele se comportará como uma variável de classe - uma única localização na memória compartilhada por todos os objetos de uma classe.

```
type
  Pessoa = class
    nome: string //variável independente para cada objeto
    ...
    function Quantidade: integer;
  end;
  ...

implementation

  var i: integer;           //variável compartilhada entre objetos

  function Pessoa.Quantidade: integer;
  begin
    ..
    Quantidade := i;
  end;
```

Um método classe, declarado com a palavra-chave `class`, é um método que não pode acessar os dados de instâncias de objetos, mas que pode ser chamado ao se referir a uma classe (ou a um objeto desta classe).

```
type
```

```

MinhaClasse = class
  ...
  class function Valor: byte;           //método de classe
end;
...

begin
  i := MinhaClasse.Valor
end;

```

### 6.7.7 Ponteiro de Método

Um tipo ponteiro de método é semelhante a um tipo procedural, a diferença é que a sua declaração é seguida das palavras-chave `of object`.

```

type
  IntMethodPointer = procedure (Num: Integer) of object;

  MinhaClasse = class
    Value: Integer;
    Operation: IntMethodPointer;
  end;

```

### 6.7.8 Referências a Classes

Este tipo é declarado com a participação das palavras-chave `class of`.

```

type
  CRefType = class of TControl;

var
  ClassRef: CRefType;

...
begin
  ClassRef := TRadioButton;
end;

```

### 6.7.9 Herdando de Tipos Existentes

Para herdar de um tipo existente<sup>3</sup>, você precisa somente colocar esse tipo entre parênteses após a palavra-chave `class` na declaração da classe descendente.

```

type
  NovaData = class (Data)
  ...
end;

```

Como regra geral, você pode usar um objeto de uma classe descendente toda vez que um objeto de uma classe ancestral for esperado, porém o inverso não é permitido.

```

type
  Animal = class
  ...

```

---

<sup>3</sup> Um novo tipo pode herdar de uma única classe existente, isto é, não é permitida a herança múltipla.

```

end;

type
  Cao = class (Animal)
    ...
  end;

var
  MeuAnimal, MeuAnimal2: Animal;

begin
  MeuAnimal := Animal.Create;
  MeuAnimal2 := Cao.Create;
end;

```

### 6.7.10 Métodos Virtuais e Dinâmicos

As diretivas `virtual` (ou `dynamic`) e `override` são alguns dos recursos para a aplicação do polimorfismo. Um método de uma classe ancestral declarado como `virtual` (ou `dynamic`) pode ser ignorado (redefinido) em suas classes descendentes como `override`, contanto que tenha os mesmos parâmetros e (tipo de retorno no caso das funções).

Sem estas diretivas, se uma classe descendente declarar um método com o mesmo nome de um método da classe ancestral, o novo método substituirá o anterior ao invés de ignorá-lo.

Os métodos virtuais são mais utilizados que os dinâmicos pois favorecem a velocidade de execução em detrimento do tamanho do código, enquanto que os métodos dinâmicos fazem o contrário.

```

type
  Animal = class
    ...
    function Verse: string; virtual
  end;

type
  Cao = class (Animal)
    ...
    function Verse: string; override
  end;

function Animal.Verse: string;
begin
  Verse := 'Ruído do Animal';
end;

function Cao.Verse: string;
begin
  Verse := 'Au Au';
end;

```

Um método redefinido pode substituir o antigo como no caso anterior, ou complementá-lo. Neste caso, a palavra-chave `inherited` é usada para chamar o método da classe ancestral.

```

function Cao.Verse: string;
begin
  inherited;           //chama o método anterior
  Verse := 'Au Au';    //complemento
end;

```

### 6.7.11 Manipuladores de Mensagens

A diretiva `message` serve para definir métodos de manipulação de mensagens. Estes métodos devem ser procedimentos protegidos com um único parâmetro `var` do tipo `TMessage`, e a diretiva `message` deve ser seguida do identificador da mensagem Windows a que o procedimento se refere.

Este manipulador de mensagem é chamado quando o tamanho da janela é alterado.

```
type
  TForm1 = class (TForm)
    protected
      procedure WMMinMax (var Message: Tmessage);
        message WM_GETMINMAXINFO;
    end;

procedure TForm1. WMMinMax (var Message: TMessage);
begin
  ShowMessage ('Evento capturado.');
```

```
  inherited;
```

```
end;
```

### 6.7.12 Métodos Abstratos

Outro recurso de polimorfismo é a diretiva `abstract`, usada para declarar métodos (virtuais ou dinâmicos) que serão definidos somente em subclasses da classe atual.

```
type
  Animal = class
    ...
    function Verse: string; virtual; abstract;
  end;

  Cao = class (Animal)
    ...
    function Verse: string; override
  end;
...

// a função Animal.Verse não é definida

function Cao.Verse: string;
begin
  Verse := 'Au Au';
end;
```

### 6.7.13 Informações de Tipo em Tempo de Execução (RTTI)

Uma vez que cada objeto conhece seu tipo e sua herança, podemos requisitar estas informações através de certos operadores. O operador `is` retorna `True` se um determinado objeto for do tipo de dados especificado ou se for descendente desse tipo de dados.

```
if MeuAnimal is Cao then
  MeuCao := Cao (MeuAnimal);
```

O operador `as` realiza a moldagem de um objeto para o tipo de dados especificado, se for possível. Senão uma exceção é gerada.

```
MeuCao := Cao as MeuAnimal;
```

### 6.7.14 Manipulando Exceções

Podemos substituir a manipulação padrão de exceções realizada pelo Delphi em tempo de execução criando blocos de código protegido com as palavras-chave a seguir:

`try` - delimita o início de um bloco de código protegido.

`except` - delimita o final de um bloco de código protegido e insere as instruções de manipulação de exceções, com a forma: `on [tipo de exceção] do [instruções]`; este bloco é terminado com a palavra-chave `end`.

`finally` - indica um bloco executado após a saída do bloco `try`, seja essa saída pelo fluxo normal do programa ou por exceção; este bloco é terminado com a palavra-chave `end`.

`raise` - instrução usada para suscitar uma exceção.

```
function Divide (A, B: integer): integer;
begin
  try
    Divide := A div B;
  except
    on EDivByZero do
      begin
        Divide := 0;
        MessageDlg ('Divisão por zero', mtError, [mbOK], 0);
      end;
    end;
  end;
end;
```

O exemplo acima utiliza a classe de exceção pré-definida `EDivByZero`. No entanto, os programadores podem definir suas próprias exceções simplesmente criando uma nova subclasse das classes de exceções existentes.

```
type
  EArrayFull = class (Exception);
```

A exceção é gerada criando uma instância da classe de exceção desejada e, então, chamando a instrução `raise` com o objeto criado como argumento. Este objeto não precisa ser destruído, ele será apagado pelo manipulador de exceções.

Certifique-se de desativar a característica de depuração `Break on Exception` na tela `Environment Option` (menu `Tools / Options`); isto impedirá que o depurador interrompa a execução do programa quando um erro for encontrado.

```
var MeuErro: EArrayFull
...
if MinhaArray.Full then
begin
  MeuErro := EArrayFull.Create ('Matriz cheia');
  raise MeuErro;
end;
```

Quando uma exceção é manipulada, o fluxo de execução inicia-se novamente após o manipulador, e não após o código que suscita a exceção. No exemplo abaixo, quando `B` for igual a zero a instrução `Bmp.Free` não será executada.

```

function ComputeBits (A, B: integer): integer;
var
  Bmp: TBitmap;
begin
  Bmp := Tbitmap.create;
  try
    ComputeBits := A div B;
    Bmp.Free;
  except
    on EDivByZero do
      begin
        ComputeBits := 0;
        MessageDlg ('Erro emComputeBits', mtError, [mbOK], 0);
      end;
    end;
  end;
end

```

A instrução `finally` resolve o problema citado anteriormente, garantindo a execução da instrução `Bmp.Free`, quer ocorra uma exceção, quer não;

```

function ComputeBits (A, B: integer): integer;
var
  Bmp: TBitmap;
begin
  Bmp := Tbitmap.create;
  try
    ComputeBits := A div B;
  finally
    Bmp.Free;
  end;
end;

```

Um bloco `try` pode ser seguido de um bloco `except` ou de um bloco `finally`, mas não de ambos ao mesmo tempo. No entanto, podemos obter o mesmo efeito construindo blocos `try` aninhados.

```

function ComputeBits (A, B: integer): integer;
var
  Bmp: TBitmap;
begin
  Bmp := Tbitmap.create;
  try
    try
      ComputeBits := A div B;
    finally
      Bmp.Free;
    end;
  except
    on EDivByZero do
      begin
        ComputeBits := 0;
        MessageDlg ('Erro emComputeBits', mtError, [mbOK], 0);
      end;
    end;
  end;
end;

```

## 7 A Biblioteca de Componentes Visuais

A biblioteca do sistema Delphi é chamada de Biblioteca de Componentes Visuais (VCL), não obstante ela inclua mais do que componentes.

No âmago do Delphi encontra-se uma hierarquia de classes. Desde que cada classe do sistema é uma subclasse da classe TObject, toda a hierarquia tem uma única raiz. Isto permite que você use TObject como substituto para qualquer tipo de dados do sistema. As RTTI (informações de tipo em tempo de execução) obtidas com os operadores `is` e `as` fornecem o tipo de dados de um objeto quando necessário.

### 7.1 A Hierarquia da VCL

Use o Object Browser (menu View / Browser) para visualizar a hierarquia da VCL; se este comando de menu estiver desabilitado, compile um projeto primeiro.

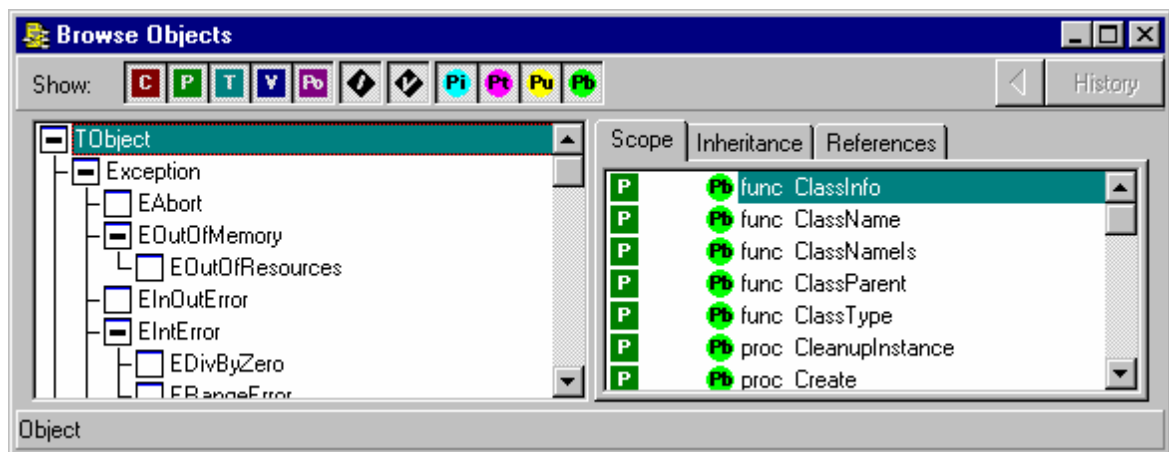


Figura 7-A

Object Browser no topo da hierarquia da VCL.

#### 7.1.1 Componentes

Componentes são os elementos centrais dos Aplicativos Delphi. Eles podem ser visuais ou não-visuais. Os não-visuais aparecem no formulário como um pequeno ícone, mas em tempo de execução alguns deles gerenciam recursos visuais, como uma caixa de diálogo.

Controles podem ser definidos como componentes visuais. Você pode ver como um controle se apresenta num formulário tanto em tempo de projeto como em tempo de execução.

Controle ajanelados (windowed control) são controles baseados em janela, ou seja, que possuem um manipulador de janela. Controles desta subclasse podem receber o foco ou conter outros controles. Exemplos: classes TEdit, Tbutton, TForm.

Controles gráficos ou controles não ajanelados (nonwindowed control) Ao contrário dos controles ajanelados, estes controles não possuem um manipulador de janela, e não podem receber o foco nem conter outros controles. Exemplos: classes TLabel, TSpeedButton.

## 7.1.2 Objetos

Embora a VCL seja basicamente uma coleção de componentes, existem outras classes que não se enquadram nesta categoria. Todas as classes não-componentes freqüentemente são indicadas pelo termo *objetos*, embora esta não seja uma definição precisa.

Há dois usos principais para estas classes. Geralmente elas definem os tipos de dados das propriedades dos componentes, por exemplo, a propriedade `Picture` de um componente de imagem, a qual é da classe `Tgraphic`. O segundo uso de classes não-componentes é a utilização direta na programação, como armazenar o valor de uma propriedade na memória e modificá-la enquanto ela não se relaciona a nenhum componente.

Há diversos grupos de classes de não-componentes na VCL:

- **Objetos gráficos:** `TBitmap`, `TBrush`, `Tcanvas`, `TFont`, `TGraphic`, `TGraphics-Object`, `TIcon`, `TMetaFile`, `TPen` e `Tpicture`.
- **Objetos de fluxo / arquivo:** `TBlobStream`, `TFileStream`, `THandleStream`, `TIniFile`, `TMemoryStream`, `Tfiler`, `TReader` e `Twriter`.
- **Coleções:** `TList`, `TStrings`, `TStringList` e `TCollection`.

## 7.2 Usando Componentes e Objetos

Existem basicamente duas maneiras de criar um objeto no Delphi. Você pode definir uma nova instância adicionando um componente a um formulário, ou pode criar um objeto dinamicamente. No segundo caso, é preciso declarar um objeto, alocar sua instância (chamando o construtor `create`) e definir sua propriedade `parent`. A partir deste ponto, você pode agir sobre o objeto exatamente como se ele fosse definido em tempo de projeto.

Código para criar um botão dinamicamente:

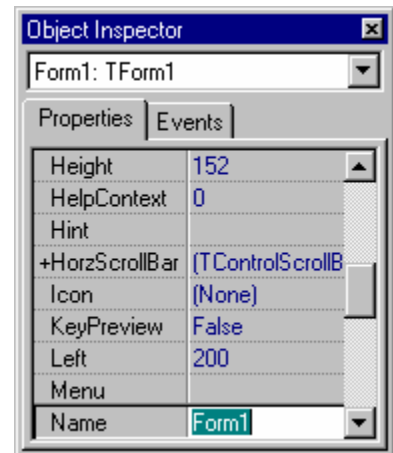
```
var MeuBotao: TButton;  
  
begin  
  MeuBotao := TButton.create (self);           //cria instância  
  MeuBotao.left := 100;                       //define a posição  
  MeuBotao.top := 50;  
  MeuBotao.parent := self;                   //exibe o botão no formulário  
end;
```

## 7.3 Propriedades

Propriedades são atributos de classes que determinam o status de um objeto, como sua posição e sua aparência, e também seu comportamento. A diferença de uma propriedade para um campo de dados é que, ao ler ou alterar o valor de uma propriedade um método pode ser chamado, embora algumas propriedades mapeiem diretamente a campos de dados, particularmente quando você lê o seu valor.

Para saber o valor de uma propriedade em tempo de projeto, ou para mudá-la, você pode usar o Object Inspector. Em tempo de execução, você pode acessar uma propriedade lendo-a ou escrevendo-a com algum código.

Existem propriedades de tempo de projeto, declaradas na seção `published`, e propriedades de tempo de execução, declaradas na seção `public`. O Object Inspector lista somente as primeiras.



**Figura 7-B**

O Object Inspector acessa as propriedades dos componente em tempo de projeto.

Usualmente você pode atribuir ou ler um valor de uma propriedade, e até mesmo usá-la em expressões, mas nem sempre pode passar uma propriedade como parâmetro para uma rotina. Isto acontece porque uma propriedade não é uma localização de memória; deste modo, ela não pode ser usada como um parâmetro passado por referência.

Nem todas as classes VCL têm propriedades. Elas estão presentes nos componentes e em todas as demais subclasses da classe `TPersistent`, porque as propriedades usualmente podem se escoadas (streamed) e gravadas num arquivo.

A palavra-chave `property` inicia a declaração de uma propriedade; as diretivas `read` e `write` especificam uma rotina ou campo de leitura e escrita, respectivamente.

```
type
  TAnObject = class(TObject)
    property Color: TColor //valor do tipo TColor
      read GetColor;      //leitura chama GetColor
      write SetColor;    //escrita chama SetColor
  end;
```

A omissão da diretiva `write` torna uma propriedade somente-leitura (comum). De forma equivalente, a omissão da diretiva `read` a torna somente-escrita (raro).

```
type
  TAnObject = class(TObject)
    property AProperty: TypeX //propriedade somente-leitura
      read GetAnObject;
  end;
```

### 7.3.1 Propriedades Mais Comuns

A tabela abaixo lista algumas das propriedades comuns à maioria dos componentes e é seguida de observações sobre algumas propriedades.

<b>Propriedade</b>	<b>Disponível para</b>	<b>Descrição</b>
Align	Todos os controles	Determina como o controle é alinhado dentro do componente-pai.
BoundsRect	Todos os controles	Indica o retângulo demarcador do controle.
Caption	Todos os controles	A legenda do controle.
ComponentCount	Todos os componentes	O número de componentes possuídos pelo atual.
ComponentIndex	Todos os componentes	Indica a posição do componente na lista de componentes do proprietário.
Components	Todos os componentes	Um vetor dos componentes possuídos pelo atual.
ControlCount	Todos os controles	O número de controles que são filhos do atual.
Controls	Todos os controles	Um vetor dos controles que são filhos do atual.
Color	Muitos Objetos e componentes	Indica a cor da superfície, do fundo, ou a cor atual.
Ctrl3D	A maioria dos Componentes	Determina se o controle tem uma aparência tridimensional.
Cursor	Todos os controles	O cursor usado quando o ponteiro do mouse está sobre o controle.
DragCursor	A maioria dos controles	O cursor usado para indicar que o controle aceita ser arrastado.
DragMode	A maioria dos controles	Determina o comportamento “arrastar-e-soltar” do controle como componente inicial para a operação de arrastar.
Enabled	Todos os controles e alguns outros componentes	Determina se o controle está ativo ou inativo.
Font	Todos os controles	Determina a fonte do texto exibido dentro do componente.
Handle	Todos os controles	O manipulador da janela usado pelo sistema.
Height	Todos os controles e alguns outros componentes	O tamanho vertical do controle.
HelpContext	Todos os controles e os componente de caixa de diálogo	Um número contextual usado para chamar a ajuda contextual automaticamente.
Hint	Todos os controles	A string usada para exibir dicas instantâneas sobre o controle.
Left	Todos os controles	A coordenada horizontal do canto superior esquerdo do componente.
Name	Todos os componentes	O Nome único do componente, o qual pode ser usado no código-fonte.
Owner	Todos os componentes	Indica o componente proprietário.
Parent	Todos os controles	Indica o controle de origem (pai).
ParentColor	Muitos objetos e componentes	Determina se o componente deve usar sua própria propriedade Color ou a do componente-pai.
Parent3D	Maioria dos componentes	Determina se o componente deve usar sua própria propriedade Ctrl3D ou a do componente-pai.
ParentFont	Todos os controles	Determina se o componente deve usar sua própria propriedade Font ou a do componente-pai.
ParentShowHint	Todos os controles	Determina se o componente deve usar sua própria propriedade ShowHint ou a do componente-pai.
PopupMenu	Todos os controles	Indica o menu suspenso a ser usado quando o usuário der

		um clique sobre o controle com o botão esquerdo.
ShowHint	Todos os controles	Determina se as dicas estão ativadas.
Showing	Todos os controles	Determina se o controle está visível quando seu controle-pai está visível.
TabOrder	Todos os controles (exceto TForm)	Determina a ordem de tabulação do controle.
TabStop	Todos os controles (exceto TForm)	Determina se o usuário pode tabular para o controle.
Tag	Todos os componentes	Um long integer disponível para armazenar dados personalizados.
Top	Todos os controles	A coordenada vertical do canto superior esquerdo do componente.
Visible	Todos os controles e alguns outros componentes	Determina se o controle é visível.
Width	Todos os controles e alguns outros componentes	O tamanho horizontal do controle.

**Tabela 7.3-A**

#### A Propriedade Name

- O nome deve ser único dentro do componente proprietário, geralmente um formulário.
- Se o nome e a legenda de um controle forem idênticos, uma mudança do nome também mudará a legenda.
- O Delphi usa o nome do componente para criar os nomes dos métodos relacionados a seus eventos. Se posteriormente você mudar o nome do componente, o Delphi modificará os nomes dos métodos relacionados de acordo.

#### Propriedades Relacionadas ao Tamanho e à Posição

- A posição de um componente sempre se relaciona à área cliente de seu componente-pai. Para um formulário, a área cliente é a superfície incluída em suas fronteiras, mas sem as bordas.

#### As Propriedades Enabled, Visible e Showing

- Quando a propriedade Enabled de um componente é definida como falsa, o usuário fica impedido de interagir com o componente, embora ele continue visível.
- A propriedade Showing de um componente pode ser falsa enquanto sua propriedade Visible é verdadeira. A primeira indica que o componente está atualmente visível e a segunda indica que o componente estará visível quando seu contêiner (componente-pai) também estiver.

#### A Propriedade Tag

- Esta propriedade assemelha-se mais a um campo de dados já que não exerce nenhum efeito sobre o componente quando é alterada.
- Usando typecasting na propriedade Tag, você pode armazenar um ponteiro, um objeto ou qualquer dado que tenha quatro bytes de largura. Isto permite que se associe virtualmente qualquer coisa a um componente.

## 7.4 Métodos de Componentes

Os métodos de componentes são exatamente iguais a quaisquer outros métodos  
A tabela abaixo lista alguns métodos disponíveis para a maioria dos componentes.

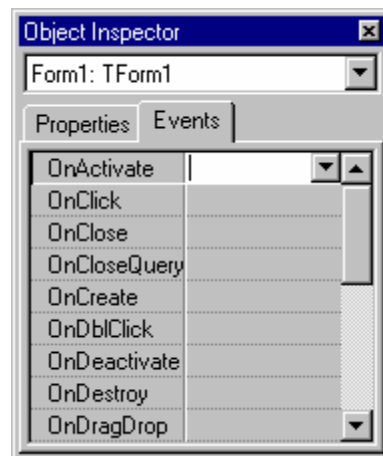
<b>Método</b>	<b>Disponível para</b>	<b>Descrição</b>
BeginDrag	Todos os controles	Inicia o arrasto manual.
BringToFront	Todos os controles	Coloca o componente na frente de todos os outros.
CanFocus	Todos os controles	Verifica se o controle pode receber o foco.
ClientToScreen	Todos os controles	Traduz coordenadas do cliente para coordenadas globais da tela.
ContainsControl	Todos os controles ajanelados	Verifica se certo controle é contido pelo atual.
Create	Todos os objetos e componentes	Cria uma instância.
Destroy	Todos os objetos e componentes	Destrói uma instância (ver método Free).
Dragging	Todos os controles	Indica se o controle está sendo arrastado.
EndDrag	Todos os controles	Termina manualmente o arrasto.
FindComponent	Todos os componentes	Retorna o componente da propriedade matricial Components que tiver o nome solicitado.
Focused	Todos os controles ajanelados	Verifica se o controle tem o foco.
Free	Todos os objetos e componentes (não sugerido para formulários)	Apaga uma instância se ela for não-nula.
GetTextBuf	Todos os controles	Copia o texto ou a legenda do controle para uma string terminada em nulo.
GetTextLen	Todos os controles	Retorna a extensão do texto ou da legenda do controle.
HandleAllocated	Todos os controles	Retorna True se o manipulador de janela do controle existir.
HandleNedded	Todos os controles	Cria um manipulador de janela para o controle se ele não existir ainda.
Hide	Todos os controles	Torna o controle invisível.
InsertComponent	Todos os componentes	Adiciona um elemento à lista de componentes possuídos pelo atual.
InsertControl	Todos os controles	Adiciona um elemento à lista de controles filhos do atual.
Invalidate	Todos os controles	Força um redesenho do controle.
RemoveComponent	Todos os componentes	Remove um elemento da lista de componentes possuídos pelo atual.
ScaleBy	Todos os controles	Gradua o tamanho controle em determinada porcentagem do seu tamanho anterior.
ScreenToClient	Todos os controles	Traduz coordenadas globais da tela para coordenadas do cliente.
ScrollBy	Todos os controles	Rola o conteúdo do controle.
SendToBack	Todos os controles	Coloca o componente atrás de todos os outros.
SetBounds	Todos os controles	Muda a posição e tamanho do controle de uma só vez.
SetFocus	Todos os controles	Coloca o foco de entrada no controle.
SetTextBuf	Todos os controles	Define o texto ou legenda do controle.
Show	Todos os controles	Torne o controle visível.
Update	Todos os controles	Redesenha imediatamente o controle, mas apenas se o redesenho tiver sido solicitado (invalidate).

**Tabela 7.4-A**

## 7.5 Eventos de Componentes

Os eventos podem ser gerados tanto pelas ações do usuário, um clique sobre um componente por exemplo, quanto pelo sistema, que pode responder a métodos e a mudanças de propriedades.

É possível alterar um manipulador de eventos em tempo de projeto e de execução, desde que o novo método seja compatível com os parâmetro do evento. Para que um método esteja disponível em tempo de projeto no Object Inspector ele deve ser declarado na seção `published` da classe do formulário.



**Figura 7-C**

Eventos do formulário Form1 na página Events do Object Inspector

Outro recurso utilizado é o compartilhamento de manipuladores de eventos por componentes. A tabela abaixo lista alguns eventos disponíveis para a maioria dos componentes.

Evento	Disponível para	Descrição
OnChange	Muitos objetos e componentes	Ocorre quando o objeto (ou seu conteúdo) muda.
OnClick	Muitos Controles	Ocorre quando se dá um clique como botão primário do mouse sobre o componente.
OnDblClick	Muitos Controles	Ocorre quando o usuário dá um duplo-clique com o botão primário do mouse sobre o componente.
OnDragDrop	Muitos Controles	Ocorre quando uma operação de arrastar termina sobre um componente.
OnDragOver	Muitos Controles	Ocorre quando um usuário está arrastando sobre um componente.
OnEndDrag	Muitos Controles	Ocorre quando a operação de arrastar é encerrada.
OnEnter	Todos os controles ajanelados	Ocorre quando o componente fica ativo, ou seja, quando ele recebe o foco.
OnExit	Todos os controles ajanelados	Ocorre quando o componente perde o foco.
OnKeyDown	Muitos Controles	Ocorre quando qualquer tecla é pressionada e o componente tem o foco.
OnKeyPress	Muitos Controles	Ocorre quando uma tecla ou seqüência de teclas é pressionada e o componente tem o foco.
OnKeyUp	Muitos Controles	Ocorre quando qualquer tecla é liberada e o componente tem o foco.
OnMouseDown	Muitos Controles	Ocorre quando o usuário pressiona um dos botões do mouse sobre um componente.
OnMouseMove	Muitos Controles	Ocorre quando o usuário move o mouse sobre um componente.

<b>Evento</b>	<b>Disponível para</b>	<b>Descrição</b>
OnChange	Muitos objetos e componentes	Ocorre quando o objeto (ou seu conteúdo) muda.
OnClick	Muitos Controles	Ocorre quando se dá um clique como botão primário do mouse sobre o componente.
OnDblClick	Muitos Controles	Ocorre quando o usuário dá um duplo-clique com o botão primário do mouse sobre o componente.
OnDragDrop	Muitos Controles	Ocorre quando uma operação de arrastar termina sobre um componente.
OnDragOver	Muitos Controles	Ocorre quando um usuário está arrastando sobre um componente.
OnEndDrag	Muitos Controles	Ocorre quando a operação de arrastar é encerrada.
OnEnter	Todos os controles ajanelados	Ocorre quando o componente fica ativo, ou seja, quando ele recebe o foco.
OnExit	Todos os controles ajanelados	Ocorre quando o componente perde o foco.
OnKeyDown	Muitos Controles	Ocorre quando qualquer tecla é pressionada e o componente tem o foco.
OnKeyPress	Muitos Controles	Ocorre quando uma tecla ou seqüência de teclas é pressionada e o componente tem o foco.
OnKeyUp	Muitos Controles	Ocorre quando qualquer tecla é liberada e o componente tem o foco.
OnMouseDown	Muitos Controles	Ocorre quando o usuário pressiona um dos botões do mouse sobre um componente.
OnMouseUp	Muitos Controles	Ocorre quando o usuário solta um dos botões do mouse que havia sido pressionado sobre um componente.
OnStartDrag	Muitos Controles	Ocorre quando um componente inicia uma operação de arrasto.

**Tabela 7.5-A**

Conforme o quadro acima, os componente distinguem entre 3 “tipos de cliques”:

- O evento OnClick envolve pressionar e soltar o botão primário do mouse sobre um componente.
- O evento OnMouseDown é disparado apenas com o pressionar de qualquer botão do mouse sobre um componente.
- O evento OnMouseUp é disparado quando qualquer botão do mouse é liberado sobre um componente.

Estes dois últimos eventos incluem parâmetro para indicar qual botão foi pressionado, se alguma tecla de shift (Alt, Control e Shift) estava pressionada simultaneamente e a posição (X e Y) em que se deu o clique.

O evento OnClick costuma se adequar aos casos mais comuns, em que não são importantes os detalhes fornecidos pelos outros dois eventos.

- Os eventos OnDragOver e OnDragDrop são utilizados em conjunto por um componente capaz de receber um outro componente arrastado: o código do primeiro evento verifica se o componente sendo arrastado pode ser solto e ativa ou desativa o parâmetro `Accept`; o segundo contém a ação a ser tomada quando o componente é solto, e só ocorre se o parâmetro `Accept` estiver ativo.

## 7.6 Usando Coleções Delphi

Entre os vários objetos, ou seja, as classes de não componentes, um grupo importante são as coleções. Basicamente, há quatro diferentes classes de coleções:

- TList define uma lista de objetos de alguma classe.
- TStrings é uma classe abstrata para representar todas as formas de listas de strings, independente de quais sejam suas implementações de armazenagem. Esta classe define uma lista de strings sem oferecer a adequada armazenagem. Por esta razão, objetos TStrings são usados somente como propriedades de componentes capazes de armazenar as próprias strings, como uma caixa de listagem.
- TStringList define uma lista de strings com sua própria armazenagem. Você pode usar esta classe para definir suas próprias listas de strings num programa.
- TCollection define uma lista homogênea de objetos, que pertence à classe de coleções. Os objetos devem ser persistentes (derivados de TPersistent), porque a coleção inteira pode ser direcionada (gravada em arquivo).

Você pode operar com listas usando a notação de matrizes ( [ e ] ), tanto para ler quanto para mudar elementos.



## Parte II - Usando Componentes

Esta pode ser considerada a parte mais importante para desenvolvedores de aplicações, já que a programação visual usando componentes é a característica fundamental do Delphi.

O objetivo agora é mostrar como usar algumas das características oferecidas pelos componentes pré-definidos na criação de aplicações.

### 8 Uma Viagem Pelos Componentes Básicos

Aqui detalharemos algumas propriedades, métodos e eventos já citados anteriormente e sua utilização em situações específicas para cada componente.

Mas antes algumas informações sobre os componentes:

- Se, quando um componente for selecionado na palheta, a tecla shift estiver pressionada, uma borda surgirá em volta dele. Ela indica que você pode criar vários componentes do tipo selecionado, de uma só vez. Ao terminar, desmarque o componente selecionando a seta do mouse na palheta.
- Podemos criar teclas de atalho ao sublinhar uma letra da legenda do componente, precedendo-a com o caracter “&”. Assim, o usuário poderá acionar um controle ou menu através da combinação Alt + Tecla.
- A menos que nunca se cogite em referenciar um componente no código da aplicação, ele deve receber um nome. Um modelo que pode ser adotado é juntar o tipo do componente ao seu nome. Por exemplo, um botão que chama a tela de ajuda poderia se chamar AjudaButton.
- A ordem com que os componentes são acessados no formulário ao se teclar <Tab> (ordem de tabulação) pode ser alterada através do speedmenu Tab Order do formulário. Além disso, cada componente pode ser excluído da ordem de tabulação desativando a propriedade TabStop.
- No Windows 95, os controles de edição possuem por padrão um speedmenu que, no Delphi, podem ser personalizados.
- Se a propriedade ShowHint estiver ativada, o texto da propriedade Hint será exibido como dica quando o usuário posicionar o mouse sobre o componente.
- A propriedade HelpContext contém o número de uma tela do arquivo de help (.hlp) da aplicação (se houver um arquivo de help), para chamá-la quando o usuário teclar <F1> dentro do componente. Isto permite a criação de um sistema de help sensível ao contexto. Se o valor desta propriedade for zero para o componente, será usado o HelpContext do pai do componente.

#### 8.1 Seta do Mouse



Usada simplesmente para desmarcar a seleção de algum componente na palheta.

Figura 8-A

#### 8.2 Botão (Button)



Os Botões são utilizados para permitir que o usuário dispare determinadas ações ao clicá-los.

Figura 8-B

#### Propriedades

1. Defina a legenda (Caption) pois é ela que indica primariamente ao usuário a função do botão.
2. Se o evento OnClick deve ser chamado quando se pressionar a tecla <Esc>, ative a propriedade Cancel.
3. Se o evento OnClick deve ser chamado quando se pressionar a tecla <Enter> fora de um botão, ative a propriedade Default.
4. Escreva o código do evento OnClick.

### Observações

1. Um Botão utilizado para fechar um formulário modal não precisa ter código associado ao seu evento OnClick; se a propriedade ModalResult for diferente de zero (mrNone), quando for clicado, o botão definirá o valor da propriedade ModalResult do formulário para o mesmo valor da sua. Isto faz com que ele seja fechado e a função ShowModal que o chamou retorne o valor ModalResult definido.

## 8.3 Rótulo (Label)



Figura 8-C

Os Rótulos são apenas textos ou comentários, normalmente estáticos em tempo de execução. Costumam ser usados para descrever outros componentes, particularmente as caixas de texto e as caixas combinadas ou de listagem, porque elas não têm títulos.

Os Rótulos são componentes mais “leves”. Portanto, devemos empregá-los preferencialmente em situações onde não é necessário alterar o texto em tempo de execução.

### Propriedades

1. Align permite fixar a posição do Rótulo dentro do formulário.
2. Alignment permite fixar a posição do texto dentro do Rótulo.
3. Se quiser que o tamanho do rótulo se altere automaticamente de forma a envolver a legenda, ative a propriedade AutoSize.
4. Defina a legenda (Caption).
5. Se você quiser que a tecla de atalho do rótulo coloque o foco em algum componente, selecione o seu nome na propriedade FocusControl.
6. Defina o tipo, estilo, tamanho e cor da fonte a ser usada na legenda (Font).
7. Se quiser que a legenda se enquadre no espaço do rótulo ative a propriedade WordWrap.

## 8.4 Caixa de Diálogo Color (DialogColor)



Este componente permite ao usuário escolher uma das cores da palheta apresentada.

Figura 8-D

### Propriedades

1. Defina a cor inicial da palheta.
2. Ative a propriedade Option.cdFullOpen para exibir automaticamente a área de definição de cores personalizadas.
3. Ative a propriedade Option.cdPreventFullOpen para impedir que o usuário defina cores personalizadas.

4. Ative a propriedade `Option.ShowHelp` para que esteja disponível um botão de ajuda.
5. Ative a propriedade `Option.SolidColor` para impedir a disponibilidade de cores mistas.

### Observações

1. Chame a caixa de diálogo com o método `Execute`.
2. Quando a caixa de diálogo for fechada, a cor escolhida estará disponível na propriedade `Color` da `FontDialog`.

## 8.5 Caixa de Edição (*Edit*)



As Caixas de Edição são usadas para que o usuário veja ou altere uma única linha de texto.

**Figura 8-E**

### Propriedades

1. Ative a propriedade `AutoSelect` para que o texto seja automaticamente selecionado quando o foco passar para o componente.
2. `CharCase` força a caixa da letra para minúscula, normal ou maiúscula.
3. `Enabled` controla a resposta do componente a eventos de mouse e do teclado.
4. Defina um valor diferente de zero para a propriedade `MaxLength` se quiser limitar o número de caracteres que podem ser digitados.
5. Ative a propriedade `OEMConvert` se para converter os caracteres ANSI para OEM, o que deve ser feito quando o texto consistir de nomes de arquivos.
6. Defina a propriedade `PasswordChar` para um caractere diferente de nulo (`#0`) se quiser usar o componente para digitar dados secretos.
7. `ReadOnly` controla a permissão de alteração do conteúdo do componente.
8. Defina a propriedade `Text` para o valor inicial a ser exibido.

### Observações

1. A propriedade de tempo de execução `Modified` indica se o componente sofreu alguma alteração.

## 8.6 Caixa de Edição com Máscara



A Caixa de Edição com Máscara é semelhante à Caixa de Edição comum; só que ela nos permite formatar a entrada com o recurso das máscaras, de forma que o usuário é obrigado a entrar com dados válidos.

**Figura 8-F**

### Propriedades

1. Ative a propriedade `AutoSelect` para que o texto seja automaticamente selecionado quando o foco passar para o componente.
2. `CharCase` força a caixa da letra para minúscula, normal ou maiúscula.

3. Defina a máscara de edição através da propriedade EditMask. O Delphi oferece uma série de máscaras comumente utilizadas, mas se for necessário, você pode criar a sua seguindo as normas descritas no arquivo de ajuda on-line.
4. Enabled controla a resposta do componente a eventos de mouse e do teclado.
5. Defina um valor diferente de zero para a propriedade MaxLength se quiser limitar o número de caracteres que podem ser digitados.
6. Defina a propriedade PasswordChar para um caractere diferente de nulo (#0) se quiser usar o componente para digitar dados secretos.
7. ReadOnly controla a permissão de alteração do conteúdo do componente.
8. Defina a propriedade Text para o valor inicial a ser exibido.

### Observações

1. A propriedade de tempo de execução Modified indica se o componente sofreu alguma alteração.
2. A propriedade Text de uma MaskEdit se refere aos caracteres “reais” presentes na MaskEdit. Já a propriedade MaskEdit, se refere a todos os caracteres presente na MaskEdit, incluindo os caracteres da máscara.

## 8.7 Memo



Figura 8-G

O componente Memo é semelhante a uma Caixa de Edição. No entanto, pode exibir diversas linhas e conter até 32 K de texto.

### Propriedades

1. Align permite fixar a posição do Memo dentro do formulário.
2. Alignment permite fixar a posição do texto dentro do Memo.
3. Enabled controla a resposta do componente a eventos de mouse e do teclado.
4. Ative a propriedade HideScrollBars para esconder as barras de rolagem automaticamente quando elas não forem necessárias.
5. Lines é semelhante à propriedade Text dos componentes Edit. Com ela nós podemos definir os dados de cada linha do Memo separadamente.
6. Defina um valor diferente de zero para a propriedade MaxLength se quiser limitar o número de caracteres que podem ser digitados.
7. ReadOnly controla a permissão de alteração do conteúdo do componente.
8. Defina se o Memo deve conter barras de rolagem vertical e horizontal com a propriedade ScrollBars.
9. Ative a propriedade WantReturns se você quiser que, ao pressionar a tecla <Enter> no Memo, ocorra uma quebra de linha.
10. Ative a propriedade WantTabs se você quiser que, ao pressionar a tecla <Tab> no Memo, uma tabulação seja inserida. Desative se quiser que a tecla <Tab> mude o foco para o próximo componente.
11. Ative a propriedade WordWrap para obrigar o texto a se acomodar no espaço horizontal disponível, quebrando a linha quando necessário.

### Observações

1. A propriedade de tempo de execução Modified indica se o componente sofreu alguma alteração.
2. Se a propriedade WantReturns estiver desativada e o usuário teclar <Enter> no Memo, este caractere será enviado ao formulário. Para quebrar a linha, use a seqüência <Ctrl> + <Enter>.
3. Quando propriedade WantTabs for verdadeira, o usuário não será capaz de sair do Memo com a tecla <Tab>.
4. A propriedade de tempo de execução Text dá acesso a todo o texto do componente, em oposição ao acesso linha a linha da propriedade Lines.

## 8.8 Caixa de Diálogo Font (FontDialog)



Com este componente o usuário pode escolher uma fonte e definir seus atributos.

**Figura 8-H**

### Propriedades

1. Escolha na propriedade Device os dispositivos que devem ter a fonte afetada pela configuração da FontDialog.
2. Defina os tamanhos de fonte mínimo e máximo que o usuário pode escolher através das propriedades MaxFontSize e MinFontSize.
3. Ative a propriedade Option.fdAnsiOnly para disponibilizar somente as fontes com conjunto de caracteres do Windows.
4. Ative a propriedade Option.fdTrueTypeOnly para listar apenas as fontes TrueType.
5. Ative a propriedade Option.fdEffects para disponibilizar os efeitos de cores, riscado e sublinhado.
6. Ative a propriedade Option.fdFixedPitchOnly para disponibilizar somente as fontes mono-espaçadas.
7. Ative a propriedade Option.fdForceFontExist para impedir o usuário de escolher uma fonte inexistente.
8. Ative a propriedade Option.fdNoFaceSel para que nenhuma fonte esteja selecionada quando a caixa de diálogo for aberta.
9. Ative a propriedade Option.fdNoOEMFont para impedir a listagem de fontes vetoriais.
10. Ative a propriedade Option.fdNoSimulations para impedir a listagem de fontes GDI simuladas.
11. Ative a propriedade Option.fdNoSizeSel para que nenhum tamanho de fonte esteja selecionado quando a caixa de diálogo for aberta.
12. Ative a propriedade Option.fdNoStyleSel para que nenhum estilo esteja selecionado quando a caixa de diálogo for aberta.
13. Ative a propriedade Option.fdNoVectorFonts para obter o mesmo efeito da propriedade Option.fdNoOEMFont.
14. Ative a propriedade Option.fdShowHelp para disponibilizar um botão de ajuda na caixa de diálogo.
15. Ative a propriedade Option.fdWysiwyg para listar apenas as fontes disponíveis tanto para tela quanto para impressão.
16. Ative a propriedade Option.fdLimitSize para que as propriedades MaxFontSize e MinFontSize tenham efeito.
17. Ative a propriedade Option.fdScalableOnly para listar apenas as fontes escaláveis.
18. Ative a propriedade Option.fdApplyButton para disponibilizar o botão Apply.

### Observações

1. Chame a caixa de diálogo com o método `Execute`.
2. Quando a caixa de diálogo for fechada, a fonte escolhida estará disponível na propriedade `Font` da `FontDialog`.

## 8.9 RichEdit



Figura 8-1

Este componente é um descendente do componente `Memo`, capaz de manipular textos RTF (Rich Text Format).

### Propriedades

1. `Align` permite fixar a posição do `RichEdit` dentro do formulário.
2. `Alignment` permite fixar a posição do texto dentro do `RichEdit`.
3. A propriedade de tempo de execução `DefAttributes` contém a descrição do texto default do `RichEdit`. A propriedade de tempo de execução `SelAttributes` contém a descrição do texto selecionado.
4. `Enabled` controla a resposta do componente a eventos de mouse e do teclado.
5. Ative a propriedade `HideScrollBars` para esconder as barras de rolagem automaticamente quando elas não forem necessárias.
6. `Lines` é semelhante à propriedade `Text` dos componentes `Edit`. Com ela nós podemos definir os dados de cada linha do `RichEdit` separadamente.
7. Defina uma valor diferente de zero para a propriedade `MaxLength` se quiser limitar o número de caracteres que podem ser digitados.
8. A propriedade de tempo de execução `Paragraph` contém sub-propriedades que controlam o alinhamento, indentação, numeração e tabulação (`Alignment`, `FirstIndent`, `LeftIndent`, `Numbering`, `RightIndent`, `Tab array`, and `TabCount`) dos parágrafos selecionados.
9. `PlainText` controla se só o `RichEdit` trata seu texto como RTF ou como texto não-formatado (TXT).
10. A propriedade de tempo de execução `Print` imprime o conteúdo do `RichEdit`.
11. `ReadOnly` controla a permissão de alteração do conteúdo do componente.
12. Defina se o `RichEdit` deve conter barras de rolagem vertical e horizontal com a propriedade `ScrollBars`.
13. Ative a propriedade `WantReturns` se você quiser que, ao pressionar a tecla `<Enter>` no `Memo`, ocorra uma quebra de linha.
14. Ative a propriedade `WantTabs` se você quiser que, ao pressionar a tecla `<Tab>` no `Memo`, uma tabulação seja inserida. Desative se quiser que a tecla `<Tab>` mude o foco para o próximo componente.
15. Ative a propriedade `WordWrap` para obrigar o texto a se acomodar no espaço horizontal disponível, quebrando a linha quando necessário.

### Observações

1. A propriedade de tempo de execução `Modified` indica se o componente sofreu alguma alteração.
2. Se a propriedade `WantReturns` estiver desativada e o usuário teclar `<Enter>` no `RichEdit`, este caractere será enviado ao formulário. Para quebrar a linha, use a seqüência `<Ctrl> + <Enter>`.
3. Quando propriedade `WantTabs` for verdadeira, o usuário não será capaz de sair do `RichEdit` com a tecla `<Tab>`.
4. A propriedade de tempo de execução `Text` dá acesso a todo o texto do componente, em oposição ao acesso linha a linha da propriedade `Lines`.

5. As propriedades `DefAttributes` e `SelAttributes` são do tipo `TTextAttributes` e não `TFont`. Para permutarmos valores entre essas propriedade e a propriedade `Font` de outro componente, podemos usar o método `Assign` da propriedade receptora.

## 8.10 Caixa de Verificação (*CheckBox*)



Figura 8-J

A Caixa de Verificação representa uma opção que pode ser marcada ou desmarcada pelo usuário.

### Propriedades

1. `Align` permite fixar a posição da legenda em relação à Caixa de Verificação.
2. Se a propriedade `AllowGreyed` estiver desativada, Caixa de Verificação terá dois estados possíveis: marcada ou desmarcada. Se estiver ativa, haverá um terceiro estado intermediário: acinzentado.
3. `Caption` é a legenda do componente.
4. `Enabled` controla a resposta do componente a eventos de mouse e do teclado.
5. Defina o estado inicial da Caixa de Verificação com a propriedade `State`.

## 8.11 Botão de Rádio (*RadioButton*)



Figura 8-K

O Botões de Rádio representam opções mutuamente exclusivas que podem ser marcadas pelo usuário. Para delimitar os Botões de Rádio interrelacionados, costuma-se agrupá-los em componentes contêineres, como Caixas de Grupos ou Painéis.

### Propriedades

1. Crie um componente contêiner, no qual os Botões de Rádio serão agrupados. Eles também podem ser colocados diretamente no formulário.
2. Insira os Botões de Rádio dentro do componente contêiner.
3. `Align` permite fixar a posição da legenda em relação o Botão de Rádio.
4. `Caption` é a legenda do componente.
5. Defina o estado inicial do Botão de Rádio com a propriedade `Checked`.
6. `Enabled` controla a resposta do componente a eventos de mouse e do teclado.

### Observações

1. O Delphi possui um componente específico, chamado `RadioGroup`, que simula uma Caixa de Grupo preenchida com Botões de Rádio.
2. Outros componentes que possibilitam a seleção de opções são as Caixas de Listagem e as Caixa Combinada.

## 8.12 Caixa de Listagem (*ListBox*)



Figura 8-L

Como o nome diz, uma Caixa de Listagem possui uma lista capaz de armazenar até 32 K elementos, sendo que você pode mostrar apenas uma parte

limitada do conteúdo. Além disso, você pode facilmente inserir ou remover elementos em tempo de execução.

### **Propriedades**

1. Align permite fixar a posição da Caixa de Listagem em relação ao formulário.
2. Columns especifica o número de colunas de elementos no componente.
3. Enabled controla a resposta do componente a eventos de mouse e do teclado.
4. Ative a propriedade ExtendedSelect para permitir que o usuário selecione vários elementos com o auxílio das teclas <Control> e <Shift>. Esta propriedade só tem efeito quando a propriedade MultiSelect está ativa.
5. Ative a propriedade IntegralHeight para que a Caixa de Listagem adapte sua altura de forma a exibir um número inteiro de elementos (sem deixar um deles semi-oculto). Esta propriedade só tem efeito quando a propriedade Style não possui o valor lbOwnerDrawVariable.
6. ItemHeight determina a altura dos itens do componente. Esta propriedade só tem efeito quando a propriedade Style não possui o valor lbStandard.
7. Items lista as strings que devem aparecer na caixa de listagem.
8. Ative a propriedade MultiSelect para permitir que o usuário selecione vários itens, ou desative para permitir que ele selecione só um item (ver propriedade ExtendedSelect).
9. Ative a propriedade Sorted para que os itens sejam colocados em ordem alfabética automaticamente.
10. Style determina se os elementos da lista são apenas strings de altura fixa (lbStandard), ou se podem ser elemento gráficos de altura fixa (lbOwnerDrawFixed) ou elemento gráficos de variável (lbOwnerDrawVariable).

### **Observações**

1. A propriedade de tempo de execução ItemIndex indica qual item possui o foco.
2. Você pode atualizar uma Caixa de Listagem usando os métodos Add, Insert e Delete da propriedade Items.
3. A propriedade de tempo de execução Selected indica se um item está selecionado.
4. A propriedade de tempo de execução SelCount retorna o número de itens atualmente selecionado.
5. Outros componentes que possibilitam a seleção de opções são os Botões de Rádio e as Caixa Combinada.

## **8.13 Caixa Combinada (ComboBox)**

Este componente pode ser considerado a combinação de dois outros: uma Caixa de Edição e uma Caixa de Listagem. Normalmente, somente a parte de edição fica visível e o usuário deve clicar o botão à direita do componente para abrir a listagem (drop-down list) abaixo do componente.



**Figura 8-M**

### **Propriedades**

1. DropDownCount determina o número máximo de itens que serão exibidos na lista drop-down.
2. Enabled controla a resposta do componente a eventos de mouse e do teclado.
3. ItemHeight determina a altura dos itens da listagem. Esta propriedade só tem efeito quando a propriedade Style não possui os valores csOwnerDrawFixed e csOwnerDrawVariable.

4. Items lista as strings que devem aparecer na listagem.
5. Defina um valor diferente de zero para a propriedade MaxLength se quiser limitar o número de caracteres que podem ser digitados.
6. Ative a propriedade Sorted para que os itens da listagem sejam colocados em ordem alfabética automaticamente.
7. Style determina um dos seguintes estilos de Caixa Combinada:
  - csDropDown: Caixa de Edição e lista drop-down, com cada item sendo uma string e tendo a mesma altura.
  - csDropDownList: Lista drop-down sem Caixa de Edição, com cada item sendo uma string e tendo a mesma altura.
  - csOwnerDrawFixed: Lista drop-down sem Caixa de Edição, com cada item sendo um gráfico e tendo a altura definida pela propriedade ItemHeight.
  - csOwnerDrawVariable: Lista drop-down sem Caixa de Edição, com cada item sendo um gráfico e tendo a altura variável.
  - csSimple: Caixa de Edição e lista drop-down permanente, com cada item sendo uma string e tendo a mesma altura.

#### **Observações**

1. A propriedade de tempo de execução Text indica o texto da Caixa de Edição.
2. A propriedade de tempo de execução ItemIndex indica qual item da lista está selecionado.
3. Você pode atualizar a lista usando os métodos Add, Insert e Delete da propriedade Items.
4. Outros componentes que possibilitam a seleção de opções são os Botões de Rádio e as Caixa de Listagem.

## **8.14 Barra de Rolagem (ScrollBar)**



Este componente permite a seleção de um conjunto de valores numéricos através do deslocamento de um retângulo (caixa de rolagem) dentro de uma região de rolagem.

**Figura 8-N**

#### **Propriedades**

1. Enabled controla a resposta do componente a eventos de mouse e do teclado.
2. Kind indica se a Barra de Rolagem será Horizontal (sbHorizontal) ou Vertical (sbVertical).
3. LargeChange indica quantas posições a caixa de rolagem será deslocada quando o usuário clicar dentro da região de rolagem ou teclar <Page Down> / <Page Up>.
4. Max indica o valor máximo assumido pela caixa de rolagem.
5. Min indica o valor mínimo assumido pela caixa de rolagem.
6. Position indica a posição inicial da caixa de rolagem.
7. SmallChange indica quantas posições a caixa de rolagem será deslocada quando o usuário clicar as setas nas extremidades do componente pressionar as setas do teclado.

#### **Observações**

1. O método `SetParams` permite a alteração das propriedades `Min`, `Max` e `Position` simultaneamente.

## 9 Criando e Manipulando Menus

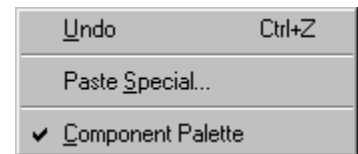
### 9.1 A Estrutura de um Menu

Os menus são caracterizados pelo número de níveis, isto é, de acordo com a quantidade de menus suspensos aninhados. Uma forma de diminuir o número de níveis é separar as opções afins com barras verticais, ao invés de colocá-las em submenus.

#### 9.1.1 Diferentes Papeis dos Itens do Menu

Há três tipos fundamentais de itens de menu, independentemente da posição que eles ocupam no menu:

1. **Comandos**: usados para executar um comando; não oferecem pista visual.
2. **Definidores de estado**: usados para alternar entre opções; usualmente possuem uma marca de verificação à esquerda do texto quando estão ativos.
3. **Itens de diálogo**: fazem com que uma caixa de diálogo apareça; são caracterizados pela presença de reticências após o texto.



**Figura 9-A**

Tipos de itens de menu.

### 9.2 Menu Principal (MainMenu)

O componente Menu Principal encapsula uma barra de menu com os respectivos menus drop-down.

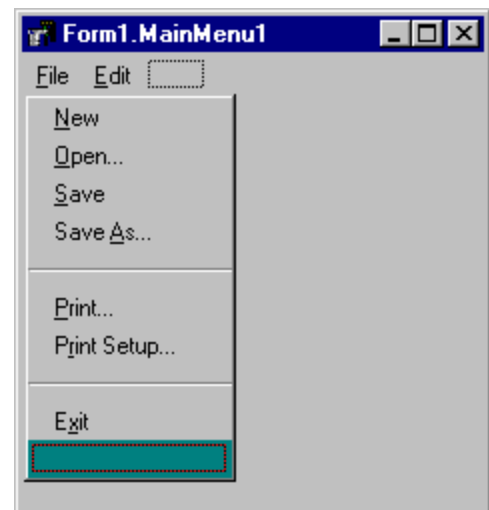


**Figura 9-B**

#### 9.2.1 Editando um Menu com o Menu Designer

O Delphi incluir um editor de menus, o Menu Designer. Para utilizar esta ferramenta, coloque um componente MainMenu no formulário e dê um clique-duplo sobre ele.

Para criar um item de menu, clique sobre o retângulo tracejado do Menu Designer e edite as propriedades exibidas no Object Inspector:



**Figura 9-C**

Edição de um menu através do Menu Editor.

1. A propriedade Break permite que você quebre o menu drop-down em dois, e separe cada parte por uma barra vertical (mbBarBreak) ou não (mbBreak).
2. A propriedade Caption é a legenda do item de menu. Um caractere “&” sublinha a letra que o segue, e um sinal de menos “-” sozinho cria uma barra horizontal usada para separar grupos de itens.

3. A propriedade Checked indica o estado inicial de um item de menu definidor de estado.
4. Ative a propriedade Default para que o item de menu apareça em negrito e seja chamado se o usuário der um duplo-clique no menu ao qual ele pertence.
5. A propriedade Enabled controla a resposta do componente a eventos de mouse e do teclado.
6. A propriedade GroupIndex indica quais itens formam um grupo; é usada também em aplicativos MDI, quando se quer que o menu de uma janela-filha se junte ou substitua o menu da janela-pai.
7. Ative a propriedade RadioItem para que o item de menu funcione com um grupo de Botões de Rádio (opções exclusivas) em relação aos itens de mesmo GroupIndex.
8. Para atribuir teclas de atalho para um item de menu, selecione uma das seqüências de teclas da propriedade ShortCut.

### 9.2.2 Observações

1. A propriedade Items pode ser usada para acessar um item de menu usando sua posição como índice. Para saber quantos itens um menu contém, use a propriedade Count.
2. Se você definir apenas a propriedade Caption do componente, ele será nomeado segundo a convenção abaixo:
  - Qualquer caracter especial ou em branco (inclusive o E-comercial (&) e o hífen (-) é removido.
  - Se não houver nenhum caracter à esquerda, uma letra é adicionada (N).
  - Um número é adicionado ao final do nome (1 se este for o primeiro item de menu com este nome; um número mais elevado, caso contrário).

Você pode criar um menu a partir de um modelo ou criar seu próprio modelo de menu através dos comandos Insert From Template e Save as Template do speedmenu do Menu Designer.

### 9.2.3 Respondendo a Comando de Menu

Para responder aos comandos de menu você precisa definir um método para o evento OnClick para cada item de menu. O evento OnClick de menus suspensos é usado somente em casos especiais - por exemplo, para verificar se os itens do menu devem ser desativados. O evento OnClick de um separador é inútil.

### 9.2.4 Mudando Menus em Tempo de Execução

Quatro propriedades comumente são usadas para alterar modificar um item de menu em tempo de execução. Dentre as propriedades abaixo, apenas Checked não se aplica a menus suspensos.

1. Caption: Alterar o texto do item de menu para indicar um status de programa diferente ao usuário.
2. Checked: Adiciona ou remove uma marca do lado do item de menu.
3. Enabled: Acinzentado um item de menu e impede que ele possa ser selecionado pelo usuário.
4. Visible: "Remove" um item de menu. Normalmente é preferível desativar um item de menu a torná-lo invisível, pois o usuário pode perder tempo procurando pelo item em outros menus.

Outra forma de modificar menus em tempo de execução é ter vários componentes MainMenu, cada um com uma configuração de barra de menu diferente. Assim, pode-se trocar a barra de menu de um formulário definindo a sua propriedade Menu para o MainMenu desejado.

```
procedure TForm1.MenusCurtosClick (Sender: TObject);
begin
```

```
Form1.Menu := MainMenu3;  
end;
```

### 9.3 Menu Pop-Up (PopupMenu)

No Windows 95 é comum ver aplicações que possuem menus locais especiais<sup>4</sup> que você ativa dando um clique com o botão secundário do mouse. Estes menus tendem a ser mais fáceis de usar, uma vez que agrupam somente os comandos relacionados ao elemento que foi clicado. Eles também são mais rápidos pois não é necessário mover o mouse para a barra de menus e depois voltar.

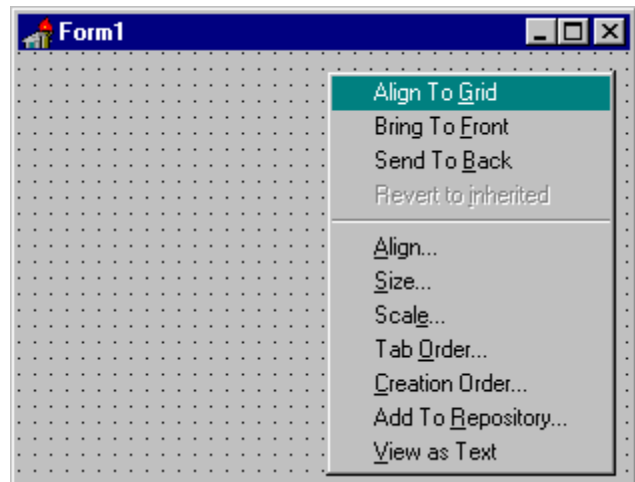


Figura 9-D

Menu pop-up do formulário Form1.

#### 9.3.1 Criando Menus Pop-Up

Menus pop-up são semelhantes aos menus principais, com exceção do fato de um menu pop-up ser um único menu, e não um conjunto de menus. Assim, o que foi apresentado até aqui pode ser aplicado também para menus pop-up.



Figura 9-E

1. Inicie colocando um componente PopupMenu no formulário, e dando um clique-duplo sobre ele para chamar o Menu Designer.
2. A propriedade Alignment determina o alinhamento do menu pop-up em relação ao ponteiro do mouse.
3. A propriedade AutoPopup determina se o menu pop-up deverá ser exibido automaticamente quando o usuário clicar o componente que o contém com o botão secundário do mouse.
4. Adicione alguns itens de menu ao menu pop-up.

Você pode deixar que o Delphi manipule os menus pop-up automaticamente (padrão) ou pode escolher uma técnica manual.

#### 9.3.2 Manipulando Menus Pop-Up Automaticamente

Nesta abordagem, o menu pop-up é exibido sempre que o usuário clica o componente com o botão secundário do mouse.

Para definir um menu pop-up para um componente:

1. Ative a propriedade AutoPopup do menu pop-up.
2. Selecione o menu pop-up na propriedade PopupMenu de algum componente da sua aplicação.

---

<sup>4</sup> Conhecidos genericamente por menus pop-up no Windows. No Delphi são designados de speedmenus.

### 9.3.3 Manipulando Menus Pop-Up Manualmente

Nesta abordagem, o menu pop-up é exibido de acordo com o critério adotado pelo desenvolvedor, e não apenas quando o usuário clica o componente com o botão secundário do mouse.

Para exibir um menu pop-up quando ocorrer um evento com um componente:

1. Desative a propriedade `AutoPopup` do menu pop-up (se o menu estiver definido como `PopupMenu` do componente, o que é desnecessário).
2. No evento desejado do componente, exiba o menu pop-up com o método `Popup`. Este método requer como parâmetros as coordenadas `x` e `y` **de tela** onde o menu será exibido. A função `ClientToScreen` pode ser usada para converter um ponto em coordenadas cliente<sup>5</sup> para coordenadas de tela.

Este código exibe o menu pop-up `PopupMenu1` quando o usuário clica o formulário `Form1` com o botão principal do mouse:

```
procedure TForm1.FormMouseDown(Sender: TObject;
  Button: TMouseButton; Shift: TShiftState; X, Y: Integer);

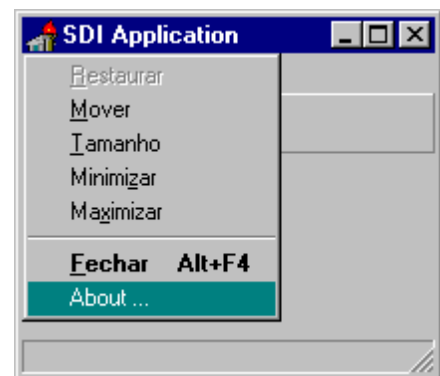
var
  PontoCliente, PontoTela: TPoint;

begin
  if (Button = mbLeft) then
  begin
    PontoCliente.X := X;
    PontoCliente.Y := Y;
    PontoTela := ClientToScreen (PontoCliente);
    PopupMenu1.Popup (PontoTela.X, PontoTela.Y);
  end;
end;
```

### 9.4 Alterando o Menu de Sistema

O menu de sistema, que está presente no canto esquerdo superior da maioria das janelas, pode ter seus itens alterados, bem como a resposta desses itens.

Para alterar os itens de menu do menu de sistema, usamos as funções da API: `GetSystemMenu` para obter o manipulador do menu de sistema, `AppendMenu` para adicionar um item de menu e `DeleteMenu` para remover um item de menu.



**Figura 9-F**

Menu de sistema com a adição do item “About”.

Este código adiciona o item ‘About ...’ ao menu de sistema.

```
const
  IdSysAbout = 100;          //Identificador do item de menu
  ...
begin
  AppendMenu (GetSystemMenu (Handle, FALSE), MF_STRING,
```

---

<sup>5</sup> Coordenadas da janela.

```
    IdSysAbout, 'About ...');  
end;
```

Este código remove o primeiro item do menu de sistema.

```
begin  
    DeleteMenu (GetSystemMenu (Handle, FALSE), 0, MF_BYPOSITION);  
end;
```

A definição ou alteração da resposta de um item do menu de sistema envolve a definição de um manipulador de mensagens `wm_SysCommand`, que são geradas quando um item do menu de sistema é selecionado.

Este código define um manipulador de mensagens `wm_SysCommand`.

```
Type  
TForm1 = class (TForm)  
    ...  
public  
    procedure WMSysCommand (var Msg: Tmessage)  
        message wm_SysCommand;  
    ...  
procedure TForm1.WMSysCommand (var Msg: TMessage);  
begin  
    if Msg.Param = IdSysAbout then  
        ShowMessage ('About selecionado.');
```

inherited;  
end;

## 10 De Volta ao Formulário

Neste capítulo examinaremos algumas das propriedades e estilos de formulários. Também dedicaremos algum tempo à entrada e saída em um formulário.

### 10.1 Formulários Versus Janelas

No Windows, a maioria dos elementos de interface com o usuário é formada por janelas. Para tornar as coisas mais claras, analisemos as seguintes definições de janela:

- **Do ponto de vista do usuário**, uma janela é uma área da tela contornada por uma borda, usualmente, tendo uma legenda. Elas podem ser divididas em duas categorias: janelas principais e caixas de diálogo.
- **Tecnicamente falando**<sup>6</sup>, uma janela é uma entrada numa área de memória do sistema Windows, freqüentemente correspondendo a um elemento visível na tela, que tenha algum código associado. O sistema atribui a cada janela um número único chamado de manipulador (handler). Algumas dessas janelas são percebidas pelo usuário conforme a primeira definição de janela, outras têm o papel de controles e outras permanecem ocultas.

O Windows reserva uma porção da memória para gerenciar as janelas existentes no sistema. Como essa memória é limitada, é aconselhável não exagerar na quantidade de janelas (tecnicamente falando) presentes no aplicativo, utilizando, sempre que possível, componente não-ajanelados, como os rótulos.

Com estes conceitos em mente, podemos definir um formulário como uma janela do ponto de vista do usuário. Logo, eles podem ser usados para criar janelas principais, janelas MDI, caixas de diálogo, etc. Os outros componentes ajanelados podem ser definidos como janelas somente de acordo com a definição técnica.

### 10.2 Janelas Sobrepostas, Pop-Up e Filhas

Cada janela que você cria tem um dos três estilos gerais que determinam seu comportamento:

1. **Sobreposto**: Janelas comuns.
2. **Pop-up**: Semelhantes às janelas sobrepostas. Normalmente usadas para caixas de diálogo e caixas de mensagens.
3. **Filho**: Janelas que não podem se mover para fora da área cliente da janela-pai.

Qualquer janela pode ter uma janela-proprietário, que troca mensagens continuamente com a janela possuída, por exemplo, quando ela é reduzida a um ícone. Usualmente a janela-pai é também a proprietária.

### 10.3 O Aplicativo é uma Janela

Os formulários estão conectados a um proprietário: a janela do aplicativo, que permanece oculta a visão, exceto quando o aplicativo está minimizado.

---

<sup>6</sup> O Delphi possui a ferramenta WinSight que lista todas as janelas do sistema.

O papel do objeto Application é fornecer algum código de inicialização antes de criar qualquer formulário, como pode ser observado no arquivo do projeto, com o comando View / Project Source. Já a janela relacionada a este objeto serve para manter o vínculo de todas as janelas da aplicação.

## 10.4 Definindo Estilos e Comportamentos de Formulários

Entre as propriedades de um formulário, duas delas determinam as regras fundamentais de seu comportamento: FormStyle e BorderStyle.

### 10.4.1 O Estilo do Formulário

Os valores de FormStyle são:

1. fsMDIChild: O formulário é uma janela MDI filha.
2. fsMDIForm: O formulário é uma janela MDI pai, ou seja, janela da moldura do aplicativo MDI.
3. fsNormal (padrão): O formulário é uma janela SDI normal ou uma caixa de diálogo.
4. fsStayOnTop: O formulário é uma janela SDI sempre visível. Ela sempre permanece sobre todas as demais janelas da aplicação, exceto as que tenham este mesmo estilo. Se o formulário principal for sempre visível, ele também ficará sobre as janelas das outras aplicações.

### 10.4.2 O Estilo da Borda

A propriedade BorderStyle se refere a um elemento visual do formulário, mas tem uma influência mais profunda no seu comportamento.

Os valores de BorderStyle são:

1. bsDialog: O formulário possui uma borda espessa<sup>7</sup> que o usuário não pode arrastar, como a de uma caixa de diálogo. Além da borda, o formulário ganha outros atributos de um caixa de diálogo: o menu de sistema é diferente e a propriedade BorderIcons é ignorada.
2. bsNone: O formulário não possui borda, legenda, botões de maximizar, minimizar e menu de sistema.
3. bsSingle (padrão): Também conhecida como borda fixa O formulário possui uma borda fina que não pode ser reajustada.
4. bsSizeable: O formulário possui uma borda espessa que o usuário pode arrastar.
5. bsSizeToolWin: O formulário possui uma legenda fina e somente uma miniatura do botão fechar. Este é um estilo especial para caixas de ferramentas, com bordas reajustáveis.
6. bsToolWindow: O formulário possui uma legenda fina e somente uma miniatura do botão fechar. Este é um estilo especial para caixas de ferramentas, com bordas não reajustáveis.

## 10.5 Os Ícones da Borda

Você pode definir várias opções utilizando a propriedade BorderIcons, um conjunto com até quatro valores:

1. biSystemMenu: Disponibiliza o menu de sistema.

---

<sup>7</sup> No Windows 95 não há uma diferença clara entre uma borda espessa e uma borda fina.

2. biMinimize: Disponibiliza o botão de minimizar.
3. biMaximize: Disponibiliza o botão de maximizar.
4. biHelp: Disponibiliza o botão de ajuda.

Alguns destes elementos estão diretamente relacionados a outros: Se você remover o menu de sistema, todos os ícones da borda desaparecerão; se retirar ou o botão de maximizar ou o de minimizar, ele ficará desativado, mas se retirá-los simultaneamente, eles desaparecerão; quando um botão é retirado, a opção correspondente do menu de sistema é desativada.

## 10.6 Configurando Mais Estilos de Janelas

As propriedades `BorderStyle` e `FormStyle` correspondem principalmente a diferentes configurações no estilo e estilo estendido de uma janela, que por sua vez refletem dois parâmetros da função da API `CreateWindowsEx` que o Delphi utiliza para criar formulários. Você pode modificar estes dois parâmetros livremente sobrepondo o método virtual `CreateParams`.

Declaração do método `CreateParams`:

```
type
  TForm1 = class (TForm)
    ...
  public
    procedure CreateParams (var Params: TCreateParams); override;
  end;
```

Adiciona o flag de estilo de janela transparente:

```
procedure TForm1.CreateParams (var Params: TCreateParams);
begin
  inherited CreateParams (Params);
  Params.ExStyle := Params.ExStyle or WS_EX_TRANSPARENT;
end;
```

Esta é a única forma de utilizar alguns estilos peculiares de janelas que não estão disponíveis através das propriedades do formulário. Mais informações podem ser encontradas no help da API sob o tópico “`CreateWindow`” e “`CreateWindowEx`”.

## 10.7 Formulários em Diferentes Resoluções de Tela

Quando um formulário é criado em uma dada resolução de tela e é usado em outra resolução menor, ele pode ficar maior que a tela ou ter controles seus desarrumados.

O reajuste do tamanho dos componentes, denominado *gradação*, existe para contornar este problema. É importante notar que: se você quiser que apenas os componentes do formulário sejam graduados, a propriedade `AutoScroll` deve ser ativada. Para forçar a gradação do formulário também, desative-a; a gradação apropriada das legendas e textos dos componentes exige a utilização de fontes `TrueType`.

### 10.7.1 Graduação Manual do Formulário

Todas as vezes que quiser graduar um componente ou um formulário, incluindo seus componentes, você poderá usar o método `ScaleBy`, o qual tem dois parâmetros inteiros, um multiplicador e um divisor - uma fração. A instrução

```
ScaleBy (3, 4);
```

reduz o formulário a três quartos do seu tamanho original.

A multiplicação por números fracionários tende a gerar coordenadas e tamanhos reais que, ao serem truncados para valores inteiros, podem causar pequenas diferenças de graduação. Este problema se agrava quando o componente é redimensionado diversas vezes, caso em que é preferível utilizar como base o formulário com as medidas originais.

### 10.7.2 Graduação Automática do Formulário

Você pode deixar que o Delphi faça o reajuste do formulário quando o aplicativo iniciar ativando a propriedade `Scaled` em tempo de projeto.

## 10.8 Definindo a Posição e o Tamanho do Formulário

O comportamento padrão de um formulário é que ele apareça na tela do mesmo modo como foi projetado. As propriedades a seguir alteram este comportamento:

### 10.8.1 Propriedade `Position`

Indica a posição e o tamanho do formulário quando ele é criado:

1. `poDesigned`: O formulário aparece com a mesma posição e o mesmo tamanho com que foi projetado.
2. `poDefault`: O Windows determina a posição e o tamanho do formulário usando um layout em cascata.
3. `poDefaultPosOnly`: O formulário usa o tamanho que você determinou em tempo de projeto, mas o Windows escolhe sua posição na tela.
4. `poDesigned`: O formulário usa a posição que você determinou em tempo de projeto, mas o Windows escolhe seu tamanho.
5. `poDesigned`: O formulário aparece no centro da tela, com o tamanho que você definiu em tempo de projeto.

### 10.8.2 Propriedade `WindowState`

Indica o estado do formulário quando ele é criado:

1. `wsMaximized`: O formulário aparece maximizado.
2. `wsMinimized`: O formulário aparece como um ícone.
3. `wsMinimized`: O formulário de acordo com a propriedade `Position`.

O estado de um formulário em tempo de execução pode ser alterado definindo-se esta propriedade. Porém, quando se tratar do formulário principal da aplicação, é mais adequado utilizar os métodos `Maximize`, `Minimize` e `Restore` do objeto `Application`. Um dos motivos é que ao minimizarmos uma janela da primeira forma, uma janela minimizada é criada na área de trabalho e, da segunda, o aplicativo só aparece na barra de tarefas do Windows.

## 10.9 O Tamanho de um Formulário e sua Área Cliente

A área cliente de um formulário é a sua parte interna, excluindo as bordas, a legenda e a barra de menus. Há quatro propriedades relacionadas ao tamanho de um formulário:

1. Height: é a altura do formulário completo.
2. Width: é a largura do formulário completo.
3. ClientHeight: é a altura da área cliente do formulário.
4. ClientWidth: é a largura da área cliente do formulário.

Os valores das propriedades Height e ClientHeight, bem como os valores de Width e ClientWidth são mutuamente dependentes, bastando alterar uma de cada.

## 10.10 O Tamanho Máximo e Mínimo de um Formulário

Quando você escolhe uma borda redimensionável para um formulário, os usuários geralmente podem redimensioná-lo de acordo com o que desejarem, e também maximizá-lo em uma tela cheia. Todavia, freqüentemente é útil definir um limite para o tamanho de um formulário, particularmente o tamanho mínimo. Para fazer isso, devemos manipular a mensagem do Windows `wm_GetMinMaxInfo`:

```
type
  TForm1 = class(TForm)
    ...
  public
    procedure GetMinMax (var MinMaxMessage : TWMGetMinMaxInfo);
      message wm_GetMinMaxInfo;
    end;
```

O parâmetro `MinMaxMessage` é uma estrutura que contém o campo `MinMaxInfo`. Este campo é um ponteiro para a estrutura a seguir:

- ptReserved é um campo não documentado, reservado para uso interno do Windows.
- ptMaxSize especifica, nos campos x e y, a largura e a altura da janela quando maximizada.
- ptMaxPosition especifica a posição da janela quando maximizada.
- ptMinTrackSize especifica, nos campos x e y, a largura e a altura mínimas da janela em estado normal.
- ptMaxTrackSize especifica, nos campos x e y, a largura e a altura máximas da janela em estado normal.

No método manipulador da mensagem, nós limitamos o redimensionamento definimos somente os valores dos campos em que estivermos interessados, já que eles já vêm preenchidos com valores-padrão:

```
procedure TForm1.GetMinMax (var MinMaxMessage : TWMGetMinMaxInfo);
begin
  with MinMaxMessage.MinMaxInfo^ do
    begin
      ptMaxPosition.x := 200;
      ptMaxPosition.y := 100;
```

```

    ptMinTrackSize.x := 200;
    ptMinTrackSize.y := 100;
    ptMaxTrackSize.x := 400;
    ptMaxTrackSize.y := 300;
end;
end;

```

## 10.11 Criação Automática dos Formulários

A criação automática dos formulários é feita no arquivo do projeto (.dpr) pelo método `CreateForm` do aplicativo. Este método cria uma nova instância da classe passada como o primeiro parâmetro e a referencia pela variável passada como segundo parâmetro.

```

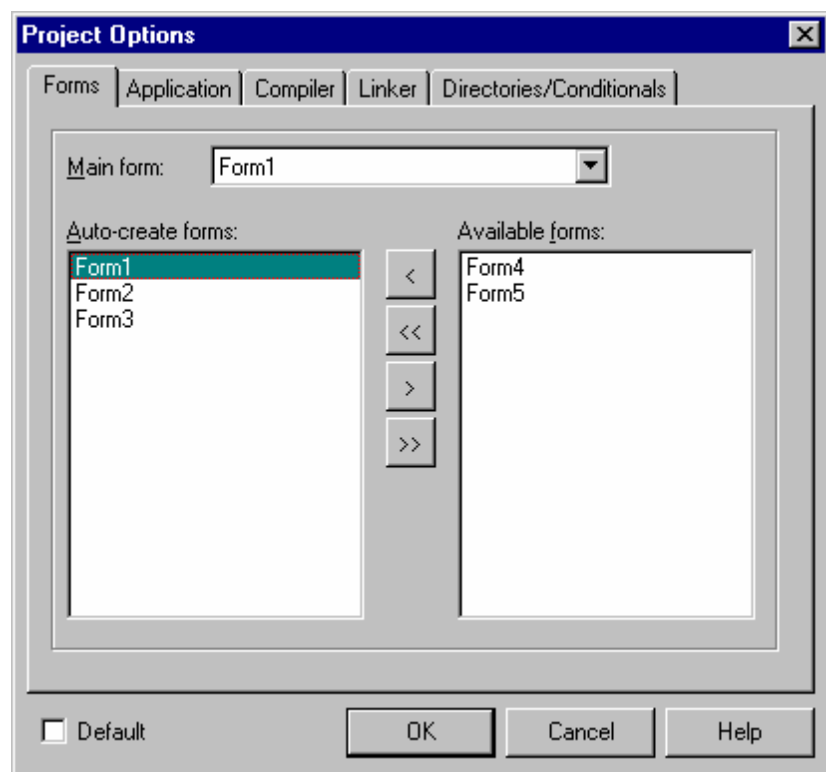
begin
    Application.Initialize;
    Application.CreateForm(TForm1, Form1);
    Application.CreateForm(TForm2, Form2);
    Application.CreateForm(TForm3, Form3);
    Application.Run;
end.

```

Para pular a criação automática de um formulário, você pode remover a linha correspondente à sua criação, ou usar a página Forms da caixa de diálogo Project / Options.

O primeiro formulário criado é atribuído à propriedade `MainForm` do aplicativo. Por esta razão, o formulário que você escolher como principal será deslocado para o primeiro lugar na lista.

Uma das características do formulário principal é que o seu fechamento determina o término do aplicativo.



**Figura 10-A**

Os formulário criado automaticamente são o Form1 o Form2 e o Form3, sendo que o Form1 será o principal por ser o primeiro a ser criado.

## 10.12 `zFechando um Formulário

Quando você fecha o formulário, o Evento `OnCloseQuery` é chamado. Nele você pode pedir que o usuário confirme o fechamento manipulando o parâmetro `CanClose`.

```
procedure TForm1.FormCloseQuery(Sender: TObject;
  var CanClose: Boolean);
begin
  if MessageDlg (Quer mesmo sair?', mtConfirmation,
    [mbYes, mbNo], 0) = idNo) then
    CanClose := False;
  end;
end.
```

Se `CanClose` indicar que o formulário deve ser fechado, o evento `OnClose` será chamado. O parâmetro `Action` deste evento é que vai determinar o que acontecerá com o formulário. Podemos atribuir os seguintes valores a `Action`:

- caNone: Não é permitido fechar o formulário.
- caHide (padrão): O formulário não é fechado, apenas ocultado.
- caFree: O formulário é fechado liberando sua memória. Se este for o formulário principal, o programa é encerrado.
- caMinimize: O formulário não é fechado, apenas minimizado.

## 10.13 Recebendo Entrada do Mouse

Quando o usuário pressiona um dos botões do mouse sobre um formulário, o Windows envia algumas mensagens ao aplicativo. O Delphi define alguns eventos para escrever código em resposta a estas mensagens:

- OnMouseDown é recebido quando um dos botões do mouse é pressionado.
- OnMouseUp é recebido quando um dos botões do mouse é liberado.
- OnMouseMove é recebido quando um mouse é movido sobre o formulário.
- OnClick é recebido quando o botão primário do mouse é pressionado e liberado sobre o formulário.

### 10.13.1 O Papel dos Botões do Mouse

O botão primário do mouse é usado para selecionar e mover elementos na tela, acionar comandos de menu e pressionar botões. O botão secundário é usado para exibir menus pop-up.

Apesar de o usuário poder inverter os papéis dos botões, é comum se referir ao botão primário como o botão esquerdo do mouse, e ao botão secundário como o botão direito.

O botão do meio, por estar presente apenas em alguns modelos de mouse, raramente é usado. Um clique neste botão pode simular um duplo-clique no botão primário.

### 10.13.2 Usando o Windows sem um Mouse

Um usuário sempre deve ter condições de usar qualquer aplicativo Windows sem o mouse, não só porque ele pode não ter um mouse conectado ao seu computador, como para permitir que os comandos possam ser executados mais rapidamente através do teclado.

Por esta razão, você sempre deve configurar uma ordem de tabulação apropriada para os componentes de um formulário e adicionar teclas de atalho.

## 10.14 Desenhando no Formulário

Antes de mais nada, é preciso alertar que o comportamento padrão do Windows é de não armazenar o bitmap resultante de um desenho pois isso requer uma quantidade razoável de memória. Por isso, quando você cobre os desenhos de uma janela com outra janela eles se perdem. Salvar o desenho em um bitmap exige o uso dos componentes apropriados.

Para desenhar no formulário, usamos a propriedade Canvas, que é uma tela de pintura<sup>8</sup>, e tem duas características: ela mantém uma coleção de ferramentas de desenho (caneta, pincel e fonte), e possui um grande número de métodos de desenho (Rectangle, Ellipse, ...) que usam as ferramentas atuais.

### 10.14.1 As Ferramentas de Desenho

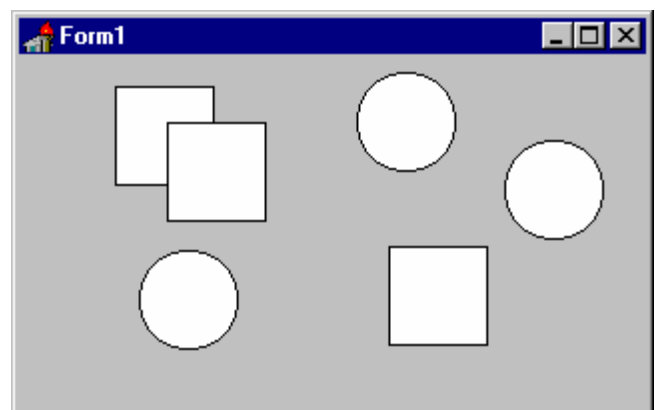
Eis uma lista das ferramentas de desenho (ou ferramentas GDI, de Graphics Device Interface) de uma tela de pintura:

- **Brush:** determina a cor das superfícies envolvidas. O pincel é usado para preencher formas fechadas, tais como círculos ou retângulos. As propriedades de um pincel são sua cor, estilo e bitmap.
- **Pen:** determina a cor e o tamanho das linhas e bordas das formas. As propriedades de uma caneta são sua cor, tamanho se ela for uma linha sólida, ou outros estilos, inclusive pontilhado e tracejado.
- **Font:** determina a fonte usada para escrever texto no formulário usando o método `TextOut` da tela de pintura. Uma fonte tem um nome, um tamanho, um estilo, uma cor e assim por diante.

Quando o adaptador de vídeo é incapaz de exibir cores RGB de 24 bits o Windows utiliza a técnica de dithering, que simula a cor solicitada, para aplicar a cor de um pincel. A caneta e a fonte, ao contrário, usam somente cores puras, ou seja, cores diretamente disponíveis. Quando a cor solicitada não está disponível, é escolhida a cor pura mais próxima.

### 10.14.2 Desenhando Formas

O exemplo a seguir utiliza o evento `OnMouseDown` para desenhar formas a cada pressionar de botão do mouse sobre o formulário. Um quadrado será desenhado se a tecla `<Shift>` estiver pressionada, e um círculo será desenhado se a tecla `<Alt>` estiver pressionada:



**Figura 10-B**

Formas desenhadas com a propriedade Canvas do formulário Form1.

```
procedure TForm1.FormMouseDown(Sender: TObject;
```

---

<sup>8</sup> Área na qual o aplicativo pode desenhar.

```

    Button: TMouseButton; Shift: TShiftState; X, Y: Integer);
begin
    if ssShift in Shift then           //tecla <Shift> pressionada
        Canvas.Rectangle (X, Y - 50, X + 50, Y)
    else if ssAlt in Shift then       //tecla <Alt> pressionada
        Canvas.Ellipse (X, Y - 50, X + 50, Y)
    end;
end;

```

## 10.15 Desenhando e Pintando no Windows

Existe uma diferença entre desenhar e pintar:

Desenhar é acessar a tela de pintura do formulário e chamar alguns de seus métodos. Desde que a imagem não seja salva ela corre o risco de se perder porque o aplicativo não sabe redesenhá-la.

Pintar é seguir uma abordagem que permita ao aplicativo repintar a sua superfície, recriando a saída que tiver sido destruída.

Pintar é a técnica comum usada para manipular a saída no Windows, bem como em programas particulares orientados a gráficos. A abordagem usada para implementar a pintura é tem um nome muito descritivo: *armazenar e pintar*. De fato, precisamos armazenar as ações do usuário para usar estas informações na hora de repintar os elementos.

Você pode usar diversos métodos para invocar a repintura de componentes. Os dois primeiros correspondem às funções da API do Windows, enquanto os dois últimos forma introduzidos pelo Delphi:

- Invalidate: solicita, mas não impõe uma operação de pintura. O Windows armazena o pedido responde a ele somente quando não houver nenhum outro evento pendente, uma vez que a pintura é uma das operações que mais consomem tempo
- Update: solicita uma atualização imediata ao Windows. Porém esta operação se realizará somente se houver uma área inválida, ou seja, se o método Invalidate tiver sido chamado anteriormente. Caso contrário ela não surtirá qualquer efeito.
- Repaint: chama os métodos Invalidate e Update em seqüência, forçando a pintura.
- Refresh: para formulários, é equivalente ao método Repaint, para componentes pode ser ligeiramente diferente.

Quando precisar pedir uma operação de pintura, geralmente você deve chamar Invalidate, seguindo a abordagem do Windows. Há momentos, porém, em que você quer que o aplicativo repinte a superfície tão rapidamente quanto possível. Nestes casos pouco freqüentes, chamar Repaint é o melhor caminho a ser seguido.

### 10.15.1 Pintando uma Única Forma

Empregaremos agora a técnica armazenar e desenhar. O método `FormMouseDown` será responsável pelo armazenamento das informações sobre a imagem e o método `FormPaint` será responsável pela pintura.

```

type
    TShapesForm = class(TForm)
    ...
    private
        Centro: TPoint;
        Circulo: Boolean;

```

```
end;
```

Quando o usuário pressiona o botão esquerdo do mouse, a variável `Centro` armazena o centro da forma e a variável `Circulo` armazena se a forma é um círculo.

```
procedure TShapesForm.FormMouseDown(Sender: TObject;
  Button: TMouseButton; Shift: TShiftState; X, Y: Integer);
begin
  if Button = mbLeft then
  begin
    {armazena o valor do centro}
    Centro.X := X;
    Centro.Y := Y;
    {armazena o tipo de shape}
    if ssShift in Shift then
      Circulo := False
    else
      Circulo := True;
    {solicita a repintura do formulário}
    Invalidate;
  end;
end;
```

Sempre que o formulário é repintado pelo Windows este procedimento é chamado, garantindo a repintura da forma.

```
procedure TShapesForm.FormPaint(Sender: TObject);
begin
  if Circulo then
    Canvas.Ellipse (Centro.X-10, Centro.Y-10,
      Centro.X+10, Centro.Y+10)
  else
    Canvas.Rectangle (Centro.X-10, Centro.Y-10,
      Centro.X+10, Centro.Y+10);
end;
```

### 10.15.2 Pintando uma Série de Formas

Uma solução simples para armazenar um grande número de formas é usar a classe `TList`.

Este exemplo define um tipo de dados `ShapeData` para armazenar os atributos de cada forma.

```
type
  ShapeDado = class (TObject)
    Circulo: Boolean;
    X, Y, Tamanho, TamanhoPen: Integer;
    CorPen, CorBrush: TColor;
  end;

  TShapesForm = class(TForm)
  ...
  private
    Raio: Integer;
    ShapesLista: Tlist;
  end;
```

Um novo objeto é adicionado à lista cada vez que o usuário cria uma forma.

```

procedure TShapesForm.FormMouseDown(Sender: TObject;
  Button: TMouseButton; Shift: TShiftState; X, Y: Integer);
var
  Shape: ShapeDado;
begin
  if Button = mbLeft then
  begin
    Shape := ShapeDado.Create;
    if (ssShift in Shift) then
      Shape.Circulo := False
    else
      Shape.Circulo := True;
    Shape.X := X;
    Shape.Y := Y;
    Shape.Tamanho := Raio;
    Shape.TamanhoPen := Canvas.Pen.Width;
    Shape.CorPen := Canvas.Pen.Color;
    Shape.CorBrush := Canvas.Brush.Color;
    ShapesLista.Add (Shape);

    Repaint;
  end;
end;

```

O método acima tem o inconveniente de repintar toda a superfície do formulário cada vez que você adiciona uma forma. A função da API do Windows `InvalidateRect` nos permite restringir a área a ser repintada:

```

procedure InvalidateRect (Wnd: Hwnd; Rect: Prect; Erase: Bool);

```

`Wnd` é o manipulador da janela (Handle do formulário); `Rect` é a área (retângulo) a ser repintada; `Erase` indica se você quer apagar a área antes de repintá-la.

Para aplicar esta técnica no método anterior, devemos declarar:

```

var
  InvRect: TRect;

```

e substituir a chamada `Repaint` por:

```

InvRect := Rect (X-Raio-Shape.TamanhoPen, Y-Raio-Shape.TamanhoPen,
  X+Raio+Shape.TamanhoPen, Y+Raio+Shape.TamanhoPen);
InvalidateRect (Handle, @InvRect, False);

```

O Windows pula automaticamente as operações de saída fora do retângulo, evitando o desperdício de tempo e efeitos colaterais como um leve tremor da exibição.

## 11 Componentes Gráficos

Além dos controles básicos tomados emprestados do Windows, o Delphi exibe uma série de componentes dotados de uma interface moderna com o usuário. Alguns deles são versões gráficas dos controles tradicionais, como os botões e caixas de listagem gráficos, outros são definidos pelo próprio Delphi.

Vimos no capítulo anterior que a pintura no Windows requer algum trabalho. Por esta razão, ter controles gráficos prontos para serem usados é uma grande vantagem sobre a abordagem tradicional.

### 11.1 O Botão de Bitmap (*BitBtn*)



Semelhantes ao componente Button, com a capacidade de exibir uma imagem além do texto da legenda. **Figura 11-A**

#### Propriedades

1. Glyph nos permite escolher a imagem que irá aparecer no botão.
2. Kind oferece uma série de tipos básicos de botões. Ao selecionar um valor diferente de `bkCustom`, a imagem, a legenda e o valor de retorno serão automaticamente definidos.
3. Layout indica a posição da imagem em relação à legenda.
4. NumGlyphs indica quantas imagens existem no arquivo especificado pela propriedade Glyph.
5. Spacing determina a distância entre a imagem e a legenda.
6. Style determina se o botão usará um visual estilo Windows 3.x ou estilo Windows 95.

Um conceito importante é o da cor transparente: a cor encontrada no canto inferior esquerdo do bitmap é considerada a cor transparente da imagem. Muitos aplicativos usam bitmaps com um fundo amarelo-escuro (`clOlive`) por ser uma cor raramente usada em outras partes do bitmap.

#### 11.1.1 Muitas Imagens em um Bitmap

O `BitBtn` é capaz de manipular bitmaps com até quatro imagens, dividindo-o em subbitmaps de mesma largura de acordo com a propriedade `NumGlyphs`.

- O primeiro subbitmap é usado quando o botão está liberado.
- O segundo subbitmap é usado quando o botão está desativado.
- O terceiro subbitmap é usado quando o botão está sendo pressionado pelo mouse.
- O quarto subbitmap é usado quando o botão permanece pressionado por se comportar como uma caixa de verificação.

Se você não fornecer todos os subbitmaps, os que faltarem serão computados a partir do primeiro, por meio de algumas alterações gráficas.

## 11.2 O Image



Este componente usualmente é considerado um visualizador de imagens. Ele aceita arquivos de bitmap (BMP), arquivos de ícones (ICO), ou metaarquivos (WMF e EMF).

Figura 11-B

### Propriedades

1. `AutoSize` determina se o controle será redimensionado para acomodar a imagem.
2. `Center` indica se a imagem deve ser centralizada ou deve permanecer do lado esquerdo-superior do controle.
3. `Picture` determina a imagem que o controle deve exibir.
4. `Stretch` indica se a imagem deve se ajustar ao tamanho do controle. Arquivos de ícones não são afetados por esta propriedade.

## 11.3 Desenhando em um Bitmap

Ao invés de utilizar a técnica de pintura para “armazenar” imagens, podemos desenhar em bitmaps presentes na memória, de forma que a imagem possa ser recuperada sempre que necessário.

As vantagens são a rapidez, já que, em vez de ter que repintar a imagem basta, copiá-la da memória para um componente, e a simplicidade do código. As desvantagens são o gasto de memória e uma menor flexibilidade.

O componente `Image` possui um bitmap no qual podemos desenhar, e não precisamos nos preocupar com o redesenho, que é controlado pelo componente.

```
procedure TShapesForm.Image1MouseDown(Sender: TObject;
  Button: TMouseButton; Shift: TShiftState; X, Y: Integer);
begin
  if Button = mbLeft then
    if ssShift in Shift then
      Image1.Canvas.Rectangle (X-10, Y-10, X+10, Y+10)
    else
      Image1.Canvas.Ellipse (X-Raio, Y-Raio, X+Raio, Y+Raio);
end;
```

## 11.4 Outline



O `Outline` permite que você adicione um bitmap antes do texto dos itens e construa uma lista com uma hierarquia gráfica.

Figura 11-C

### Propriedades

1. `ItemSeparator` é a string usada como separadora do texto dos itens no caminho da propriedade `FullPath` de um nó da lista.
2. `Lines` armazena os itens da árvore em tempo de projeto. No `String List Editor`, cada linha representa um item, e os espaços ou tabulações antes do texto indicam o nível do item na hierarquia.
3. `Options` controla as seguintes opções:
  - `ooDrawTreeRoot`: Define a presença de uma raiz na árvore.
  - `ooDrawFocusRect`: Exibe um retângulo de foco ao redor do item selecionado.

- ooStretchBitmaps: Redimensiona os bitmaps-padrão (propriedades PictureLeaf, PictureOpen, PictureClosed, PicturePlus, PictureMinus) de forma a se ajustarem ao tamanho dos textos dos itens.
4. OutlineStyle indica se estrutura do Outline será exibida como apenas texto, texto e imagem, texto e sinais de + e -, etc.
  5. PictureClose e PicturePlus determinam as imagens que serão exibidas quando um item que contém subitens está retraído (não está exibindo seus subitens).
  6. PictureLeaf determina a imagem que será exibida para os item que não contém subitens (folhas da árvore).
  7. PictureMinus e PictureOpen determinam as imagens que serão exibidas quando um item que contém subitens está expandido (está exibindo seus subitens).

### Observações

1. Quando você estiver usando o String List Editor para enumerar os itens da lista e quiser inserir um caractere de tabulação use a combinação de teclas <Ctrl> + <Tab>.
2. A propriedade Lines pode ser preenchida a partir de um arquivo texto, com o comando Load do speedmenu no String List Editor.
3. Em tempo de execução, pode acessar os itens com maior rapidez através da propriedade Items, ao invés da propriedade Lines.

## 11.5 ImageList



Figura 11-D

Este componente é capaz de armazenar uma lista de imagens e tem como uma de suas utilidades a disponibilização dessas imagens para os componentes TreeView e ListView.

### Propriedades

1. AllocBy define o espaço adicional (quantas imagens) que será alocado cada vez que for preciso aumentar a estrutura de armazenamento.
2. Height e Width definem a altura e largura das imagens.

Para adicionar ou excluir as imagens da lista em tempo de projeto, selecione ImageList Editor no speedmenu do componente.

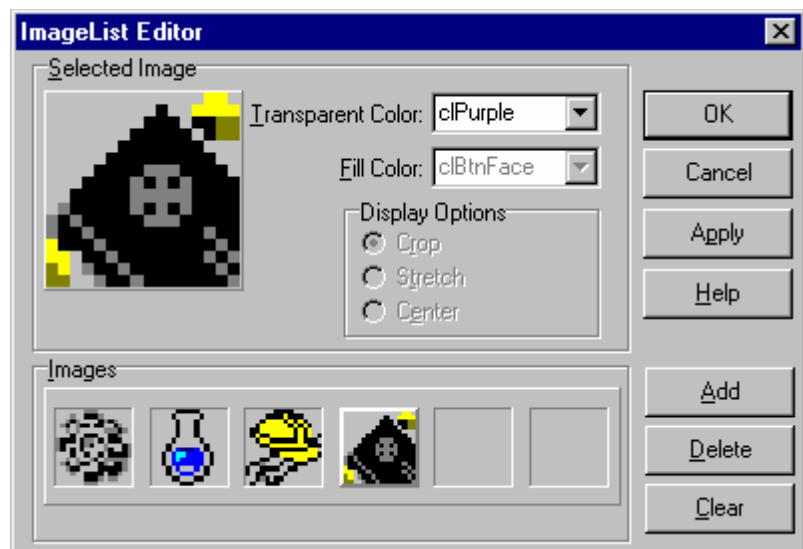


Figura 11-E

Adição de quatro imagens ao ImageList.

## 11.6 TreeView



O TreeView possui uma interface de usuário flexível, com suporte para editar e arrastar elementos, e padronizada, porque é a interface do Windows Explorer.

Figura 11-F

### Propriedades

1. Images indica um componente ImageList que contém uma lista de imagens que podem ser exibidas à esquerda dos rótulos dos itens.
2. Indent indica o tamanho da indentação de um item em relação ao seu pai.
3. Items armazena os itens da árvore em tempo de projeto. No TreeView Items Editor, pode-se acrescentar um item de mesmo nível do atual (New Item), um filho do atual (New Subitem), ou carregar a árvore de itens a partir de um arquivo. Os campos Image Index e Selected Index especificam os índices das imagens de um ImageList (propriedade Images) que devem ser exibidas quando o item for exibido e quando estiver selecionado, respectivamente. O campo State Index é o índice de uma imagem adicional de um ImageList (propriedade StateImages) que pode ser exibida à esquerda do item.
4. ShowButtons determina se os botões de expansão (+) e retração (-) devem ser exibidos à esquerda do item.
5. ShowLines determina se haverá linhas ligando os itens filhos aos seus pais.
6. ShowRoot determina se haverá uma raiz.
7. SortType determina se e como os itens serão ordenados.
8. StateImages indica um componente ImageList que contém uma lista de imagens de estado (propriedade Items) que podem ser exibidas à esquerda dos rótulos dos itens.

## 11.7 ListView



O ListView é outro controle comum do Windows 95. Ele permite diversas formas de visualização de uma lista.

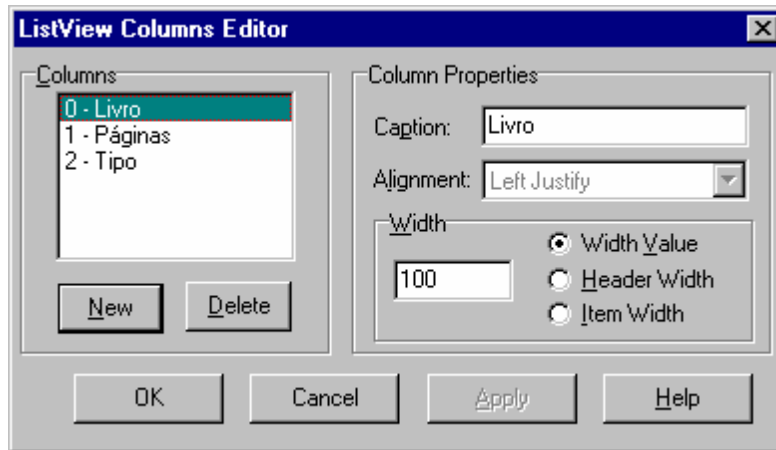
Figura 11-G

### Propriedades

1. AllocBy define o espaço adicional (quantas imagens) que será alocado cada vez que for preciso aumentar a estrutura de armazenamento.
2. ColumnClick determina se os cabeçalhos das colunas da lista devem se comportar como botões.

1. Columns nos permite editar as propriedades das colunas da lista.

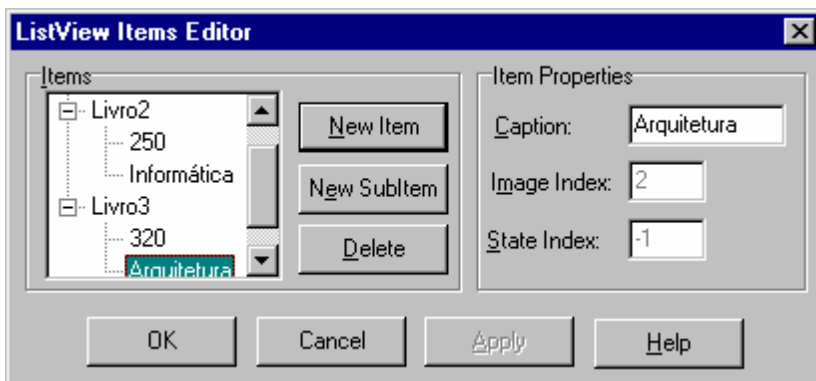
6. IconOptions determina o posicionamento (Arrangement), o auto-ajuste (AutoArrange) e o comportamento das legendas (WrapText) dos ícones.



**Figura 11-H**

Definição das colunas exibidas no ListView.

6. Items permite a edição dos itens da lista em tempo de projeto. Os subitens aparecerão na lista da segunda



**Figura 11-I**

Definição dos valores das colunas do ListView coluna em diante.



**Figura 11-J**

ListView com os valores definidos nas duas figuras 11-K e 11-L.

6. LargeImage indica o ImageList que contém as imagens grandes da lista.

7. MultiSelect indica se o usuário pode selecionar vários itens da lista simultaneamente.

8. ShowColumnHeader determina se os cabeçalhos das colunas serão exibidos.

9. SmallImage indica o ImageList que contém as imagens pequenas da lista.

10.SortType determina se e como os itens serão ordenados.

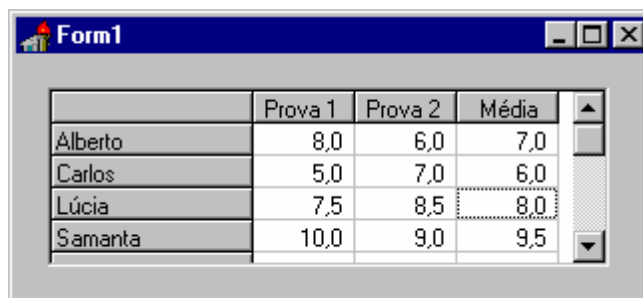
11.StateImages indica um componente ImageList que contém uma lista de imagens de estado (propriedade Items) que podem ser exibidas à esquerda dos rótulos dos itens.

## 11.8 Construindo (Tabelas)

## Grades

O Delphi possui quatro diferentes componentes de grade: grade de strings (StringGrid), grade de imagens (DrawGrid), grade de banco de dados (DBGrid) e grade de cores (ColorGrid). Com exceção da StringGrid, as grades definem apenas a organização e não o armazenamento dos dados, ou seja, elas são visualizadores, mas não são contêineres.

O DBGrid não é descrito nesta seção por se tratar de um dos tópicos do capítulo dedicado a aplicativos de bancos de dados.



	Prova 1	Prova 2	Média
Alberto	8,0	6,0	7,0
Carlos	5,0	7,0	6,0
Lúcia	7,5	8,5	8,0
Samanta	10,0	9,0	9,5

Figura 11-M

StringGrid para controle de notas de alunos.

### 11.8.1 StringGrid e DrawGrid

Como já foi mencionado, a principal diferença entre o StringGrid e o DrawGrid é que o primeiro possui sua própria estrutura de armazenamento, e o segundo não. O preenchimento das células do DrawGrid é efetuado através da utilização da propriedade Canvas dentro do manipulador do evento OnDrawCell, que é chamado sempre que uma célula precisa ser repintada (rolagem da grade, por exemplo).



Figura 11-N



Figura 11-O

#### Propriedades

1. ColCount e RowCount representam o número de colunas e de linhas.
2. DefaultColWidth e DefaultRowHeight representam o comprimento das colunas e a altura das linhas. Em tempo de execução, o comprimento (altura) de cada coluna (linha) pode ser alterado separadamente pela propriedade ColWidths (RowHeights).
3. FixedColor é a cor das linhas e colunas fixas.
4. FixedCols e FixedRows determinam o número de colunas e de linhas fixas.
5. Option controla algumas características da grade, como permitir que os usuário redimensione as colunas e linhas (goColSizing e goRowSizing), ou arraste colunas e linhas (goColMoving e goRowMoving).

### 11.8.2 ColorGrid

Este componente é usado para permitir que o usuário escolha uma cor de fundo (clicando o botão secundário do mouse sobre uma cor) e uma cor de primeiro plano (clicando o botão primário do mouse sobre uma cor).



Figura 11-P

#### Propriedades

1. BackgroundEnabled indica se o usuário pode selecionar uma cor de fundo, que fica caracterizada pelo símbolo BG. ForeGroundEnabled indica se o usuário pode selecionar uma cor de primeiro plano, que fica caracterizada pelo símbolo FG.
2. BackgroundIndex e ForegroundIndex são os índices da cor de fundo e da cor de primeiro plano.
3. GridOrdering determina o número de colunas e de linhas da grade.

## 12 Uma Barra de Ferramentas e uma Barra de Status

A barra de ferramentas, também chamada de speedbar ou de barra de controle, usualmente fica localizada no topo da janela, abaixo da barra de menus, e abriga uma série de pequenos botões e caixas combinadas que nos permitem emitir comandos e escolher opções. As barras de ferramentas mais recentes oferecem, ainda, a possibilidade de serem movidas para outras posições da janela.

A barra de status fica localizada na parte inferior da janela, e possui uma ou mais áreas destinadas à descrições textuais do estado atual do aplicativo. Exemplos de dados presentes na barra de status são a página atual de um documento, a descrição do comando de menu selecionado no momento, etc.

### 12.1 Agrupando Controles com o Painel



Figura 12-A

Para criar uma barra de ferramentas ou uma barra de status, você pode usar o componente Panel, já que ele é capaz de dividir a área cliente de uma janela e de agrupar outros componentes.

#### Propriedades

1. Align determina como o painel se alinha em relação ao formulário. As barras de ferramentas costumam ter o valor alTop, e as barras de status, alBottom. Quando esta propriedade é diferente de alNone, o painel é redimensionado juntamente com o formulário.
2. BevelInner determina a aparência do chanfro interno.
3. BevelOuter determina a aparência do chanfro externo.
4. BevelWidth determina o tamanho dos chanfros interno e externo.
5. BorderWidth determina a distância entre o chanfro interno e o externo.

### 12.2 SpeedButton



Figura 12-B

Um SpeedButton é semelhante a um BitBtn na aparência, mas pode se comportar como botão de pressão, caixa de verificação e botão de rádio. Além disso, por de ser um componente gráfico, consome menos recursos do sistema e não pode receber o foco de entrada.

#### Propriedades

1. AllowAllUp indica se todos os botões de uma mesmo grupo (ver propriedade GroupIndex) podem estar levantados ao mesmo tempo, ou se um deles tem que estar pressionado.
2. Down determina se um botão está pressionado.
3. GroupIndex determina se o SpeedButton deve se comportar como um botão de pressão (0), ou como uma opção de um grupo (> 0). No último caso, os botões que possuírem o mesmo GroupIndex farão parte do mesmo grupo e funcionarão como opções exclusivas.

#### Observações

Para simular uma caixa de verificação, basta criar um grupo de apenas um botão e ativar a sua propriedade AllowAllUp.

## 12.3 Criando uma Barra de Ferramentas

Para criar uma barra de ferramentas típica, você precisa colocar um painel alinhado com a parte superior do formulário e incluir nele uma série de componentes SpeedButton.

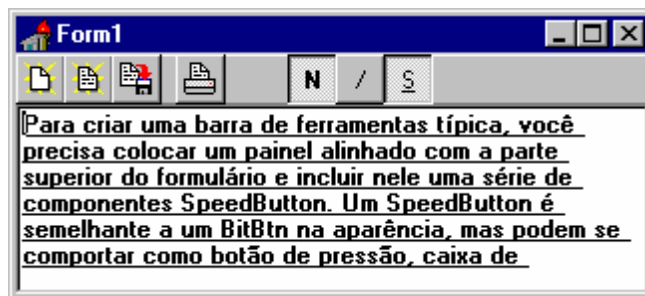


Figura 12-C

Formulário com uma barra de ferramentas contendo quatro SpeedButtons. Os botões de negrito, itálico e sublinhado comportam-se como caixas de verificação. Os demais comportam-se como botões de pressão comuns.

### 12.3.1 Adicionando Dicas à Barra de Ferramentas (Hint)

A dica, ou balloon help, é um texto que descreve brevemente um componente. Usualmente a dica é exibida numa caixa amarela que surge quando o ponteiro do mouse permanece fixo sobre o componente durante um intervalo de tempo.

Para adicionar dicas a um SpeedButton, ative a sua propriedade ShowHint e digite o texto da dica na propriedade Hint.

O objeto Application possui uma série de propriedades relativas às características das dicas que podem ser personalizadas, como a cor do retângulo (HintColor) e o atraso antes da apresentação (HintPause).

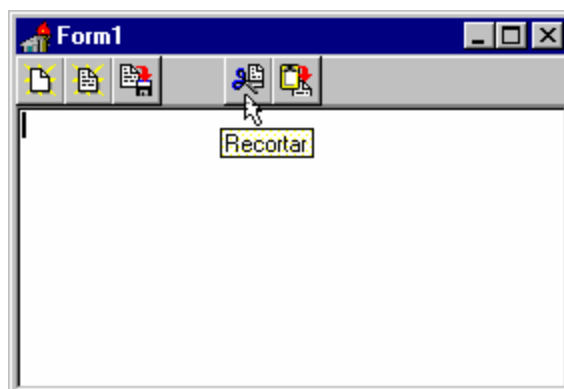


Figura 12-D

Dica do botão Recortar da barra de ferramentas.

## 12.4 Criando uma Barra de Status

Muito embora você possa utilizar um Painel para simular uma barra de status, é aconselhável empregar o componente StatusBar, cuja vantagem é permitir a configuração de diversos sub-painéis. Outra característica desse componente é a alça de reajuste presente no seu canto inferior direito que auxilia no reajuste do tamanho do formulário, comum nos aplicativos do Windows 95.

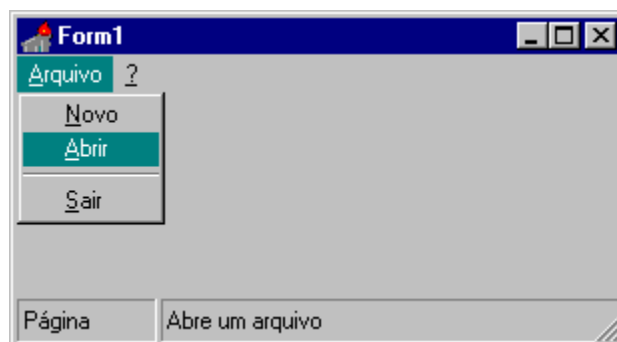


Figura 12-E

## Propriedades

1. Panels abre o editor de painéis da barra de status, onde podemos definir o texto, comprimento, estilo, chanfro e alinhamento de cada sub-painel. Esta propriedade só tem efeito quando a propriedade SimplePanel, descrita a seguir, está desativada.

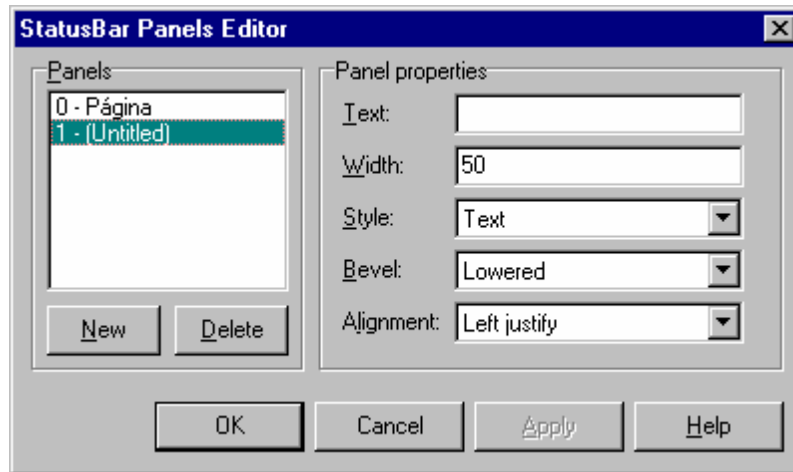


Figura 12-F

2. SimplePanel determina se a barra de status conterá um ou diversos painéis.
3. SimpleText é o texto exibido na barra de status quando a propriedade SimplePanel estiver ativada.

### 12.4.1 Adicionando Dicas à Barra de Status

Existe uma forma simples de exibir dicas adicionais sobre um componente na barra de status: na propriedade hint do componente, acrescente após a dica padrão o caracter pipe “|” e a dica adicional. O Delphi usará a primeira parte da string para a dica padrão e a segunda para a dica adicional da barra de status.

Agora crie um procedimento (eu chamei de dica) para copiar a dica adicional do componente para um dos painéis da barra de status, e associe o evento OnHint do aplicativo a esse procedimento.

```
type
  TForm1 = class(TForm)
    procedure dica (Sender: TObject);
    ...
  end;
  ...

procedure TForm1.dica (Sender: TObject);
begin
  StatusBar1.Panels[1].Text := Application.Hint;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  Application.OnHint := dica;
end;
```

## 13 Formulários Múltiplos e Caixas de Diálogo

Apesar de ser possível montar um aplicativo completamente funcional com um único formulário e alguns componentes, é bem mais comum ter mais de um formulário. Os aplicativos mais complexos podem ter uma estrutura MDI (Multiple Document Interface), ou seja, uma janela de estrutura com diversas janelas-filhas dentro da sua área cliente. Este tipo de aplicativo será discutido mais adiante. Este capítulo enfoca aplicativos não-MDI que compostos por vários formulários ou que tenham caixas de diálogo.

O Delphi possui caixas de diálogo pré-definidas: para exibir mensagens, selecionar cores, fontes, arquivos, etc. Agora nós vamos explorar a capacidade de definir nossas próprias caixas de diálogo.

### 13.1 Caixas de Diálogo Versus Formulários

As caixas de diálogo são um tipo especial de formulário, caracterizado pela borda e interface com o usuário. Outro aspecto marcante nas caixas de diálogo é que a maioria delas, mas nem todas, são janelas modais. Quando uma janela modal<sup>9</sup> é aberta, ela impede o acesso às outras janelas do aplicativo até ser fechada. As caixas de mensagem do Windows costumam ter este comportamento.

### 13.2 Adicionando um Segundo Formulário a um Programa

Para incluir um segundo formulário em um projeto, selecione o menu File / New Form. Você pode navegar entre os formulário e units de um projeto através dos comando View / Forms e View / Units.

Uma vez que já existam outros formulário além do formulário principal, nós podemos criar uma hierarquia de exibição entre eles, com o principal chamando um ou mais formulário subordinados e assim por diante. Isso é feito através dos métodos Show e ShowModal. O primeiro exibe o formulário como não-modal e o segundo, como modal.

Além do método Show (ou ShowModal), sempre que quisermos chamar um outro formulário, temos que incluir sua unit associada na cláusula uses da seção interface (ou da seção implementation) da unit chamadora. Isso acontece porque a variável que representa um formulário está declarada na sua unit associada e, quando “usamos” uma unit, passamos a enxergar todas as suas variáveis, rotinas, etc. não privadas.

Código para exibir o formulário DadosCompra como uma janela não-modal a partir do Menu Principal:

```
implementation

uses
    UnitCompra;    //unit do formulário Compra
    ...

procedure TMenuPrincipal.CompraButtonClick(Sender: TObject);
begin
    Compra.Show;
    ...
end;
```

---

<sup>9</sup> Outro termo utilizado para janela modal é janela restrita.

### 13.3 Formulários Modais e Não-Modais

Além da característica de restrição de foco dos formulários modais, existe uma outra distinção entre eles e os formulários não-modais: o método `Show` é um procedimento que abre um formulário e retorna o controle para o programa chamador; já o método `ShowModal`, é uma função que abre um formulário como janela modal e só retorna o controle para o programa chamador quando esta janela for fechada. Quando a propriedade `ModalResult` do formulário modal assume um valor não nulo, o formulário é fechado e método `ShowModal` retorna aquele valor.

Se o usuário pressionar o botão OK no formulário modal, o bloco de comandos após o método `ShowModal` é executado:

```
procedure TMenuPrincipal.CompraButtonClick(Sender: TObject);
begin
  Compra.ShowModal;    //Só retorna quando Compra é fechado.
  if Compra.ModalResult = mrOK then
  begin
    ...
  end;
end;
```

No formulário `Compra`, o botão OK fecha o formulário ao definir a propriedade `ModalResult` como `mrOK`.

```
procedure TCompra.OKButtonClick(Sender: TObject);
begin
  ModalResult := mrOK;    //Fecha o formulário modal
end;
```

Ao invés de escrever o procedimento anterior, você poderia definir, em tempo de projeto, a propriedade `ModalResult` do botão OK como `mrOK`.

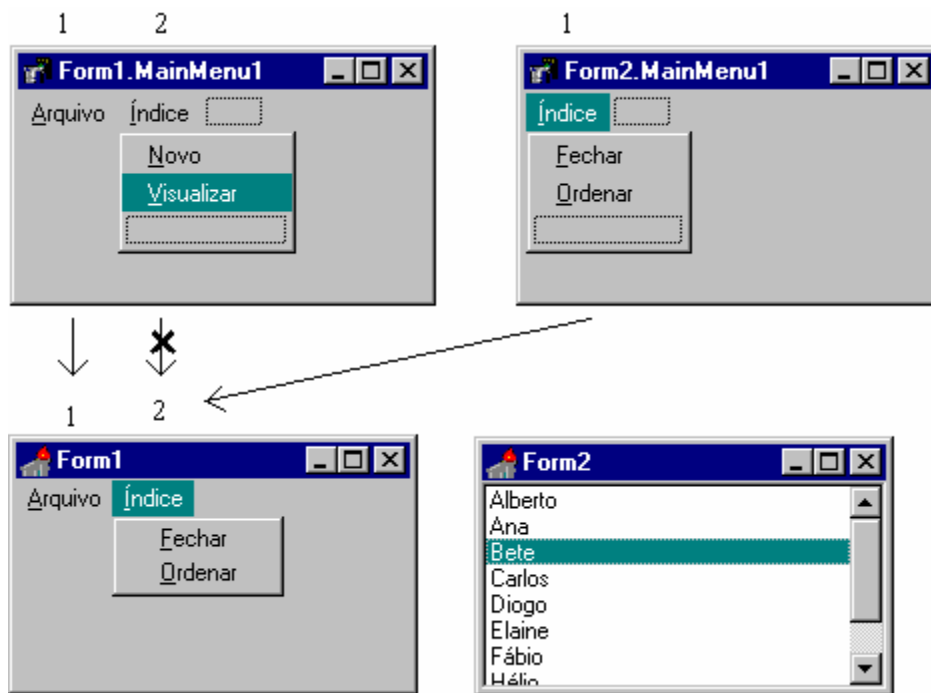
### 13.4 Mesclando Menus de Formulários

Imagine uma aplicação composta por um formulário principal que abre um formulário secundário não modal. Quando o formulário secundário estiver aberto, é possível que existam alguns menus do formulário principal que se tornam desnecessários por executarem ações que devem estar disponíveis somente quando o formulário secundário estiver fechado. Similarmente, outros menus podem passar a ser necessários.

Nestes casos, uma opção seria alternar a visibilidade dos menus. No entanto, a técnica de mesclar menus é mais simples, já que não exige codificação. Essa técnica é implementada da seguinte forma:

- O formulário principal e o formulário secundário devem ter barras de menus. Abaixo, elas são referenciadas como barra de menus principal e barra de menus secundária.
- A propriedade `AutoMerge` da barra de menus secundária deve ser ativada.
- Cada menu da barra de menus principal será substituído pelo menu da barra de menus secundária que tiver o mesmo valor para a propriedade `GroupIndex`.
- O formulário secundário não exibirá sua barra de menus.
- Os menus serão ordenados por ordem ascendente de `GroupIndex`.

A mesclagem de menus só tem sentido quando você abre um formulário não-modal, pois um formulário modal não permite acesso à barra de menus do formulário principal, sendo inútil alterá-la.



**Figura 13-A**

Acima são mostrados os GroupIndex menus do formulário principal (Form1) e do formulário secundário (Form2). Abaixo é mostrado o aplicativo em execução, com a substituição do menu “Índice” do formulário principal pelo menu “Índice” do formulário secundário, quando este está aberto.

### 13.5 Criando uma Caixa de Diálogo

Há um truque simples para se montar uma caixa de diálogo a partir de um formulário. Selecione o valor `bsDialog` para a propriedade `BorderStyle` do formulário. Além de alterar a borda, esta opção modifica o menu de sistema e retira os botões de maximizar e minimizar.

Os botões de bitmap (`BitBtn`) agilizam a criação de alguns tipos de botão característicos das caixas de diálogo (OK, Cancelar, Fechar, ...) através da propriedade `Kind`. Quando ela é definida com algum valor diferente de `bkCustom`, as propriedades `Cancel`, `Caption`, `Default`, `Glyph` e `ModalResult` são alteradas convenientemente.

Assim que a caixa de diálogo estiver pronta, use o métodos `Show` ou `ShowModal` para exibi-la como janela não-modal ou modal.

### 13.6 Usando Caixas de Diálogo Pré-definidas

O Delphi permite que você use caixas de diálogo pré-definidas pelo Windows, caixas de mensagem e de entrada, além de alguns modelos e experts.

#### 13.6.1 Caixas de Diálogo Comuns do Windows

A página `Dialog` da palheta de componentes contém as caixas de diálogo comuns do Windows. Algumas delas já foram apresentadas anteriormente, e costumam ser utilizadas de um modo padrão. Basicamente, você precisa:

- Colocar o componente no formulário e alterar algumas das suas propriedades.

- Chamar a caixa de diálogo com o método `Execute`. Algumas vezes é preciso inicializar algumas propriedades da caixa de diálogo de acordo com os objetos da aplicação afetados por ela.
- Se o usuário confirmar as alterações na caixa de diálogo, repassá-las para os objetos apropriados.

### 13.6.2 Caixas de Mensagem e de Entrada

Há seis procedimentos e funções do Delphi que você pode usar para exibir caixas de diálogo simples:

- MessageDlg: função que exibe no centro da tela uma mensagem que pode ser personalizada com um ou mais botões e um bitmap. Retorna o botão pressionado pelo usuário.
- MessageDlgPos: semelhante à função `MessageDlg`, com a exceção de permitir o posicionamento da mensagem na tela.
- ShowMessage: procedimento que exibe no centro da tela uma mensagem com um botão OK.
- ShowMessagePos: semelhante ao procedimento `ShowMessage`, com a exceção de permitir o posicionamento da mensagem na tela.
- InputBox: função que exibe no centro da tela uma pergunta, uma caixa de texto para ser preenchida pelo usuário com a resposta e os botões OK e Cancelar. Retorna uma string não importando o botão pressionado.
- InputQuery: função que exibe no centro da tela uma pergunta, uma caixa de texto para ser preenchida pelo usuário com a resposta e os botões OK e Cancelar. Retorna o botão pressionado pelo usuário.

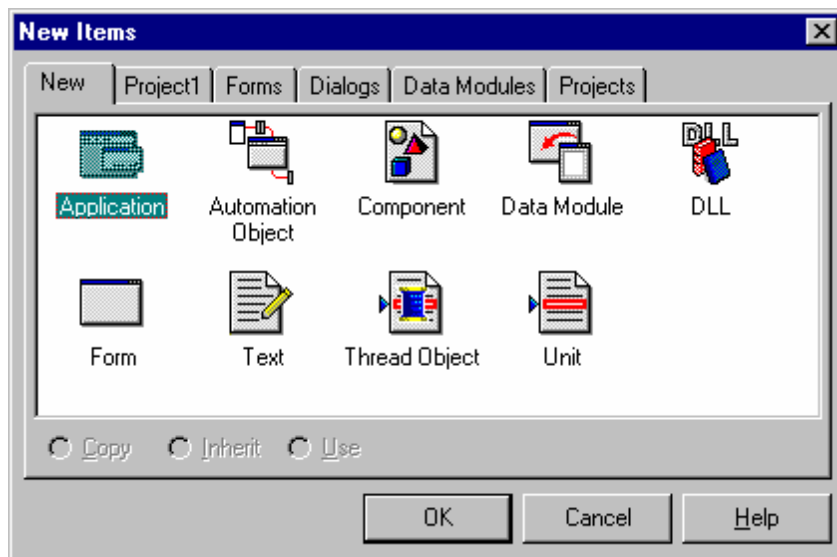
## 13.7 Herança Visual de Formulários

De forma análoga ao conceito de herança de classes das linguagens de programação, a herança de formulários nos permite criar interfaces visuais e códigos genéricos para serem usados em formulários descendentes.

### 13.7.1 Herdando de um Formulário-Base

O formulário-base pode ser um formulário qualquer da aplicação que já tenha alguma interface e funcionalidade.

Para criar um formulário descendente, selecione o menu `File / New`; na página com o nome do projeto do Repositório de Objetos, selecione o formulário escolhido para se base e clique o botão OK.



**Figura 13-B**

A página Project1 (nome do projeto atual) do Repositório de Objetos lista os formulários que podem servir como base.

De acordo com as regras que governam a herança visual de formulários:

- O formulário de subclasse possui os mesmos componentes que seu

formulário-base. Estes componentes podem ter suas propriedades e manipuladores de eventos alterados, mas não podem ser excluídos.

- O formulário de subclasse pode possuir novos componentes.
- Alterações nos componentes herdados do formulário-base se repercutem no formulário derivado. Se um componente herdado tiver uma propriedade alterada no formulário derivado, alterações subsequentes desta propriedade no formulário-base não terão efeito no formulário derivado. Entretanto, você pode re-sincronizar um propriedade do componente selecionando-a no Object Inspector e executando o comando Revert to Inherited do menu local<sup>10</sup>. Para re-sincronizar todas as propriedades, execute o comando Revert to Inherited do menu local do componente.
- Um formulário de subclasse possui todos os métodos do formulário-base.
- Você pode adicionar novos manipuladores de eventos e sobrepor os existentes.

Executa o manipulador herdado do formulário-base:

```
procedure TForm2.Button1Click(Sender: TObject);
begin
    inherited;
end;
```

Executa o manipulador herdado do formulário-base e algum código adicional:

```
procedure TForm2.Button1Click(Sender: TObject);
begin
    inherited;
    ShowMessage (Msg);
end;
```

Sobrepo o manipulador herdado do formulário-base:

```
procedure TForm2.Button1Click(Sender: TObject);
begin
```

<sup>10</sup> Isso remove a linha na descrição textual do formulário que define um valor para a propriedade do componente herdado, diferenciando-a da propriedade do componente-base.

```
    ShowMessage (Msg);  
end;
```

Executa um outro manipulador herdado do formulário-base:

```
procedure TForm2.Button1Click(Sender: TObject);  
begin  
    inherited Button2Click(Sender);  
end;
```

## 14 Rolagem e Formulários Multipáginas

O uso de fichários e abas tornou-se tão importante que a Microsoft incluiu suporte direto para eles no Windows 95.

Mas, primeiro discutiremos uma técnica mais simples, porém efetiva, quando se precisa inserir diversos componentes em um formulário: a rolagem de tela.

### 14.1 Rolando um Formulário

A forma mais simples de se permitir a rolagem de um formulário é ativar a sua propriedade AutoScroll. Quando isto é feito e a área-cliente do formulário é reduzida deixando componentes ocultos, as barras de rolagem necessárias são automaticamente adicionadas.

Entretanto, as vezes podemos querer criar um formulário com um número de componentes maior que o comportado pela área-cliente. As propriedades de formulário HorzScrollBar e VertScrollBar fornecem os seguintes recursos como sub-propriedades:

1. Increment é o número de posições que a barra de rolagem se desloca quando o usuário clica nas setas laterais.
2. Margin é a distância mínima que os componentes devem manter da borda do formulário para não causarem o aparecimento da barra de rolagem. Este recurso funciona quando a propriedade AutoScroll está ativa.
3. Position indica a posição da caixa de rolagem.
4. Range é o tamanho virtual do formulário. Sempre que a área visível for menor que o valor especificado em Range, a barra de rolagem aparecerá.
5. Tracking determina se o deslocamento da caixa de rolagem deve causar a rolagem simultânea do formulário, ou se a rolagem deve ser efetuada somente quando o usuário soltar a caixa de rolagem
6. Visible determina se a barra de rolagem deve ficar visível.

### 14.2 Criando Fichários

Existem diversos componentes que podem ser empregados na criação de fichários. Porém, é recomendável utilizar os componentes PageControl e TabControl, já que eles são baseados em controles comuns do Windows. As outras alternativas são os componente NoteBook e TabSet, ou TabbedNotebook, presente no Delphi 2.0 por propósito de compatibilidade.

#### 14.2.1 TabControl, PageControl e TabSheet



Figura 14-A

TabControl



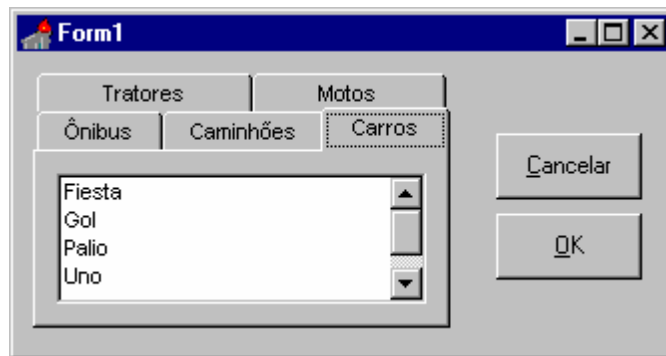
Figura 14-B

PageControl

O TabControl é usado para construir abas isoladas, ou seja, abas não conectadas com páginas. O componente PageControl, ao invés, é usado para construir um fichário completo, onde as páginas são componentes TabSheet. Como o PageControl é capaz de, sozinho, oferecer uma interface completa do tipo fichário, vamos nos concentrar nas suas características.

#### Propriedades do PageControl

1. `ActivePage` define a página apresentada no momento. Outra maneira de selecionar uma aba em tempo de projeto é clicando sobre ela.
2. `MultiLine` define se as abas devem ser enfileiradas em camadas, ou se elas devem aparecer em uma única fila. Neste caso, se o número de abas for maior que a largura do `PageControl`, um par de setas aparecerá para permitir a rolagem até as abas que ficaram ocultas.
3. `TabHeight` e `TabWidth` determinam a altura e largura das abas. Quando estas propriedades tem o valor zero, as dimensões das abas se ajustam de forma a comportar as legendas.



**Figura 14-C**

PageControl multilinha com a página “Carros” selecionada.

A criação de páginas em um componente `PageControl` é feita através do comando `New Page` do seu menu local.

Se você inserir um componente em uma página, ele estará disponível somente naquela página. Mas você pode simular a presença de um componente em todas as páginas, criando-o fora do `PageControl` e, depois, posicionando-o sobre as páginas.

Como já foi dito, cada página de um `PageControl` é um componente `TabSheet`. As propriedades mais importantes deste componente são:

1. `Caption` é a legenda exibida na aba da página.
2. `PageIndex` é o índice usado para referenciar uma página e determina também a posição no `PageControl`.
3. `TabVisible` determina se a aba da página deve estar visível.

Observe que a página exibida inicialmente em tempo de execução é a página que estava selecionada em tempo de projeto quando o aplicativo foi compilado. Desta forma, pode ser uma boa idéia definir em tempo de execução a página que deve estar ativa quando o `PageControl` for apresentado.

```
procedure TForm1.FormCreate(Sender: TObject);
begin
  PageControl1.ActivePage := TabSheet1;
end;
```

### 14.2.2 Abas sem Páginas e Páginas sem Abas

Embora a situação mais comum seja empregar abas e páginas em conjunto para formar um fichário, nada o impede de criar interfaces que utilizem abas sem páginas ou páginas sem abas. Um exemplo conhecido do segundo caso é o expert ou wizard.

## 15 Dividindo Janelas

Uma interface interessante com o usuário que está se tornando comum é o divisor (splitter), um elemento que você pode arrastar para alterar o tamanho ou mover alguns componente em uma janela. Um exemplo típico é o divisor entre a lista de diretórios e a lista de arquivos do Windows Explorer.

### 15.1 Técnicas de Divisão de Formulários

Você pode usar as seguintes técnicas para separar formulários:

1. Usando um componente Header.
2. Colocando um componente (normalmente um Panel ou GroupBox) entre dois outros elementos.
3. Usando um componente que suporte internamente a operação de arrastar, por exemplo os componentes StringGrid e DrawGrid.
4. Deixando um espaço entre os componentes e use a operação de arrasto sobre este espaço.



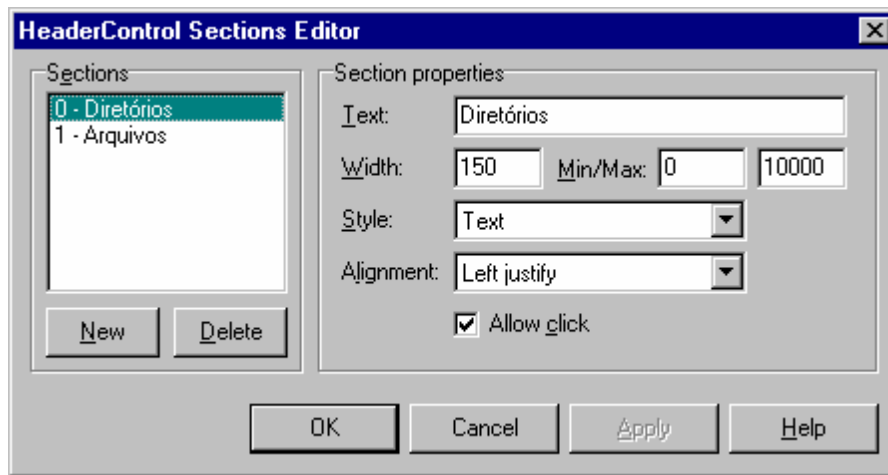
### 15.2 Dividindo com um Cabeçalho (HeaderControl)

Figura 15-A

O HeaderControl do Windows 95 é a técnica mais utilizada para implementar um separador vertical. Como alternativa, e por uma questão de compatibilidade com o Windows 3.xx, você pode utilizar o componente Header do próprio Delphi.

#### Propriedades

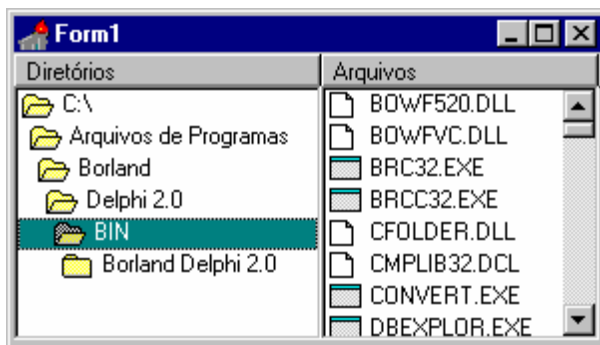
1. Align determina a posição do Cabeçalho dentro do formulário. As posições mais usuais são o topo (alTop) ou toda a área-cliente (alClient).
2. Sections define as seções do Cabeçalho com base nas seguintes propriedades:
  - Text é a legenda da seção.
  - Width é o comprimento atual da seção.
  - Max / Min são os tamanhos máximo e mínimo a que uma seção pode ser redimensionada.
  - Style determina se os dados da seção serão exibidos como strings, ou se serão desenhados em tempo de execução.
  - Alignment determina o alinhamento da legenda da seção.
  - AllowClick determina se a seção deve se comportar como um botão, podendo ser clicada.



**Figura 15-B**

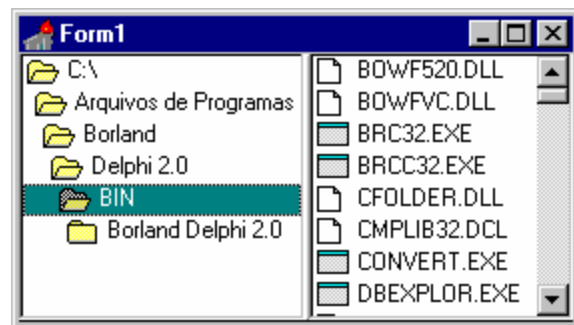
Definição das seções “Diretórios” e “Arquivos” de um cabeçalho.

Com base nas propriedades das seções do cabeçalho, podemos definir a largura de outros componentes do formulário.



**Figura 15-C**

Cabeçalho no topo da área-cliente.



**Figura 15-D**

Cabeçalho em background ocupando a área-cliente.

Código para redimensionamento das listagens da figura 15-E:

```
procedure TForm1.HeaderControl1SectionResize(HeaderControl:
  THeaderControl; Section: THeaderSection);
begin
  With HeaderControl1 do
  begin
    DirListBox1.Width := Sections[0].Width;
    FileListBox1.Left := Sections[1].Left;
    FileListBox1.Width := Sections[1].Width;
  end;
end;
```

Código para redimensionamento das listagens da figura 15-F:

```
procedure TForm1.HeaderControl1SectionResize(HeaderControl:
  THeaderControl; Section: THeaderSection);
begin
  With HeaderControl1 do
  begin
    DirListBox1.Width := Sections[0].Width;
    FileListBox1.Left := Sections[1].Left + 2;
    FileListBox1.Width := Sections[1].Width;
  end;
end;
```

## 16 Criando Aplicativos MDI

Além de usar caixas de diálogo ou formulários secundários, há uma terceira abordagem comum de distribuição de funções em aplicativos Windows, conhecida como MDI (Multiple Document Interface). Os aplicativos MDI são compostos por diversos formulário (janelas-filhas) que aparecem dentro de um único formulário principal, chamado de moldura (frame) ou aplicativo.

O Microsoft Word 6.0 é um aplicativo MDI. A janela de moldura, intitulada “Microsoft Word” abriga os documentos de texto (janelas-filhas).

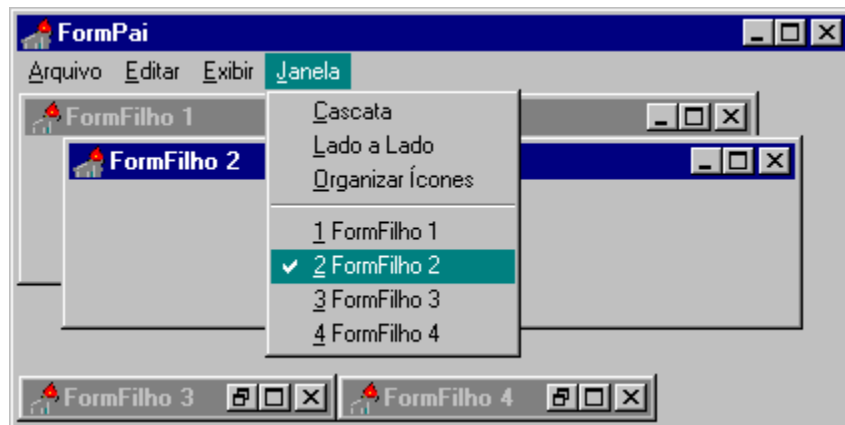


Figura 16-A

Aplicativo MDI com quatro janelas-filhas.

### 16.1 Janelas-Filhas e Janelas de Moldura

Um aplicativo MDI exige pelo menos dois formulários, um com a propriedade `FormStyle` definida para `fsMDIForm` (janela de moldura) e outro com a mesma propriedade definida para `fsMDIChild` (janela-filha).

Cada formulário-filho será usado como matriz na criação de um tipo de janela-filha, que pode ser feita através de um comando Nova Janela no menu Arquivo.

```
procedure TFormPai.Novajanela1Click(Sender: TObject);
var
  filha: TFormFilho;
begin
  filha := TFormFilho.create (self);
  Inc(i);
  filha.Caption := filha.Caption + ' ' + IntToStr(i);
  filha.show;
end;
```

Se você não quiser que uma janela-filha seja criada automaticamente na inicialização do aplicativo, use a página Forms do comando de Project / Options.

## 16.2 Criando um Menu Janela Completo

O Windows faz a manipulação automática do menu que contém a lista de janelas-filhas abertas. Para isso, insira uma barra de menus no formulário de moldura e crie um menu (Janela é a legenda mais comum). Depois, selecione esse menu na lista de opções da propriedade `WindowMenu`<sup>11</sup>.

Podemos, ainda, usar alguns dos métodos dos formulário com estilo `fsMDIForm`:

1. `Cascade` organiza as janelas-filhas em cascata.
2. `Tile` organiza as janelas-filhas lado a lado. A propriedade `TileMode` determina se o método `Tile` fará um arranjo horizontal (`tbHorizontal`) ou vertical (`tbVertical`). Os ícones das janelas-filhas minimizadas também são organizados.
3. `ArrangeIcons` organiza somente os ícones das janelas-filhas minimizadas.

## 16.3 Montando uma Janela Filha

Quando fechamos uma janela-filha ela é minimizada. Isso ocorre porque o comportamento padrão de um formulário-filho é ficar minimizado ao ser “fechado”. Precisamos alterar a variável `Action` do método executado ao fechar a janela-filha de forma a mudar este comportamento.

```
procedure TFormFilho.FormClose(Sender: TObject;  
    var Action: TCloseAction);  
begin  
    Action := caFree;  
end;
```

Uma janela-filha não pode conter sua própria barra de menus. Se definirmos uma, ela será exibida na janela de moldura, com o padrão de mesclagem de menus discutido na seção 13.4 (mesclando menus de formulários).

---

<sup>11</sup> Parece ser necessária a presença de algum item de menu no menu Janela para que esse recurso funcione.

## 17 Utilizando Controles OLE

Os controles OLE (OCX) seguem um padrão de desenvolvimento que proporciona uma interface semelhante àquela dos componentes do Delphi.

Além de utilizar controles OLE (OCX), o Delphi permite que você herde um novo componente a partir de um OCX, adicionando-lhe novas capacidades.

### 17.1 Controles OLE Versus Componentes Delphi

- Controles OLE são baseados em DLL (Dynamic-Link Library), de forma que você tem instalar o seu código (arquivo OCX) juntamente com a aplicação que o utiliza. O código dos componentes Delphi são ligado estaticamente ao arquivo executável.
- Aplicações diferentes podem compartilhar o código de um mesmo controle OLE, em disco e em memória. No entanto, podem surgir problemas de compatibilidade se aplicações diferentes utilizarem versões diferentes de um OCX.
- A abordagem de componentes Delphi é ligeiramente mais rápida. Em compensação, o arquivo executável tende a ser maior.
- Em relação ao licenciamento, você pode ter que pagar royalties para poder distribuir um arquivo OCX, diferentemente do Delphi.
- Os controles OLE existem em maior número que os componentes Delphi e estão disponíveis para diversos ambientes de desenvolvimento, pois seguem o padrão OCX que é amplamente aceito no mercado.

### 17.2 Instalando um Novo Controle OLE

Para tornar um controle OLE disponível ao sistema, você precisa instalá-lo:

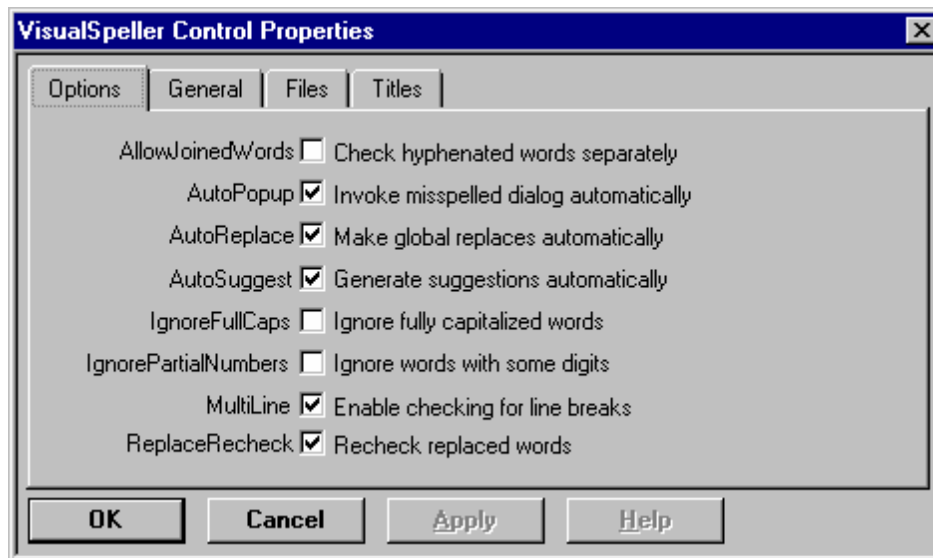
1. Selecione o menu Component / Install e, na caixa de diálogo Install Components, clique o botão OCX.
2. Na caixa de diálogo Import OLE Control você pode ver a lista de controles OLE atualmente registrados no Delphi. Clique o botão Register e localize o arquivo OCX a ser instalado<sup>12</sup>.
3. A parte inferior da caixa de diálogo permite que você especifique um nome para o arquivo Object Pascal de interface para o controle OLE. Você pode, também, alterar a página da Palheta de Componentes onde o OCX será inserido, ou o nome da nova classe.
4. Assim que você fechar as duas caixas de diálogo, o Delphi reconstruirá a Biblioteca de Componentes Visuais, exibindo o novo controle na Palheta de Componentes.

---

<sup>12</sup> Arquivos OCX costumam ficar localizados no diretório Windows\System



Outra diferença é a possibilidade de editar as propriedades usando o Object Inspector ou o editor de propriedades específico do controle OLE, que pode ser invocado através do comando Properties do speedmenu.



**Figura 17-B**

Editor de propriedades específico do controle OLE VCSpeller.

## 18 Criando Aplicativos de Bancos de Dados

O suporte a banco de dados é um dos recursos-chave do ambiente Delphi, especialmente da versão Client-Server. Além de acessar bancos de dados locais, como a versão Desktop, o Delphi Client-Server é capaz de acessar bancos de dados SQL<sup>13</sup> ou via ODBC<sup>14</sup> em computadores servidores.

### 18.1 Acesso a Bancos de Dados

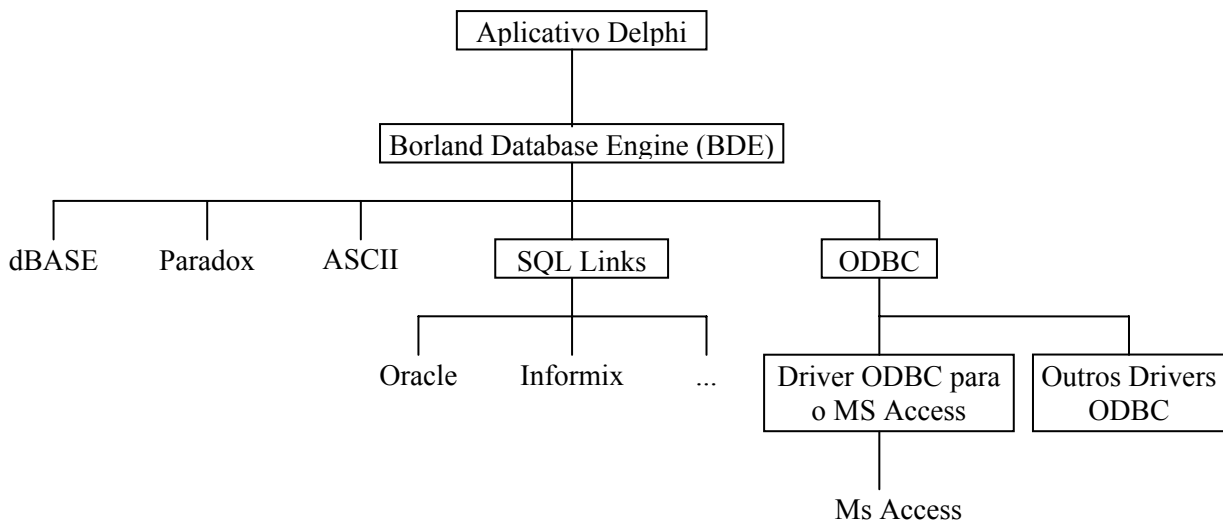
As duas técnicas mais comuns de armazenamento de bancos de dados, ambas acessíveis ao Delphi, são:

1. O armazenamento de todos os dados (tabelas, índices, ...) em um arquivo único.
2. O armazenamento dos dados em arquivos separados, normalmente dentro do mesmo diretório.

Você se refere a um banco de dados pelo seu nome ou por meio de um alias (apelido). Essa referência pode ser a um arquivo, ou ao diretório que contém os dados, dependendo da técnica de armazenamento em questão.

Os aplicativos Delphi não têm acesso direto às fontes de dados, eles fazem interface com o Borland Database Engine (BDE) que acessa diretamente diversas fontes de dados, incluindo dBASE, Paradox e tabelas ASCII (por meio de drivers apropriados).

O BDE também pode fazer interface com SQL Links da Borland, que permitem acesso a vários servidores SQL remotos e locais. O servidor local disponível é o InterBase for Windows; entre os servidores remotos incluem-se o Oracle, o Sybase, o Informix e o Interbase. Se você precisar acessar um banco de dados ou formato de dados diferente, o BDE pode fazer interface com drivers ODBC.



**Figura 18-A**

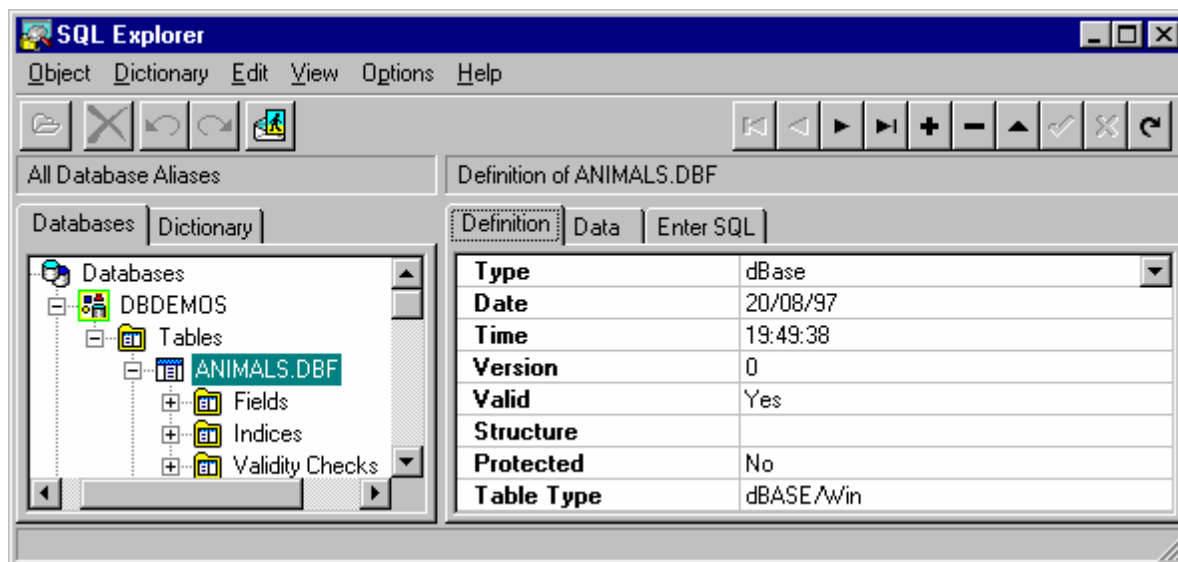
Ilustração do mecanismo de acesso a bancos de dados no Delphi.

<sup>13</sup> Structured Query Language: uma linguagem de manipulação de dados.

<sup>14</sup> Open Database Connectivity: padrão de acesso a bancos de dados. É o método menos eficiente de acesso a dados, devendo, portanto, ser evitado.

O fato de o Delphi não suportar acesso direto a bancos de dados significa que você terá que instalar o BDE junto com seus aplicativos clientes para que eles funcionem. Isto não é difícil, já que a Borland tem um programa de instalação do BDE que pode ser distribuído gratuitamente.

Para visualizar a estrutura e os dados de um banco de dados, podemos usar o SQL Explorer, que acompanha o Delphi e pode ser acessado pelo menu Database / Explore.



**Figura 18-B**

Estrutura da tabela ANIMALS.DBF no SQL Explorer.  
Podemos visualizar os dados selecionando o página Data.

## 18.2 Componentes de Bancos de Dados

A página Data Access da palheta contém os componentes usados para interagir com bancos de dados, em sua maioria, invisíveis em tempo de execução, já que envolvem conexões, tabelas, queries, etc. Os componentes empregados na visualização e edição de dados ficam localizados na página Data Control e são chamados de componentes relativos a dados (data-aware).

### 18.2.1 DataSource

Para acessar um banco de dados, geralmente você precisa de uma fonte de dados, simbolizada pelo componente DataSource. Todavia, este componente não se refere aos dados diretamente, e sim a um “data set”, que pode ser um componente Table, Query ou StoredProc. O papel do DataSource é conectar os componente relativos a dados com os “data sets”.



**Figura 18-C**

#### Propriedades

1. AutoEdit determina os registros da tabela poderão ser alterados diretamente nos componente relativos a dados associados ao DataSource.
2. DataSet indica o componente data set que contém os dados efetivamente. Para que esta propriedade liste o data set de outro formulário, insira a unit associada ao formulário na seção uses da porção implementariion (comando de menu File / Use Unit).

3. **Enabled** determina se os componente relativos a dados associados ao DataSource estarão disponíveis e serão atualizados com os dados do registro corrente.

### 18.2.2 Data Sets



**Figura 18-D**

Table



**Figura 18-E**

Query

Um componente Table refere-se a uma tabela de um banco de dados ou a um arquivo de um diretório. As tabelas disponíveis são listadas assim que a propriedade Database for definida.

O componente Query é similar ao Table. A diferença é que ao invés de selecionar uma tabela, deve-se entrar com uma declaração SQL na propriedade SQL.

Por último, temos o componente StoredProc, que se refere a procedures locais de um servidor de banco de dados SQL. Você pode executar essas procedures e obter os resultados na forma de uma tabela.

### 18.2.3 Outros Componentes de Acesso a Dados

Além de Table, Query, StoredProc e DataSource, há cinco outros componentes na página Data Access:

1. **Database**: é usado para controle de transação, segurança e controle de conexão. Geralmente, ele é usado somente para conexões a bancos de dados remotos em aplicativos cliente/servidor.
2. **Session**: fornece controle global sobre as conexões de bancos de dados para uma aplicação, incluindo uma lista dos bancos de dados existentes e um evento para personalizar o log-in.
3. **BatchMove**: é usado para executar operações em lote, em um ou mais bancos de dados.
4. **UpdateSQL**: permite que você escreva instruções SQL para executar as diversas operações de atualização de um data set. Este componente é utilizado como valor da propriedade UpdateObject de tabelas e queries.
5. **Report**: é uma interface para o aplicativo ReportSmith da Borland.

Estes podem ser considerados componentes avançados de banco de dados e alguns deles não têm sentido para ambientes locais.

### 18.2.4 Componentes Relativos a Dados

Para mostrar os dados de uma fonte, utilizamos os componentes semelhantes aos controles normais do Windows localizados na página Data Controls da palheta.

- **DBGrid**: é uma grade capaz de exibir toda uma tabela de dados de uma só vez. Ela permite rolagem e navegação e você pode editar seu conteúdo.
- **DBNavigator**: é um conjunto de botões usado para navegar e executar ações no banco de dados.
- **DBLabel**: é usado para exibir o conteúdo de um campo que não pode ser modificado.
- **DBEdit**: é usado para permitir que o usuário veja e modifique um campo.
- **DBMemo**: é usado para permitir que o usuário veja e modifique um campo grande.
- **DBImage**: é usado para mostrar uma figura armazenada em um campo BLOB (Objeto Binário Grande)
- **DBListBox** e **DBComboBox**: são usados para permitir que o usuário selecione um valor de um conjunto especificado. Se este conjunto for extraído de uma tabela do banco de dados ou for

resultado de uma query, você deve utilizar os componentes DBLookupListBox ou DBLookupComboBox.

- **DBCheckBox**: pode ser usado para mostrar e ativar/desativar uma opção.
- **DBRadioGroup**: é usado para fornecer uma série de opções exclusivas.
- **DBCtrlGrid**: é uma grade multi-registro, que pode acomodar diversos outros controles relativos a dados.

Todos esses componentes são conectados à fonte de dados usando a propriedade DataSource. Uma vez que esta propriedade esteja definida, a propriedade DataField exibirá uma lista dos campos existentes na fonte.

### 18.3 Acessando os Campos de uma Tabela ou Query

Normalmente, os campos de tabelas e queries são criados em tempo de execução e armazenados na propriedades vetorial Fields. Podemos acessar os valores do campo usando seu nome ou seu número de ordem dentro do conjunto de campos.

Atribuição do valor do primeiro campo da tabela Table1 ao componente Edit1:

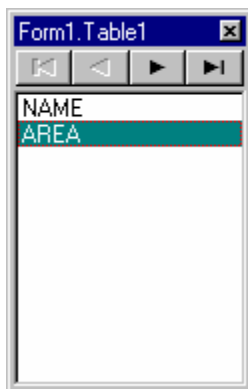
```
Edit1.Text := Table1.Fields[0].Value;
```

Atribuição do valor do campo Nome da tabela Table1 ao componente Edit1:

```
Edit1.Text := Table1.FieldName('Nome').Value;
```

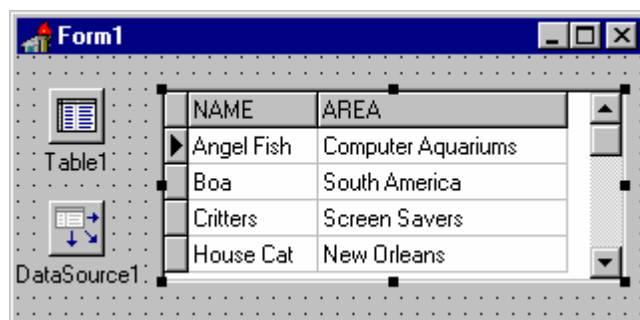
Como alternativa, os componentes fields podem ser criados em tempo de projeto com o Fields Editor e passarão a ser listados no Object Inspector com suas respectivas propriedades.

Para abrir o Editor de Campos de uma tabela, ative o speedmenu do componente Table e escolha o comando Fields Editor. Uma vez no Fields Editor, você pode usar o seu speedmenu para adicionar (Add Fields) campos da tabela à lista de campos, ou definir novos campos (New Field)



**Figura 18-G**

Campos “NAME” e “AREA”  
definidos no Fields Editor.



**Figura 18-F**

Apenas os campos definidos no Fields Editor se  
encontram disponíveis.

Um outro recurso do Fields Editor é que você pode arrastar os campos para a superfície de um formulário para gerar componentes apropriados aos seus tipos de dados.

## 18.4 Usando Campos para Manipular uma Tabela

### 18.4.1 Procurando Registros em uma Tabela

Existem vários métodos usados para localizar (`GotoKey`, `FindKey`, `GotoNearest`, `FindNearest`) e navegar pelos dados de uma tabela (`First`, `Last`, `Next`, `MoveBy`).

Ir para o primeiro registro da tabela `Table1`:

```
Table1.First;
```

Avançar cinco registros da tabela `Table1`:

```
Table1.MoveBy(5);
```

Localizar o registro da tabela `Table1` que tenha o valor 'Marta' para o campo `Nome`:

```
Table1.IndexName := 'Nome';           //Índice usado para a busca  
Table1.FindKey(['Marta']);
```

### 18.4.2 Atualizando Registros em uma Tabela

Use os métodos `Append`, `Insert`, `AppendRecord` ou `InsertRecord` para incluir registros; `Delete` para excluir o registro corrente; `Edit` para colocar o registro corrente em estado de alteração; `Post` para gravar o registro corrente; `Cancel` para descartar quaisquer alterações efetuadas no registro corrente.

Inclui um registro na tabela `Table1`:

```
Table1.Append;  
Table1('Nome') := 'Humberto';  
Table1('Cidade') := 'Porto Alegre';  
Table1.Post;
```

Exclui o último registro da tabela `Table1`:

```
Table1.Last;  
Table1.Delete;
```

Inclui um registro na tabela `Table1` e verifica se o usuário quer gravar as alterações:

```
Table1.Append;  
Table1('Nome') := 'Roberto';  
if MessageDlg('Gravar Registro?', mtConfirmation, [mbYes, mbNo],  
    0) = mrYes then  
    Table1.Post  
else  
    Table1.Cancel;
```

## 18.5 Criando um Formulário Mestre-Detalhe com o Form Expert

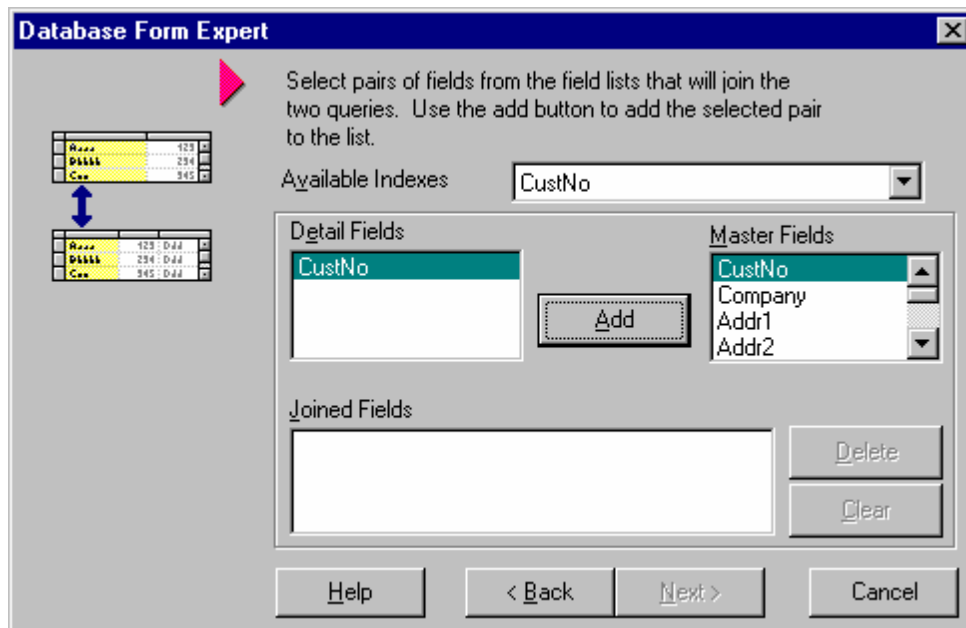
Um formulário mestre-detalhe exibe os dados de duas tabelas que se relacionam na forma um-para-muitos.

Para gerar este tipo de formulário com o Form Expert, acione o menu `Database / Form Expert` e selecione a opção "Create a master/detail form". A seguir será pedido o nome da tabela-mestra, os campos a serem exibidos e a organização do formulário; depois, ocorre o mesmo processo para a tabela-detalhe. Finalmente, você deverá selecionar os campos das duas tabelas que servirão para relacioná-las. Estes campos devem fazer parte de algum índice da tabela-detalhe.

O que o Form Expert faz é definir algumas propriedades do componente `Table` referente à tabela-detalhe:

- `MasterSource` é o nome do `DataSource` associado à tabela primária.

- MasterFields é o campo usado para relacionar a tabela-detalle à tabela-mestra.
- IndexName é o índice que contém o campo (MasterField) usado para definir o relacionamento.



**Figura 18-H**

O índice da tabela-detalle CustNo, formado pelo campo CustNo, é selecionado; o campo CustNo é selecionado em Detail Field e em Master Field; o relacionamento entre a tabela-mestra e a tabela-detalle é criado clicando-se o botão Add.

## 18.6 Caixas Combinadas Relativas a Dados

Existem dois tipos de caixas combinadas relativas a dados:

1. **DBComboBox**: a lista de valores é extraída da propriedade Items, e o valor selecionado é armazenado no campo da tabela. DataSource indica onde são armazenados os dados e DataField é o campo do DataSource.
2. **DBLookupComboBox**: a lista de valores é extraída de um data set. Aqui o valor selecionado na lista pode ser diferente do valor armazenado no campo da tabela. Podemos por exemplo, exibir nomes de pessoas na lista e armazenar os seus códigos, que estão ocultos. DataSource e DataField são como no caso anterior; ListSource é o DataSource usado para preencher a lista; ListField são os campos exibidos na lista (os nomes dos campos devem ser separados por ponto-e-vírgula); KeyField é o campo (exibido ou não na lista) que será armazenado na tabela.