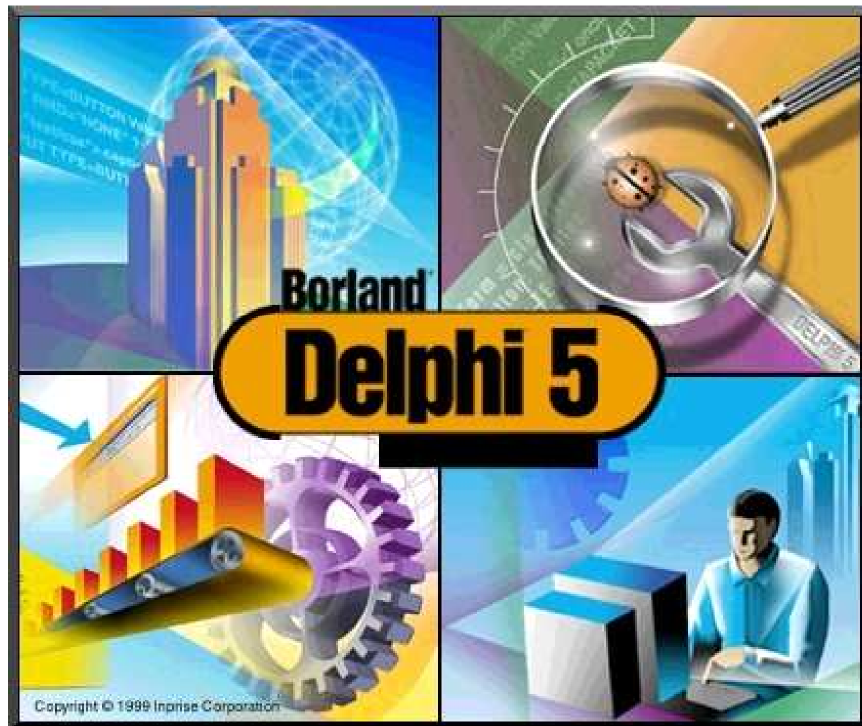
SERVIÇO NACIONAL DE APRENDIZAGEM COMERCIAL
Administração Regional em Minas Gerais



Delphi 5 – Fundamentos
01.04.01.05.218.1.06

Eduardo Ribeiro Felipe

BELO HORIZONTE
2000

Dia 1 ***Introdução, Ambiente Integrado (IDE).***	***Dia 2*** ***Projeto, Objetos, Propriedades, Eventos.***
Dia 3 ***VCL***	***Dia 4*** ***VCL, Object Pascal***
Dia 5 ***VCL, Object Pascal***	***Dia 6*** ***VCL, Diálogos***
Dia 7 ***VCL***	***Dia 8*** ***VCL***
Dia 9 ***VCL, Exceções***	***Dia 10*** ***Revisão, Exercícios***
Dia 11 ***Prova***	***Dia 12*** ***Introdução à Modelagem e Banco de Dados***

Dia 13

Banco de Dados

Dia 14

Banco de Dados

Dia 15

Banco de Dados

Dia 16

Banco de Dados

Dia 17

Banco de Dados

Dia 18

Banco de Dados

Dia 19

Banco de Dados

Dia 20

InstallShield

Dia 21

Transações

Dia 22

SQL & Delphi

Dia 23

Técnicas de Interface

Dia 24 e 25

Avaliação : 100 pts

Sumário

VERSÕES 11

IDE 12

O FORM DESIGN 12
A BARRA DE MENU PRINCIPAL 13
A PALETA DE COMPONENTES 13
A SPEEDBAR 14
OBJECT INSPECTOR 15
CODE EDITOR 16
CODE INSIGHT 17
SPEED MENUS 18
DESKTOPS TOOLBAR 18
CONFIGURAÇÕES DO AMBIENTE 19
Autosave Options 19
Compiling and running 19
Form designer 19

TECLAS IMPORTANTES 20

PROJETO EM DELPHI 21

.PAS E .DPR 23
SALVAR O PROJETO 23
ABRIR O PROJETO 24
OPÇÕES DE PROJETO 25
Forms 25
Application 25
Compiler 25
Linker 25
Directories/Conditionals 26
Version Info 26
Packages 26
A LISTA TO-DO 26

TIPOS DE COMPONENTES 26

Visíveis 26
Não-Visíveis 26

CONVENÇÃO DE NOMEAÇÃO 27

MANIPULANDO COMPONENTES 27

Utilizando o Object Inspector 28

MANIPULANDO EVENTOS 30

Construção de um manipulador de evento para o objeto button. 31
Executando a aplicação 32
COMENTÁRIOS 32
UM POUCO MAIS SOBRE EVENTOS 32

VCL 33

Objeto – Form (Formulário) 33
Objeto – Button1 (Botão) 35
Objeto – Edit (Caixa de edição) 35

Objeto – Label (Rótulo de orientação) *36*
Sugestão: Exercicio 1 .... *36*
MAIS SOBRE A PALETA STANDART 37
Objeto – Memo (Memorando) *37*
Objeto – CheckBox (Caixa de verificação) *37*
Objeto – RadioButton (Botão de 'radio') *38*
Objeto – ListBox (Caixa de listagem) *38*
Objeto – ComboBox1 (Caixa de listagem em formato de cortina) *39*
Objeto – GroupBox (Caixa de agrupamento) *39*
Objeto RadioGroup (Grupo de botões 'radio') *40*
Objeto – Panel (Painel) *40*
Objetos – MainMenu e PopupMenu (Menu principal e Menu rápido) *41*
Sugestão: Exercício 2 .... *41*

A LINGUAGEM OBJECT PASCAL 41

O MÓDULO .DPR 41
AS UNITS 42
Cabeçalho *43*
Interface *43*
Implementação *43*
Inicialização *44*
Finalização *44*
ATRIBUIÇÃO 44
DECLARAÇÃO DE VARIÁVEIS 44
TIPOS PARA MANIPULAÇÃO DE VARIÁVEIS 45
Tipos de variáveis Inteiras *45*
Tipos de números Reais *45*
Tipos de variáveis booleanas *45*
Tipos de variáveis de caracteres *45*
Tipo genérico (Variant) *45*
FUNÇÕES DE CONVERSÃO E MANIPULAÇÃO 46
EXPRESSÕES LÓGICAS 46
COMANDO IF 47
COMANDO CASE 47
COMANDO REPEAT 48
COMANDO WHILE 48
COMANDO FOR 48
COMANDO BREAK 48
COMANDO WITH 49
Sugestão: Exercício 3 .... *49*
PROCEDURES E FUNÇÕES 49
DECLARAÇÃO E ATIVAÇÃO DE PROCEDIMENTO 49
DECLARAÇÃO E ATIVAÇÃO DE FUNÇÕES 50
DECLARAÇÕES CRIADAS AUTOMATICAMENTE PELO DELPHI 51
CAIXAS DE DIÁLOGO 51
ShowMessage *51*
MessageDlg *51*
Application.MessageBox *52*
CAIXAS DE ENTRADA 53
InputBox *53*
InputQuery *54*
Exemplo *54*
Sugestão: Exercício 4 .... *55*

CHAMADA DE FORMS 55

COMPONENTES (VCL) ... **56**
Objeto – BitBtn (Botão com figuras opcionais) ... *56*
Objeto – SpeedButton (Botão para barra de ícones) ... *57*
Objeto MaskEdit – (Caixa de edição com máscara) ... *57*
Objeto – Image (Imagem) ... *58*
Objeto - PageControl ... *58*
Objeto – OpenDialog (Caixa de diálogo para abertura de arquivos) ... *59*
Sugestão: Exercício 5 ... *59*
Objeto – ImageList (Lista de imagens) ... *59*
Objeto – RichEdit (Texto com formatação) ... *60*
Objeto – ProgressBar (Barra de progresso) ... *60*
Objeto – Gauge (Barra de progresso) ... *61*
Objeto – Animate (Animações) ... *61*
Sugestão: Exercício 6 ... *61*
Objeto – DateTimePicker (Data e hora através de uma Combobox) ... *62*
Objeto – MonthCalendar (Calendário mensal) ... *62*
Objeto – StatusBar (Barra de status) ... *63*
Objeto – ToolBar (Barra de ícones) ... *63*
TRATAMENTO DE EXCEÇÕES ... **64**
O COMANDO TRY-EXCEPT ... 64
A CONSTRUÇÃO ON-DO ... 65
O COMANDO TRY-FINALLY ... 66
CLASSES BÁSICAS ... 67
BLOCOS TRY ANINHADOS ... 67
TRATAMENTO DE EXCEÇÕES DE FORMA GLOBAL ... 68
TRATAMENTO DE EXCEÇÕES SILENCIOSAS ... 69
Sugestão: Exercício 8 ... *69*
UM POUCO MAIS SOBRE COMPONENTES (VCL) ... **70**
Objeto – Timer (Temporizador) ... *70*
Objeto – FileListBox (Caixa de listagem de arquivos) ... *70*
Objeto – DirectoryListBox (Caixa de listagem de diretórios) ... *71*
Objeto - DriveComboBox (Caixa de listagem de drives) ... *71*
Objeto – FilterComboBox (Caixa de listagem de filtros) ... *71*
BANCO DE DADOS ... **72**
MODELAGEM BÁSICA ... 72
Modelo Conceitual e Lógico ... *72*
Modelo Físico ... *73*
Relacionamentos ... *73*
Visão física do banco de dados ... *74*
CONECÇÃO AO BANCO DE DADOS ... **75**
BDE ... 75
COMPONENTES DE CONTROLE E ACESSO ... 75
Exemplo ... *76*
OBJETOS TFIELD ... 78
APLICAÇÃO EM BANCO DE DADOS ... **80**
DATABASE DESKTOP ... 80
BDE – CRIAÇÃO DO ALIAS ... 83
APLICAÇÃO UTILIZANDO BD EM DELPHI ... 84
FrmPrincipal ... *84*
DATA MODULE ... 84

FORMULÁRIO DE CADASTRO DE SETOR 86
FrmCadSetor *86*
MÉTODOS E PROPRIEDADES PARA MANIPULAÇÃO DE DADOS 89
FUNÇÕES DE CONVERSÃO 91
OS ESTADOS DE UM DATASET 91
FORMULÁRIO DE CADASTRO DE FUNCIONÁRIO 94
FrmCadFuncionario *94*
MÁSCARA PARA ENTRADA DE DADOS 96
INTEGRAÇÃO DE DADOS ENTRE AS TABELAS 97
ATRIBUINDO VALORES ATRAVÉS DE ONNEWRECORD 99
CONSISTINDO DADOS ATRAVÉS DE ONVALIDADE 100
FORMULÁRIO DE CADASTRO DE DEPENDENTES 100
FrmCadDependente *100*
ATRIBUINDO VALORES DEFAULT (ONNEWRECORD) 105
MÉTODOS DE PESQUISA 105
FORMULÁRIO DE CONSULTA DE FUNCIONÁRIOS 105
FrmConFuncionario *105*
DEFININDO CAMPOS REQUERIDOS E EXCEÇÃO LOCAL 108
EXCLUSÃO COM CONSISTÊNCIA 108
EXCLUSÃO EM CASCATA 110
UM POUCO MAIS SOBRE CONSULTAS 111
FrmConSetFun *111*
RELATÓRIOS 113
FrmRelFunc *113*
CAMPOS CALCULADOS 116
RELATÓRIO MESTRE-DETALHE 118
FrmRelSetFun *118*
CONSULTAS E IMPRESSÃO 121

INSTALLSHIELD 122

Set the Visual Design *124*
Specify Components and Files *126*
Specify InstallShield Objects for Delphi 5 ..... *128*
Select User Interface Components *130*
Specify Folders and Icons *131*
Run Disk Builder *132*

INSTALLSHIELD PACKAGEFORTHEWEB 135

TRANSAÇÕES 141

DELPHI & TRANSAÇÕES 141

DELPHI & SQL 146

TÉCNICAS DE INTERFACE 150

SPLASH SCREEN 150
CRIANDO E DESTRUINDO FORMS 153
MANIPULANDO CURSORES 154

TOOLBAR 155

APLICAÇÕES MDI 159

Criando um lista das janelas abertas *163*
Mesclando Menus *163*
Reaproveitamento de código *164*

EXERCÍCIOS **165**
EXERCÍCIO 1 165
EXERCÍCIO 2 167
EXERCÍCIO 3 170
EXERCÍCIO 4 170
EXERCÍCIO 5 174
EXERCÍCIO 6 178
EXERCÍCIO 7 179
EXERCÍCIO 8 181
EXERCÍCIO 9 181
EXERCÍCIO 10 183
EXERCÍCIO 11 184
EXERCÍCIO 12 184
DICAS **186**
CRIAR UM HOTLINK 186
ENVIAR UM MAIL 186
EXECUTANDO UM PROGRAMA DE ACORDO COM A EXTENSÃO DE UM ARQUIVO 186
COMO SABER QUAL BIBLIOTECA INSERIR NA USES? 187
DATAS 187
SAIR DE UMA APLICAÇÃO 187
REDUZINDO TESTES `IF` 188
HINTS COM DUAS OU MAIS LINHAS 188
SUBSTITUIR A TECLA TAB POR ENTER NA MUDANÇA DE FOCO 188
ÍCONES 189
EXECUTAR UM PROGRAMA 189
LINKS 189
Nacionais *189*
Internacionais *189*
VIRTUAL KEYS 189
CONFIRMAÇÃO DE GRAVAÇÃO DE REGISTROS EM PARADOX 192
DISPARANDO SONS DO SISTEMA (MULTIMÍDIA) 192
LIMITE EM TABELAS PARADOX 192
GARANTINDO UMA INSTÂNCIA DO APLICATIVO 193
PERMITIR APENAS NÚMEROS EM UM TEDIT 193
APÊNDICE **194**
INTRODUÇÃO À ORIENTAÇÃO A OBJETOS 194
REFERÊNCIA BIBLIOGRÁFICA **197**

VERSÕES

O Delphi é distribuído em três versões:

Standart
Versão básica do produto que disponibiliza um conjunto bem limitado de componentes.

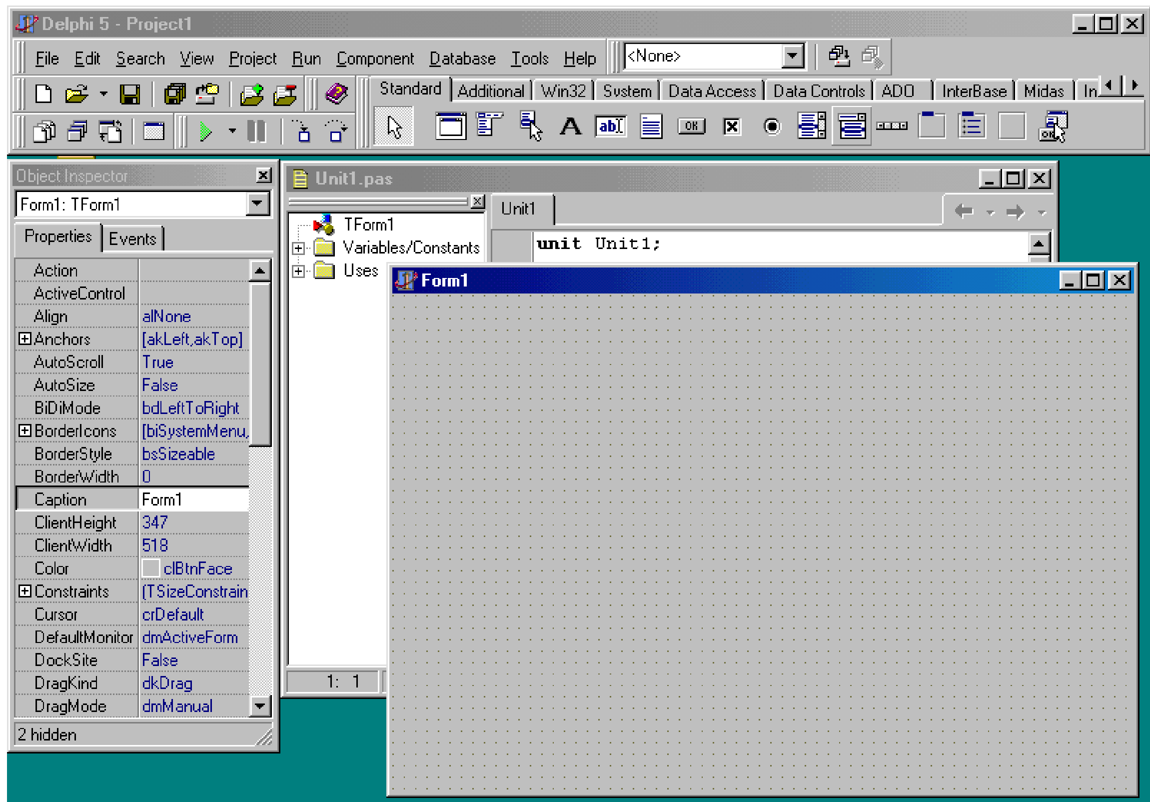
Professional
Tem por objetivos programadores profissionais. Possui de todos os recursos básicos, suporte à programação de banco de dados, suporte a servidores Web (WebBroker) e algumas ferramentas externas.

Enterprise
Conhecida como *Client/Server* nas versões anteriores; enfoca programadores que estão construindo aplicativos empresariais. Possui o SQL Links para conexões Cliente/Servidor BDE nativas, vários componentes para Internet, componentes ADO e InterBase Express, suporte a aplicativos multiusuários, internacionalização e arquitetura de três camadas.

Todas as versões possuem a mesma definição de ambiente integrado e a capacidade de expandir sua VCL original.

IDE

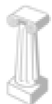
O ambiente de desenvolvimento do Delphi é composto de várias 'partes' compondo um conjunto integrado de 'janelas' que interagem entre si.



Vamos abordar cada uma separadamente:

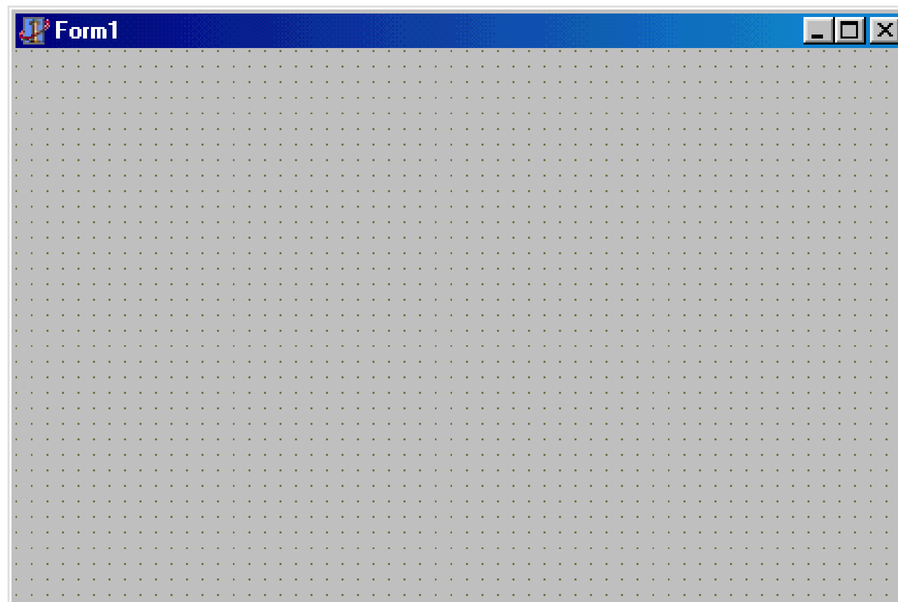
O FORM DESIGN

Form é o termo utilizado para representar as *janelas* do Windows que compõem uma aplicação. Os forms servem como base para o posicionamento dos *componentes*, que são responsáveis pela interação entre usuário e máquina.



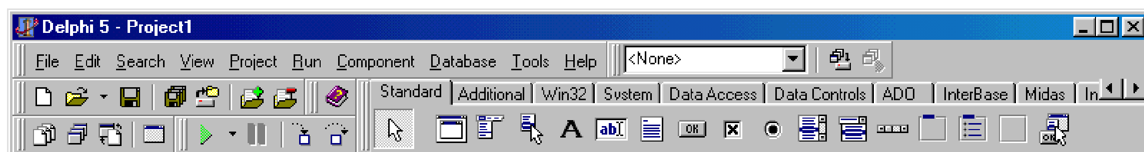
Para *selecionarmos* o form devemos clicar (uma vez) em sua área interna ou na object inspector, e não simplesmente em seu título.

As características iniciais do form como tamanho, botões (minimizar, maximizar, fechar, controle) e ícone podem (e serão) ser modificadas através de recursos que veremos adiante.



A BARRA DE MENU PRINCIPAL

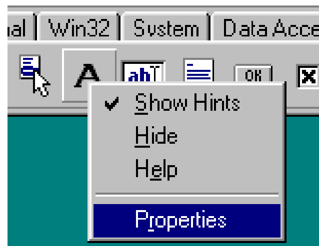
Como todo programa padrão Windows, há uma janela onde estão situados os *menus* da aplicação, a barra que contem os menus também agrupa outras partes.



A PALETA DE COMPONENTES

Aplicativos orientados a objetos trabalham com elementos que denominamos *componentes*. No Delphi, os componentes encontram-se em uma paleta com várias *guias*.

Pode-se configurar a ordenação das *guias* clicando com o **botão direito** do mouse sobre qualquer componente e clicar na opção **Properties**.



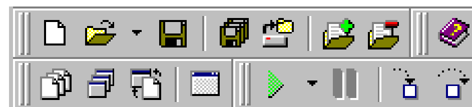
Há basicamente três maneiras de **inserirmos os componentes** no formulário:

- Clicar uma vez no componente, e clicar dentro do formulário (*não* arrastar *para* o form).
- Clicar duas vezes rapidamente no componente desejado.
- Segurar a tecla *Shift* e clicar no componente desejado; clicar no form várias vezes.

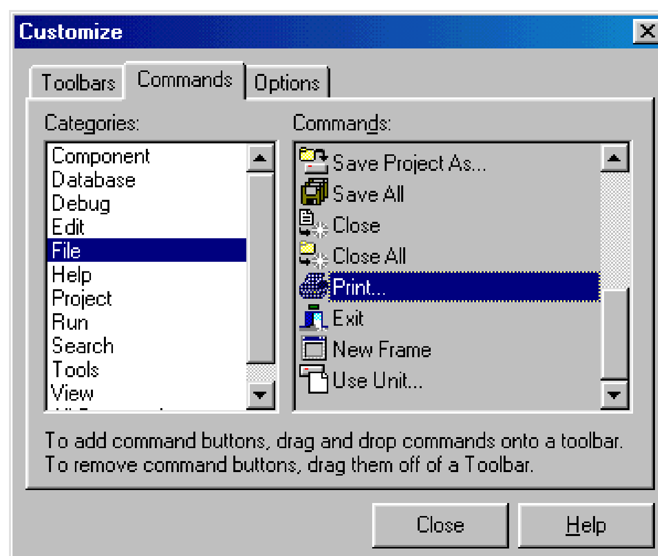
Na terceira opção, o componente será '*travado'* ao mouse. Para '*destravá-lo*' clique no ícone da *seta*, o primeiro ícone da paleta.

A SPEEDBAR

A speedbar está posicionada ao lado esquerdo da barra principal do Delphi. Possui diversos botões (ícones) que representam comandos muito utilizados durante o desenvolvimento.



Pode-se customizar a speedbar adicionando ou retirando algum botão através do botão direito em qualquer ícone (da speedbar) e escolher o comando **customize**. Na janela aberta, seleciona-se a guia **Commands**. Neste momento pode-se arrastar nos dois sentidos, para adicionar ou retirar botões.



OBJECT INSPECTOR

Uma das 'partes' mais importantes da orientação a objeto é a possibilidade de definir características personalizadas aos componentes.

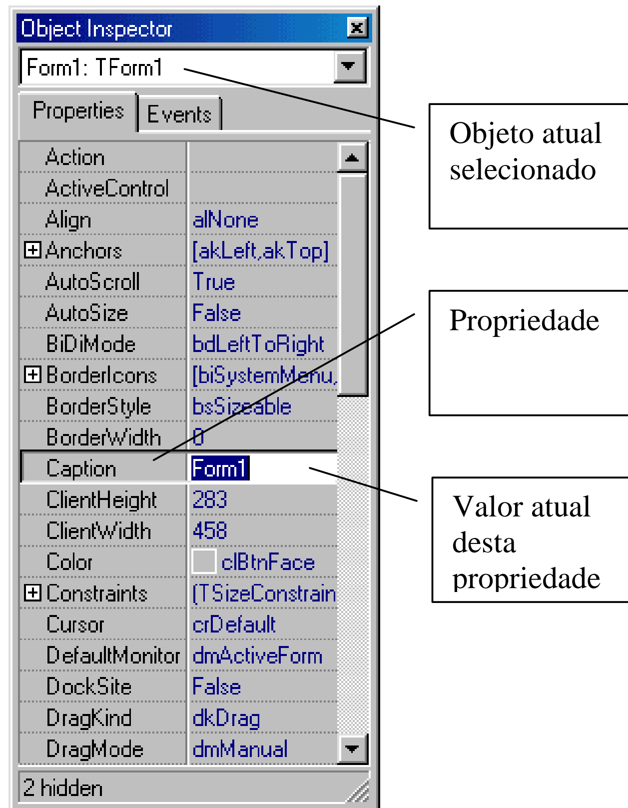
No Delphi, utilizamos a janela object inspector para realizar esta tarefa.

Há uma *caixa de listagem*[1] que permite a escolha de qual componente deverá ser selecionado.

Duas *guias*:

Properties – Define as propriedades e valores do Objeto selecionado.

Events – Define quais os eventos serão manipulados pelo desenvolvedor.



Algumas ***propriedades*** trazem opções diferenciadas para alteração.

Por exemplo:

Caption – Permite a inserção de uma string de caracteres.

Color – Permite a inserção de um dos valores *pré-definidos* na caixa de listagem.

BorderIcons – Toda propriedade que possui o sinal de + tem a característica de mostrar *subpropriedades*. Deve-se clicar no sinal de + para expandir e no sinal de – para ocultar.

Icon – Exibe um botão de reticências (...) que dará origem a uma caixa de diálogo.

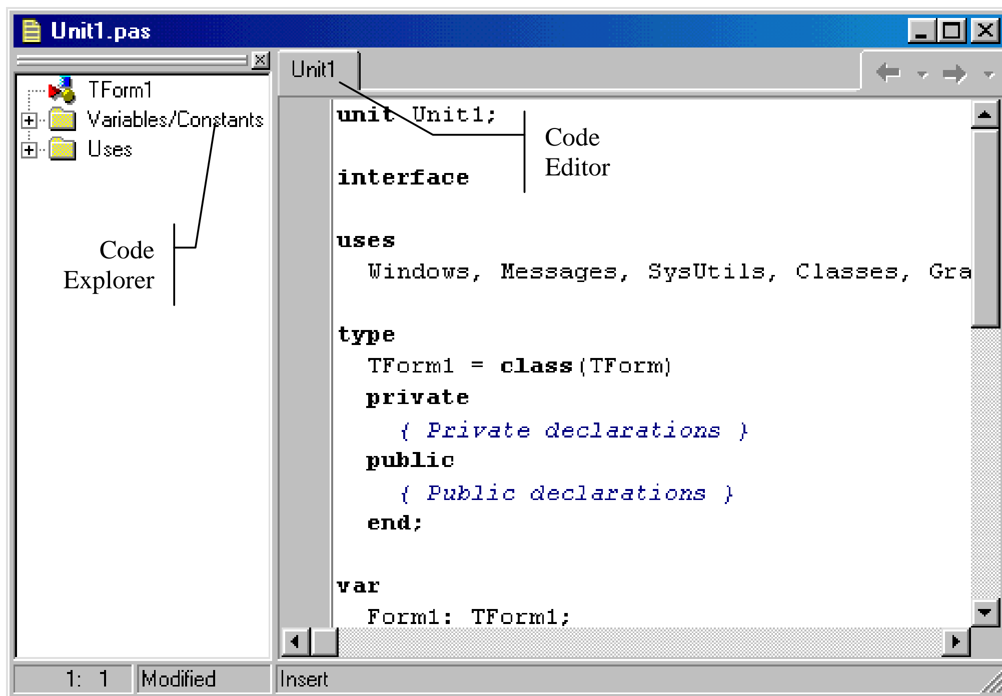
Os nomes definidos como valores das propriedades na object inspector serão os nomes usados na construção do código em Object Pascal.

[1] Componente ComboBox.

CODE EDITOR

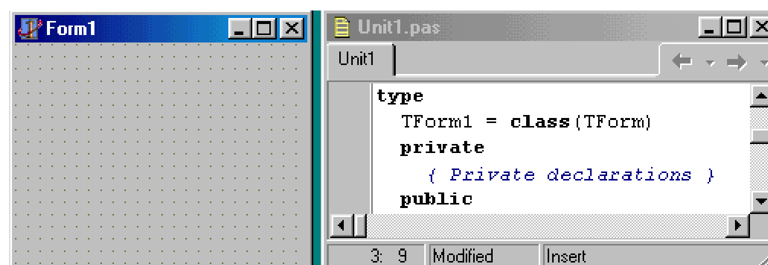
O editor de código é responsável por receber todas as declarações criadas pelo Delphi e *handlers*[2] criados pelo desenvolvedor.

E no ambiente Code Editor que implementamos o algoritmo na linguagem Object Pascal.

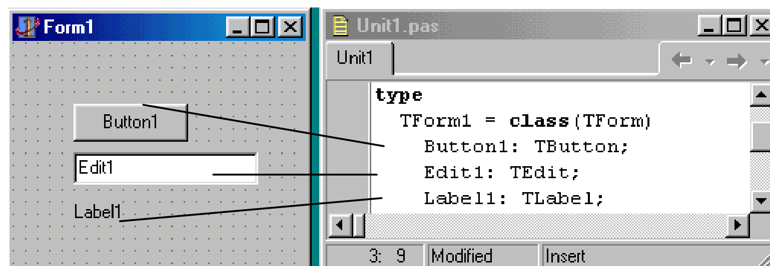


Na janela do editor *pode* haver uma outra janela denominada ***Code Explorer***. É a parte esquerda da janela, onde podemos ter uma orientação sobre os objetos, procedimentos, funções e classes utilizadas na aplicação. Para desligar o *code explorer* clique no pequeno X ao lado da guia do *code editor*, para visualiza-lo clique com o botão direito dentro do editor e escolha **View Explorer** ou pelo teclado Ctrl+Shift+E.

Uma característica muito importante do *Code Explorer* é que quando inserirmos componentes no form, a sua declaração é feita pelo *Delphi* de maneira automática.



[2] Manipulador de eventos.



Podemos considerar também o seguinte fato:

‘Tudo o que o *Delphi* escrever, é problema *dele*’.

Agora, ‘Tudo o que *você* escrever é problema *seu*’.

Problema no sentido de atualização de código, como veremos adiante. Quando alteramos o nome do objeto, deve-se utilizar o mesmo nome nas rotinas implementadas. Agora, o que o Delphi declarou, ele se encarrega de atualizar.

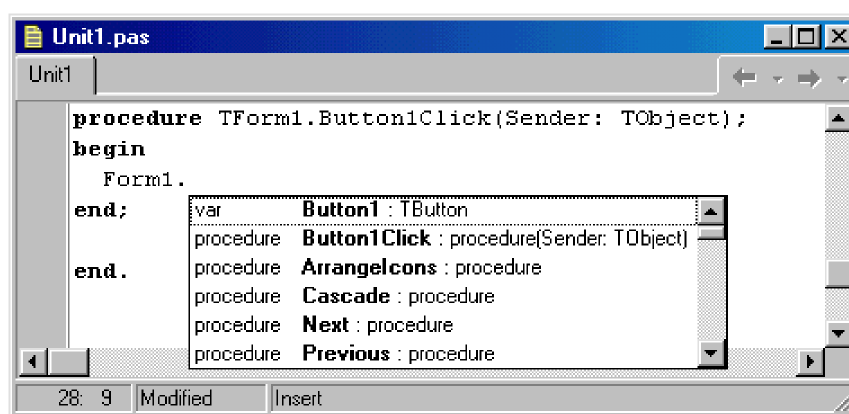
Pode-se personalizar o Editor através do menu **Tools** | **Editor Options**.

CODE INSIGHT

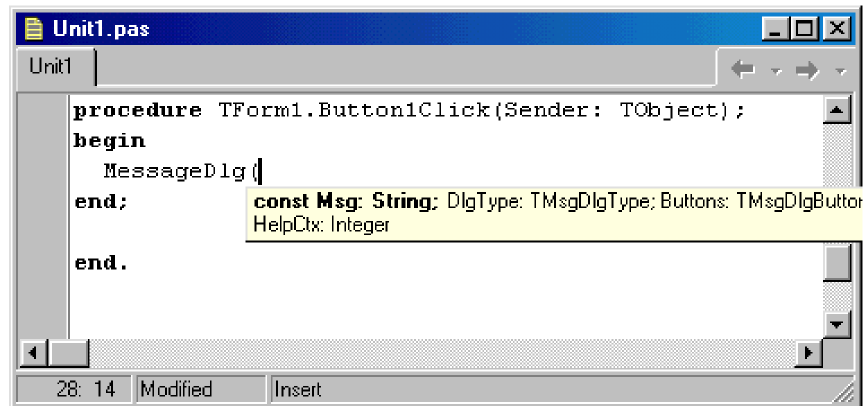
Um recurso que vai facilitar nossa vida no momento de desenvolvimento de código é o ***Code Insight*** do *Code Editor* do Delphi.,

Ele atua como um *ajudante de complemento* junto ao código do desenvolvedor. Por exemplo, ao digitar o nome de um objeto seguido de ponto (.) abre-se uma listagem de *métodos* e propriedades que podem ser utilizadas neste objeto.

Esta lista pode ser ordenada por nome, clicando com o *botão direito* dentro da listagem.



No momento de chamada de procedimentos ou métodos:



Para forçar o ***code insight*** em determinada situação, utiliza-se:

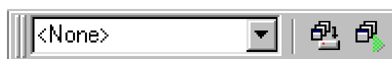
- Ctrl + Barra de Espaço - Para complemento de objetos; seus métodos e propriedades.
- Ctrl + Shift + Barra de Espaço – Para complemento de parâmetros.

SPEED MENUS

Speed Menus ou 'Menus Rápidos' é a característica de podermos selecionar comandos 'rápidos' através do botão direito do mouse. Em várias situações (já citadas anteriormente) utilizamos o botão direito para escolher algum comando ou ação.

DESKTOPS TOOLBAR

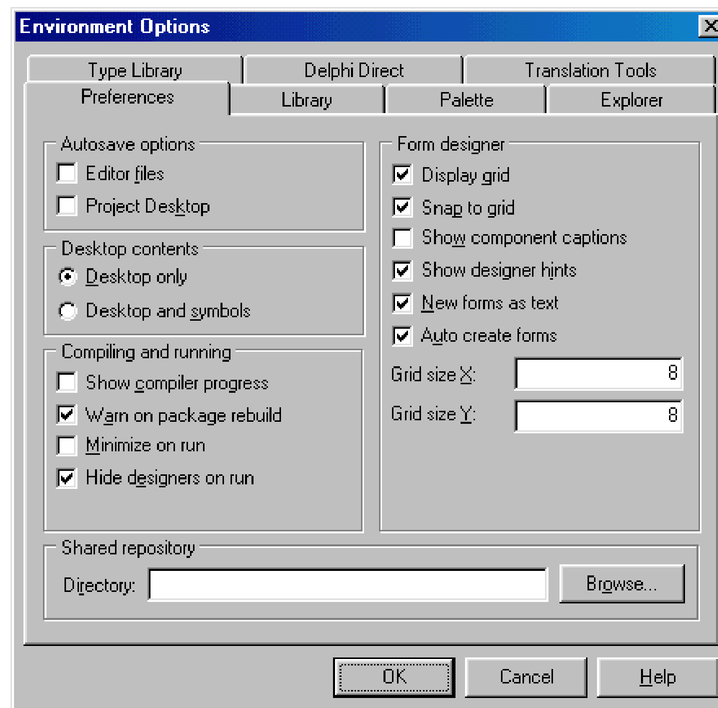
Este novo recurso permite gravar vários layouts[3] de tela personalizando seu ambiente de trabalho.
Estas opções (layouts) podem ser manipuladas pelos ícones ou pelo menu: **View - Desktops**



[3] Disposições das janelas no monitor.

CONFIGURAÇÕES DO AMBIENTE

O Delphi permite que você personalize o ambiente através do menu **Tools | Environment Options**.



Algumas opções da janela *Environment Options* que a princípio, podemos julgar importantes:

Autosave Options

Editor files – Grava os arquivos fonte (.PAS) no momento da compilação, evitando perda de código em caso de travamento da máquina. Porém, não permite compilar um determinado projeto sem salva-lo antes.
Project Desktop - Grava a posição das janelas do projeto atual.

Compiling and running

Minimize on run – Para minimizar o Delphi no momento da compilação em efeito de testes. Evita confusões de janelas.

Form designer

New forms as text –Para tornar ***compatível*** os arquivos *de definição de formulário (.DFM)* criados no ***Delphi5*** para o ***Delphi4***, desligue esta opção.

TECLAS IMPORTANTES

Tecla	*Função*
F12	Alterna entre o *code editor* e o *form designer*.
F11	Alterna entre o *code editor*, *form designer* e a *oject inspector*.
F10	Torna o foco para a janela *principal.*
F9	(RUN) Permite *compilar* e *executar* o projeto para testes. Este processo gera *automaticamente* o arquivo .EXE no diretório onde foi gravado o arquivo de projeto (.DPR).
CTRL + F9	Permite *compilar* o projeto *sem* executar. Ideal para conferência de código.
SHIFT + F12	Permite alternar entre os formulários do projeto. Equivalente ao ícone **View Form** na SpeedBar.
CTRL + F2	Permite 'destravar' o Delphi em caso de testes onde ocorram *exceções*, como veremos mais adiante.

PROJETO EM DELPHI

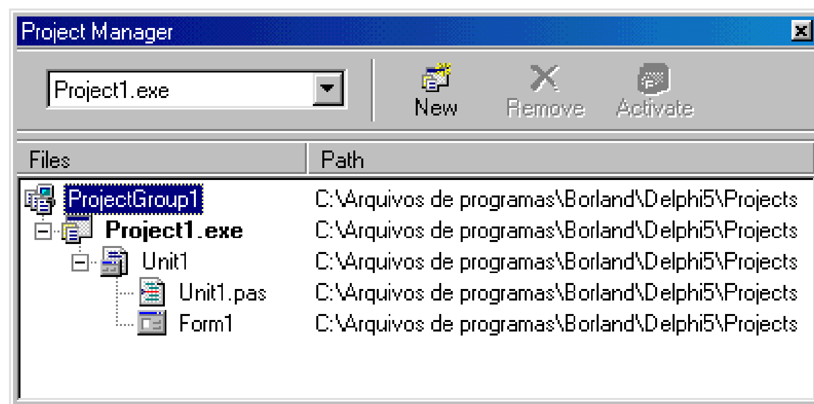
O conceito de projeto em Delphi é baseado em um conjunto de arquivos necessários para gerar uma aplicação.

Vamos destacar os principais arquivos:

Extensão	*Tipo e descrição*	*Criação*	*Necessário para compilar?*
.PAS	Arquivo Pascal: o código-fonte de uma unidade Pascal, ou uma unidade relacionada a um formulário ou uma unidade independente.	Desenvolvimento	Sim.
.DPR	Arquivo Delphi Project. (Contém código-fonte em Pascal.)	Desenvolvimento	Sim.
.DFM	Delphi Form File: um arquivo binário (na versão 5 pode ser convertido para texto) com a descrição das propriedades de um formulário e dos componentes que ele contém.	Desenvolvimento	Sim. Todo formulário é armazenado em um arquivo PAS e em um arquivo DFM.
.DCU	Delphi Compiled Unit: o resultado da compilação de um arquivo Pascal.	Compilação	Apenas se o código-fonte não estiver disponível. Os arquivos DCU para as unidades que você escreve são um passo intermediário; portanto, eles tornam a compilação mais rápida.
.BMP, .ICO, .CUR	Arquivos de bitmap, ícone e cursor: arquivos padrão do Windows usados para armazenar imagens de bitmap.	Desenvolvimento: Image Editor	Normalmente não, mas eles podem ser necessários em tempo de execução e para edição adicional.
.CFG	Arquivo de configuração com opções de projeto. Semelhante aos arquivos DOF.	Desenvolvimento	Necessário apenas se opções de *compilação* especiais foram configuradas.
.DOF	Delphi Option File: um arquivo de texto com as configurações atuais para as opções de projeto.	Desenvolvimento	Exigido apenas se opções de *compilação* especiais foram configuradas.

.DSK	Arquivo de Desktop: contém informações sobre a posição das janelas do Delphi, os arquivos abertos no editor e outros ajustes da área de trabalho.	Desenvolvimento	Não. Você deve excluí-lo se copiar o projeto em um novo diretório.
.EXE	Aquivo executável: o aplicativo Windows que você produziu.	Compilação: Ligação (linking)	Não. Esse é o arquivo que você vai distribuir. Ele inclui todas as unidades compiladas, formulários e recursos.
.~PA	*Backup* do arquivo Pascal Pode ser ativado ou desativado através do Menu **Tools – Editor Options -** *guia display* – Item: Create backup file.	Desenvolvimento	Não. Esse arquivo é gerado automaticamente pelo Delphi, quando você salva uma nova versão do código-fonte.
.TODO	Arquivo da lista **to-do**, contendo os itens relacionados ao projeto inteiro.	Desenvolvimento	Não. Esse arquivo contém notas para os programadores.

O Delphi possui um mecanismo de *gerência de arquivos de projeto* informando os *principais* arquivos e seu *path*. Clique em **View – Project Manager**



A figura acima é um exemplo de um projeto inicial, ainda não salvo. O diretório padrão para criação dos arquivos é *projects*, obviamente devemos definir na gravação pasta e nomes de arquivos mais específicos.

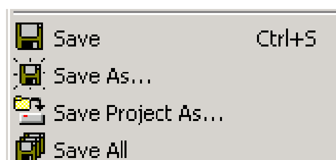
.PAS E .DPR

Para visualizarmos o código fonte da unidade (.PAS) em Delphi basta selecionarmos o *code editor* (F12).
Para visualizarmos o código fonte no arquivo de projeto (.DPR) basta selecionarmos o menu **Project – View Source**. O arquivo de projeto é exibido em uma *guia* no *code editor*.
Para fechar a guia basta clicar com o botão direito e escolher *close page*.

Não feche as janelas através do **X** (botão fechar do Windows)

SALVAR O PROJETO

Como vimos anteriormente, o conceito de projeto em Delphi se faz através de um conjunto de arquivos. No Menu **File** do Delphi temos quatro opções para a gravação do projeto:

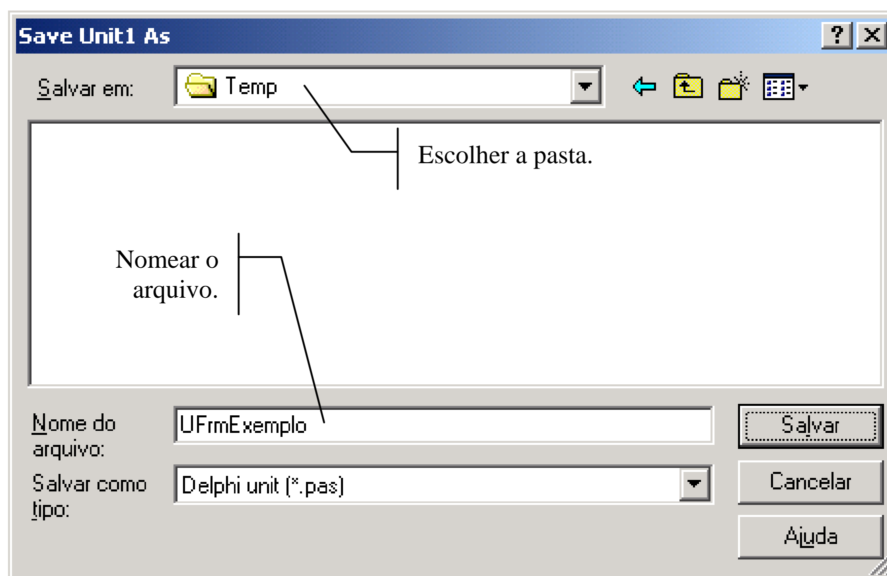


Onde:

Comando	*Objetivo*
Save	Salvar apenas a unidade selecionada
Save As...	Salvar a unidade selecionada como... pode-se renomear ou trocar de pasta (duplicando) o arquivo.
Save Project As...	Salvar o projeto como… pode-se renomear ou trocar de pasta (duplicando) o arquivo.
Save All	Grava todos os arquivos do projeto, e atualiza-os caso já sejam salvos.

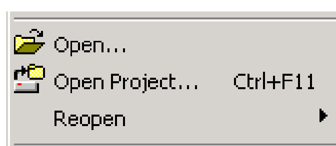
Ao clicar em Salve All abre-se uma caixa de diálogo padrão do Windows onde deve ser preenchido o nome do arquivo e escolhida uma pasta para armazenar o projeto.

Observe o título da janela, pois após a gravação do ‘arquivo da unidade’, será exibida a mesma caixa (com título diferente) para a gravação do ‘arquivo de projeto’.



ABRIR O PROJETO

O projeto em Delphi é determinado através do arquivo com extensão .DPR. Desta forma, para abrir um projeto, deve-se abrir o arquivo .DPR. No menu **File** do Delphi podemos utilizar mais de uma opção:



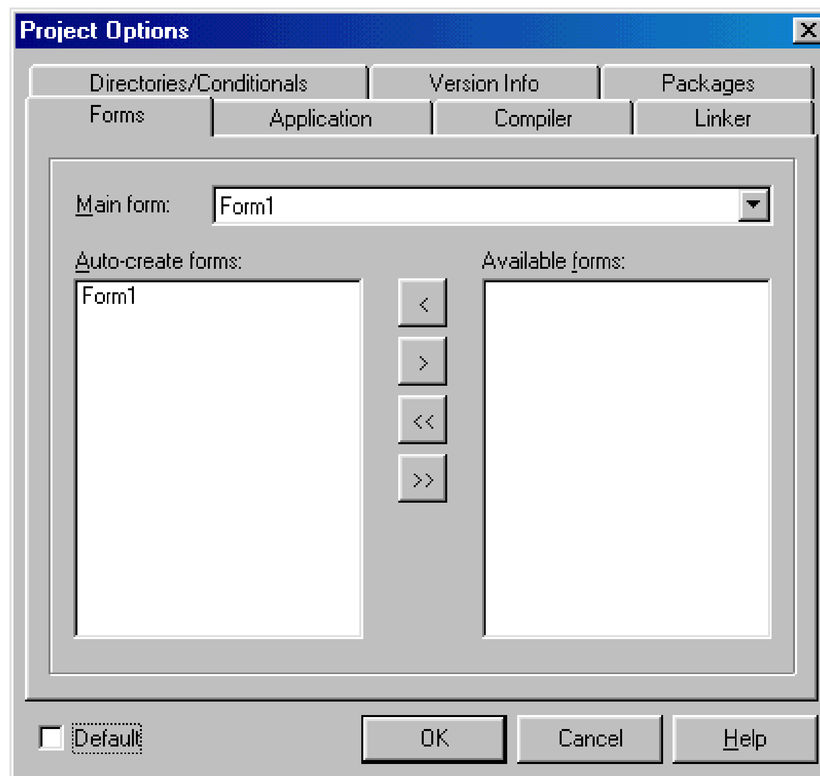
Onde:

Comando	*Objetivo*
Open	Permite abrir um arquivo .DPR, .PAS entre grupos de projeto.
Open Project...	Permite abrir um arquivo de projeto.
Reopen	Permite reabrir um arquivo (DPR ou PAS) utilizado anteriormente.

Não abra um arquivo .PAS através destas opções, a menos que saiba o que está fazendo. Os arquivos .PAS devem ser abertos através do menu **View | Units** após a abertura do .DPR.

OPÇÕES DE PROJETO

O Delphi permite a configuração de vários itens do sistema através do menu **Project – Options**.



Forms

- Main form - Nesta guia permite a escolha do formulário *principal* da aplicação.
- Available form - Os formulários *available* (disponíveis) em caso de criação em tempo de execução.

Application

- Title - Define um nome para a sua aplicação diferente do nome do arquivo de projeto (.DPR).
- Help file – Define o nome do arquivo de Help (.HLP) associado à aplicação.
- Icon – Define o ícone utilizado no arquivo executável. (.EXE)

Compiler

- Estas opções permitem especificar uma *compilação personalizada*, ou seja, cada projeto pode ser compilado com uma característica.

Linker

- Estas opções incluem informações para a depuração.

Directories/Conditionals

- Nesta guia pode-se configurar o diretório de saída para os arquivos gerados pela aplicação.

Version Info

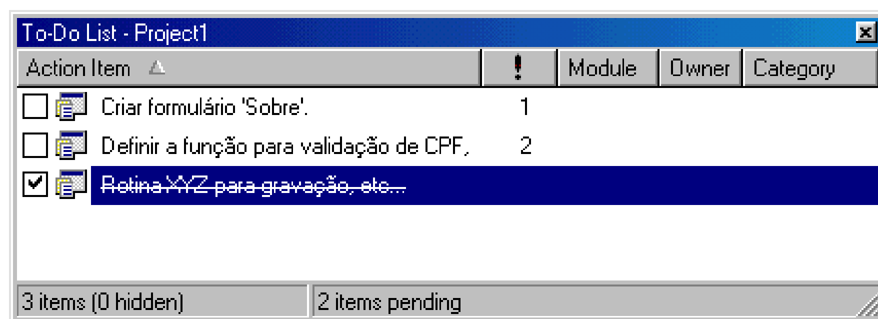
- Estas informações podem ser visualizadas no Windows através do menu rápido do mouse no arquivo executável.

Packages

- Os packages permitem um controle de distribuição através de DLL's básicas *externas* ao executável entre outros recursos.

A LISTA TO-DO

O Delphi adicionou um recurso de controle/gerenciamento de projeto na versão 5, denominado ***To-Do List***. Você pode incluir incluir ou modificar itens a serem feitos através de diálogos que podem ter dois estados, *prontos* ou à *fazer*.
Para utilizar a **To-Do List** clique no menu **View** | **To-Do List**.



Clique com o botão direito dentro da janela e escolha **Add**.
O arquivo gerado gerado pela lista é gravado no diretório do projeto com a extensão *.todo*

TIPOS DE COMPONENTES

Fazemos uma distinção de duas categorias básicas quando tratamos dos componentes, são: Componentes Visíveis e Componentes Não-Visíveis.

Visíveis

Quando um componente pode ser visto pelo usuário em tempo de execução. Exemplo Button e Edit.

Não-Visíveis

Alguns componentes aparecem no form durante o tempo de projeto na aparência de um ícone, mas não podem ser vistos pelo usuário em tempo de execução. Exemplo: Timer e MainMenu.

CONVENÇÃO DE NOMEAÇÃO

A propriedade mais importante de um componente é a propriedade *Name*. É ela que define o nome *interno* com relação ao código escrito em Object Pascal. Para organizar e facilitar o processo de desenvolvimento/manutenção do sistema, grande parte dos desenvolvedores adota uma nomenclatura para tornar o código mais legível possível.

O Delphi adota como nomenclatura padrão o nome da classe da qual o componente é *instanciado* e um número crescente de acordo com o número de ocorrência deste componente no form. Exemplo: Button1, Button2, etc... são componentes *instanciados* da classe TButton .

Não é obrigatória a utilização da convenção de nomes utilizados nesta apostila, mas *é muito importante* fazer uso de uma convenção mais clara possível.

Exemplo:

Nome gerado pelo Delphi	***Objetivo***	***Convenção***
Button1	Sair da aplicação	BtnSair
Edit1	Receber nome do usuário	EdtNome
Label1	Indicar componente Edit	LblNome

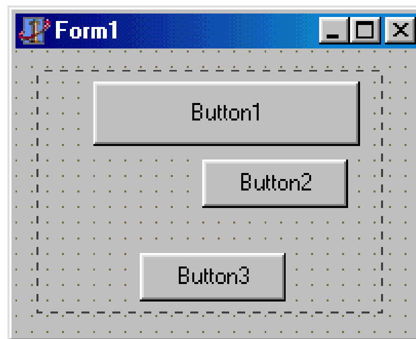
MANIPULANDO COMPONENTES

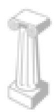
Podemos adicionar os componentes ao formulário de três maneiras, (citadas anteriormente) e utilizar ferramentas e técnicas de alinhamento para aumentar nossa produtividade.

Para selecionar um componente, basta clicá-lo uma vez ou na object inspector selecioná-lo na caixa de listagem.
Pode-se então arrastá-lo com o Mouse ou utilizar as teclas CTRL+SETAS para mover o componente. As teclas SHIFT+SETAS alteram a largura e altura.

Para selecionar mais de um componente ao mesmo tempo, utiliza-se a tecla SHIFT, pode-se mover ou alterar o conjunto.

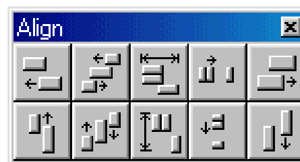
O recurso de *arrastar e selecionar* (Paint, por exemplo) é válido quando a base é o Form. Quando inserirmos componentes em cima de outro objeto (Panel, por exemplo) é necessário segurar a tecla CTRL no processo de arrastar.



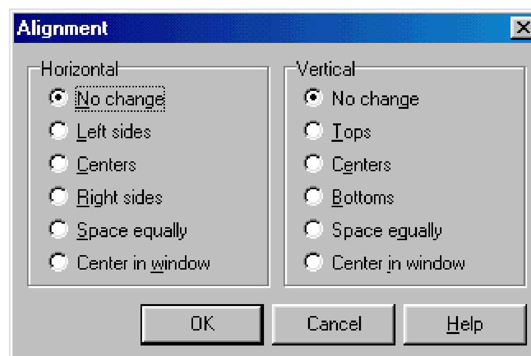


Para definir vários componentes baseados em uma propriedade de outro, altere o componente '*modelo*', selecione-o primeiro e com SHIFT selecione os outros. Na object inspector selecione a propriedade a ser definida para todos Width (largura, por exemplo) e aperte a tecla ESC.

O Delphi dispõe de uma ferramenta para auxílio ao alinhamento dos componentes. Clique no menu **View – Alignment Palette**

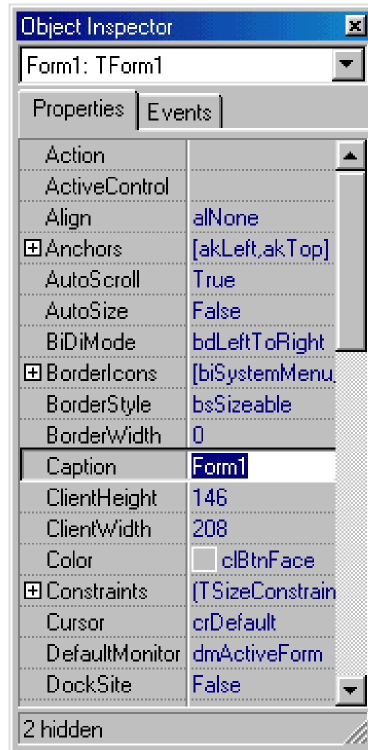


Uma outra opção é clicar sobre o componente selecionado e no Speed Menu (Menu rápido) selecionar **Align**.



Utilizando o Object Inspector

O object inspector é uma janela importantíssima na programação orientada a objetos, é através dela que podemos alterar as propriedades e definir os eventos de acordo com o objetivo da aplicação.

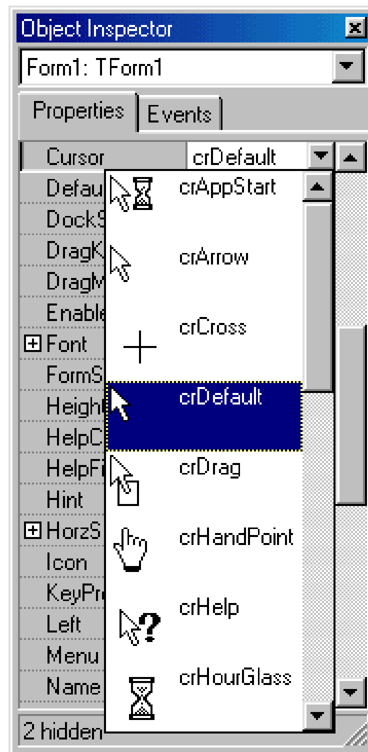


Na parte superior da janela há uma caixa de listagem que permite a seleção de componentes já inseridos no formulário. Duas guias (*Properties e Events*) separam as listas de propriedades e eventos.

As propriedades são definidas através de tipos. Podemos citar no exemplo com o objeto form:

Tipos Simples
São tipos String ou valores numéricos definidos ao digitar um valor na frente da propriedade.
Exemplo: Name, Caption, Height e Width entre outros.

Tipos Enumerados
São tipos definidos por uma quantidade limitada de opções que devem ser previamente selecionadas, não simplesmente definidas pelo usuário.
Exemplo: Cursor, BorderStyle e WindowState entre outros.



Tipo Set
Algumas propriedades podem conter múltiplos valores. Um exemplo é a propriedade BorderIcons com o sinal + indicando subpropriedades.

Tipos com Editor de Propriedades
As propriedades que são acompanhadas de um ícone de reticências (...) indicam que uma janela de diálogo irá auxiliar na escolha de seu(s) valor(es). Exemplo: Icon.

Manipulando Eventos

A guia *Events* permite o desenvolvedor definir um ***handler***[4] em Object Pascal para um determinado evento que pode ser disparado pelo usuário ou pelo sistema.

Um evento é uma ***ação*** disparada dentro de uma aplicação orientada a Objeto. Podemos citar as ocorrências dos principais eventos que são disponibilizados na maioria dos componentes em Delphi:

Evento	***Ocorrência***
OnClick	Quando o usuário clicar uma vez com o botão esquerdo do mouse sobre o componente.
OnDblClick	Quando o usuário dá um duplo clique no componente com o botão esquerdo do mouse.

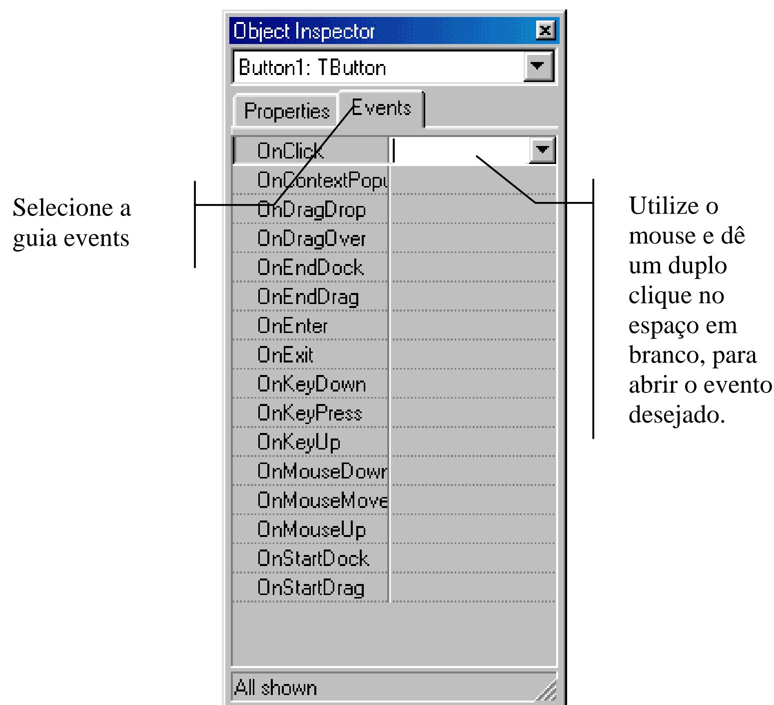
[4] Manipulador de evento

OnEnter	Quando o componente recebe o foco.
OnExit	Quando o componente perde o foco.
OnKeyPress	Quando pressiona uma única tecla de caractere.

Construção de um manipulador de evento para o objeto button.

- Insira ***um*** componente button no Form, não é necessário mudar nenhuma propriedade.
- Selecione a object inspector a guia *events* e localize o evento *OnClick*.
- Dê um duplo clique no espaço em branco do evento.

Os componentes possuem um evento '**padrão**' para a construção do código, por isso é possível clicar *no componente* duas vezes para abrir um evento.



No *Code Editor* é criada uma declaração do evento na cláusula *Interface* e a implementação do procedimento na cláusula *Implementation.*

Como veremos com mais detalhes nos próximos capítulos, todo código em object pascal é delimitado pelas palavras reservadas "**begin**" e "**end**".
Defina apenas as duas linhas de código dentro dos delimitadores.

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  Form1.Caption := 'Curso de Delphi';
  Showmessage('Exemplo de caixa de diálogo');
end;
```

Observe a construção do procedimento criado pelo próprio Delphi:

```
procedure TForm1.Button1Click(Sender: TObject);
begin

end;
```


Executando a aplicação

Para executar o programa e visualizar os dois comandos codificados no evento *OnClick* basta teclar **F9** ou o ícone **Run**.

COMENTÁRIOS

Os comentários no código fonte são importantes e podem ser feitos através dos seguintes símbolos:

```
//Comentário de linha
{ Comentário de bloco }
(*Comentário de bloco *)
```

UM POUCO MAIS SOBRE EVENTOS

A programação baseada em eventos (POE[5]), em resumo, tem a característica de obedecer as ações do usuário. Se você já programou em alguma linguagem para o sistema operacional MS-DOS sabe do que estamos falando.

É mais fácil programar com POE. Um programa estruturado difere em muito deste raciocínio porque seu escopo é rígido e baseado em rotinas, ou seja, pode haver (e na maioria há) momentos em que o usuário deve seguir determinados passos orientados pelo programa; enquanto que na POE existe a execução do evento associado à ação do usuário ou do sistema.

Há dois tipos básicos de eventos:

- ***Usuário*** – São eventos disparados pelo usuário, por exemplo: OnClick, OnKeyPress, OnDblClik.
- ***Sistema*** – São eventos que podem ocorrer baseados no sistema operacional, por exemplo: O evento OnTimer executa um procedimento a cada intervalo em milisegundos. O evento OnCreate ocorre quando uma *instância* do objeto está sendo criada.

É importante notar que o usuário pode disparar mais de um evento em uma única ação, na verdade isso irá ocorrer com freqüência, de maneira que devemos ter consciência que ***os eventos obedecem uma ordem***.

[5] Programação Orientada a Eventos

Supondo a existência de três manipuladores de eventos para um objeto da classe Tbutton: OnMouseDown, OnEnter e OnClick. A ordem destes eventos será:

- OnEnter
- OnMouseDown
- OnClick

Isso é uma observação importante no manuseio e na construção de manipuladores.

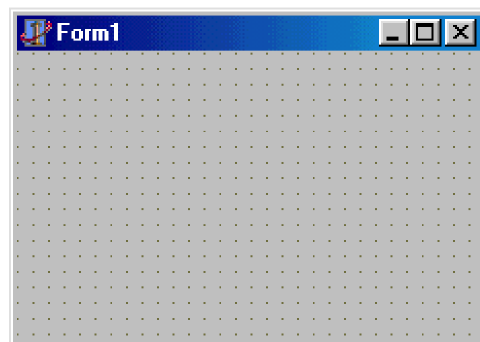
VCL

Vamos considerar alguns objetos e suas principais *propriedades e métodos*.

Objeto – Form (Formulário)

Paleta – Standart

Importância: É o principal componente *container* pois permite posicionar os demais componentes em si mesmo.
É literalmente a implementação do conceito de 'janelas' do sistema operacional Windows.



Propriedades

ActiveControl	Permite definir qual o *primeiro* componente a receber ***foco*** assim que o formulário é criado.
Align	Altera o alinhamento e preenchimento do objeto.
AutoScroll	Permite habilitar as barras de rolagem.
AutoSize	Determina se o controle será automaticamente redimensionado.
BorderIcons	Determina os ícones a serem exibidos na barra de título do formulário.
BorderStyle	Define o estilo da borda do formulário. bsDialog – Borda não redimensionável, comum em caixa de diálogo bsSingle – Borda simples e redimensionável. bsNone – Borda invisível, não redimensionável, sem botões de controle. bsSizeable – Borda padrão redimensionável.
BorderWidth	Define a espessura da borda.
Caption	Indica o rótulo exibido para o componente.
ClientHeight / ClientWidth	Define a altura e largura da área cliente.
Color	Define a cor de fundo de um componente.
Cursor	Indica a imagem exibida pelo ponteiro do mouse quando este ficar sobre o objeto.
DefaultMonitor	Associa o form a um monitor específico em uma aplicação que utiliza

	vários monitores.
Enabled	Define se o componente está habilitado ou não.
Font	Permite controlar os atributos do texto exibido em um componente.
FormStyle	Determina o estilo do formulário. fsNormal – Definição padrão do formulário. fsMDIChild – O formulário será uma janela-filha de uma aplicação MDI. fsMDIForm – O formulário será o formulário-pai de uma aplicação MDI. fsStayOnTop – O formulário permanece *sobre* todos os outros formulários do projeto, exceto aqueles que também têm a propriedade FormStyle igual a fsStayOnTop.
Height	Define a altura do objeto.
HelpContext	Define o tópico do arquivo help que será exibido ao pressionar a tecla F1.
HelpFile	Define um arquivo de help específico.
Hint	Permite exibir um texto de auxílio no momento em que o ponteiro do mouse permanece sobre o controle.
HorzScrollBar	Define o comportamento de uma barra de rolagem horizontal.
Icon	Define o ícone que será usado pelo formulário.
KeyPreview	Define se o formulário deve ou não responder a um pressionamento de tecla, através do evento OnKeyPress, por exemplo.
Left	Define a coordenada da extremidade esquerda de um componente.
Menu	Permite escolher entre mais de um componente MainMenu.
Name	Define o nome *interno* que identifica o componente dentro da aplicação.
PopupMenu	Define o componente PopupMenu a ser utilizado pelo objeto.
Position	Permite definir o tamanho e posição de um formulário no momento em que ele aparece na sua aplicação.
ShowHint	Define se a string de auxílio deve ou não ser exibida quando o usuário mantém o ponteiro do mouse sobre um controle.
Tag	A propriedade Tag é uma variável do tipo Longint que o Delphi coloca à disposição do usuário, que pode atribuir o significado mais conveniente.
Top	Define a coordenada da extremidade superior de um componente.
VertScrollBar	Define o comportamento de uma barra de rolagem vertical.
Visible	Define se o componente aparece ou não na tela.
Width	Define a largura do objeto.
WindowMenu	Permite definir qual o menu responsável por manipular as janelas-filhas de uma aplicação MDI.
WindowState	Define o estado de exibição de um formulário.

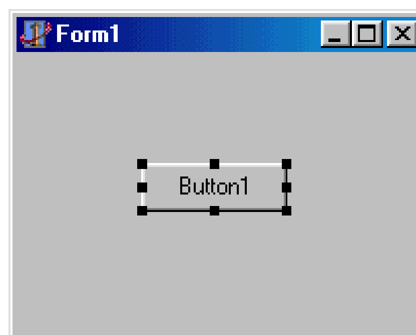
Métodos

Show	Exibe o formulário de manipulação não-modal.
ShowModal	Exibe o formulário de manipulação modal.
Close	Permite fechar o formulário.

Objeto – Button1 (Botão)

Paleta – Standart

Importância: É um dos objetos mais utilizados para confirmar e disparar rotinas associadas.



Propriedades

Action	Referencia uma ação definida em um objeto TActionList.
Anchors	Permite manter a posição relativa do objeto ao objeto 'parente' quando este é redimencionado.
Cancel	Associa o evento OnClick do objeto ao pressionamento da tecla Esc.
Default	Associa ao evento OnClick do objeto ao pressionamento da tecla Enter.
ModalResult	Propriedade utilizada para encerrar a execução de um formulário Modal quando selecionado um valor diferente de mrNone.
Parent...	As propriedades *Parent* permitem que o componente receba a mesma formatação do objeto proprietário.
TabOrder	Define a ordem na passagem de foco no momento de pressionamento da tecla TAB.
TabStop	Define se o foco *pára* no componente.

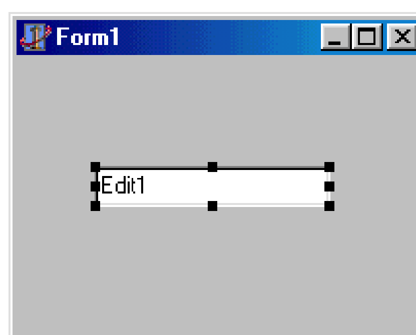
Métodos

SetFocus	Envia o foco do windows para o componente.

Objeto – Edit (Caixa de edição)

Paleta – Standart

Importância: Um dos principais componentes para a entrada de dados do usuário do sistema.



Propriedades

AutoSelect	Define se o texto exibido pelo controle será selecionado quando este receber o foco da aplicação.
AutoSize	Para componentes TEdit a propriedade determina se a altura do controle

	será redimensionada quando o tamanho da fonte **for** alterado.
BorderStyle	Determina o tipo da borda do componente.
CharCase	Determina o se tipo da fonte será maiúscula, minúscula ou normal.
HideSelection	Define se o texto perde a seleção ao perder o foco.
Maxlength	Define um limite para a inserção de caracteres.
PasswordChar	Define qual caractere será usado para ocultar o texto inserido no componente.
Text	Permite manipular os caracteres inseridos no componente pelo usuário.

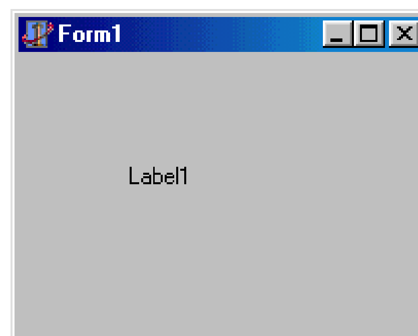
Métodos

Clear	Limpa o conteúdo da propriedade text.
SetFocus	Envia o foco do windows para o componente.

Objeto – Label (Rótulo de orientação)

Paleta – Standart

Importância: Orientar o usuário à escolha de componentes bem como sua utilização.



Propriedades

Alignment	Define o alinhamento da string na área do componente.
AutoSize	Para componentes TDBText e TLabel, esta propriedade define se o controle será automaticamente redimensionado para acomodar o texto.
FocusControl	Define qual o componente receberá foco quando o usuário selecionar a combinação de teclas aceleradoras (atalho) se existir.
Layout	Define o alinhamento vertical do texto na área do componente.
ShowAccelChar	Define se o caracter ‘&’ será um literal ou tecla de aceleradora (atalho).
Transparent	Define se o fundo do componente será ‘transparente’ ou não.
WordWrap	Define se o texto poderá utilizar o ‘retorno automático’ em caso de ultrapassar a largura definida e se a propriedade *AutoSize* estiver falsa.

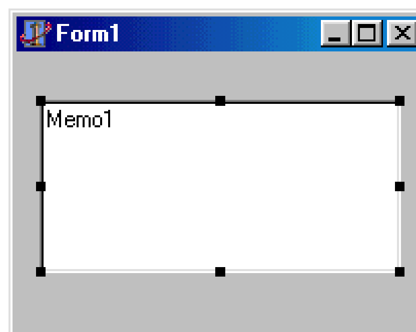
Sugestão: Exercicio 1

MAIS SOBRE A PALETA STANDART

Objeto – Memo (Memorando)

Paleta – Standart

Importância: Permite o usuário entrar com dados do tipo TStrings, compara-se à funcionalidade do software bloco de notas.



Propriedades

Lines	Propriedade do tipo TStrings que contém as linhas de texto do componente.
MaxLength	Define o limite máximo de caracteres no componente em sua propriedade Lines.
ReadOnly	Define se o componente é do tipo somente leitura.
ScrollBars	Define se o componente pode trabalhar com barras de rolagem.
WantReturns	Define se a tecla ENTER será utiliza para 'quebra de linha'.
WantTabs	Define a tecla Tab como tabulação ou mudança de foco. Caso falso pode-se utilizar CTRL+TAB para produzir o efeito desejado.

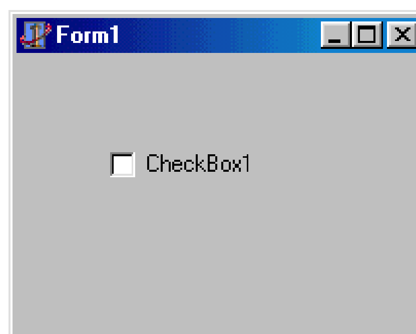
Métodos

LoadFromFile	Permite 'carregar' um arquivo para a propriedade Lines.
SaveToFile	Permite salvar o conteúdo da propriedade Lines em um arquivo especificado.

Objeto – CheckBox (Caixa de verificação)

Paleta – Standart

Importância: Permite verificar opções boolenas pré-definidas ou re-definidas pelo usuário.



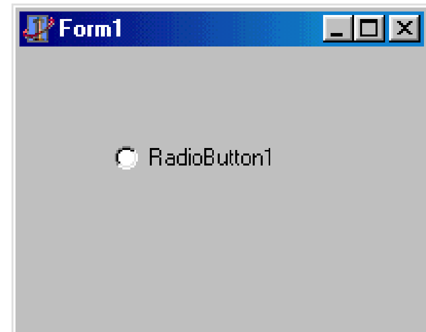
Propriedades

AllowGrayed	Define caso verdadeiro, três estados possíveis para o checkbox: checked (ligado), unchecked (desligado) e grayed (parcial). Caso falso, dois

	estados: checked (ligado) e unchecked (desligado).
Checked	Define se o componente está ligado ou não, caso tenha apenas dois estados.
State	Permite definir três estados se AllowGrayed **for** verdadeiro.

Objeto – RadioButton (Botão de 'radio')

Paleta – Standart



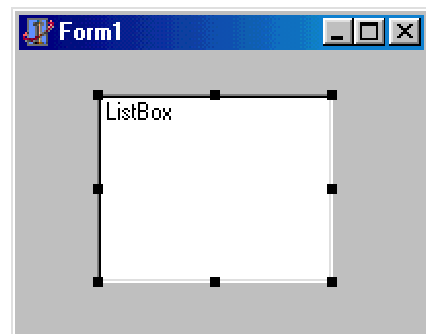
Importância: Permite escolher entre um grupo, *pelo menos* uma opção.

Propriedades

Checked	Define se o componente está ligado ou desligado.

Objeto – ListBox (Caixa de listagem)

Paleta – Standart



Importância: Permite o usuário entrar ou manipular uma lista de dados.

Propriedades

Items	Define uma lista de Strings que aparece no componente.
MultiSelect	Permite selecionar vários itens (Strings) no componente.
Sorted	Define se a lista de Strings será ordenada ou não.

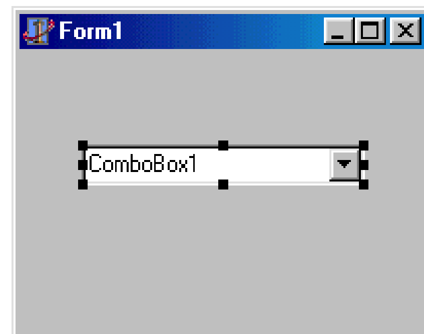
Métodos

Clear	Permite limpar o conteúdo da propriedade Items.
LoadFromFile	Permite 'carregar' um arquvo para a propriedade Items.
SaveToFile	Permite salvar o conteúdo da propriedade Items para um arquivo.

Objeto – ComboBox1 (Caixa de listagem em formato de cortina)

Paleta – Standart

Importância: Permite o usuário entrar ou manipular uma lista de dados.



Propriedades

Items	Define uma lista de Strings que aparece no componente.
Sorted	Define se os dados serão ordenados.
Text	Define o texto atual da Combobox.

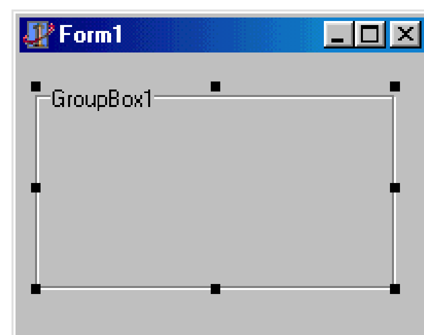
Métodos

Clear	Permite limpar o conteúdo da propriedade Items.
LoadFromFile	Permite 'carregar' um arquvo para a propriedade Items.
SaveToFile	Permite salvar o conteúdo da propriedade Items para um arquivo.

Objeto – GroupBox (Caixa de agrupamento)

Paleta – Standart

Importância: Permite agrupar componentes e estabelecer um título na propriedade Caption.



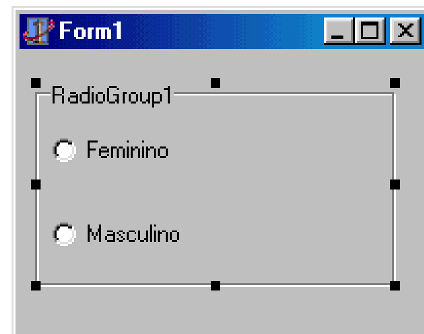
Propriedades

Align	Pemite definir um alinhamento no objeto proprietário.
Caption	Define o texto informativo na parte superior do componente.

Objeto RadioGroup (Grupo de botões 'radio')

Paleta – Standart

Importância: Permite estabelecer um grupo de botões de radio e manipula-los pela propriedade ItemIndex.



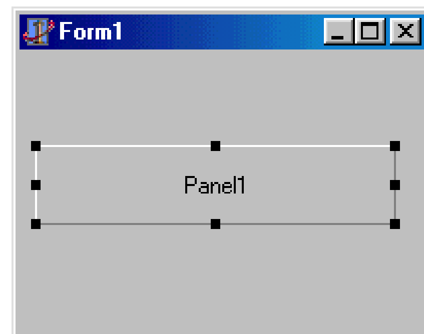
Propriedades

Items	Define os itens disponíveis ao usuário.
ItemIndex	Define qual dos itens está selecionado.
Columns	Define o número de colunas para organização dos componentes.

Objeto – Panel (Painel)

Paleta – Standart

Importância: Permite agrupar outros objetos e estabelecer um efeito visual nas aplicações.



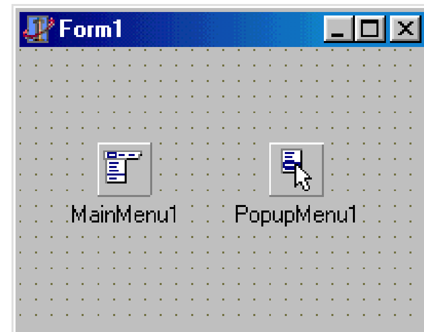
Propriedades

Align	Define o alinhamento do componente em relação ao seu proprietário.
Bevel...	Define a característica das bordas (interna e externa) bem como sua espessura.
BorderStyle	Define o tipo da borda.

Objetos – MainMenu e PopupMenu (Menu principal e Menu rápido)

Paleta – Standart

Importância: Define os Menus utilizados pelo usuário pelo botão esquerdo (MainMenu) ou pelo botão direito (PopupMenu) do Mouse.



Propriedades

Items	Define um novo item de Menu.
Images	Define um objeto do tipo 'ImageList'.

O objeto MainMenu permite a construção de *sub-menus* através de seu *construtor* clicando no item com o botão direito e escolhendo a opção *Create submenu*. Pode-se também excluir ou incluir items aleatoriamente através do botão direito no item desejado.
Para criar um separador de menus, utilize o operador de subtração ('-') e confirme com a tecla Enter.

Sugestão: Exercício 2

A LINGUAGEM OBJECT PASCAL

Por mais recursos gráficos que as linguagens orientadas a objetos possuam, em determinado momento não há como fugir do *código*. A programação em Delphi é definida através da Linguagem Object Pascal, uma extensão do Pascal proposto por Niklaus Wirth.

Consideramos uma aplicação em Delphi baseada em um conjunto de arquivos, (citados anteriormente .DPR .PAS e .DFM) básicos. Vamos examinar alguns arquivos de fundamental importância:

O MÓDULO .DPR

Todo programa em Object Pascal possui um arquivo .DPR, considerado como *arquivo de projeto*, o seu formato é composto inicialmente da seguinte definição:

```
program Project1;

uses
  Forms,
  Unit1 in 'Unit1.pas' {Form1};
```

```
{$R *.RES}

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

A palavra *program* define o nome do programa, este nome será alterado quando for gravado o arquivo .DPR do projeto.
Na cláusula *uses*, são listadas as *units* usadas pelo módulo principal. As *units* (que serão vistas adiante) são responsáveis pela capacidade de dividir o programa em uma visão *modularizada*. Em cada um, declaramos uma série de objetos (funções, variáveis, procedimento, etc...) que podem ser usados por outras *units* e pelo módulo principal.
Em seguida vem um conjunto de comandos (denominado *comando composto*) através de dois delimitadores ***begin*** *e* ***end***.

AS UNITS

Um programa em Object Pascal é constituído de um módulo principal (.DPR) e de uma ou mais unidades de compilação (.PAS). O compilador gera um arquivo com o código objeto correspondente, e considera o mesmo nome do arquivo .PAS com a extensão .DCU.

As *units* são entidades independentes, ou seja, no momento da criação não há vínculo lógico (nem físico) entre uma *unit* e um programa principal que a utiliza. Com esta característica, podemos utilizar as *units* em qualquer projeto.
A principal característica do conceito de *unit* é que possibilita estruturar o programa em módulos funcionais, com cada *unit* provendo um conjunto de funções e procedimentos. Cada formulário corresponde a uma *unit*. Mas, podemos criar *units* independentes, não associadas a um *form*.

Se considerarmos o código uma *unit* com um componente *Button* e um manipulador de evento, teremos o seguinte código:

```
unit Unit1;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms,
  Dialogs,  StdCtrls, ExtCtrls, ComCtrls;
type
  TForm1 = class(TForm)
    Button1: TButton;
    procedure Button1Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;
```

```
var
  Form1: TForm1;

implementation

{$R *.DFM}

procedure TForm1.Button1Click(Sender: TObject);
begin
  Form1.Caption := 'Curso de Delphi';
  Showmessage('Exemplo de caixa de diálogo');
end;

end.
```

Uma *unit* possui cinco partes:

Cabeçalho

Contém a palavra reservada *unit* seguida de um identificador que é o nome da *unit*. Este nome é o mesmo nome do arquivo com extensão .PAS

```
unit Unit1;
```

Interface

Contém tudo o que a *unit* exporta: constantes, tipos, variáveis, procedimentos, funções, etc... Na declaração dos procedimentos e funções que a *unit* exporta, deve constar apenas o cabeçalho (nome e parâmetros). A declaração completa fica na parte da *implementação*.

```
interface
uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms,
  Dialogs,  StdCtrls, ExtCtrls, ComCtrls;
type
  TForm1 = class(TForm)
    Button1: TButton;
    procedure Button1Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;
var
  Form1: TForm1;
```

Implementação

Contém a definição completa das *funções* e *procedimentos* que constam na *interface*. Se na implementação são usados identificadores definidos em outra *unit*, o nome desta outra *unit* deve ser incluído na lista de *units* da cláusula *uses* da implementação.

```
implementation
{$R *.DFM}
procedure TForm1.Button1Click(Sender: TObject);
begin
  Form1.Caption := 'Curso de Delphi - SENAC MG';
  Showmessage('Exemplo de caixa de diálogo');
end;
```

Inicialização

É uma parte opcional. Quando usada, não pode conter nenhuma declaração. Apenas comandos são permitidos nesta parte. Ela começa com a palavra *initialization*, e os comandos de inicialização são executados "antes" do programa começar.

```
initialization
  <comandos>
```

Finalização

É também uma parte opcional, com uma observação: ela só pode existir se na *unit* houver também uma parte de inicialização e só pode conter comandos, que serão executados dentro do processo de finalização do programa, após a execução do programas principal.

```
finalization
  <comandos>
```

Toda *unit* termina com a palavra ***end*** seguida de um ponto final ('**.**').

ATRIBUIÇÃO

Ao declarar uma variável, o compilador cuida de alocar na memória uma área que seja suficiente para armazenar qualquer dos valores definidos através do seu tipo. Os valores que podem ser atribuídos à variável são definidos através de um *comando de atribuição* que pode ser considerado da seguinte forma:

Variável := *expressão* ;

DECLARAÇÃO DE VARIÁVEIS

As variáveis podem ser classificadas em:

Globais: Quando são feitas diretamente na seção *interface* de uma *unit* (ou seja, fora dos procedimentos e funções). Pode-se ter varíaveis *públicas* e *privadas*.
Locais: Quando é feita a declaração *dentro* de um procedimento ou função.

```
var
  N: Single;
  S: String;
  I: Integer;
```

TIPOS PARA MANIPULAÇÃO DE VARIÁVEIS

Tipos de variáveis Inteiras

Tipo	*Faixa de Valores*	*Formato*
Integer	-2147483648.. 2147483647	32 bits
Cardinal	0..4294967295	32 bits, sem sinal
Shortint	-128..127	8 bits
Smallint	-32768..32767	16
Longint	-2147483648.. 2147483647	32
Int64	-2^63..2^63-1	64
Byte	0..255	8 bits, sem sinal
Word	0..65535	16 bits, sem sinal
Longword	0..4294967295	32 bits, sem sinal

Tipos de números Reais

Tipo	*Faixa de Valores*
Real	2.9*10E-39..1.7*10E38
Single	1.5*10E-45..3.4*10E38
Double	5.0*10E-324..1.7*10E308
Extended	3.4*10E-4932..1.1*10E4932
Comp	-2*10E63+1..2*10E63-1
Currency	-9.22*10E14..9.22*10E14

Tipos de variáveis booleanas

Tipo	*Faixa de Valores*
Boolean	False ou True
ByteBool	*
WordBool	*
LongBool	*

Tipos de variáveis de caracteres

Tipo	*Valores*
Char	Permite armazenar um caractere ASCII.
ShortString	Permite armazenar uma cadeia de até 255 caracteres.
String	Permite armazenar uma cadeia ‘ilimitada’ de caracteres.

Tipo genérico (Variant)

Objetos variant são essencialmente variáveis *sem tipo* podendo assumir diferentes tipos, automaticamente. Esta vantagem aparente tem a característica de ser ineficiente se utilizada indiscriminadamente.

FUNÇÕES DE CONVERSÃO E MANIPULAÇÃO

Os objetos do Delphi para entrada e/ou exibição de dados utilizam propriedades do tipo String, as propriedades Text e Caption são bons exemplos. O problema ocorre quando tentamos realizar cálculos matemáticos com os dados que devem ser manipulados por estas propriedades.
Desta maneira precisamos de funções para converter dados String em tipos Inteiros ou Reais ou Datas, por exemplo.

Função	*Objetivo*
StrToInt(const S: String)	Converte um dado String em tipo Inteiro.
IntToStr(value: Integer)	Converte um dado Inteiro em tipo String.
StrToFloat(const S: String)	Converte um dado String em tipo Ponto Flutuante.
FloatToStr(Value: Extended)	Converte um dado Ponto Flutuante em tipo String.
DateToStr(Date: TdateTime)	Converte um dado TDateTime em String.
DateTimeToStr(DateTime: TDateTime)	Converte um dado TDateTime em String.
StrToDate (const S: String)	Converte um dado String em tipo TDateTime.
StrToDateTime(const S: String)	Converte um dado String em tipo TDateTime
FormatFloat(const Format: string; Value: Extended): string	Permite formatar um tipo ponto flutuante retornando uma string. `Edit2.Text := FormatFloat('###,###.00',soma);` Sendo soma uma variável real.

O tipo TDateTime é internamente manipulado como tipo Ponto Flutuante.

EXPRESSÕES LÓGICAS

São expressões que retornam valor *booleano* (falso ou verdadeiro).

Operador	*Operação*
not	Negação
and	E lógico
or	OU lógico
xor	OU EXCLUSIVO lógico

O operador *not* é unário, por exemplo: `if not (X > Z) then`
Devemos usar parênteses ao compararmos expressões lógicas, por exemplo:

```
if (X > Z) or (W > Y) then
```

COMANDO IF

O comando condicional ***if*** pode ser composto de uma ou mais condições de processamento, por exemplo:

- ```
  if (A > B) then
    B := B + 1; // ou INC(B);
  ```

- ```
  if (A > B) then
     B := B + 1
  else
     A := A - 1; // ou DEC(A);
  ```

- ```
  if (A > B) then
    begin
     B := B + 1;
     X := B + A;
    end
  else
    begin
     A := A - 1;
     Y := Y + B;
    end;
  ```

No último exemplo para representar um bloco de comandos em caso verdadeiro ou falso, utiliza-se dos delimitadores ***begin*** e ***end***.
O comando ***if-then-else*** é considerado como único, portanto, não há ponto e vírgula ('**;**')antes da palavra reservada **else**.

COMANDO CASE

O comando *case..of* oferece uma alternativa para comandos ***if-then-else*** com um 'grande' número de testes. Por exemplo:

```
case Key of
  'A'..'z':           Label1.Caption := 'Letras';
  '0'..'9':           Label1.Caption := 'Números';
  '+', '-', '*', '/': Label1.Caption := 'Operador'
else
  Label1.Caption := 'Caracter especial';
end; //fim do case
```
```

COMANDO REPEAT

O comando *repeat..until* é uma opção para estruturas de repetição. A grande diferença com o comando *while* é o fato do comando *repeat* ser executado pelo menos uma vez.

```
repeat
  X := X + 1;
  INC(Z,3); //equivale a Z := Z + 3;
  DEC(AUX,2);
until X >= 200;
```

COMANDO WHILE

O comando *while..do* também permite a construção de estruturas de repetição, com diferença de não executar o laço no início do teste lógico.

```
while X <= 200 do
  begin
    X := X + 1;
    INC(Z,3);
    DEC(AUX,2);
  end;
```

COMANDO FOR

O comando *for..do* estabelece uma estrutura de repetição considerando um controle inicial e final. Pode ser construído de maneira crescente ou decrescente.

```
for i:=0 to 500 do
  Label1.Caption := IntToStr(i);

for i:=500 downto 100 do
  begin
    Label1.Caption := IntToStr(i);
    Edit1.Caption := IntToStr(i);
  end;
```

COMANDO BREAK

O comando *break* é usado para alterar o fluxo normal de comandos de repetição, o controle é desviado para o comando seguinte ao comando repetitivo.

```
frase := Edit1.Text;
for i:=1 to length(frase) do
  begin
    if frase[I] = 'A' then
      break;
    aux := aux + frase[I];
  end;
Label1.caption := aux; //Label1 recebe o conteudo de frase até a letra 'A'
```

COMANDO WITH

O comando *with..do* é usado para abreviar a referência a campos de registro, ou a métodos, e *propriedades* de um objeto.

```
begin
  Form1.Caption := 'Senac';
  Form1.Color := ClBlue;
  Form1.Top := 95;
end;
```

```
//Equivalente à:
with Form1 do
  begin
    Caption := 'Senac';
    Color := ClBlue;
    Top := 95;
  end;
```

Sugestão: Exercício 3

PROCEDURES E FUNÇÕES

Procedimentos e funções são blocos de código (rotinas) em Object Pascal que podem ou não receber parâmetros (valores) para processamento. Uma vez definida a rotina pode-se ativa-la de diversas partes do programa através de seu nome.

A grande diferença entre as formas de definição destas rotinas (se procedimentos ou funções) está no fato de que:

- *Procedimento* – **NÃO** retorna valor.
- *Função* – Retorna valor.

DECLARAÇÃO E ATIVAÇÃO DE PROCEDIMENTO

Podemos declarar um procedimento da seguinte maneira maneira:
Dentro da cláusula **private *ou* public**, defina a declaração do procedimento:

```
procedure Soma(X, Y: String);
```

Com o cursor posicionado na mesma linha, pressione: **CTRL+SHIFT+C** e perceba que o próprio Delphi realiza a construção do procedimento dentro da cláusula **implementation**. Esse recurso é chamado **Class Completion.** Nossa tarefa é apenas definir o código a ser realizado pelo procedimento.

```
procedure TForm1.Soma(X, Y: String);
begin
  Label1.Caption := FloatToStr(StrToFloat(X)+StrToFloat(Y));
end;
```

Supondo a existência de dois componentes *Edit*, um componente *Button* e um componente *Label*, este código pode ser ativado da seguinte forma:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  Soma(Edit1.Text, Edit2.Text);
end;
```

DECLARAÇÃO E ATIVAÇÃO DE FUNÇÕES

A construção de funções tem o mesmo raciocínio diferindo na característica de retorno da função.

Podemos declarar um procedimento da seguinte maneira maneira:
Dentro da cláusula **private *ou* public**, defina a declaração da função:

```
function Subtrai(X, Y: String): String;
```

Observe que agora, depois dos parâmetros há um tipo de definição de retorno da função (String).

Pode-se utilizar a mesma dica de construção do procedimento, na linha da declaração tecle **CTRL+SHIFT+C** (Class Completion) e perceba que o próprio Delphi realiza a construção da função dentro da cláusula **implementation**.

Nossa tarefa é apenas definir o código a ser realizado pela função.

```
function TForm1.Subtrai(X, Y: String): String;
begin
   result := FloatToStr(StrToFloat(X)-StrToFloat(Y));
end;
```

A palavra reservada ***result*** é o recurso usado pela Object Pascal para estabelecer o retorno da rotina. Não se deve declarar esta variável, ela é declarada no momento da utilização da ***função***.
Supondo a existência de dois componentes *Edit*, 'um' componente *Button* e um componente *Label*, esta função pode ser ativada da seguinte forma:

```
function TForm1.Button2Click(Sender: TObject);
begin
  Label1.Caption := Subtrai(Edit1.Text, Edit2.Text);
end;
```

Neste caso, o Label recebe o **result** de subtrai, ou seja, a subtração dos dados passados nos parâmetros.

DECLARAÇÕES CRIADAS AUTOMATICAMENTE PELO DELPHI

Se você é um bom observador, percebeu que o Delphi também gera as declarações e seus respectivos procedimentos quando você deseja manipular um determinado evento.

CAIXAS DE DIÁLOGO

Podemos utilizar alguns tipos de caixas de diálogo pré-definidas pelo Delphi facilitando em muito o desenvolvimento pela ferramenta:

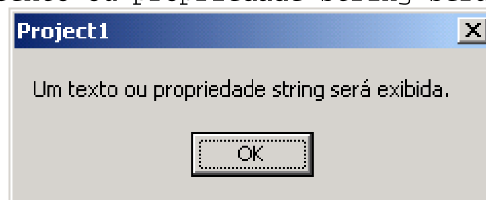
ShowMessage

A caixa de diálogo ShowMessage é declarada internamente pelo Delphi desta forma:

```
procedure ShowMessage(const Msg: string);
```

Onde o parâmetro Msg é um dado String. Exemplo:

```
ShowMessage('Um texto ou propriedade string será exibida.');
```



MessageDlg

A caixa de diálogo MessageDlg é declarada internamente pelo Delphi desta forma:

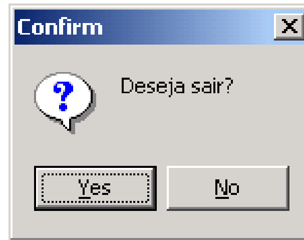
```
function MessageDlg(const Msg: string; DlgType: TMsgDlgType;
                    Buttons: TMsgDlgButtons; HelpCtx: Longint): Word;
```

Onde:

const Msg: string	É uma constante string ou propriedade deste tipo.
DlgType: TmsgDlgType	mtWarning *Contém um ícone exclamação amarelo.* mtError *Contém um ícone vermelho de 'parada'.* mtInformation *Contém um ícone 'i' azul.* mtConfirmation *Contém uma interrogação verde.* mtCustom *Não contém BitMap.*
Buttons: TMsgDlgButtons	mbYes mbNo mbOK mbCancel mbAbort mbRetry mbIgnore mbAll mbNoToAll mbYesToAll mbHelp
HelpCtx: Longint	Define um número para o help de contexto. Por padrão, zero '0'.

O ***retorno*** da função é o tipo do botão como `mr`
Desta maneira pode-se fazer testes lógicos como no exemplo:

```
if MessageDlg('Deseja sair?', mtConfirmation, [mbYes, mbNo], 0)=mrYes then
```



Application.MessageBox

Uma outra caixa de diálogo é o método MessageBox do objeto Application. Esta função está definida da seguinte maneira:

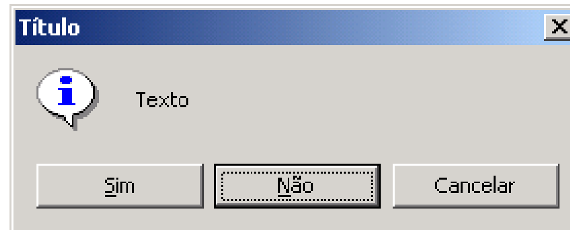
```
function MessageBox(const Text, Caption: PChar; Flags: Longint): Integer;
```

Onde:

const Text	É uma constante string ou propriedade deste tipo.
Caption: PChar	Define uma string para o título da janela.
Flags	Define os botões, ícones e a possibilidade de focar um determinado botão. Os valores para ***botões*** são: MB_ABORTRETRYIGNORE, MB_OK, MB_OKCANCEL, MB_RETRYCANCEL, MB_YESNO, MB_YESNOCANCEL Os valores para os ***ícones*** são: MB_ICONEXCLAMATION, MB_ICONWARNING, MB_ICONINFORMATION, MB_ICONASTERISK, MB_ICONQUESTION, MB_ICONSTOP, MB_ICONERROR, MB_ICONHAND Os valores para a definição do ***botão default*** pode ser: MB_DEFBUTTON1, MB_DEFBUTTON2, MB_DEFBUTTON3, MB_DEFBUTTON4

O ***retorno*** da função é o tipo do botão como `id`
`(IDABORT IDCANCEL IDIGNORE IDNO IDOK IDRETRY IDYES)`
Desta maneira pode-se fazer testes lógicos como no exemplo:

```
if Application.MessageBox('Texto','Título',MB_YESNOCANCEL +
MB_ICONINFORMATION + MB_DEFBUTTON2) = IdYes then
```



CAIXAS DE ENTRADA

Podemos obter dados do usuário através de caixas de diálogo pré-defindas.

InputBox

A função InputBox retorna um tipo String, que é dado digitado pelo usuário na sua utilização.
Sua definição interna é a seguinte:

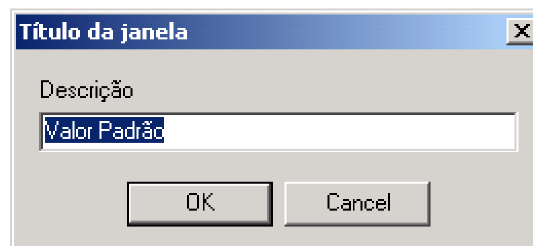
```
function InputBox(const ACaption, APrompt, ADefault: string): string;
```

Onde:

const ACaption	Define o título da janela
APrompt	Define um rótulo para orientação dentro da caixa.
ADefault	Define um valor default para a caixa.

Exemplo:

```
InputBox('Título da janela','Descrição','Valor Padrão')
```



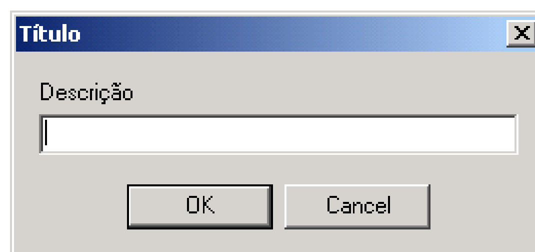
InputQuery

A função InputQuery retorna um tipo Booleano, o dado digitado pelo usuário será colocado em uma variável do tipo string *previamente* declarada.
Sua definição interna é a seguinte:

```
function InputQuery(const ACaption, APrompt: string; var Value: string):
Boolean;
```

Exemplo:

```
if InputQuery('Título', 'Descrição', aux) and (aux <> '') then
```



Neste exemplo acima, a janela só retornará verdade **se** houver algum valor digitado **e** o usuário clicar no botão OK, caso contrário o retorno será falso.

Exemplo

Vamos examinar uma *unit* e exemplicar os principais conceitos:

```
type
  TForm1 = class(TForm)
    Button1: TButton;
    Edit1: TEdit;
    Edit2: TEdit;
    Label1: TLabel;
    procedure Button1Click(Sender: TObject);
    function Soma(X, Y: String): Integer; //definida pelo desenvolvedor
  private
    { Private declarations }
    aux: Integer; //variável privada
  public
    { Public declarations }
    frase: String; //variável pública
  end;
var
  Form1: TForm1;
implementation

{$R *.DFM}

const MAX = 50; //definição de constante
```

```
procedure TForm1.Button1Click(Sender: TObject);
var titulo: String; //variavel local
    i: Integer;
begin
  if (Edit1.Text <> '') and (Edit2.Text <> '') then
  //atribui à variavel private - aux o retorno da função soma
    aux := Soma(Edit1.Text,Edit2.Text);
  titulo := 'Curso de Delphi'; //atribui à variàvel local
  frase := titulo+' – Versão 5'; //atribui à variavel public
  Form1.Caption := frase; //atribui à propriedade Caption do form
  ShowMessage('A soma dos valores é: '+IntToStr(Aux));
  for i:=0 to MAX do
    Label1.Caption := IntToStr(i);
end;

function TForm1.Soma(X, Y: String): Integer;
begin
  result := StrToInt(X)+StrToInt(Y);
end;

end.
```

Sugestão: Exercício 4

CHAMADA DE FORMS

Uma característica importante da apresentação dos formulários em uma aplicação, é o fato de ser apresentado como **MODAL** ou **NÃO-MODAL**. Há dois métodos para executar a visualização, mas antes vamos entender como isso funciona.

- MODAL – O foco fica *preso* no formulário e não é liberado para outro form até que ele seja fechado. O usuário pode ativar outra aplicação do Windows, mas não poderá trabalhar em outra janela daquele programa cuja janela foi aberta como modal (até que seja fechada).
- NÃO MODAL – O foco pode ser transferido para outra janela sem que esta precise ser fechada.

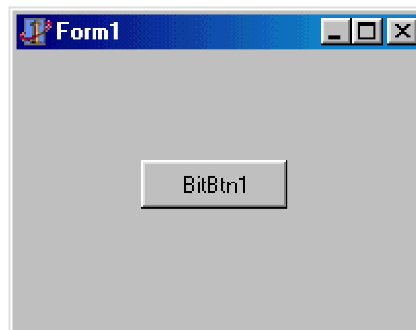
Entendido este conceito, os métodos que o Delphi utiliza para apresentar os forms são:
Show para apresentar forms NÃO-MODAIS, ou **ShowModal** para apresentar forms MODAIS.

COMPONENTES (VCL)

Objeto – BitBtn (Botão com figuras opcionais)

Paleta – Additional

Importância: Permite inserir figuras para uma melhor orientação do usuário, além de funções pré-definidas.



Propriedades

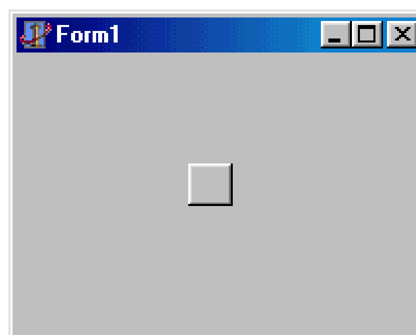
Glyph	Define um Bitmap para o componente. (Arquivo com extensão .BMP)
Kind	Define o tipo de Bitmap exibido pelo usuário. bkCustom *Bitmap definido pelo usuário.* bkOk *Botão OK padrão, com uma marca de verificação na cor verde e propriedade Default igual a True.* bkCancel *Botão Cancel padrão, com um "x" na cor vermelha e propriedade Cancel igual a True.* bkYes *Botão Yes padrão, com uma marca de verificação na cor verde e propriedade Default igual a True.* bkNo *Botão No padrão, com uma marca vermelha representando um círculo cortado e propriedade Cancel igual a True.* bkHelp *Botão de auxílio padrão, com uma interrogação na cor cyan. Quando o usuário clica sobre o botão, uma tela de auxílio deve ser exibida (baseada no código do desenvolvedor).* bkClose *Botão Close padrão, com o desenho de uma porta. Quando o usuário clica sobre o botão, o formulário a que ele pertence se fecha.* bkAbort *Botão Abort padrão, com um "x" na cor vermelha e propriedade Cancel igual a True.* bkRetry *Botão Retry padrão, com uma seta circular verde.* bkIgnore *Botão ignore padrão, com o desenho de um homem verde se afastando.*

	bkAll *Botão All padrão, com uma marca de verificação dupla na cor verde e propriedade default igual a True.*
ModalResult	Permite encerrar a execução de um formulário *Modal* quando o seu valor for diferente de *mrNone*.

Objeto – SpeedButton (Botão para barra de ícones)

Paleta – Additional

Importância: Permite ao usuário manipular os botões individuais ou através do conceito de grupo.



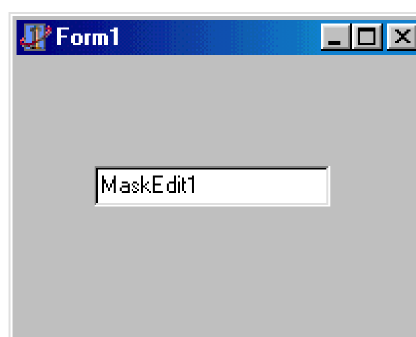
Propriedades

Glyph	Define um Bitmap para o componente.
GroupIndex	Permite *agrupar* um conjunto de SpeedButtons quando ao serem selecionados, tiverem a propriedade diferente de zero.
AllowAllUp	Permite que o componente possa ter o relevo suspenso ao ser clicado. Só pode ser utilizado junto ao conceito de agrupamento.
Flat	Define um efeito visual interessante.
Down	Permite determinar qual componente foi pressionado. Só pode ser utilizado junto ao conceito de agrupamento.

Objeto MaskEdit – (Caixa de edição com máscara)

Paleta – Additional

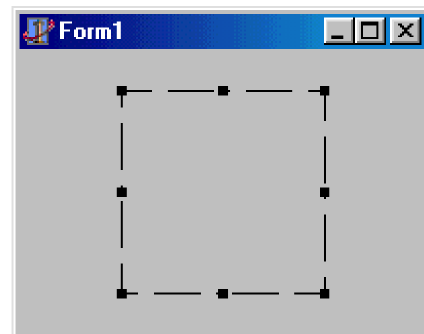
Importância: Permite estabelecer uma máscara para a entrada de dados no componente. Pode ser considerado literalmente um componente 'Edit com máscara'.



Propriedades

CharCase	Define o tipo dos caracteres.
EditMask	Permite definir uma máscara para entrada de dados.
PasswordChar	Define um caracter para ocultar a entrada de dados.

Objeto – Image (Imagem)



Paleta – Additional

Importância: Permite inserir uma figura para uso geral na aplicação.

Propriedades

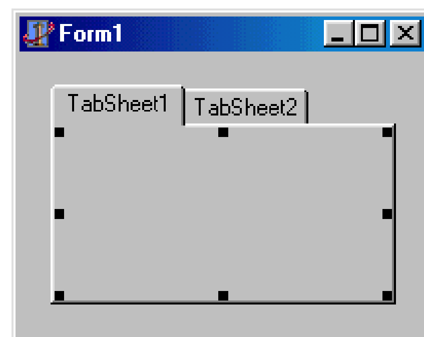
AutoSize	Permite alterar o tamanho do *componente* baseado no tamanho da figura.
Picture	Define a figura a ser exibida.
Stretch	Permite alterar o tamanho da *figura* baseado no tamanho do componente.

Métodos

LoadFromFile	Permite 'carregar' um arquivo de figura na propriedade Picture.

Para trabalhar com imagens jpg, é necessário acrescentar na cláusula uses da interface a biblioteca **jpeg**.

Objeto - PageControl



Paleta – Win32

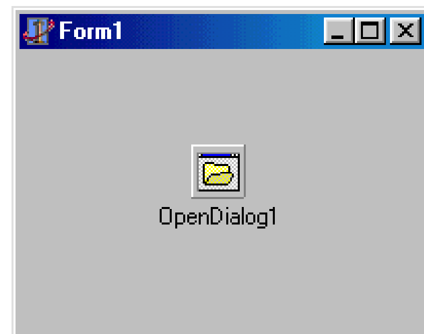
Importância: Permite definir guias para agrupar os demais componentes.
Cada guia representa um componente TabSheet do tipo TTabSheet, uma espécie de 'sub-objeto' do PageControl.

Propriedades

ActivePage	Permite determinar qual a guia foi selecionada pelo usuário.

Para criar novas páginas, clique com o botão direito no componente *PageControl* e escolha New Page.

Objeto – OpenDialog (Caixa de diálogo para abertura de arquivos)



Paleta – Dialogs

Importância: Permite utilizar uma caixa de diálogo pronta com recursos padronizados pelo sistema operacional.

Propriedades

DefaultExt	Especifica a extensão a ser adicionada ao nome de um arquivo quando o usuário digita o nome de um arquivo sem a sua extensão.
FileName	Define o arquivo selecionado no componente.
Filter	Permite definir as máscaras de filtro de arquivo a serem exibidas.
FilterIndex	Define o filtro default a ser exibido na lista drop-down que define os tipos de arquivos selecionáveis.
InitialDir	Define o diretório default quando a caixa de diálogo é aberta.
Options	Neste componente, options define uma série de valores booleanos.
Title	Define o título da caixa de diálogo.

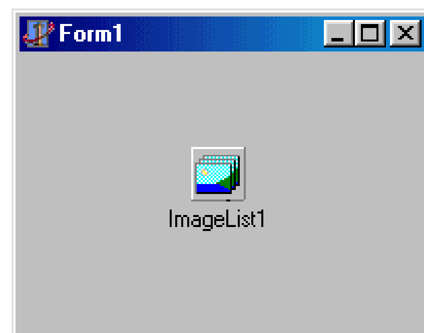
Os componentes da paleta dialogs são executados através do método ***execute.*** Este método é uma função que retorna um valor booleano, assim para exibir uma caixa de diálogo, podemos escrever:

```
    if OpenDialog1.Execute then
```

Se o usuário escolher algum arquivo e confirmar a caixa, *execute* retorna verdadeiro, caso contrário, falso.

Sugestão: Exercício 5

Objeto – ImageList (Lista de imagens)



Paleta – Win32

Importância: Permite definir um conjunto de ícones para serem re-utilizados por diversos componentes de recebem este objeto como provedor de uma lista de imagens.

Para incluir imagens no componente ImageList, clique 2 vezes rapidamente no componente e clique no botão Add.

O Delphi possui um conjunto de ícones e imagens em uma pasta padrão[6]:
`C:\Arquivos de programas\Arquivos comuns\Borland Shared\Images`

Objeto – RichEdit (Texto com formatação)

Paleta – Win32

Importância: Permite formatar o texto (Negrito, Itálico, Sublinhado, Fontes, etc...) para a leitura de outros editores compatíveis com o padrão RTF.

Propriedades

Lines	Define o texto exibido no componente.
WantReturns	Define a tecla Enter como quebra de linha.
WantTabs	Define a tecla Tab como tabulação ou mudança de foco. Caso falso pode-se utilizar CTRL+TAB para produzir o efeito desejado.
WordWrap	Define a quebra de linha automática de texto.

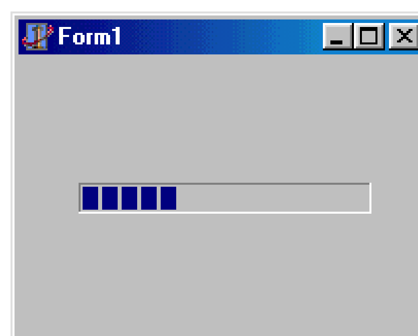
Métodos

Clear	Permite limpar o conteúdo do componente.
LoadFromFile	Permite 'carregar' um arquivo para a propriedade Lines.
SaveToFile	Permite salvar o conteúdo da propriedade Lines em um arquivo.

Objeto – ProgressBar (Barra de progresso)

Paleta – Win32

Importância: Permitir ao usuário ter um acompanhamento de uma rotina demorada.



[6] Caso você tenha instalado o software nos diretórios sugeridos como padrão.

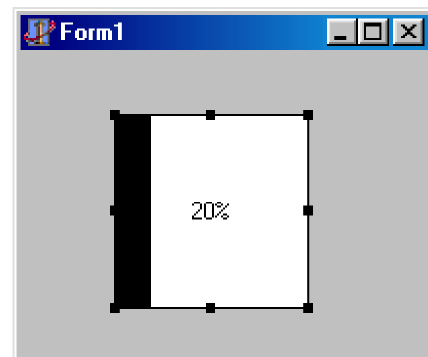
Propriedades

Max	Permite definir o valor máximo para a faixa de valores no componente.
Min	Permite definir o valor mínimo para a faixa de valores no componente.
Orientation	Define se o componente deverá ser vertical ou horizontal.
Position	Define a posição corrente do controle no componente.
Step	Define o incremento usado na variação do valor da propriedade position.

Objeto – Gauge (Barra de progresso)

Paleta – Samples

Importância: Permitir ao usuário ter um acompanhamento de uma rotina demorada.



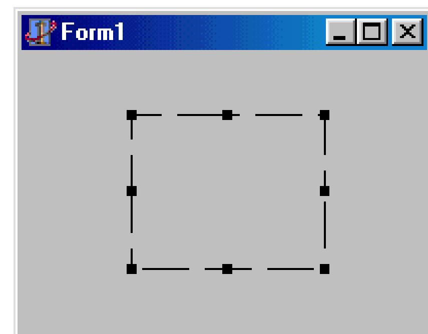
Propriedades

Kind	Permite definir aparências diferenciadas no componente.
Progress	Define a posição corrente do controle no componente.

Objeto – Animate (Animações)

Paleta – Win32

Importância: Permite exibir um 'filme' .AVI para ilustrar tarefas (rotinas) em andamento.

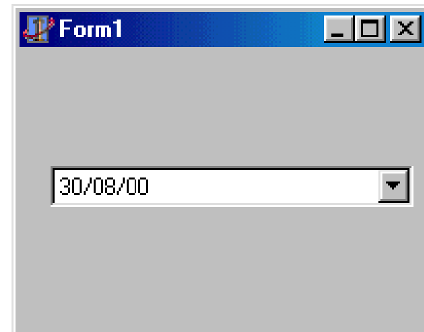


Propriedades

CommonAVI	Define o AVI a ser exibido.
Active	Liga e desliga a exibição do AVI.
Repetitions	Define um número inteiro correspondente ao número de repetições. Zero define repetições indefinidas.

Sugestão: Exercício 6

Objeto – DateTimePicker (Data e hora através de uma Combobox)



Paleta – Win32

Importância: Permite ao usuário escolher uma data através de um componente que possui um importante impacto visual e facilidade operacional.

Propriedades

CalColors	Define as cores do calendário.
Date	Define a data selecionada no componente.
DateFormat	Define o formato da apresentação da data.
DateMode	Define o estilo da caixa de listagem.
Kind	Define se o componente deve trabalhar com data ou hora.
MaxDate	Define uma data máxima para uma faixa de valores.
MinDate	Define uma data mínima para uma faixa de valores.

Objeto – MonthCalendar (Calendário mensal)

Paleta - Win32

Importância: Permite ao usuário escolher uma data através de um componente que possui um importante impacto visual e facilidade operacional.

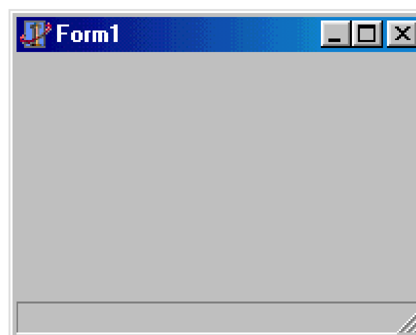
Propriedades

Date	Define a data selecionada no componente.
FirstDayOfWeek	Define qual o primeiro dia da semana.
WeekNumbers	Permite numerar as semanas.

Objeto – StatusBar (Barra de status)

Paleta – Win32

Importância: Um dos principais componentes de informações sobre operações gerais no sistema.



Propriedades

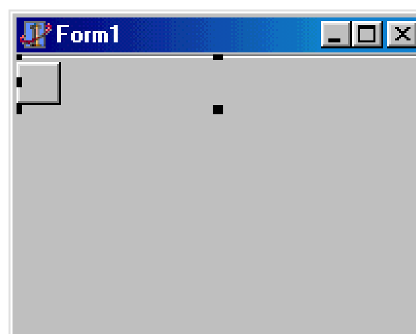
AutoHint	Permite exibir o hint do componente automaticamente na barra de status. Se não houver painéis, a barra deve ter a propriedade SimplePanel ligada.
SimplePanel	Define que a barra de status será *sem* divisões.
SimpleText	Define o texto a ser exibido pela barra de status.
Panels	Permite a criação e edição de *painéis* na barra de status. A propriedade SimplePanel deve estar desligada. Pode-se também dar um duplo clique na barra de status.

Ao utilizar a propriedade AutoHint, pode-se definir o hint dos objetos com duas strings separadas por *pipe* ('|').
A primeira string será utilizada como hint local, a segunda como hint na barra de status.

Objeto – ToolBar (Barra de ícones)

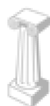
Paleta – Win32

Importância: Permite criar barras de ícones de maneira rápida e simples.



Propriedades

Flat	Define um efeito visual com relevo através do mouse nos botões.
Images	Permite definir um objeto do tipo *ImageList*.
HotImages	Permite definir um objeto do tipo *ImageList* a ser usado no momento em que o mouse passa (ou clica) sobre o componente.
ShowCaptions	Permite exibir a propriedade *caption* dos botões.



Para adicionar botões ou separadores na ToolBar, clique com o botão direito sobre o componente e escolha New Button ou New Separator.

Sugestão: Exercício 7

TRATAMENTO DE EXCEÇÕES

Quando criamos e executamos nossos programas, estamos sujeitos à situações de ***erros*** em tempo de *execução*, a isto denominamos ***exceção***.
As exceções devem ser tratadas de maneira a **não** permitir:

- Travar a aplicação
- Emitir mensagens 'técnicas' ao usuário leigo
- Deixar o SO instável

Quando uma exceção ocorre, o fluxo de controle é automaticamente transferido para blocos de código denominados *handlers*[7] de exceções, definidos através de comandos específicos do Object Pascal.

No Object Pascal, uma *exceção é uma classe*. A definição de exceções como classes permite agrupar exceções correlatas. Esse agrupamento é feito através da própria hierarquia de classes, de modo que podemos ter várias classes dependentes de uma única.

O que ativa o mecanismo de tratamento de erros através de exceções é o uso da unit *SysUtils*. Ela permite detectar os erros e convertê-los em exceções.

O COMANDO TRY-EXCEPT

Podemos tratar as exceções através do comando *try-except*.
Sua sintaxe:

```
try
      <comandos a serem executados>
except
      <bloco de exceção>
end; //finaliza o bloco
```

Os *comandos a serem executados* são tratados seqüencialmente na ordem em que foram criados, caso não haja alguma exceção o *bloco de exceção* é ignorado. O programa prossegue normalmente obedecendo aos eventos provocados pelo usuário.

Caso ocorra alguma exceção, o fluxo de controle é desviado para o *bloco de exceção*. É importante lembrar que podemos inserir qualquer comando, inclusive fazer chamadas a procedimentos e funções que por sua vez, podem chamar outros procedimentos e funções.

O *bloco de exceção* pode ser definido através de uma construção **genérica**, exemplo:

---

[7] *Handler* = Manipulador de evento.

```
try
  Abre(Arq);
  while not Fim(Arq) do
    processa(Arq);
except
    Showmessage ('Houve um erro inesperado.');
end; //bloco try
```

No exemplo acima tratamos os erros com uma mensagem genérica dentro de um um bloco try-except.

A CONSTRUÇÃO ON-DO

```
try
  Abre(Arq);
  while not Fim(Arq) do
    processa(Arq);
except
  on EInOutError do //erro de entrada e saída
     begin
       Showmessage('Problemas...');
       Fecha(Arq);
     end;
  on EdivByZero do //erro de divisão de nº inteiro por zero
     Showmessage('Erro ao dividir por zero');
  on EconvertError do //erro de conversão de tipos
     Showmessage('Erro de conversão de tipos de dados');
end; //bloco try
```

Podemos ainda definir utilizando a cláusula *on-do* com um *handler* genérico usando *else*, da seguinte forma:

```
try
  Processa;
except
  on Exceção1 do Trata1;
  on Exceção2 do Trata2;
  else TrataOutras;
end;
```

Os principais tipos de exceção da RTL (*RunTime Library*) do DELPHI, a serem tratadas nos blocos **on ... do** são:

Nome	Descrição
EaccessViolation	Ocorre quando se tenta acessar uma região de memória inválida (ex: tentar atribuir valor a um ponteiro cujo conteúdo é nil).
EconvertError	ocorre quando se tenta converter um string em um valor numérico (ex: utilizar a função StrToInt em uma letra).
EdivByZero	ocorre na divisão de um número **inteiro** por zero.
EinOutError	ocorre numa operação incorreta de I/O (ex: abrir um arquivo que não existe).

EintOverFlow	ocorre quando o resultado de um cálculo excedeu a capacidade do registrador alocado para ele (para variáveis inteiras).
EinvalidCast	ocorre quando se tenta realizar uma operação inválida com o operador as (ex: tentar usar um Sender com uma classe que não corresponde a seu tipo).
EinvalidOp	ocorre quando se detecta uma operação incorreta de ponto flutuante.
EinvalidPointer	ocorre quando se executa uma operação invalida com um ponteiro (ex: tentar liberar um ponteiro duas vezes).
EoutOfMemory	ocorre quando se tenta alocar memória mas já não existe mais espaço suficiente.
EoverFlow	ocorre quando o resultado de um cálculo excedeu a capacidade do registrador alocado para ele (para variáveis de ponto flutuante).
ErangeError	ocorre quando uma expressão excede os limites para a qual foi definida (ex: tentar atribuir 11 ao índice de um vetor que pode ir no máximo até 10).
EstackOverflow	ocorre quando o sistema não tem mais como alocar espaço de memória na Stack.
Eunderflow	ocorre quando o resultado de um cálculo é pequeno demais para ser representado como ponto flutuante.
EzeroDivide	ocorre quando se tenta dividir um valor **de ponto flutuante** por zero.

O COMANDO TRY-FINALLY

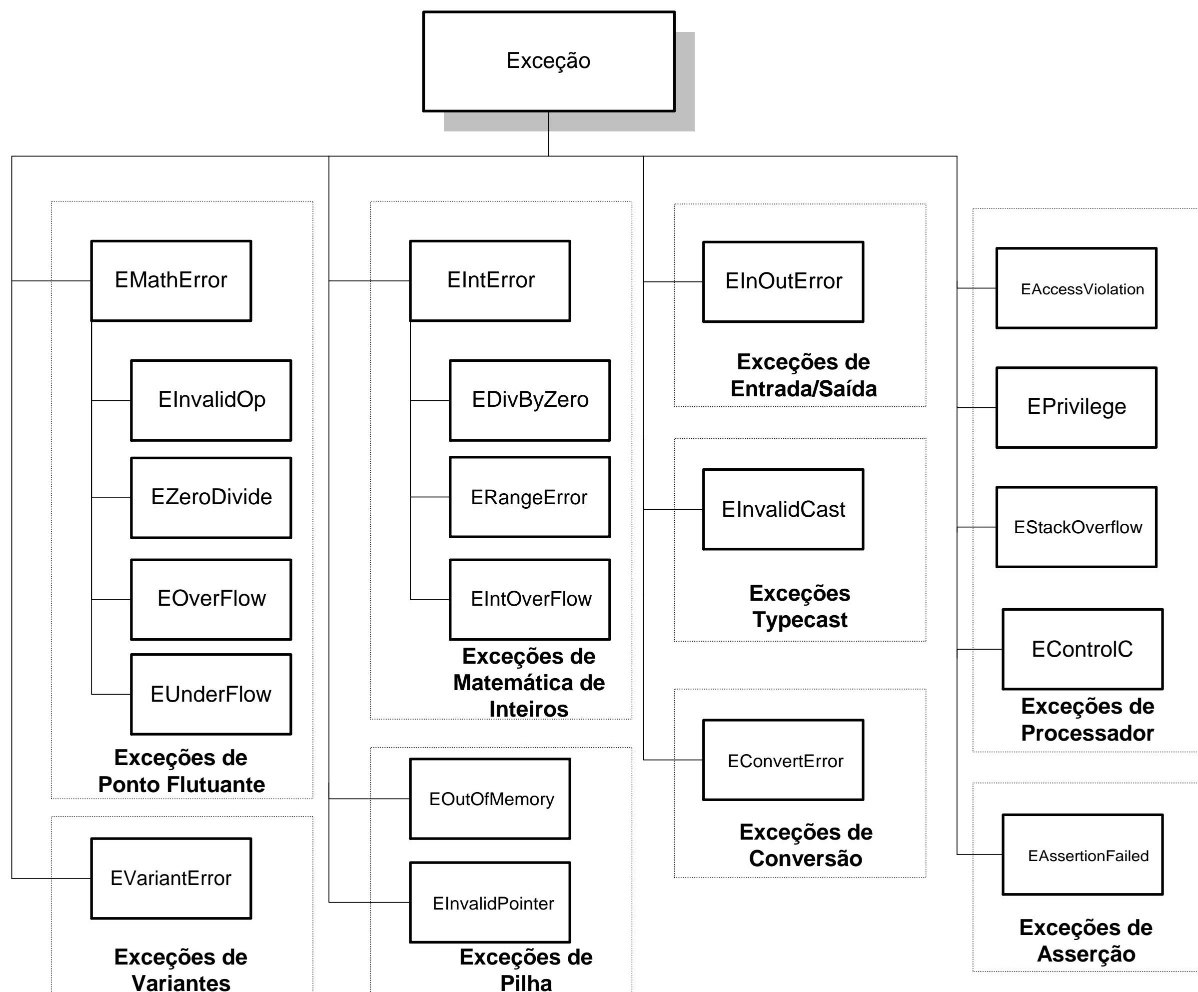
Há um outro comando cuja sintaxe começa com ***try***. Este controle de finalização nos permite lidar de forma estruturada com as situações em que alocamos algum tipo de recurso e, *haja o que houver*, precisamos depois liberá-lo.

```
  <aloca o recurso>
try
  <usa o recurso>
finally
  <libera o recurso com ou sem exceção>
end;
```

O comando funciona da seguinte forma: os comandos especificados após o *Try* são executados seqüencialmente. Se não ocorrer nenhuma exceção, os comandos especificados após *finally* **são** executados, e o programa prossegue com a execução normal, com o comando seguinte ao *try-finally*. Porém, se houver alguma exceção – qualquer uma – durante a execução da lista de comandos do *try*, o trecho após o *finally* é executado e, no final, a exceção é reativada.

Em resumo: Os comandos do bloco *finally* sempre são executados, *haja ou não* alguma exceção durante a execução dos comandos especificados após o *try*.

CLASSES BÁSICAS



BLOCOS TRY ANINHADOS

Blocos try aninhados permitem maior versatilidade na construção de blocos protegidos, lembrando que se a exceção ocorrer, os comandos inseridos em except serão executados. Já os comandos inseridos em finally serão executados havendo ou não a ocorrência de erros.

Embora no próximo exemplo, não exista margem para exceções dentro do laço *for..do* (a menos que seja digitado errado) podemos ter uma idéia de como criar blocos aninhados para garantir a execução de rotinas sujeitas a erros mais graves.

```
procedure TForm1.Button1Click(Sender: Tobject);
var i, aux:integer;
begin
  aux := 500;
  try           {inicio do bloco try-finally.
                 Screen controla uma serie de recursos do sistema operacional
                 neste exemplo, muda-se a aparencia do cursor para ampulheta}
    Screen.Cursor := crHourGlass;
    try//inicio do bloco try-except
      for i:=0 to aux do
      begin
        Edit1.Text := IntToStr(i);
        Application.ProcessMessages;
              {O método ProcessMessages é necessário para forçar que as
               mensagens do windows sejam processadas, desenhando o numero
               no Edit. Sem ele, apenas o valor final seria visto.}
      end;
    except
      Showmessage('Ocorreu um erro.');
   end;//fim do bloco try-except
  finally
    Screen.Cursor := crDefault;
  end;//fim do bloco try-finally
end; //fim da procedure
```

TRATAMENTO DE EXCEÇÕES DE FORMA GLOBAL

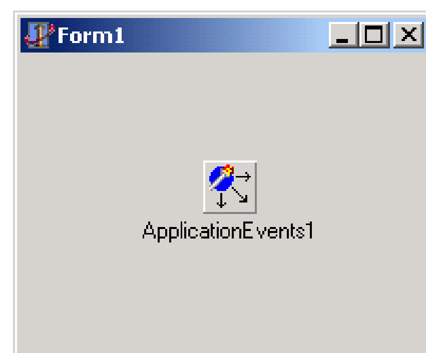
Há no Delphi um objeto chamado *Application* criado sem a decisão do desenvolvedor em todo o projeto. Este objeto representa a applicação como um todo e possui um evento muito importante: *OnException*. Este evento permite manipular as exceções em um nível global, podemos afirmar que os tratamentos de erro através do comando *try* são tratamento *locais*.

Na versão 5 do Delphi, os eventos deste objeto estão disponível na paleta Win32, nas versões anteriores este objeto era manipulado apenas no Code Editor tendo o desenvolvedor o trabalho de declarar os procedimentos desejados.

Como o objeto tem como finalidade generalizar e centralizar tratamentos, deve haver um único objeto na aplicação.

Paleta – Additional

Importância: Permite a manipulação de exceções em um nível global para toda a aplicação.



A utilização do evento OnException pode ser criado da seguinte forma, utilização um **if** na variável **E** (que recebe o erro atual) tomando uma decisão na condição verdadeira:

```
procedure TForm1.ApplicationEvents1Exception(Sender: TObject;
  E: Exception);
begin
   if E is EConvertError then
     ShowMessage('Erro de conversão de dados.');
end;
```

Neste exemplo acima, em *qualquer lugar do programa* (e não apenas em uma determinada rotina) que venha a levantar um erro do tipo EConvertError, uma mensagem genérica será exibida.
Este objeto deve estar inserido ou no formulário principal ou no formulário especial denominado Data Module como veremos adiante.

TRATAMENTO DE EXCEÇÕES SILENCIOSAS

Podemos utilizar o comando *Abort* para gerar exceções silenciosas, ou seja, sem nenhuma mensagem.

```
  try
    Form1.Caption :=
                 FloatToStr(StrToFloat(Edit1.Text) / StrToFloat(Edit2.Text));
  except
    on EZeroDivide do
      begin
        Beep;
        ShowMessage('Divisão por zero');
      end;
    on EInvalidOp do ShowMessage('Operação inválida');
    else
      Abort;
  end;
```

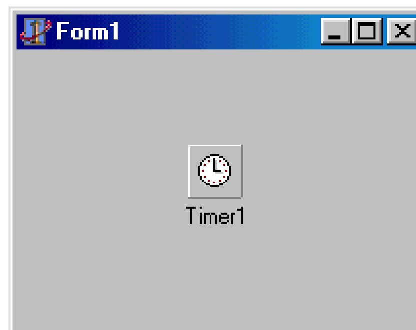
Sugestão: Exercício 8

UM POUCO MAIS SOBRE COMPONENTES (VCL)

Objeto – Timer (Temporizador)

Paleta – System

Importância: Permite a execução de rotinas em loop, em um intervalo pré-definido.



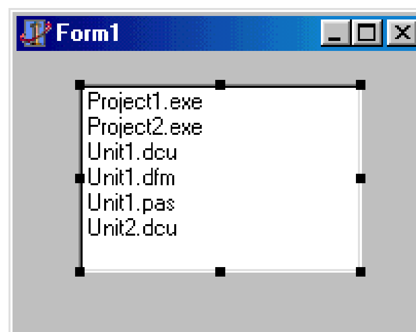
Propriedades

Enabled	Permite 'ligar' o timer, ou seja, ele entra em um loop executando o evento OnTimer até que seja atribuído falso ou terminada a aplicação.
Interval	Define em milisegundos o intevalo de repetição do evento OnTimer.

Objeto – FileListBox (Caixa de listagem de arquivos)

Paleta – Win 3.1

Importância: Permite listar arquivos de determinado diretório.



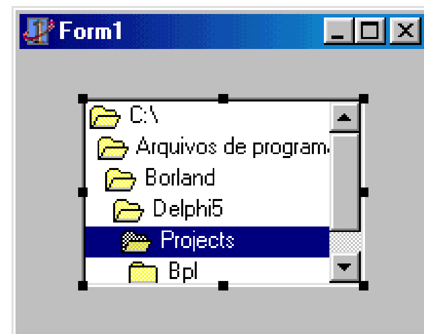
Propriedades

FileEdit	Define uma caixa de edição (TEdit) que exibirá o arquivo atualmente selecionado.
FileName	Define o nome do arquivo selecionado. Válido em tempo de execução.
Mask	Define máscaras de filtro (separadas por ponto e vírgula) para a exibição dos arquivos.

Objeto – DirectoryListBox (Caixa de listagem de diretórios)

Paleta: Win 3.1

Importância: Permite listar os diretórios do drive desejado.



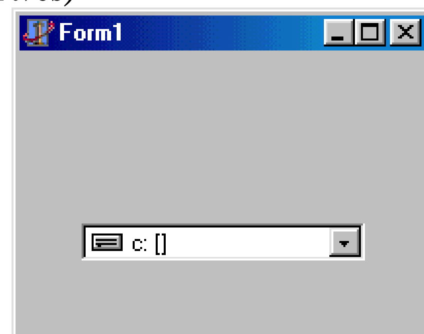
Propriedades

DirLabel	Permite exibir o diretório corrente com a propriedade Caption de um componente do tipo TLabel.
FileList	Permite a conexão com um componente TFileListBox.

Objeto - DriveComboBox (Caixa de listagem de drives)

Paleta: Win 3.1

Importância: Permite listar os drives disponíveis no computador.



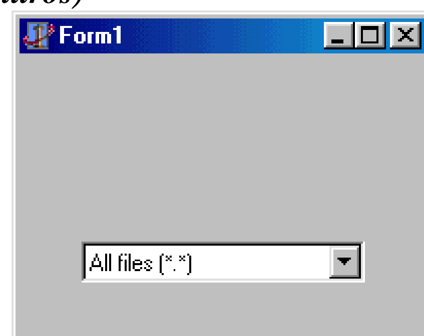
Propriedades

DirList	Permite a conexão com um componente TDirectoryListBox.

Objeto – FilterComboBox (Caixa de listagem de filtros)

Paleta: Win 3.1

Importância: Permite estabelecer filtros para visualização de arquivos.



Propriedades

FileList	Permite a conexão com um componente TFileListBox.
Filter	Permite definir as máscaras de filtro de arquivo a serem exibidas.

BANCO DE DADOS

Na maioria dos casos, as aplicações que vamos construir com o Delphi, ou outra ferramenta visual, deve manipular um banco de dados.
Um banco de dados pode ser visto de diferentes perspectivas, pode ser um arquivo ou pode ser um diretório com vários arquivos. Vamos entender essas diferenças um pouco mais adiante.

MODELAGEM BÁSICA

Se você já tem alguma experiência com a *teoria* de banco de dados, tenha paciência, vamos estudar *rapidamente* neste tópico alguns conceitos básicos que independem da plataforma no desenvolvimento de banco de dados.

- O *Modelo Conceitual* procura abstrair a realidade independente da plataforma de harware ou software, é uma visão global na construção da aplicação.
- O *Modelo Lógico* define as regras básicas e quais os dados devem ser armazenados no banco e depende das características do software, é também dependente do modelo conceitual.
- O *Modelo Físico* implementa as definições do modelo lógico na 'prática', ou seja, desenvolver o projeto definido anteriormente, depende dos recursos de software e hardware.

Modelo Conceitual e Lógico

A princípio vamos entender como podemos definir os dados que serão armazenados em um computador, através do conceito de **entidade**.
Ao pensarmos em cadastrar dados de clientes, alunos, fornecedores, etc... temos exemplos de entidades. A entidade possui propriedades que serão identificados como os dados propriamente ditos, estas propriedades são chamadas de **atributos**[8], ou seja:

Entidade	Alunos
Atributos	AluCodigo
	AluNome
	AluDataNasc

Entidade	Funcionario
Atributos	FunCodigo
	FunCPF
	FunTelefone

[8] Os atributos devem ter um tipo de dados definido (Inteiro, AlfaNumérico, Data, etc..)

Modelo Físico

No modelo físico a **entidade** se chama ***tabela*** e os **atributos** se chamam ***campos***. A linha de dados que deriva do conjunto de campos se chama **registro**. Exemplo:

Tabela: Funcionario		
FunCodigo	FunNome	FunDataNasc
1	Edileumar Jamaica	01/01/1975
2	Jorgina Alambradus	25/04/1979

Campos

Registros

É necessário definirmos nas entidades (tabelas) um *campo* que seja **idenficador** de cada *registro* ou seja, um dado que **não** deve repetir, identificando aquele registro como único.
Exemplo: FunCodigo

Esse campo recebe o nome de **chave primária** porque ele é capaz de identificar aquele determinado registro como único, além disso, a tabela começa a ser ordenada pelos dados deste campo.

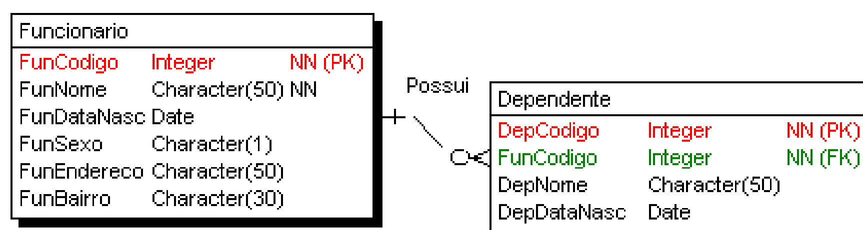
Relacionamentos

O conceito: "Banco de Dados Relacional" significa que as tabelas que compõem o banco "relacionam-se entre si". Um exemplo: A tabela Funcionarios tem um relacionamento com a tabela Dependentes.

Relacionamento é '*uma regra de associação entre duas ou mais entidades*'.

O relacionamento pode ser representado através de *cardinalidades,* onde são definidas restrições que as entidades devem obedecer.
Existem três tipos básicos de relacionamentos: **1:1** , **1:N** ou **N:N**.

Existem regras a seguir, por exemplo: no relacionamento **1:N** ou **N:1** (este é o relacionamento que é mais utilizado), a chave primária (Primary Key – PK) da entidade **1** deve aparecer como **chave estrangeira** (Foreign Key – FK) na entidade **N**.
Exemplo de uma notação em um software CASE:



No caso de um relacionamento **N:N** deverá ser criada uma 3º tabela onde haverá no *mínimo* as chaves primárias das entidades relacionadas. Estes atributos devem fazer parte da chave

primária desta nova tabela e funcionar como chave estrangeira da tabela onde se originou o relacionamento.
Exemplo de uma notação em um software CASE:

Visão física do banco de dados

O banco de dados pode ser visto (**fisicamente**) como um arquivo ou como um diretório (pasta), isso dependerá da estrutura que o fabricante do banco de dados vai abordar, exemplo:

No produto Microsoft *Access* o banco de dados é representado fisicamente como um **único arquivo** (.MDB) contendo as tabelas e demais recursos. Para o Delphi o banco de dados é representado pelo arquivo .MDB

Nome	Tamanho	Tipo
Empresa.mdb	140KB	Arquivo MDB

Para tabelas do banco de dados *Paradox* cada tabela dará origem a um arquivo .DB e arquivos separados para índices (.PX) entre outros recursos. No exemplo abaixo, a tabela (.DB) contém campos BLOB (figuras ou memorandos), estes dados serão armazenados em um arquivo .MB Neste caso, para o Delphi, o banco de dados é o **diretório** (pasta) onde se encontram os arquivos que compõem o banco de dados.

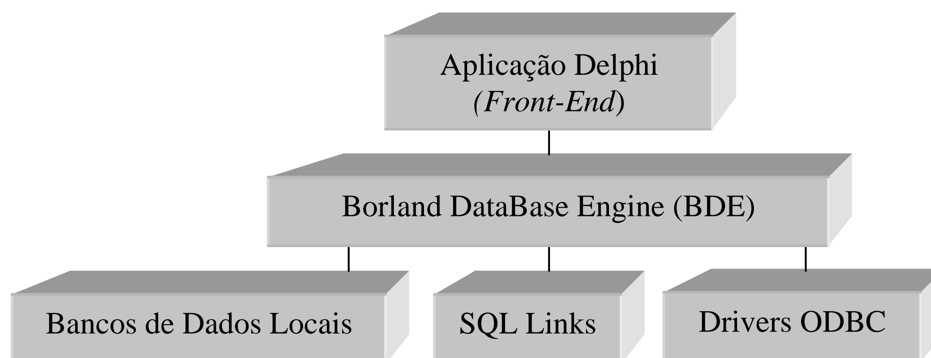
Nome	Tamanho	Tipo
biolife.px	6KB	Arquivo PX
biolife.mb	1.204KB	Arquivo MB
biolife.db	10KB	Arquivo DB

CONECÇÃO AO BANCO DE DADOS

O Delphi utiliza uma estrutura de camadas para fazer com que o *front-end* (formulário) manipulado pelo usuário venha interagir com a *base de dados*. O caminho deve ser percorrido por uma série de componentes configurados entre si, porém, há uma camada intermediária que não fica *dentro* do ambiente Delphi, nem diretamente preso ao *banco*, é o BDE.

BDE

O BDE é um conjunto de DLLs que deve acompanhar as aplicações que fazem uso de seus recursos de acesso ao banco. É nesta camada que são definidas características específicas de cada banco de dados, bem como sua localização, ou seja, o *front-end* não acessa diretamente a *base de dados*, o BDE é responsável para estabelecer este funcionamento.



Existem outras maneiras de acessarmos banco de dados SEM O BDE, a paleta InterBase disponível a partir do Delphi 5 é um bom exemplo.

COMPONENTES DE CONTROLE E ACESSO

O sistema para conexão com o banco de dados utiliza além do BDE um conjunto de componentes denominados: Session, DataBase, *DataSet*, *DataSource* e *Data-Aware*. O componente Session não será o foco de nosso estudo.

Uma visão 'geral' sobre estes componentes pode ser vista da seguinte maneira:

- ***Session***: Aplicações simples, trabalham com apenas um banco de dados. Porém o Delphi permite mais de uma conexão simultânea à bancos de dados distintos, e também mais de uma conexão com o mesmo banco de dados. O controle *global* das conexões é feito através do componente da classe TSession, criado *automaticamente* pelo Delphi na execução do programa. Esse componente representa a sessão *default* da aplicação.

- ***DataBase***: O componente DataBase é responsável pela conexão da aplicação a um banco de dados com a finalidade maior de implementar segurança (*transações*) *e* definir características de comunicação entre uma aplicação Delphi-Client/Server. Embora em aplicações locais, sua utilização *explicita* é recomendada.
- ***DataSet***: Existem três componentes que descendem de uma classe chamada TDataSet e implementam importantes métodos de manipulação de banco de dados além de suas características específicas. De fato, não utilizamos a classe TDataSet diretamente, mas através dos componentes TTable, TQuery(SQL) e TStoreProc. Estes componentes estão na paleta DataAccess
- ***DataSource***: O componente DataSource é responsável por conectar os componentes Data-Aware à uma determinada tabela representada pelo DataSet. Este componente está na paleta DataAccess.
- ***Data-Aware***: Os componentes Data-Aware são responsáveis pela visualização e manipulação direta dos dados. Todo componente Data-Aware tem uma propriedade para conectar-se ao DataSource correspondente à tabela destino. Estes componentes estão na paleta DataControls.

Exemplo

Vamos exemplificar a utilização de componentes básicos de *acesso* e *controle* através de um exemplo baseado em uma tabela (arquivo de dados) já pronta, criada na instalação do Delphi. O objetivo é entendermos o funcionamento destes componentes.

1. Crie uma nova aplicação e salve-a na pasta especificada pelo instrutor.

    O Formulário: UFrmPeixes
    Projeto: Peixes

2. Insira dois componentes: Um DataSource e um Table. Configure suas propriedades de acordo com a orientação abaixo:

```
object TbPeixes: TTable
  DatabaseName = 'DBDEMOS'
  TableName = 'biolife.db'
  Active = True
  Name = TbPeixes
end
object DsPeixes: TDataSource
  AutoEdit = False
  DataSet = TbPeixes
  Name = DsPeixes
end
```

3. Insira um componente DBGrid e configure-o:

```
object DBGrid1: TDBGrid
  Align = alBottom
  DataSource = DsPeixes
end
```

4. Insira um componente DBImage e configure-o:

```
object DBImage1: TDBImage
  DataField = 'Graphic'
  DataSource = DsPeixes
  Stretch = True
end
```

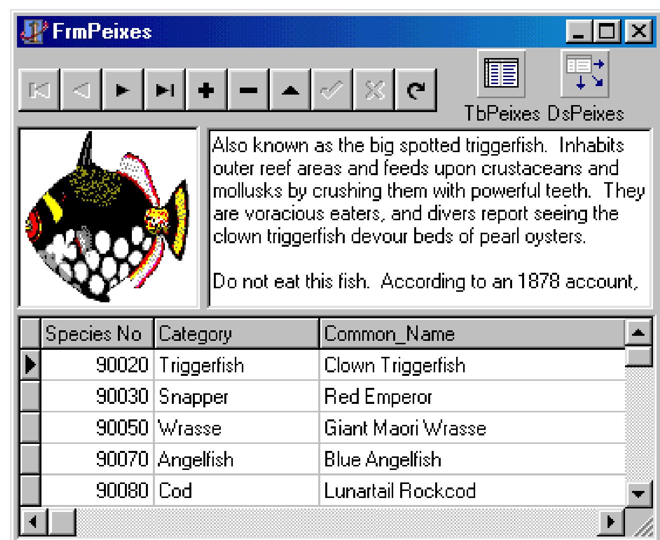
5. Insira um componente DBMemo e configure-o:

```
object DBMemo1: TDBMemo
  DataSource = DsPeixes
  DataField = 'Notes'
end
```

6. Insira um componente DBNavigator e configure-o:

```
object DBNavigator1: TDBNavigator
  DataSource = DsPeixes
end
```

Uma sugestão do visual pode ser a seguinte:



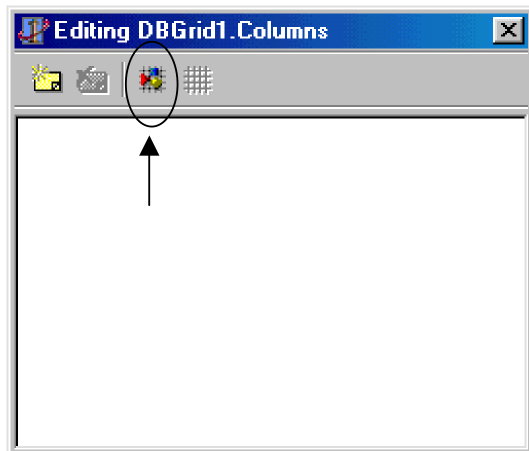
7. Salve novamente e execute a aplicação.

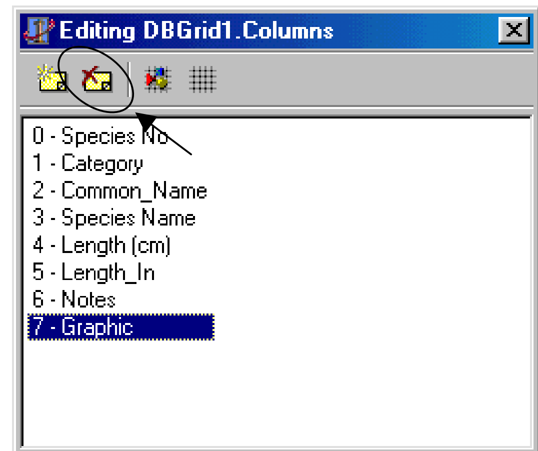
Podemos perceber que a aplicação funciona manipulando a tabela ´biolife.DB´ sem a necessidade de nenhuma linha de código explícita.

É claro que este é apenas um exemplo para entendermos o mecanismo de componentes de acesso e controle. Desse modo, veremos a seguir que linhas de código serão necessárias para implementar funcionalidades específicas (pesquisas), ou mesmo básicas como inclusão, edição e remoção.

Por último, vamos personalizar o componente *DBGrid* para *não* exibir os campos *Grafic* e *Notes* (campos BLOB).

Pode-se usar a propriedade **Columns** ou um duplo clique no componente *DBGrid*, O Editor de colunas é exibido, clique no ícone **Add All Fields;** os campos disponíveis serão exibidos, selecione o campo desejado (Grafic) e clique em **Delete Selected**





OBJETOS TFIELD

Vamos criar uma outra aplicação e exemplificar um recurso muito importante na construção de aplicações baseadas em banco de dados os ***campos persistentes***.

1. Crie uma nova aplicação e salve-a na pasta especificada pelo instrutor. Os podem ser:

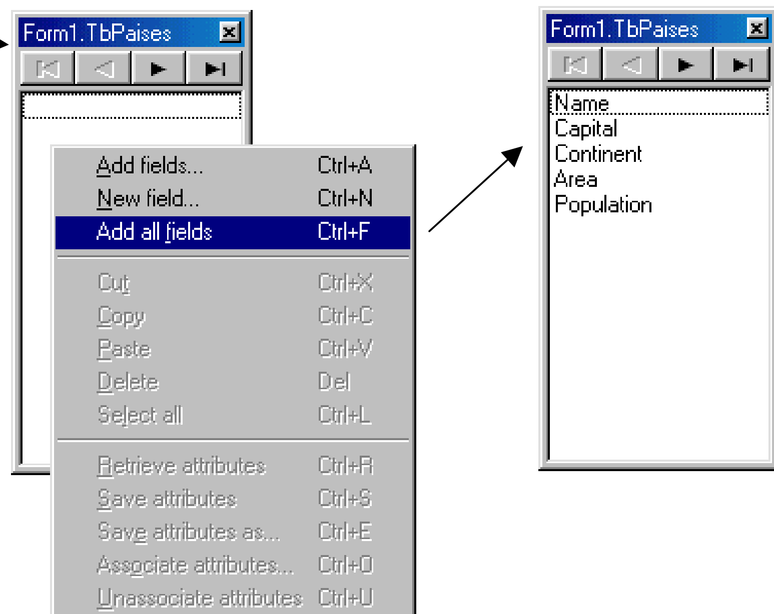
   O Formulário: UFrmPaises
   Projeto: Paises

2. Insira dois componentes: Um DataSource e um Table. Configure suas propriedades de acordo com a orientação abaixo:

```
object TbPaises: TTable
  DatabaseName = 'DBDEMOS'
  TableName = 'country.db'
  Active = True
  Name = TbPaises
end

object DsPaises: TDataSource
  AutoEdit = False
  DataSet = TbPaises
  Name = DsPaises
end
```

3. Agora, ao invés de colocarmos componentes *data-aware* (paleta DataControls) diretamente no form, vamos definir os campos persistentes através de um duplo clique componente Table surgirá o editor de campos (**Fields Editor**). Clique com botão direito no editor e escolha Add all fields.



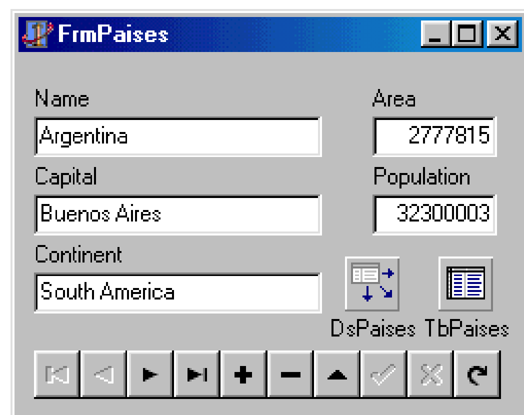
A partir de agora, cada campo na tabela é considerado um ***objeto TField*** com suas *propriedades* individualizadas na Object Inspector.

Mas, o melhor de tudo é o fato de podermos **arrastar** os campos TField diretamente **para o formulário**, e perceber que o Delphi se encarrega de construir o componente *data-aware* necessário a cada campo.
Isso realmente poupa muito tempo no desenvolvimento de aplicações em banco de dados.
Porém há a possibilidade de criar aplicativos que utilizem acesso à banco sem data-aware, nosso estudo não vai enfocar esta metodologia.

4. Insira um componente DBNavigator e configure-o:

```
object DBNavigator1: TDBNavigator
  DataSource = DsPaises
end
```

Uma sugestão para a disposição dos componentes pode ser:



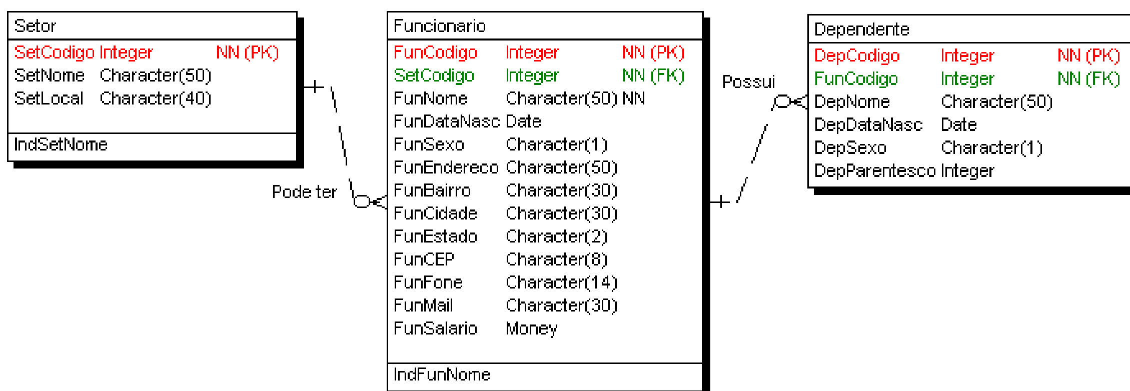
5. Salve novamente e execute a aplicação.

A principal vantagem dos campos persistentes é a facilidade de criação e a definição de propriedades importantes como EditMask e Required por exemplo.

APLICAÇÃO EM BANCO DE DADOS

Vamos iniciar um pequeno projeto que exemplificar a construção de uma aplicação de banco de dados com 'todos' os passos básicos, que vão desde a construção das tabelas até a geração de relatórios e instalação do aplicativo.

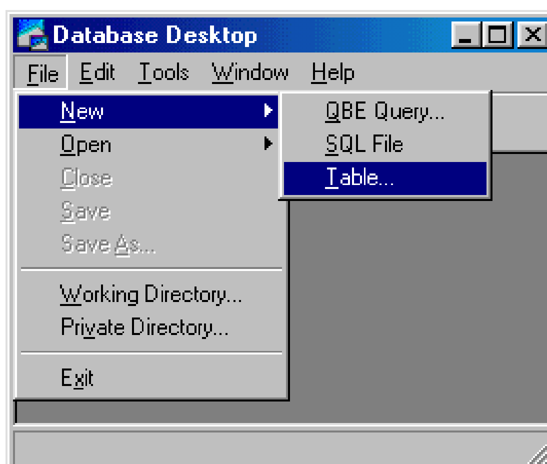
O 'projeto' será de um banco de dados local e possui três tabelas com a seguinte definição de campos e relacionamentos:



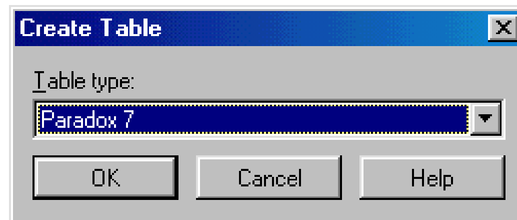
Para criarmos as tabelas devemos utilizar um gerenciador de banco de dados (Paradox, Access) ou algum programa específico para esta tarefa. O Delphi instala um aplicativo para esta finalidade denominado ***DataBase DeskTop***. Porém observe que são itens *distintos*: O Delphi **não** cria as tabelas, ele apenas **usa** o banco de dados.

DATABASE DESKTOP

Sua utilização é muito simples. Para criar as tabelas, basta clicar no comando **File | New | Table**.

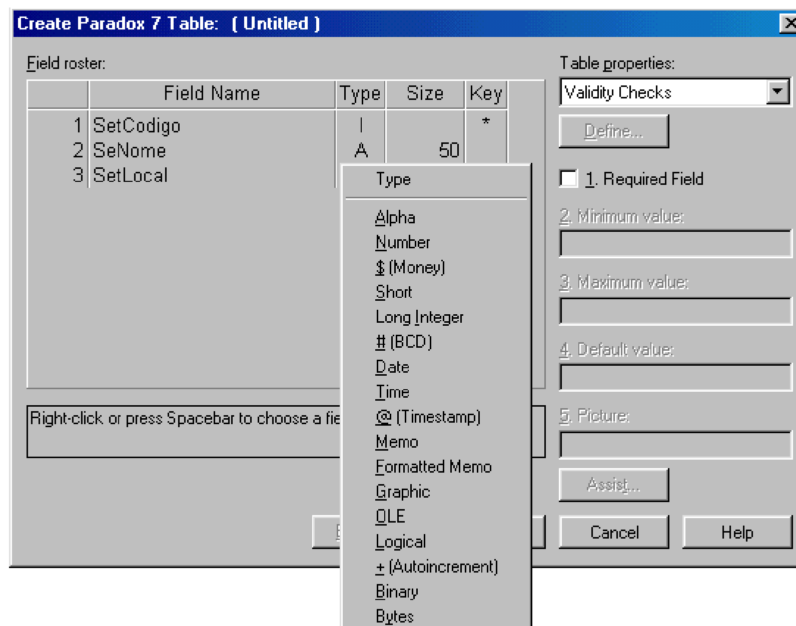


Uma janela para definição do *tipo* do banco de dados será exibida. Vamos trabalhar com o banco de dados **Paradox**.
Cada tabela em Paradox, significa um arquivo em disco com extensão .DB. Neste caso o conceito de banco de dados é definido pelo *diretório* onde estão as tabelas e não o nome do arquivo que o caso de outros bancos como Access, por exemplo.

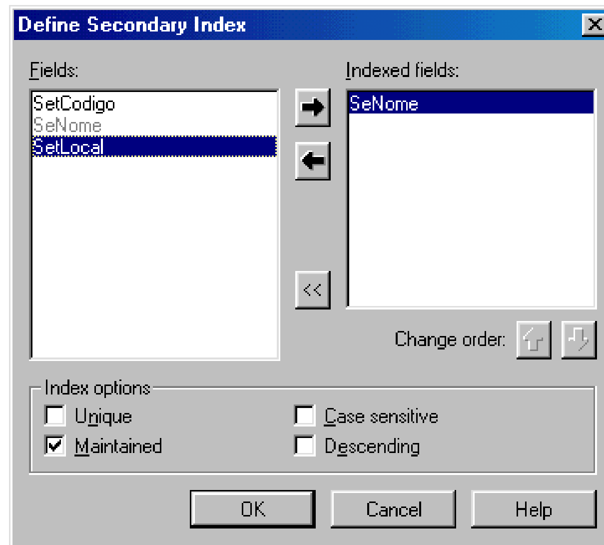


A definição dos campos deve ser feita na próxima tela, onde:

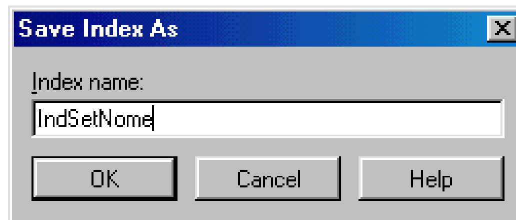
- *Field Name*: Define o nome do campo da tabela.
- *Type*: Define o tipo de dados do campo. Tecle a barra de rolagem para obter uma lista dos possíveis tipos.
- *Size*: Tamanho do tipo de dados definido no item Size. Alguns campos já são padronizados não permitindo a alteração.
- *Key*: Define se o campo é chave primária. Tecle a barra de espaço para ligar ou desligar. Um asterisco é exibido para confirmação da chave ligada.
- *Table properties*: Define uma série de itens como *Validações, Integridade referencial,* e um item muito importante – *Índices Secundários*.



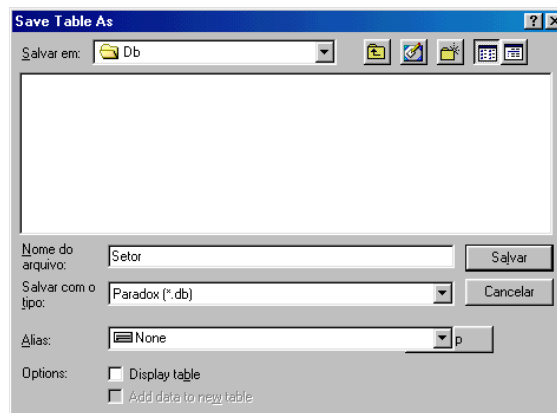
Para definir índices secundários, clique em *Table properties* e escolha *Secondary Indexes* e clique em *Define*.
Na janela exibida selecione o campo desejado para criar o índice secundário, neste exemplo SetNome. Através da seta, selecione-o para a lista *Indexed fields*.



Clique em Ok e defina um nome para o índice na próxima janela.



Após definidos os campos e chaves secundárias, grave a tabela através do botão Save As...
O diretório será definido pelo instrutor.



Crie as demais tabelas com o mesmo raciocínio.

BDE – CRIAÇÃO DO ALIAS

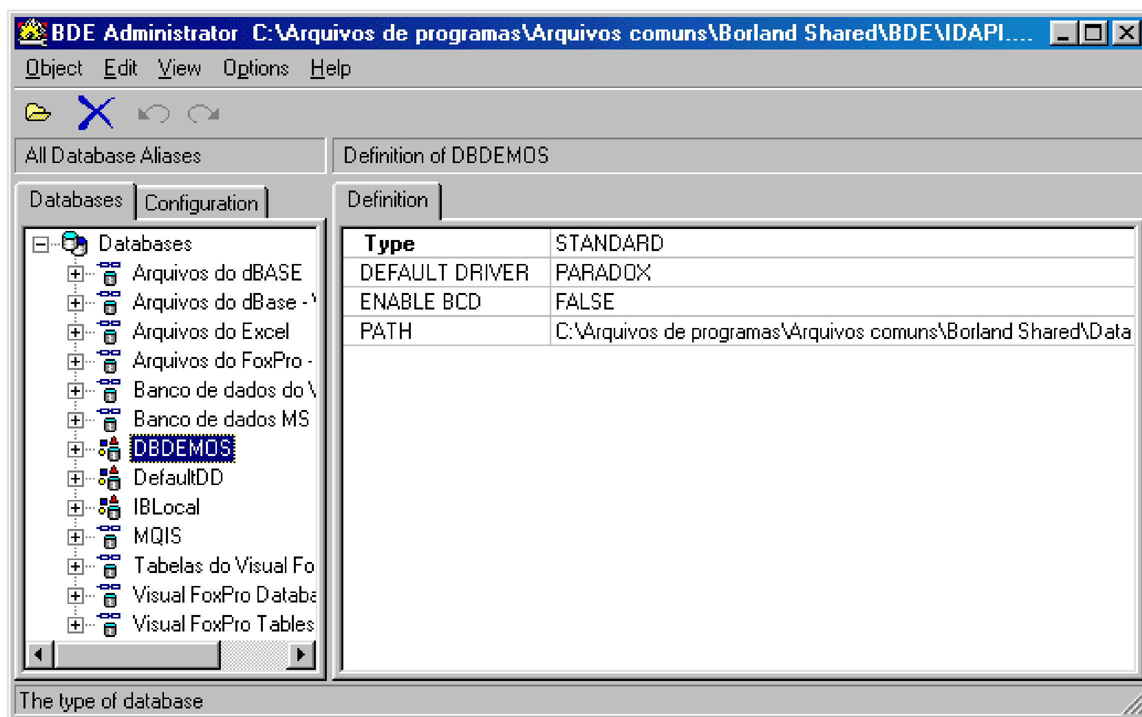
Vamos explorar com mais detalhes o conceito e funcionamento da camada BDE. O BDE assim como o DataBase DeskTop é instalado junto com o Delphi[9]. Execute o programa e examine a tela inicial.

A tela é dividida em duas partes básicas: A listagem de ***Aliases*** do lado esquerdo e as configurações de cada *Alias* no lado direito.

Um conceito importantíssimo é a definição de ***Alias***. Cada nome do lado esquerdo do BDE representa uma configuração para acesso a um determinado banco. Exemplo: O ***Alias*** BDDEMOS representa o diretório *C:\Arquivos de programas\Arquivos comuns\Borland Shared\Data,* o driver para acesso ao de banco (Paradox) entre outras configurações.

Dessa maneira o desenvolvedor utiliza o nome DBDEMOS dentro do Delphi porém o diretório pode ser alterado no BDE sem maiores problemas.

O ***Alias*** é uma string que define o path e configurações para acesso ao banco de dados.



[9] Veremos adiante que sua instalação na máquina do usuário também será necessária, caso a aplicação em Delphi utilize acesso a banco de dados.

Para criar um alias:

- Clique no menu **Object** e escolha **New**. (CTRL+N).
- Confirme o tipo de driver (Paradox).
- Defina um nome para o Alias (lado esquerdo).
- Defina o *path* através do botão reticências.
- Clique no menu **Object** e escolha **Apply**, para confirmar as alterações.
- Confirme a gravação.

APLICAÇÃO UTILIZANDO BD EM DELPHI

O próximo passo é construir a aplicação no Delphi, acessando o banco de dados através do ALIAS do BDE. O projeto terá a função básica de manipular os dados das tabelas, localiza-los através de consultas simples e emitir relatórios simples e compostos.

FrmPrincipal

Ao abrir uma nova aplicação, grave o projeto dentro de uma pasta: *Empresa*

✓ A primeira unidade como *UfrmPrincipal*
✓ E o projeto como *Empresa*

No formulário insira um componente MainMenu e crie os itens de Menu abaixo.

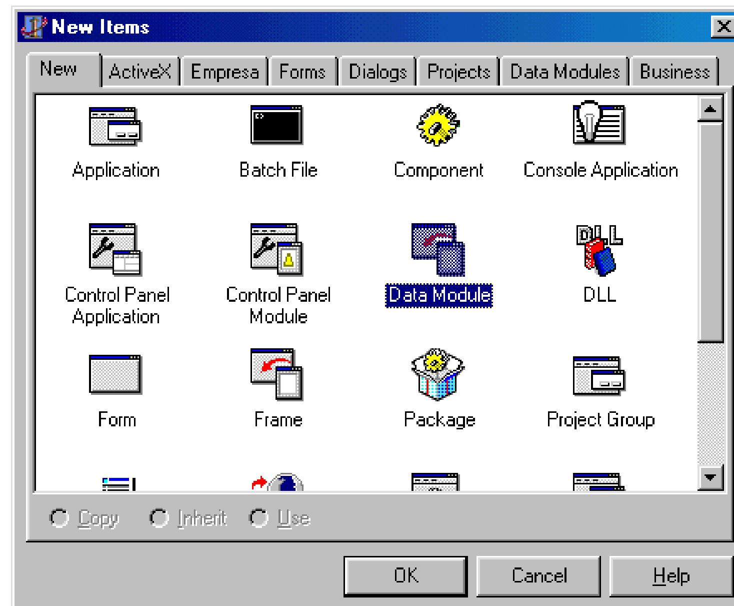
DATA MODULE

A definição dos componentes de acesso aos dados serão *centralizados* em um formulário especial chamado ***Data Module***. Este formulário tem uma característica bem específica: é um form invisível e só recebe componentes *invisíveis*[10], ou seja, o usuário nem vai imaginar o DataModule e os componentes que ele recebe. A sua finalidade é centralizar os componentes para que *qualquer formulário* possa ter acesso aos dados, sem a necessidade de *repetir* os componentes de acesso em cada formulário.

Para criar um formulário DataModule, clique em **File | New.**

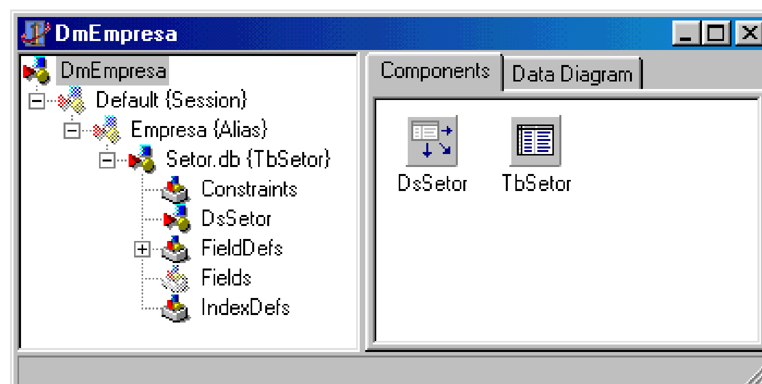
Na guia New, clique no *ícone* DataModule e confirme OK.

[10] Com relação ao usuário.



✓ Salve o novo formulário com o nome de *UFrmDmEmpresa*.

Insira dois componentes: Um DataSource e um componente Table.



As propriedades podem ser configuradas como:

```
object TbSetor: TTable
  DatabaseName = 'Empresa'
  TableName = 'Setor.db'
  Active = True
  Name = TbSetor
end

object DsSetor: TDataSource
  AutoEdit = False
  DataSet = TbSetor
  Name = DsSetor
end
```

DataSource

AutoEdit	Permite habilitar os controles para editar os dados automaticamente.
DataSet	Permite definir o componente DataSet (Table ou Query ou StoreProc)

Table

Active	Permite ‘abrir’ a tabela para permitir sua manipulação.
DataBaseName	Permite um *path* fixo ou um ALIAS para acesso ao banco de dados.
TableName	Permite escolher uma tabela após a definição da propriedade DataBaseName.

O mesmo procedimento pode ser feito para as outras duas tabelas.
Um par de componentes de acesso e as respectivas propriedades listadas acima.

FORMULÁRIO DE CADASTRO DE SETOR

FrmCadSetor

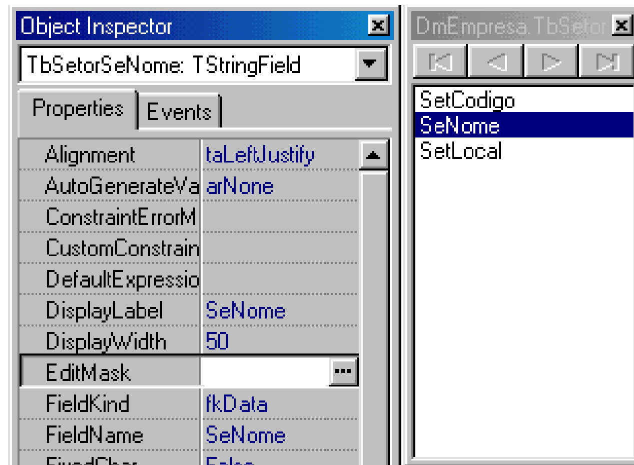
Crie um **novo formulário** e insira dois componentes Panel, um alinhado na parte inferior e outro na área cliente.

- ✓ Salve o form. O nome do arquivo será: *UFrmCadSetor*.
- ✓ A propriedade Name do form será: *FrmCadSetor*

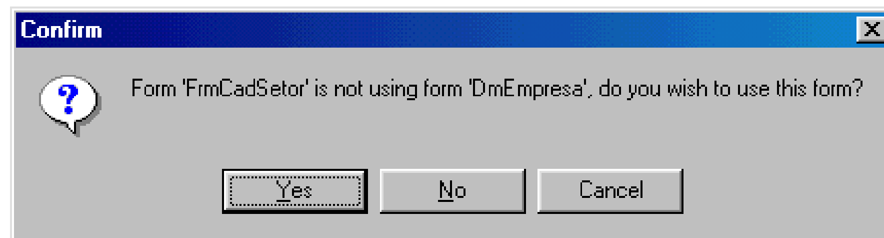
Uma característica importante na construção de aplicações utilizando banco de dados é a possibilidade de trabalharmos com campos TFIELD exemplificados anteriormente, ou seja, campos que são considerados como objetos dentro da aplicação, sendo assim estes campos possuem propriedades e eventos individuais através da object inspector ou por atribuições manipulando código.
Vamos explorar um pouco mais da funcionalidade deste recurso. Campos TFIELD são chamados de campos persistentes e sua criação é extremamente simples.

- No DataModule, localize o objeto Table referente à tabela desejada.
- Um duplo clique no objeto Table, exibirá a janela de manipulação dos campos
- Um clique *dentro da janela* com o botão direito exibirá o speed menu.
- Escolha a opção **Add All Fields**.

Após a criação dos campos persistentes, pode-se seleciona-lo e verificar propriedades específicas na Object Inspector. As propriedades serão diferentes dependendo do tipo de dados do campo, neste exemplo um tipo TStringField.



Mas o melhor está para vir. Posicione a janela dos campos persistentes em um lado da tela e com o formulário FrmCadSetor (criado anteriormente) visível, selecione os campos e arraste-os para dentro do Form. Uma pergunta do tipo:



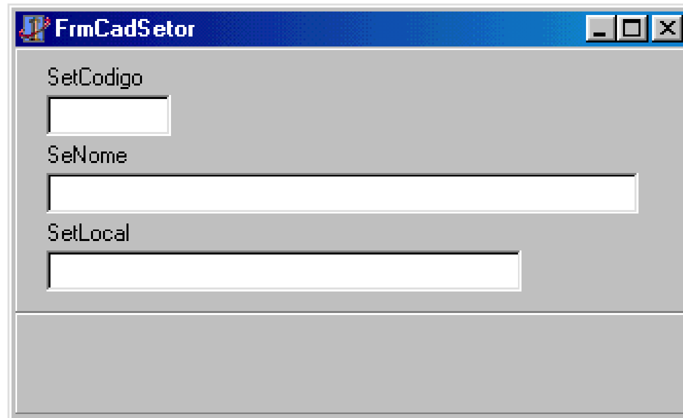
Será exibida. Confirme com Yes, ou seja, o Delphi está questionando se no formulário '*FrmCadSetor*' deve haver uma referência ao formulário '*DmEmpresa*' (DataModule). Ao confirmar o Delphi adiciona uma referência no **Code Editor** da unidade *UFrmCadSetor* na seguinte estrutura:

```
implementation

uses UDMEmpresa;
```

Embora o Delphi faça esta tarefa de maneira automática, é importante saber disto? Claro! Se você por 'falta de cuidado', executar vários forms sem salvar fazendo referências entre eles, esta cláusula utilizará os nomes padrões (Unit1, Unit2, etc...) e quando você resolver salvar... o Delphi passa a não encontrar mais estas referências de nomes antigos. Neste caso pode-se manipular o Code Editor e em algumas situações, apenas apagar o nome errado, a referência correta será refeita.

Pode-se fechar o editor de campos persistentes. Uma diferença visual que tem destaque agora são os controles de manipulação de dados que foram inseridos automaticamente pelo Delphi.



Embora se pareçam com um componente já visto anteriormente (Edit) estes componentes são provenientes da paleta Data Controls e tem o nome de DBEdit, ou seja, um controle específico para manipulação de dados provenientes de um banco de dados. Já nos referimos a estes controles nos primeiros exemplos do capítulo de Banco de Dados.

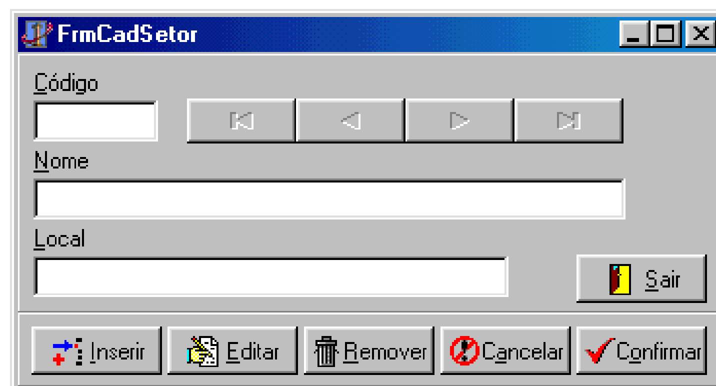
DBEdit

DataSource	Permite 'conectar' o controle à fonte de dados.
DataField	Permite especificar um campo da tabela referenciada em DataSource.

Insira um componente **DBNavigator** e modifique suas propriedades:

```
object DBNavigator1: TDBNavigator
  DataSource = DmEmpresa.DsSetor
  VisibleButtons = [nbFirst, nbPrior, nbNext, nbLast]
end
```

Insira seis componentes BitBtn, e configure apenas as propriedades Name, Caption e Glyph. Uma sugestão visual pode ser vista a seguir:



MÉTODOS E PROPRIEDADES PARA MANIPULAÇÃO DE DADOS

Para **percorrer** os registros de uma tabela podemos utilizar os seguintes *métodos*:

First	Move o cursor para o primeiro registro do dataset.
Prior	Move o cursor para o registro anterior (se houver).
Next	Move o cursor para o próximo registro (se houver).
Last	Move o cursor para o último registro do dataset.
MoveBy(N)	Move o cursor para frente ou para trás, conforme o valor do parâmetro N, que é um valor inteiro, positivo ou negativo.

Para **controlar** o início e fim da tabela podemos utilizar as *propriedades*:

BOF : Boolean	*Begin Of File*. Indica True se o cursor estiver no primeiro registro da tabela.
EOF : Boolean	*End Of File*. Indica True se o cursor estiver no último registro da tabela.

Por exemplo:

```
Table1.First;
While not Table1.EOF do
begin
  comando1;
  comando2;
  ...
  Table1.Next;
end;
```

Um incoveniente com relação ao código acima é que os controles *data-aware* (data controls) serão atualizados no monitor à medida em que o método Next percorre a tabela, gerando uma visualização indesejada e perda de tempo com a sua atualização.
Neste caso, pode-se utilizar os métodos *DisableControls e EnableControls*.

```
Table1.DisableControls;
try
 Table1.First;
 while not Table1.EOF do
 begin
  comando1;
  comando2;
  ...
  Table1.Next;
 end;
finally
 Table1.EnableControls;
end;
```

Em nosso exemplo atual, no formulário de Cadastro de Setores, os códigos para percorrer o *dataset* foram implementados de maneira automática através do componente **DBNavigator**.

Porém nada impede a criação de botões independentes e a definição de *handlers* de controle para navegação retirando o DBNavigator.

Para **alterar** (editar) os valores de um *dataset* podemos utilizar o seguinte método:

Edit	Permite editar o data set para alteração dos valores atuais para novos valores.

Exemplo:
```
  Table1.Edit;
```

Para **inserir** um novo registro em um *dataset* podemos utilizar dois métodos:

Append	Cria um novo registro após o *último* registro do dataset.
Insert	Cria um novo registro após o registro corrente.

Exemplo:
```
  Table1.Append;
Ou
  Table1.Insert;
```

Para **remover** um registro em um *dataset* podemos utilizar o método:

Delete	Remove o registro corrente.

Exemplo:
```
if MessageDLG('Está certo disto?!',mtConfirmation,[mbYes,mbNo],0) = mrYes
  then Table1.Delete;
```

Para **confirmar** (gravar) as alterações[11] no *dataset* podemos utilizar o método:

Post	Confirma as alterações em um dataset.

Exemplo:
```
Table1.Append;        |    Table1.Edit;
...                   |    ...
Table1.Post;          |    Table1.Post;
```

Para **acessar** o valor em um campo através de *TField* podemos utilizar a propriedade *Fields* ou o método *FieldByName*.

Fields	Faz referência a um campo específico através de um *array*.
FieldByName	Faz referência a um campo através de seu nome.

[11] Inserções são consideradas alterações.

Para **cancelar** algum comando enviado ao banco de dados, utiliza-se o método cancel.

Cancel	Confirma as alterações em um dataset.

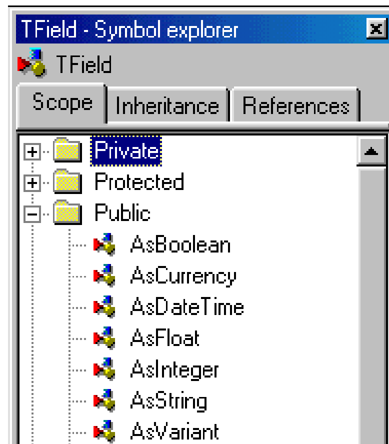
Exemplo:

```
Table1.Append;
...
Table1.Cancel;
```

```
Table1.Edit;
...
Table1.Cancel;
```

FUNÇÕES DE CONVERSÃO

Tanto a propriedade *Fields* como o método *FieldByName* utilizam *funções de conversão* para acesso aos campos.



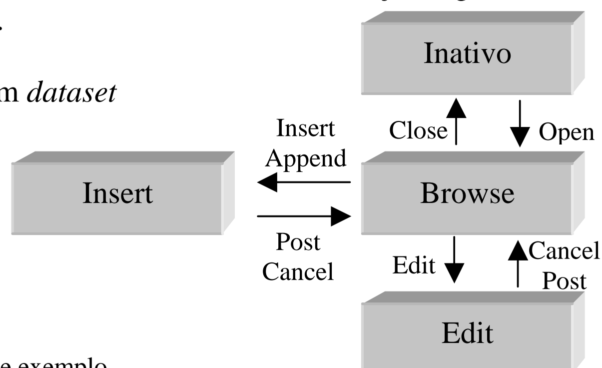
Exemplo:

```
Label1.Caption := Table1.Fields[0].AsString;
Label2.Caption := Table1.FieldByName('SetNome').AsString;
Edit1.Text := Table1.FieldByName('SetLocal').Value;
```

OS ESTADOS DE UM DATASET

Os *datasets* trabalham com 'estados'. Em cada estado uma determinada operação pode ser válida, outras não. A manipulação do *dataset* através de métodos e funções agem nestes estados em todo o tempo provocando mudanças.

Uma visão geral sobre os estados[12] de um *dataset* pode ser vista a seguir:



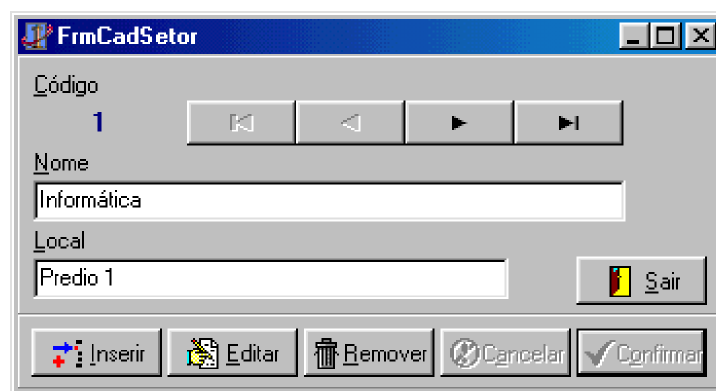
[12] O estado de pesquisa não foi considerado neste exemplo.

- ✓ Vamos utilizar no formulário **FrmCadSetor** a prática destes conceitos. Antes, troque o DBEdit1 referente ao código para um componente DBText configure as propriedades *DataSource* para a tabela Setor e *DataField* para o campo SetCodigo.

- ✓ O BitBtn referente à saída do form pode ter sua propriedade *Cancel* verdadeira.

- ✓ O BitBtn referente à confirmação das operações pode ter sua propriedade *Default* verdadeira e a propriedade *Enabled* como falso.

- ✓ O BitBtn referente ao cancelamento das operações pode ter sua propriedade *Enabled* como falso.




```
unit UFrmCadSetor;
{A interface não foi impressa}
implementation

uses UDMEmpresa;

{$R *.DFM}

procedure TFrmCadSetor.BbtSairClick(Sender: TObject);
begin
  FrmCadSetor.Close;
end;

{Para criar o procedimento TrataBotoes, digite sua declaração
(sem TFrmCadSetor) na cláusula private e utilize CTRL+SHIFT+C
Para todos os demais, selecione o objeto e utilize a object inspector}

procedure TFrmCadSetor.TrataBotoes;
begin
  BbtInserir.Enabled := not BbtInserir.Enabled;
  BbtEditar.Enabled := not BbtEditar.Enabled;
  BbtRemover.Enabled := not BbtRemover.Enabled;
  BbtCancelar.Enabled := not BbtCancelar.Enabled;
  BbtConfirmar.Enabled := not BbtConfirmar.Enabled;
  BbtSair.Enabled := not BbtSair.Enabled;
  DbNavigator1.Enabled := not DbNavigator1.Enabled;
end;
```

```
procedure TFrmCadSetor.FormCloseQuery(Sender: TObject;
  var CanClose: Boolean);
begin
  if DmEmpresa.TbSetor.State in [dsEdit, dsInsert] then
    if MessageDLG('Existem dados pendentes,'+#13+'deseja gravá-los?',
                  mtConfirmation, [mbYes,mbNo], 0) = mrYes then
       CanClose := False
    else
      begin
        DmEmpresa.TbSetor.Cancel;
        TrataBotoes;
        CanClose := True;
      end;
end;

procedure TFrmCadSetor.BbtInserirClick(Sender: TObject);
var ProxNum: Integer;
begin
  TrataBotoes;
  DmEmpresa.TbSetor.Last;
  ProxNum := DmEmpresa.TbSetor.FieldByName('SetCodigo').AsInteger + 1;
  DmEmpresa.TbSetor.Append;
  DmEmpresa.TbSetor.FieldByName('SetCodigo').AsInteger := ProxNum;
  DbEdit2.SetFocus;
end;

procedure TFrmCadSetor.BbtEditarClick(Sender: TObject);
begin
  DmEmpresa.TbSetor.Edit;
  TrataBotoes;
end;

procedure TFrmCadSetor.BbtRemoverClick(Sender: TObject);
begin
  if DmEmpresa.TbSetor.RecordCount = 0 then
     ShowMessage('Tabela vazia!')
  else
    if MessageDLG('Tem certeza que deseja remover o setor: '+#13+
                  DmEmpresa.TbSetor.FieldByName('SetNome').AsString +' ?',
                  mtConfirmation, [mbYes, mbNo], 0) = mrYes then
       DmEmpresa.TbSetor.Delete;
end;

procedure TFrmCadSetor.BbtCancelarClick(Sender: TObject);
begin
  DmEmpresa.TbSetor.Cancel;
  TrataBotoes;
end;

procedure TFrmCadSetor.BbtConfirmarClick(Sender: TObject);
begin
  DmEmpresa.TbSetor.Post;
  TrataBotoes;
end;

end.
```

Antes de prosseguir com o próximo formulário, é necessário inserir a Biblioteca **DB** na cláusula **uses** da **interface**. Consulte a dica 'Como saber' para mais informações.

FORMULÁRIO DE CADASTRO DE FUNCIONÁRIO

FrmCadFuncionario

Vamos iniciar a construção do formulário de Cadastro de Funcionário.
No menu **File** escolha o comando **New Form**.
Há um ícone para o mesmo comando na SpeedBar.

- ✓ Grave este novo formulário com o nome de: *UFrmCadFuncionario*
- ✓ A propriedade Name do form deverá ser *FrmCadFuncionario*

Chame o DataModule. Crie os campos persistentes (TFields) para a tabela Funcionário. Arraste-os para dentro do formulário. Confirme o diálogo (YES) sobre usar o DataModule e posicione os objetos em uma seqüência à sua escolha. Uma sugestão pode ser vista a seguir:

FrmCadFuncionario
Código
Código do setor
DBText1
Nome
Data Nasc.
Salário
Endereço
Sexo
Bairro
Cidade
Estado
CEP
Fone
E-Mail
Sair
Inserir
Editar
Remover
Cancelar
Confirmar

Associe o componente o componente DBText1 à tabela através da propriedade *DataSource* apontando para a tabela Funcinario e *DataField* apontando para o campo FunCodigo.

Associe o componente DBNavigator à tabela Funcionário através da propriedade *DataSource*.

Implemente os *handlers* de cada botão conforme o raciocínio do formulário anterior.

Vamos definir outros componentes e características que no formulário anterior não existiam.
Por exemplo, o campo *Sexo* foi definido na tabela como um String de um caracter. Podemos evitar que o usuário entre com uma string diferente de M ou F por exemplo, este raciocínio pode ser feito via código ou via componente.

Vamos optar pela segunda opção por uma série de vantagens.

Insira um componente **DBRadioGroup** no formulário e não se preocupe com o alinhamento (por enquanto).

Modifique as propriedades do componente:

```
object DBRadioGroup1: TDBRadioGroup
  Caption = 'Sexo'
  Columns = 2
  DataSource = DmEmpresa.DsFuncionario
  DataField = 'FunSexo'
  Items.Strings =
    Fem
    Mas
  Values.Strings =
    F
    M
end
```

Columns	Define em quantas colunas os dados da propriedade *items* serão ordenados.
Items	Define os itens a serem exibidos no componente.
Values	Define os valores a serem armazenados na tabela.

Pode-se excluir o DBEdit responsável pelo campo Sexo e alinhar o DBRadioGroup em seu lugar.

Um outro componente que pode exibir um conjunto de dados pré-definidos é o **DBComboBox**. Insira o componente no formulário e modifique suas propriedades:

```
object DBComboBox1: TDBComboBox
  DataSource = DmEmpresa.DsFuncionario
  DataField = 'FunEstado'
  Items.Strings =
     AC
     AL
     AP
     AM
     BA
     CE
     DF
     ES
     GO
     MA
     MT
     MS
     MG
     PA
     PB
     PR
     PE
     PI
```

```
        RJ
        RN
        RS
        RO
        RR
        SC
        SP
        SE
        TO
end
```

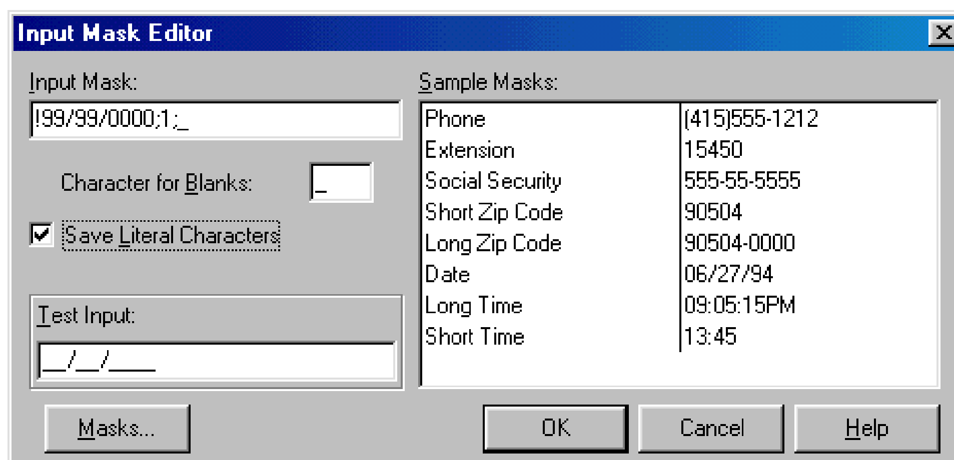
Exclua o DBEdit responsável pelo Estado e alinhe o DBComboBox em seu lugar.

MÁSCARA PARA ENTRADA DE DADOS

Podemos definir ***máscaras*** para a entrada dos dados nos campos *Data de Nascimento* e *CEP*. Porém, esta definição não é feita diretamente no componente *data-aware* e sim no objeto **TField.**

- Selecione o DataModule.
- Duplo clique no componente TbFuncionario.
- Selecione o campo persistente FunDataNasc
- Na object inspector selecione a propriedade EditMask.
- Configure a máscara desejada.

Exemplo:



Há uma série de exemplos pré-definidos, escolha o tipo Date e observe que foram acrescentados dois zeros (pelo desenvolvedor) para exibir o ano da data.

É necessário configurar no Windows: Painel de Controle | Configurações Regionais | Data/Hora para exibir os quatro dígitos no ano.

Consulte o botão Help para verificar a simbologia de cada caracter na máscara.

A máscara para o campo CEP pode ser a seguinte: `99\.999\-999;0;_`
O CEP pode ficar **sem** a opção de salvar os caracteres literais. A data **deve ter** esta opção.

Save Literal Characters permite salvar ou não, (neste caso do CEP) os separadores '`.`' e '`-`' na tabela.

INTEGRAÇÃO DE DADOS ENTRE AS TABELAS

Vamos acrescentar alguma funcionalidade para que o usuário não precise de saber o código do setor, ele poderá selecionar uma caixa de listagem através do nome do setor por exemplo.

Acrescente um **DBLookupComboBox** ao Form e faça as modificações necessárias, a sugestão visual é a seguinte:

Configure o DBLookupComboBox com as propriedades a seguir:

```
object DBLookupComboBox1: TDBLookupComboBox
  ListSource = DmEmpresa.DsSetor
  ListField = 'SetNome'
  KeyField = 'SetCodigo'

  DataSource = DmEmpresa.DsFuncionario
  DataField = 'SetCodigo'
end
```

Onde:

ListSource	Permite 'conectar' o controle à fonte **origem** dos dados.
ListField	Permite especificar um campo da tabela referenciada em ListSource.
KeyField	Permite especificar o campo chave da tabela origem dos dados.
DataSource	Permite 'conectar' o controle à fonte **destino** dos dados.

DataField	Permite especificar o campo chave estrangeira da tabela destino.

Uma sugestão de implementação para o código fonte do formulário para Cadastro de Funcionário é o seguinte:

```
unit UFrmCadFuncionario;

{A interface não foi impressa}

implementation

uses UDMEmpresa;

{$R *.DFM}

procedure TFrmCadFuncionario.FormCloseQuery(Sender: TObject;
  var CanClose: Boolean);
begin
  if DmEmpresa.TbFuncionario.State in [dsEdit, dsInsert] then
    if MessageDLG('Existem dados pendentes,'+#13+'deseja gravá-los?',
                  mtConfirmation, [mbYes,mbNo], 0) = mrYes then
       CanClose := False
    else
      begin
        DmEmpresa.TbFuncionario.Cancel;
        TrataBotoes;
        CanClose := True;
      end;
end;

{Para criar o procedimento TrataBotoes, digite sua declaração
(sem TFrmCadFuncionario) na cláusula private e utilize CTRL+SHIFT+C
Para todos os demais, selecione o objeto e utilize a object inspector}

procedure TFrmCadFuncionario.TrataBotoes;
begin
  BbtInserir.Enabled := not BbtInserir.Enabled;
  BbtEditar.Enabled := not BbtEditar.Enabled;
  BbtRemover.Enabled := not BbtRemover.Enabled;
  BbtCancelar.Enabled := not BbtCancelar.Enabled;
  BbtConfirmar.Enabled := not BbtConfirmar.Enabled;
  BbtSair.Enabled := not BbtSair.Enabled;
  DbNavigator1.Enabled := not DbNavigator1.Enabled;
end;

procedure TFrmCadFuncionario.BbtInserirClick(Sender: TObject);
var ProxNum: Integer;
begin
  TrataBotoes;
  DmEmpresa.TbFuncionario.Last;
  ProxNum := DmEmpresa.TbFuncionario.FieldByName('FunCodigo').AsInteger + 1;
  DmEmpresa.TbFuncionario.Append;
  DmEmpresa.TbFuncionario.FieldByName('FunCodigo').AsInteger := ProxNum;
  DbEdit3.SetFocus;
end;
```

```
procedure TFrmCadFuncionario.BbtEditarClick(Sender: TObject);
begin
  DmEmpresa.TbFuncionario.Edit;
  TrataBotoes;
end;

procedure TFrmCadFuncionario.BbtRemoverClick(Sender: TObject);
begin
  if DmEmpresa.TbFuncionario.RecordCount = 0 then
     ShowMessage('Tabela vazia!')
  else
    if MessageDLG('Tem certeza que deseja remover o funcionário: '+#13+
               DmEmpresa.TbFuncionario.FieldByName('FunNome').AsString +' ?',
               mtConfirmation, [mbYes, mbNo], 0) = mrYes then
       DmEmpresa.TbFuncionario.Delete;
end;

procedure TFrmCadFuncionario.BbtCancelarClick(Sender: TObject);
begin
  DmEmpresa.TbFuncionario.Cancel;
  TrataBotoes;
end;

procedure TFrmCadFuncionario.BbtConfirmarClick(Sender: TObject);
begin
  DmEmpresa.TbFuncionario.Post;
  TrataBotoes;
end;

procedure TFrmCadFuncionario.BbtSairClick(Sender: TObject);
begin
  FrmCadFuncionario.Close;
end;

end.
```

ATRIBUINDO VALORES ATRAVÉS DE ONNEWRECORD

Ainda no Cadastro de Funcionários, observe que o componente *DBRadioGroup* não tem a propriedade *ItemIndex* para iniciar o cadastro com uma das opções (Fem ou Mas) selecionada.

Isso ocorre porque os componentes *data-aware* refletem o conteúdo da tabela.

Neste caso, para definir um valor padrão, vá até o **Data Module** *selecione a tabela* (Funcionario, neste exemplo) e na **Object Inspector** na guia **Events** selecione o evento **OnNewRecord** ou seja, a cada novo registro, vamos atribuir um valor direto para tabela, se o componente reflete o conteúdo da tabela, ele vai já indicar o valor ‘default’.

```
procedure TDmEmpresa.TbFuncionarioNewRecord(DataSet: TDataSet);
begin
  TbFuncionario.FieldByName('FunSexo').AsString := 'F';
end;
```

CONSISTINDO DADOS ATRAVÉS DE ONVALIDADE

Um exemplo de consistência pode ser utilizado no campo **FunDataNasc** com o seguinte raciocínio: se a data digitada pelo usuário for maior do que a data atual, subtende-se que o funcionário cadastrado ainda não nasceu...

Para criar esta regra de validação vá até o **Data Module** e clique duas vezes na tabela (Funcionario, neste exemplo) chamando os TFields. Selecione o TField **FunDataNasc** e na Object Inspector selecione o evento **OnValidade**. Crie o seguinte código para o evento:

```
procedure TDmEmpresa.TbFuncionarioFunDataNascValidate(Sender: TField);
begin
  if TbFuncionarioFunDataNasc.Value > Date then
    raise Exception.Create('Data menor do que a atual!');
end;
```

Neste exemplo a exceção está sendo ativada pelo comando *raise*, poderíamos tratar esta situação com o comando *Abort* que gera uma exceção 'silenciosa', ou seja, uma exceção sem mensagem.

O evento *OnValidade* ocorre quando um valor é atribuído ao campo. Por exemplo uma atribuição direta:

```
  TbFuncionarioFunNome.Value := 'Edileumar';
```

dispara o evento *OnValidade* para o TField correspondente ao campo FunNome da tabela Funcionario.
Quando utilizamos controles *data-aware* esse evento ocorre quando o *foco sai do componente* e o usuário fez alguma alteração enquanto o foco estava no componente.
O evento também pode ocorrer durante o processamento do método Post, *se* existir alguma alteração pendente a ser feita pelos componentes *data-aware*.

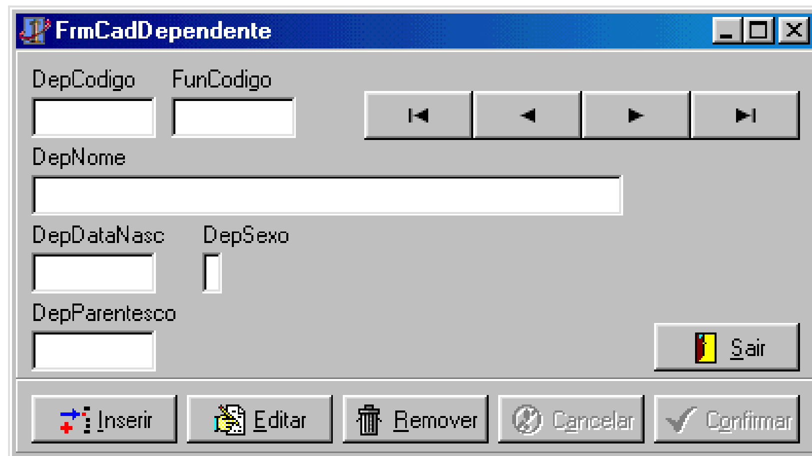
FORMULÁRIO DE CADASTRO DE DEPENDENTES

Vamos criar outro formulário com a finalidade de cadastrar os dependentes do funcionário. Utilizaremos o mesmo raciocínio visto até aqui *embora este processo possa ser bem diferente* do que vamos fazer. O objetivo maior é fixarmos os conceitos vistos anteriormente.

FrmCadDependente

Clique no menu **File | New Form...**

- ✓ Grave a unit com o nome: *UFrmCadDependente*
- ✓ A propriedade Name do form será: *FrmCadDependente*



Configure o componente DBNavigator:

```
object DBNavigator1: TDBNavigator
  DataSource = DmEmpresa.DsDependente
  VisibleButtons = [nbFirst, nbPrior, nbNext, nbLast]
end
```

Crie uma máscara para o campo Data de Nascimento através do objeto TField correspondente.

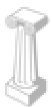
Insira dois componentes **DBRadioGroup**, um para o Sexo e outro para o Parentesco.

A configuração do DBRadioGroup para o campo Sexo pode ser a seguinte:

```
object DBRadioGroup1: TDBRadioGroup
  Caption = 'Se&xo'
  Columns = 2
  DataSource = DmEmpresa.DsDependente
  DataField = 'DepSexo'
  Items.Strings =
    Feminino
    Masculino
  Values.Strings =
    F
    M
end
```

A configuração do DBRadioGroup para o campo Parentesco pode ser a seguinte:

```
object DBRadioGroup2: TDBRadioGroup
  Caption = '&Parestesco'
  Columns = 4
  DataSource = DmEmpresa.DsDependente
  DataField = 'DepParentesco'
  Items.Strings =
    Cônjuge
    Filho(a)
    Pai
    Mãe
  Values.Strings =
    1
    2
    3
    4
end
```



Lembre-se que na definição da tabela, o campo parentesco foi definido como Integer... Porquê? Armazenar um número ocupa menos espaço do que uma string; e o componente pode *exibir* 'uma' string e *armazenar* um outro tipo de dados. Vamos entender mais adiante que o inverso também é verdadeiro, por exemplo: Quando for necessário emitir um relatório vamos ler um número e imprimir uma string.

Insira **dois** DBText e altere suas propriedades *DataSource* e *DataField.*

- ✓ Um deles deve exibir o *código do dependente* que será gerado pelo algoritmo.
- ✓ O outro (DBText2) deve exibir o *nome do funcionário* da tabela funcionário.
- ✓ O SpeedButton será utilizado adiante, ele deve ficar com a propriedade Enabled falsa e deverá ser incluído no procedimento tratabotoes.

Uma sugestão visual pode ser vista a seguir:

O código para implementar as funções dos BitBtns pode ser visto a seguir:

```
unit UFrmCadDependente;
{A interface não foi impressa}
implementation

uses UDMEmpresa;

{$R *.DFM}

procedure TFrmCadDependente.BbtInserirClick(Sender: TObject);
var ProxNum: Integer;
begin
  TrataBotoes;
  DmEmpresa.TbDependente.Last;
  ProxNum := DmEmpresa.TbDependente.FieldByName('DepCodigo').AsInteger + 1;
  DmEmpresa.TbDependente.Append;
  DmEmpresa.TbDependente.FieldByName('DepCodigo').AsInteger := ProxNum;
  DbEdit2.SetFocus;
end;

{Para criar o procedimento TrataBotoes, digite sua declaração
(sem TFrmCadFuncionario) na cláusula private e utilize CTRL+SHIFT+C
Para todos os demais, selecione o objeto e utilize a object inspector}

procedure TFrmCadDependente.TrataBotoes;
begin
  BbtInserir.Enabled := not BbtInserir.Enabled;
  BbtEditar.Enabled := not BbtEditar.Enabled;
  BbtRemover.Enabled := not BbtRemover.Enabled;
  BbtCancelar.Enabled := not BbtCancelar.Enabled;
  BbtConfirmar.Enabled := not BbtConfirmar.Enabled;
  BbtSair.Enabled := not BbtSair.Enabled;
  DbNavigator1.Enabled := not DbNavigator1.Enabled;
  SpeedButton1.Enabled := not SpeedButton1.Enabled;
end;

procedure TFrmCadDependente.BbtEditarClick(Sender: TObject);
begin
  DmEmpresa.TbDependente.Edit;
  TrataBotoes;
end;

procedure TFrmCadDependente.BbtRemoverClick(Sender: TObject);
begin
  if DmEmpresa.TbDependente.RecordCount = 0 then
     ShowMessage('Tabela vazia!')
  else
    if MessageDLG('Tem certeza que deseja remover o funcionário: '+#13+
                DmEmpresa.TbDependente.FieldByName('DepNome').AsString +' ?',
                mtConfirmation, [mbYes, mbNo], 0) = mrYes then
      DmEmpresa.TbDependente.Delete;
end;
```

```
procedure TFrmCadDependente.BbtCancelarClick(Sender: TObject);
begin
  DmEmpresa.TbDependente.Cancel;
  TrataBotoes;
end;

procedure TFrmCadDependente.BbtConfirmarClick(Sender: TObject);
begin
  DmEmpresa.TbDependente.Post;
  TrataBotoes;
end;

procedure TFrmCadDependente.BbtSairClick(Sender: TObject);
begin
  FrmCadDependente.Close;
end;

procedure TFrmCadDependente.FormCloseQuery(Sender: TObject;
  var CanClose: Boolean);
begin
  if DmEmpresa.TbDependente.State in [dsEdit, dsInsert] then
    if MessageDLG('Existem dados pendentes,'+#13+'deseja gravá-los?',
                  mtConfirmation, [mbYes,mbNo], 0) = mrYes then
       CanClose := False
    else
      begin
        DmEmpresa.TbDependente.Cancel;
        TrataBotoes;
        CanClose := True;
      end;
end;

procedure TFrmCadDependente.DBEdit2Exit(Sender: TObject);
begin
  if FrmCadDependente.DBEdit2.Text <> '' then
  if not DmEmpresa.TbFuncionario.FindKey([FrmCadDependente.DBEdit2.Text])
then
    begin
    ShowMessage('Código inválido');
    FrmCadDependente.DBEdit2.SetFocus;
    end;
end;

end.
```

ATRIBUINDO VALORES DEFAULT (ONNEWRECORD)

Para inicializar os campos Sexo e Parentesco com valores pré-definidos devemos chamar o *Data Module* e selecionar a tabela (Dependente), na Object Inspector selecione o evento **OnNewRecord** e defina o código necessário.

```
procedure TDmEmpresa.TbDependenteNewRecord(DataSet: TDataSet);
begin
  //Há diferença entre 'f' e 'F'
  TbDependente.FieldByName('DepSexo').AsString := 'F';
  TbDependente.FieldByName('DepParentesco').AsInteger := 2;
end;
```

Vamos definir um formulário para *localizar* o funcionário caso o operador do micro não saiba (e provavelmente não saberá) o código do funcionário.

MÉTODOS DE PESQUISA

Utilizaremos métodos de pesquisa para encontrar os dados da tabela Funcionário. Basicamente os métodos **FindKey** e **FindNearest**. Estes métodos fazem procura em campos indexados, ou seja, o campo a ser pesquisado deve ter um índice deve estar ordenado por ele. A diferença básica é que **FindKey** realiza uma pesquisa exata, enquanto **FindNearest** realiza uma pesquisa parcial, **FindKey** é uma função que retorna um valor boolean e **FindNearest** é um procedimento.

Para modificar a estrutura de índices da tabela, utiliza-se a propriedade **IndexName** da tabela desejada. Exemplo:

```
//utiliza o índice secundário
  DmEmpresa.TbFuncionario.IndexName := 'IndSetNome';
  DmEmpresa.TbFuncionario.FindNearest([Edit2.Text]);

//utiliza o índice primário
  DmEmpresa.TbFuncionario.IndexName := '';
  DmEmpresa.TbFuncionario.FindKey([Edit1.Text]);
```

FORMULÁRIO DE CONSULTA DE FUNCIONÁRIOS

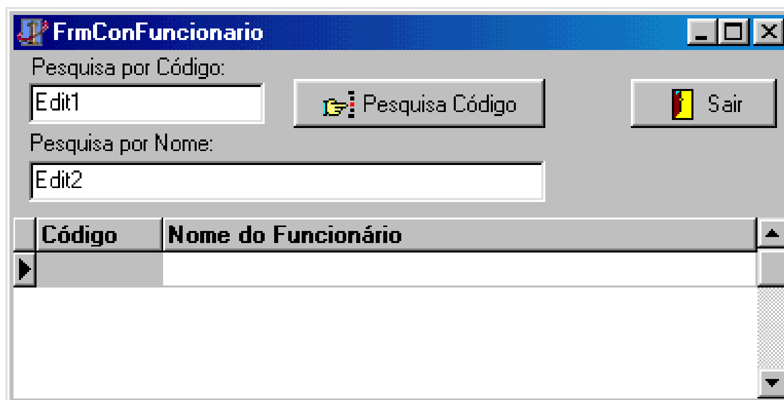
FrmConFuncionario

Clique em **File | New Form**

- ✓ Grave o nova unit como *UFrmConFuncionario*
- ✓ A propriedade Name do form será: *FrmConFuncionario*

- ✓ Insira os componentes necessários, altere suas propriedades.
- ✓ Faça a referência ao Data Module manualmente através do comando **File | Use Unit...**

Configure o **DBGrid** para acessar a tabela Funcionario e exibir apenas o código e o nome do funcionário, mudanças visuais podem ser feitas nas propriedades TileFont e Options.



Um exemplo de *algumas* propriedades que podem ser alteradas:

```
object DBGrid1: TDBGrid
  Cursor = crHandPoint
  TabStop = False
  Align = alBottom
  BorderStyle = bsNone
  DataSource = DmEmpresa.DsFuncionario
  ReadOnly = True
  TitleFont.Style = [fsBold]
end
```

✓ A propriedade Name dos BitBtn's serão o prefixo Bbt + o caption (sem espaço e sem acento).

A implementação do código do formulário FrmConFuncionario pode ser vista logo a seguir:

```
unit UFrmConFuncionario;
{A interface não foi impressa}
implementation

uses UDMEmpresa;

{$R *.DFM}

procedure TFrmConFuncionario.BbtPesquisaCodigoClick(Sender: TObject);
begin
  if Edit1.Text <> '' then
    DmEmpresa.TbFuncionario.FindKey([Edit1.Text]);
end;
```

```
procedure TFrmConFuncionario.Edit1Enter(Sender: TObject);
begin
  Edit2.Clear;
  DmEmpresa.TbFuncionario.IndexName := '';
end;

procedure TFrmConFuncionario.Edit2Enter(Sender: TObject);
begin
  Edit1.Clear;
  DmEmpresa.TbFuncionario.IndexName := 'IndFunNome';
end;

procedure TFrmConFuncionario.Edit2Change(Sender: TObject);
begin
  if Edit2.Text <> '' then
    DmEmpresa.TbFuncionario.FindNearest([Edit2.Text]);
end;

procedure TFrmConFuncionario.Edit1KeyPress(Sender: TObject; var Key: Char);
begin
  if Key = #13 then
    BbtPesquisaCodigo.Click
    else
    if ( (Key in ['0'..'9'] = FALSE) and (Word(Key) <> VK_BACK) ) then
      Key := #0;
end;

procedure TFrmConFuncionario.BbtSairClick(Sender: TObject);
begin
  FrmConFuncionario.Close;
end;

procedure TFrmConFuncionario.FormClose(Sender: TObject;
  var Action: TCloseAction);
begin
  DmEmpresa.TbFuncionario.IndexName := '';
  DmEmpresa.TbDependente.FieldByName('FunCodigo').AsInteger :=
  DmEmpresa.TbFuncionario.FieldByName('FunCodigo').AsInteger;
  FrmCadDependente.DBEdit2.SetFocus;
end;

end.
```

DEFININDO CAMPOS REQUERIDOS E EXCEÇÃO LOCAL

Podemos definir alguns campos que devem, obrigatoriamente, serem preenchidos pelo usuário, por exemplo: No formulário de *Cadastro de Dependentes* os campos *FunCodigo* e *DepNome* são muito importantes e poderiam, na sua ausência tornar o banco inconsistente. Vamos utilizar um exemplo com a tabela Dependente.

Para definir campos requeridos e evitar inconsistência de campos em branco:

- Chame o Data Module.
- Escolha a tabela desejada (TbDependente) e clique duas vezes para chamar os TFields.
- Escolha o campo TField desejado (FunCodigo, neste exemplo) e na *Object Inspector* defina **Required** como **True**.

Defina o mesmo recurso para o nome do Dependente.

Neste momento, se o usuário tentar gravar o registro *sem* digitar o código do funcionario ou o nome do dependente será levantada uma exceção da classe EDatabaseError que deverá ser tratada pelos conceitos vistos no curso.

O evento *OnClick* do botão confirmar no cadastro de dependentes pode ser (atualizado) implementado como o exemplo abaixo:

```
procedure TFrmCadDependente.BbtConfirmarClick(Sender: TObject);
begin
  try
    DmEmpresa.TbDependente.Post;
    TrataBotoes;
  except
     on E: EDataBaseError do
     Showmessage('Campo obrigatório sem preencher! ' + #13+#13 + E.Message);
  end;
end;
```

O objeto **E** da classe *Exception* recebe o valor do objeto de exceção. Este objeto não precisa ser declarado e permite comparar qual é o erro atual, para tomada de decisões.

EXCLUSÃO COM CONSISTÊNCIA

Antes de excluir em cascata, vamos exemplificar uma situação em que: havendo a chave primária como chave estrangeira na outra tabela, não se pode excluir o registro.

Neste projeto, o exemplo é o seguinte: Se houver uma ocorrência do código do setor (chave primária) da tabela setor na tabela funcionário (chave estrangeira) **não** vamos deletar o setor.

A função **locate** vai localizar na tabela *funcionário* no campo SetCodigo o valor do campo SetCodigo da tabela *setor* passado como parâmetro. Caso encontre, a função retorna um valor booleano verdadeiro.

A alteração no código do botão remover do form Cadastro de Setor pode ser visto a seguir:

```
procedure TFrmCadSetor.BbtRemoverClick(Sender: TObject);
begin
  if DmEmpresa.TbSetor.RecordCount = 0 then
     ShowMessage('Tabela vazia!')
  else if DmEmpresa.TbFuncionario.Locate('SetCodigo',
          DmEmpresa.TbSetor.FieldByName('SetCodigo').AsInteger,[]) then
          ShowMessage('Este setor possui funcionários,'+#13+
                      'favor realocá-los antes de deletar.')
    else if MessageDLG('Tem certeza que deseja remover o setor: '+#13+
                  DmEmpresa.TbSetor.FieldByName('SetNome').AsString +' ?',
                  mtConfirmation, [mbYes, mbNo], 0) = mrYes then
            DmEmpresa.TbSetor.Delete;
end;
```

A função locate é declarada internamente da seguinte forma:

```
function Locate(const KeyFields: string; const KeyValues: Variant;
                Options: TLocateOptions): Boolean; virtual;
```

Onde:

KeyFields	Permite definir o(s) campo(s) a serem pesquisados.
KeyValues	Permite definir o(s) valore(s) a serem pesquisados.
Options	Permite definir se os valores a serem pesquisados serão parciais e/ou diferenciados de maiúscula e minúscula. [loCaseInsensitive, loPartialKey]

```
  if DmEmpresa.TbFuncionario.Locate('SetCodigo',
               DmEmpresa.TbSetor.FieldByName('SetCodigo').AsInteger,[]) then
```

Neste exemplo acima, estamos pesquisando na tabela TbFuncionario o campo SetCodigo o valor que está na tabela TbSetor no campo SetCodigo. A pesquisa é exata.

EXCLUSÃO EM CASCATA

Pense nesta situação: Ao excluir um funcionario, o código correspondente na tabela dependente não será mais válido, ou seja, o dependente terá um código (FunCodigo) que não existe mais. Nosso objetivo então será deletar o(s) dependente(s) do funcionário deletado.

```
procedure TDmEmpresa.TbFuncionarioBeforeDelete(DataSet: TDataSet);
begin
  if TbDependente.Locate('FunCodigo',
                TbFuncionario.FieldByName('FunCodigo').AsInteger,[]) then
    begin
//desabilitar os controles data aware
      TbDependente.DisableControls;

{Aplicar e ligar o filtro para que a tabela considere apenas os registros que
tem a chave estrangeira com o código do funcionario a ser deletado}

      TbDependente.Filter :=
      'FunCodigo='+TbFuncionario.FieldByName('FunCodigo').AsString;
      TbDependente.Filtered := True;

//Enquanto nao e o fim da tabela, delete os registros filtrados
      while not TbDependente.Eof do
        TbDependente.Delete;

//Retirar o filtro para que a tabela possa considerar os outros registros
      TbDependente.Filter := '';
      TbDependente.Filtered := False;
    end;
end;

procedure TDmEmpresa.TbFuncionarioAfterDelete(DataSet: TDataSet);
begin
//Habilitar os controles data aware após a deleçao do funcionario
  TbDependente.EnableControls;
end;
```

As propriedades **Filter** e **Filtered** funcionam da seguinte forma:

Deve-se atribuir um valor (critério) à propriedade filter.

```
  TbDependente.Filter := ‘DepSexo=F’;
```

Depois ligar o filtro para que o critério tenha efeito.

```
  TbDependente.Filtered := True;
```

A partir deste momento a tabela (TbDependente) só vai manipular os registros que atentem ao critério, ou seja, todas as pessoas do sexo feminino. Caso o filtro *não* seja desligado, a tabela *não* vai considerar os demais registros.

Os métodos **EnableControls** e **DisableControls** servem para habilitar e desabilitar os controles *data aware*.

Isto é utilizado em nosso exemplo porque há o evento *OnChange* no DBEdit2 do Cadastro de Dependentes, ou seja, quando os registros da tabela dependentes forem deletados, o código do funcionário a ser deletado não existirá na tabela dependentes, o evento *OnChange* do *data aware* DBEdit2 será disparado e mudará o registro da tabela funcionário de posição. Quando o código for deletar o funcionário, deletará o funcionário errado.

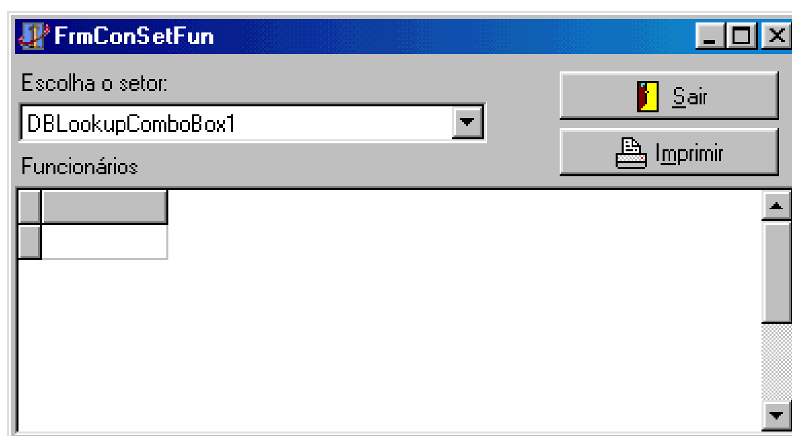
Para evitar isso, desabilitamos os controles no momento da deleção dos dependentes e habilitamos no momento *após* a deleção do funcionário.

UM POUCO MAIS SOBRE CONSULTAS

Vamos criar uma consulta para saber quais funcionários trabalham em determinado setor e ter a opção de imprimir seu resultado.

FrmConSetFun

Crie um novo formulário e grave-o com o nome de *UFrmConSetFun*
Insira os componentes necessários:



Para acessar as tabelas necessárias, use o comando **File | Use Unit** e selecione o DataModule.

O componente DBLookupComboBox será usado *apenas para listar* o conteúdo de um campo de determinada tabela. Neste exemplo as propriedades alteradas serão:

```
object DBLookupComboBox1: TDBLookupComboBox
  ListSource = DmEmpresa.DsSetor
  ListField = 'SetNome'
  KeyField = 'SetCodigo'
end
```

Ao contrário do que poderíamos pensar, as propriedades DataSource e DataField neste componente **não** serão alteradas. Elas representam uma tabela **destino** para o envio de dados da tabela **origem**, a tabela origem foi configurada através das 3 propriedades acima.

Então os nomes dos setores serão listados no DBLookupComboBox e os respectivos funcionários serão listados no DBGrid, esta é a idéia básica.

Configure o DBGrid nas suas propriedades:

```
object DBGrid1: TDBGrid
  Align = alBottom
  DataSource = DmEmpresa.DsFuncionario
end
```

Na propriedade `Columns` defina apenas duas colunas: FunCodigo e FunNome.

O procedimento para imprimir será visto após a construção dos relatórios.
Os demais procedimentos serão os seguintes:

```
unit UFrmConSetFun;
{A interface não foi impressa}
implementation

uses UDMEmpresa, UFrmRelSetFun;

{$R *.DFM}

procedure TFrmConSetFun.DBLookupComboBox1Click(Sender: TObject);
begin
  with DmEmpresa do
  begin
    TbFuncionario.Filter := 'SetCodigo='+
    TbSetor.FieldByName('SetCodigo').AsString;
    TbFuncionario.Filtered := True;
  end;
  DBGrid1.DataSource := DmEmpresa.DsFuncionario;
end;

procedure TFrmConSetFun.FormClose(Sender: TObject;
  var Action: TCloseAction);
begin
  with DmEmpresa do
  begin
    TbFuncionario.Filter := '';
    TbFuncionario.Filtered := False;
  end;
    DBGrid1.DataSource := nil;
end;

procedure TFrmConSetFun.BbtSairClick(Sender: TObject);
begin
  FrmConSetFun.Close;
end;

end.
```

A propriedade DataSource do DBGrid foi preenchida em tempo de projeto para auxiliar nas configurações, porém deverá ficar vazia após a constução e testes finais. Ela será preenchida em tempo de execução para que não liste os funcionários assim que o form é exibido.

RELATÓRIOS

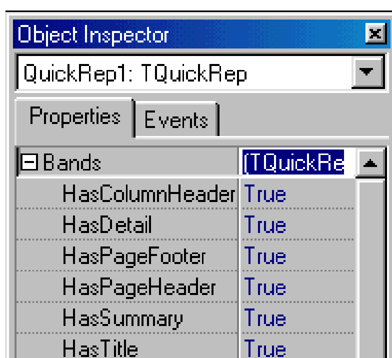
Vamos abordar inicialmente a construção de um formulário simples que exibirá os dados de *uma* tabela apenas.
Da mesma forma que para manipular a base da dados utilizamos componentes *data aware* da paleta Data Controls. Para manipular relatórios vamos utilizar componentes da paleta Qreport[13].
Mais informações (técnicas) sobre o QuickReport podem ser encontradas no site da QuSoft (www.qusoft.com).

FrmRelFunc

Crie um formulário novo e salve-o com o nome: *UFrmRelFunc*

Insira um componente QuickRep e não se preocupe com o alinhamento do componente no form. Uma propriedade muito importante do QuickRep é: *Bands*. Pois esta propriedade define as áreas úteis (bandas) para a utilização do relatório. Estas 'bandas' serão objetos do tipo TQRBand que recebem outros objetos para exibição dos dados.

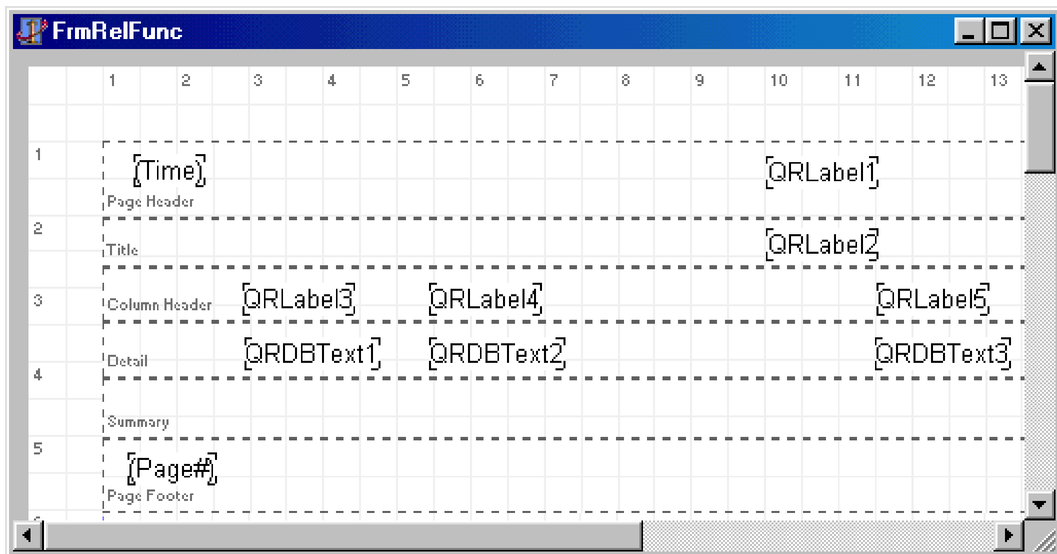
Defina as seguintes bandas para este exemplo:



HasColumnHeader	Define a área para os dados que serão repetidos em todas as páginas como rótulo das colunas.
HasDetail	Define a área para os dados (de uma tabela) a serem exibidos.
HasPageFooter	Define a área rodapé do relatório.
HasPageHeader	Define a área cabeçalho do relatório.
HasSummary	Define a área de sumário, exibida na última página do relatório.
HasTitle	Define a área para o título do formulário exibido apenas na primeira página.

[13] Quick Report.

Insira os demais componentes em suas respectivas bandas: QRSysData, QRLabel, QRDBText.



Para os componentes QRSysData altere a propriedade *Data* para o valor desejado.

Para os componentes QRLabel altere as propriedades *Alignment, AlignToBand e Font*.

Para os componentes QRDBText não altere nenhuma propriedade ainda.

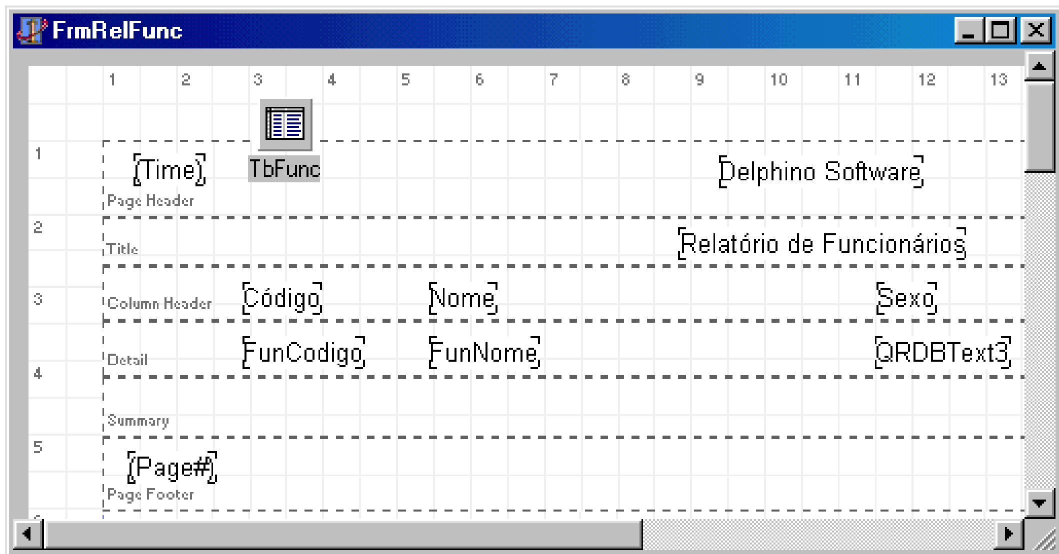
Embora seja perfeitamente possível utilizar o DataModule para ser a fonte de componentes de dados nos relatórios vamos utilizar uma metodologia diferente. Os componentes de acesso serão instanciados nos próprios relatórios.

Insira um componente Table. Configure suas propriedades:

```
object TbFunc: TTable
  DatabaseName = 'Empresa'
  TableName = 'Funcionario.db'
  Active = True
  Name = TbFunc
end
```

Configure os demais componentes:

- ✓ Para os componentes QRLabel altere a propriedade *Caption*.
- ✓ Para os componentes QRDBText altere as propriedades **DataSet** e **DataField**.



O componente QRDBText3 terá um tratamento diferenciado como veremos adiante.

Selecione o componente QuickRep1 e configure sua propriedade **DataSet**.

Com o botão direito do mouse no componente QuickRep e escolha **Preview**.
Um exemplo pode ser visto ainda em tempo de projeto.

Para fazer a 'chamada' do relatório no menu principal, deve-se chamar o método Preview do QuickRep da seguinte forma:

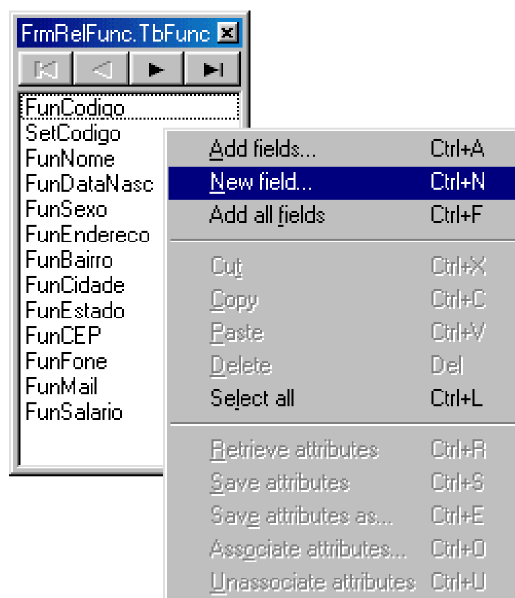
```
FrmRelFunc.QuickRep1.Preview;
```

CAMPOS CALCULADOS

Você deve lembrar-se que o campo sexo é definido como um campo AlfaNumérico de uma posição ‘F’ ou ‘M’. Mas queremos que saia a string ‘Feminino’ ou ‘Masculino’ no relatório, vamos utilizar um campo calculado para realizar esta tarefa.

Campos calculados servem para ‘calcular’ alguma rotina em *tempo de execução*. Este cálculo não é obrigatoriamente uma expressão de cálculo matemático, pode ser uma condição de atribuição após um teste lógico, por exemplo.

No relatório, selecione o *DataSet - Table*, dê um duplo clique. No editor de campos persistentes utilize o botão direito do mouse e clique em **Add All Fields** para adicionarmos todos os campos. Com o botão direito novamente, clique dentro do editor e escolha **New Field...**



Uma nova janela será exibida. Preecha os items ***Name*** e ***Type***, confirme a janela.

Um novo campo chamado SexoExtenso deverá surgir no editor.

O campo calculado funciona apenas em ***tempo de execução*** realizando o código que deve ficar no evento **OnCalcFields** da tabela desejada.
Selecione a tabela TbFunc e escreva o código no evento (*OnCalcFields*) indicado:

```
procedure TFrmRelFunc.TbFuncCalcFields(DataSet: TDataSet);
begin
{ TbFuncFunSexo e TbFuncSexoExtenso são os Name's dos campos persistentes.
Poderia ser usado a propriedade FieldByName ou Fields para acessar os campos.}

  if TbFuncFunSexo.Value = 'F' then
    TbFuncSexoExtenso.AsString := 'Feminino'
    else
    TbFuncSexoExtenso.AsString := 'Masculino';
end;
```

Selecione o QRDBText destinado ao campo sexo e altere suas propriedades **DataSet** e **DataField.** Lembrando que o campo DataField será o campo calculado *SexoExtenso.*

Para a banda *Summary* insira um componente QRLabel e um componente QRSysData. Altere o Caption do QRLabel para 'Total de funcionários'. E altere a propriedade data do QRSysData para qrsDetailCount.

Em cada banda pode-se alterar as fontes e a propriedade *Frame* para causar um efeito de divisão importante.

A largura das bandas define a distância entre os registros.

RELATÓRIO MESTRE-DETALHE

Um relatório composto de mais de uma tabela onde há um relacionamento explícito através de uma chave primária e uma estrangeira é denominado Mestre-Detalhe. A tabela que fornece a chave primária é a tabela Mestre a tabela que trabalha com a chave estrangeira é a tabela Detalhe.

Por exemplo vamos criar um relatório entre *Setor* e *Funcionário*, para cada setor (chave primária) pode haver: zero, um ou vários funcionários (a tabela funcionário têm o campo SetCodigo como chave estrangeira). Será exibido da tabela Mestre: o código e o nome do Setor, da tabela Detalhe: o nome do funcionário.

FrmRelSetFun

Crie um novo formulário e grave-o como *UfrmRelSetFun*.
Insira os componentes:
1 – QuickRep.
1 – DataSource
2 – TTables

Configure os componentes (da tabela mestre) como a seguir:

```
object DsSetor: TDataSource
  AutoEdit = False
  DataSet = TbSetor
  Name = DsSetor
end


object TbSetor: TTable
  DatabaseName = 'Empresa'
  TableName = 'Setor.DB'
  Active = True
  Name = TbSetor
end
```

Antes de definir o relacionamento, é necessário um índice secundário pelo campo chave estrangeira na tabela detalhe. Se o índice ainda não existe, crie agora através do DataBase Explorer, seu nome será IndSetCodigo

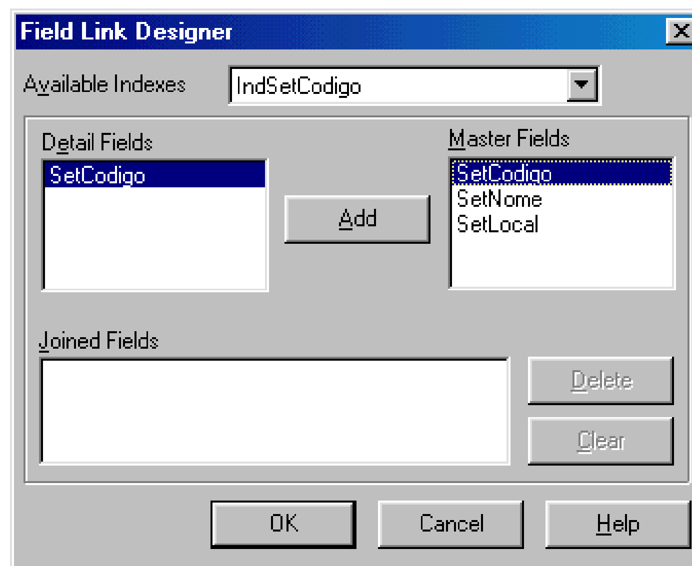
Para definirmos um relacionamento (explícito) do tipo Mestre-Detalhe selecione a tabela detalhe (TbFuncionario) preencha as propriedades básicas.

```
object TbFuncionario: TTable
  Active = True
  DatabaseName = 'Empresa'
  TableName = 'Funcionario.db'
  Name = TbFuncionario

  MasterSource = DsSetor
end
```

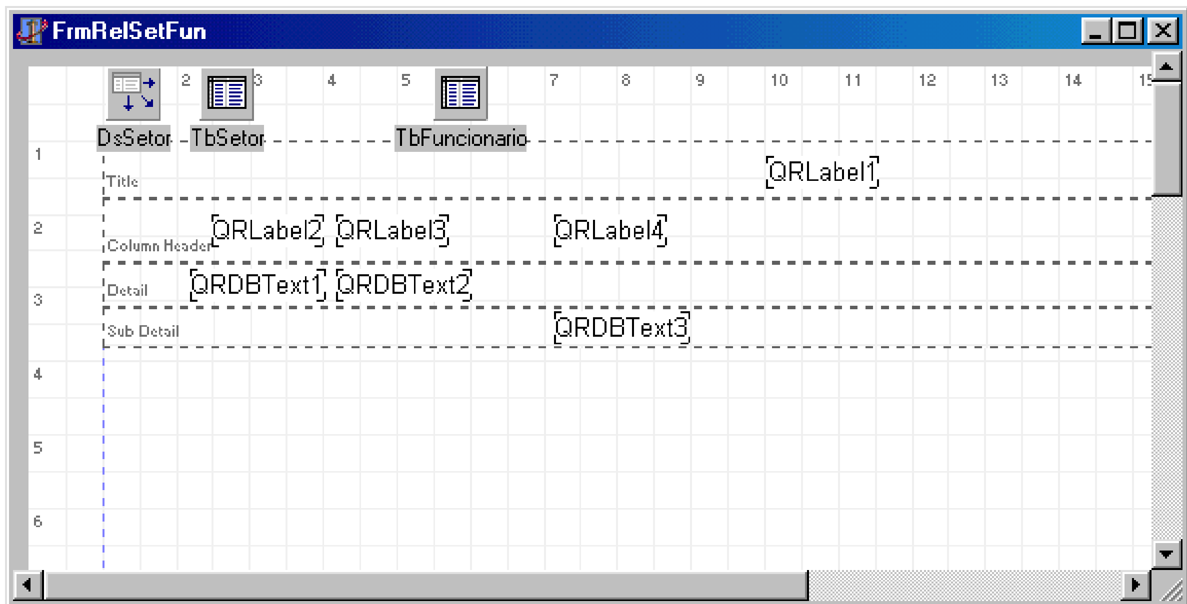
A propriedade **MasterFields** será preechida pela caixa de diálogo onde:

Available Indexes	Representa o índice secundário (da tabela funcionário) pela chave estrangeira (SetCodigo).
Detail Fields	É a chave estrangeira.
Master Fields	É a chave primária.

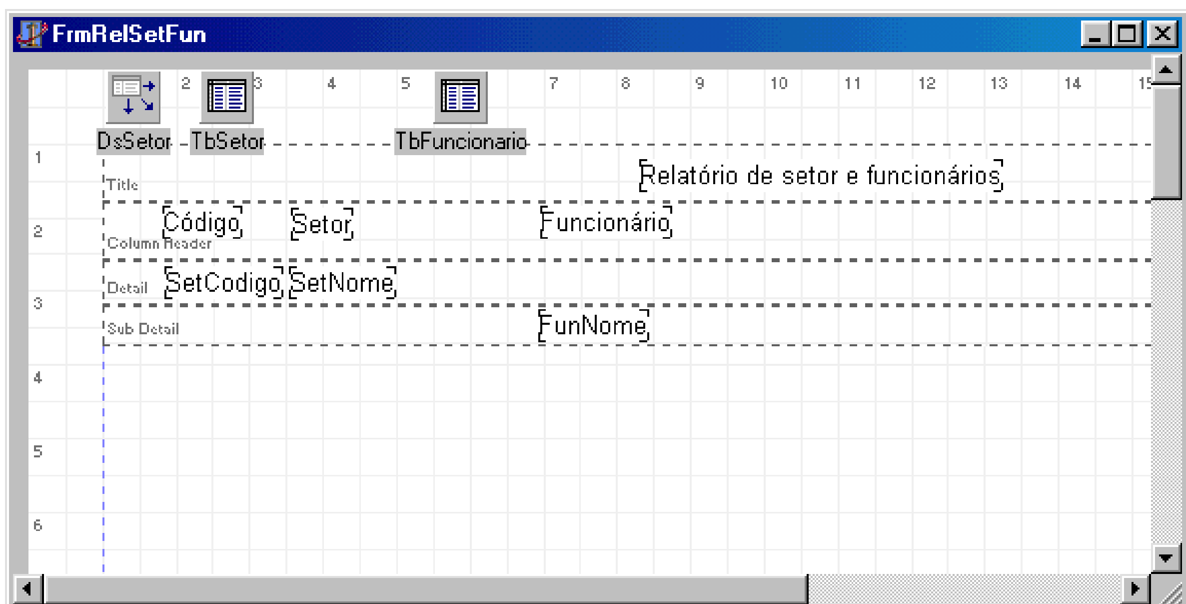


Clique em **Add** e confirme a janela com **Ok**.

Defina as bandas e insira os componentes necessários à visualização dos dados:



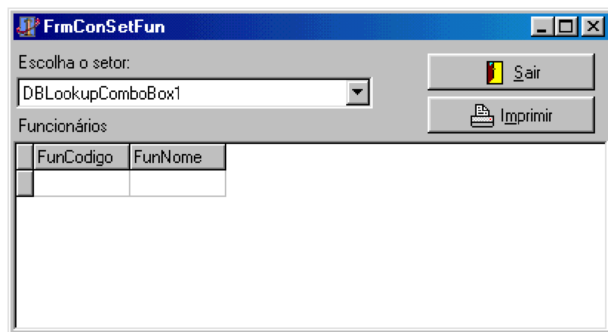
- ✓ A banda *Detail* se refere à tabela Mestre (TbSetor)
- ✓ A banda *Sub Detail* se refere à tabela Detalhe (TbFuncionario)
- ✓ A banda *Sub Detail* deve ter a propriedade DataSet ligada à tabela *detalhe*.
- ✓ Ligue os componentes QRDB às suas respectivas tabelas e campos.
- ✓ O componente QuickRep deve ter a propriedade DataSet ligada à tabela *mestre*.



CONSULTAS E IMPRESSÃO

Como foi referenciado anteriormente, na consulta de setor e funcionários (FrmConSetFun) havia um botão para imprimir a consulta.

```
procedure TFrmConSetFun.BbtImprimirClick(Sender: TObject);
begin
  with FrmRelSetFun do
    begin
      TbSetor.Filter := 'SetCodigo='+
      DmEmpresa.TbSetor.FieldByName('SetCodigo').AsString;
      TbSetor.Filtered := True;
      QuickRep1.PreviewModal;
      TbSetor.Filter := '';
      TbSetor.Filtered := False;
    end;
end;
```

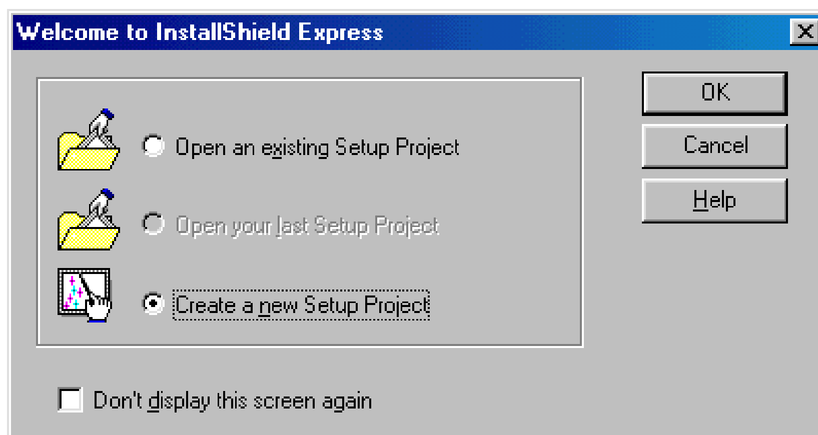


INSTALLSHIELD

O programa InstallShield permite gerar um procedimento automatizado para a instalação da sua aplicação em outros computadores por usuários experientes ou não.

Vamos exemplificar a utilização do programa gerando a instalação do projeto empresa.

Utilizaremos a versão *InstallShield Express for Delphi5*[14].
Ao executar o programa, uma tela de entrada semelhante à figura abaixo deve abrir.



Onde as opções acima representam:

Opção	*Objetivo*
Open na existing Setup Project	Permite abrir um projeto já gravado anteriormente.
Open your last Setup Project	Abrir o último projeto criado.
Create a new Setup Project	Criar um novo projeto de instalação.

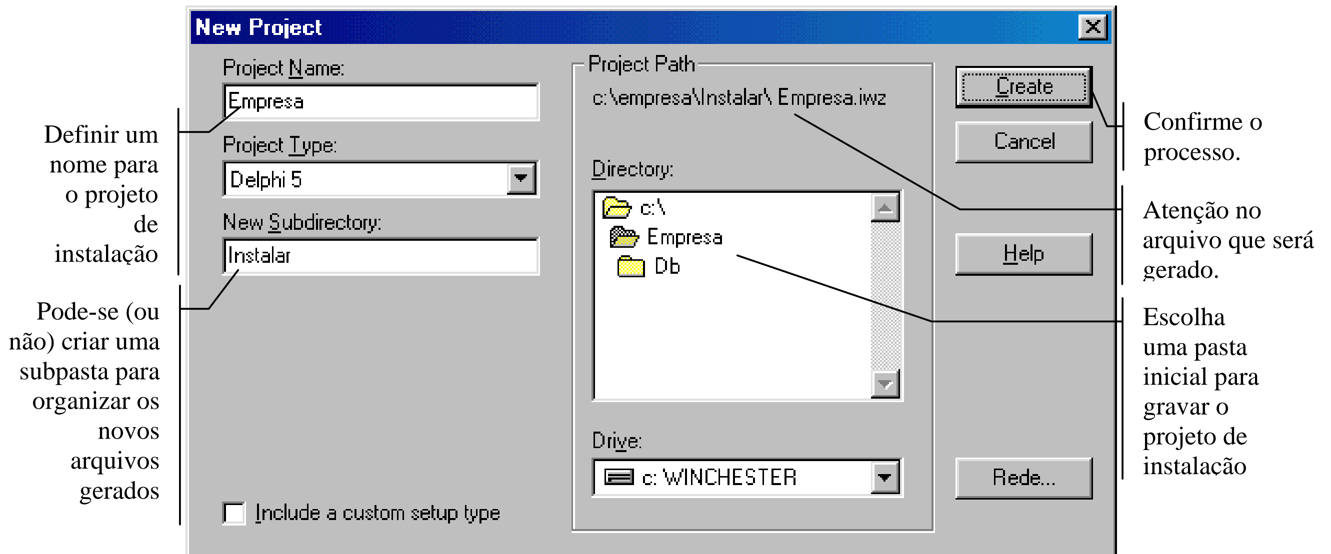
Vamos fazer inicialmente uma distinção entre *Projeto em Delphi* e *Projeto de Instalação* do InstallShield.

- Projeto em Delphi é um conjunto de arquivos (.DPR, .PAS, .DFM, ...) que serão compilados e seu resultado será o arquivo .EXE para a execução do programa.
- Projeto de instalação é um arquivo .IWZ que gravará as configurações para a instalação do aplicativo (.EXE + Banco de Dados + BDE).

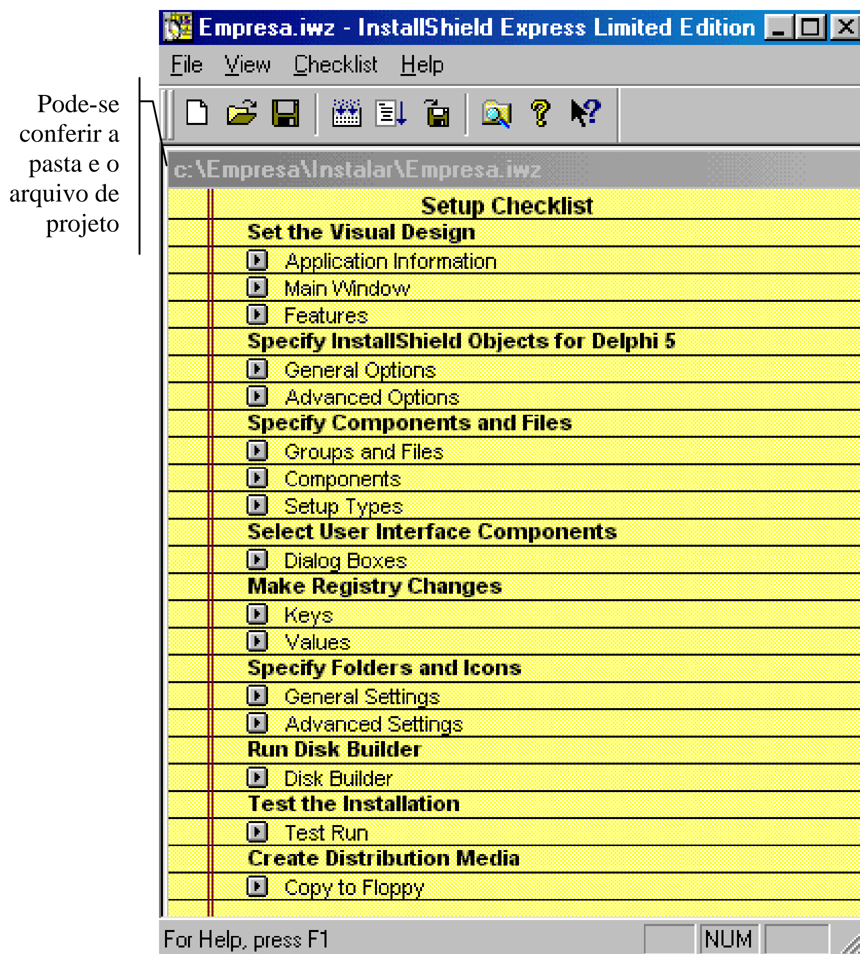
Na janela inicial, escolha a opção '*Create a new Setup Project*'.

Uma próxima janela deve ser preenchida com as seguintes informações:

[14] Embora esta versão seja encontrada no CD de instalação do Delphi é necessário instalá-la separadamente do processo de instalação do Delphi.

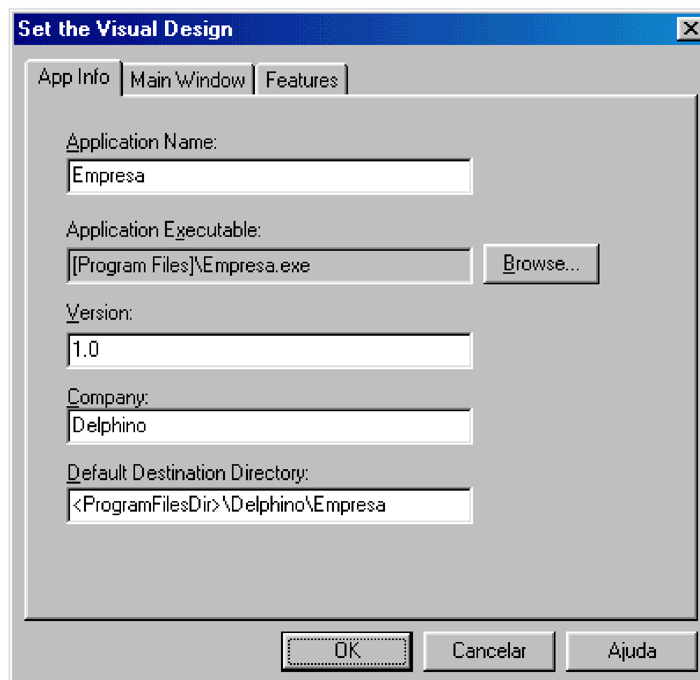


Após o preenchimento e confirmação da janela acima, uma janela principal será exibida em forma de um 'caderno' para a manipulação de items separados em grupos principais:



Set the Visual Design

Clique no item ***Application Information***. E preencha os items abaixo:



Onde:

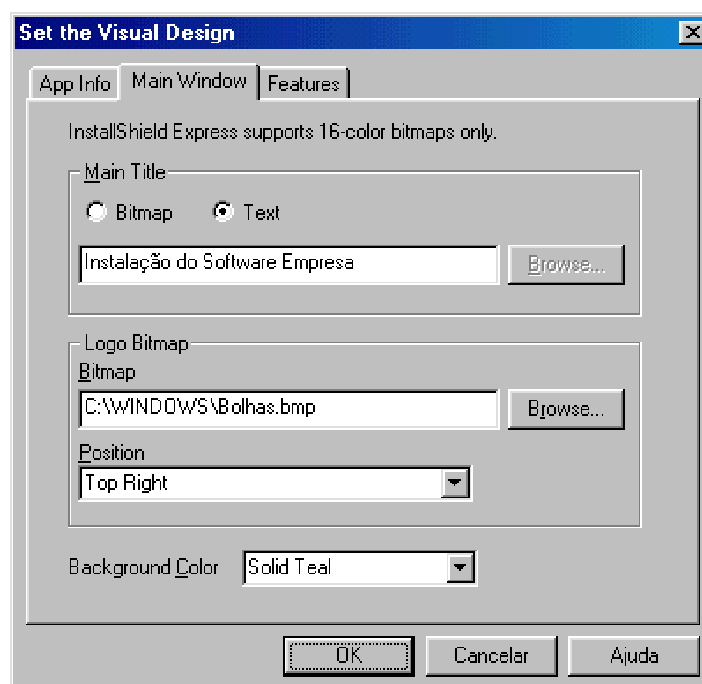
Item da janela	*Finalidade*
Application Name:	Define um nome para o aplicativo.
Application Executable:	Define o arquivo executável do aplicativo.
Version:	Define a versão do aplicativo, importante para atualizações futuras.
Company:	Nome da empresa de desenvolvimento do software.
Default Destination Directory:	Define o diretório de destino *padrão*.

Os termos entre **< >** são definidos pelo InstallShield da seguinte forma:

<ProgramFilesDir>	Define a pasta 'Arquivos de Programas'
<INSTALLDIR>	Define o caminho (path) completo escolhido pelo usuário no momento da instalação.
<WINDIR>	Define a pasta do Windows na máquina destino.
<WINSYSDIR>	Define a pasta 'System' do Windows, muito usada para receber arquivos do tipo DLL.
<CommonFilesDir>	Define a pasta 'Arquivos Comuns', subpasta de 'Arquivos de Programas'.
<FONTDIR>	Define a pasta onde são armazenados os arquivos .TTF no Windows, ou seja, as fontes.

Maiores informações, clique no botão Ajuda dentro da janela.

Ainda na mesma janela, clique na *guia* '*Main Window*'.



Onde:

Main Title	Define um título durante a instalação.
Logo BitMap	Define um arquivo de bitmap utilizado durante a instalação, provavelmente o logotipo de sua empresa.
Background Color	Defina a cor de fundo da tela durante a instalação.

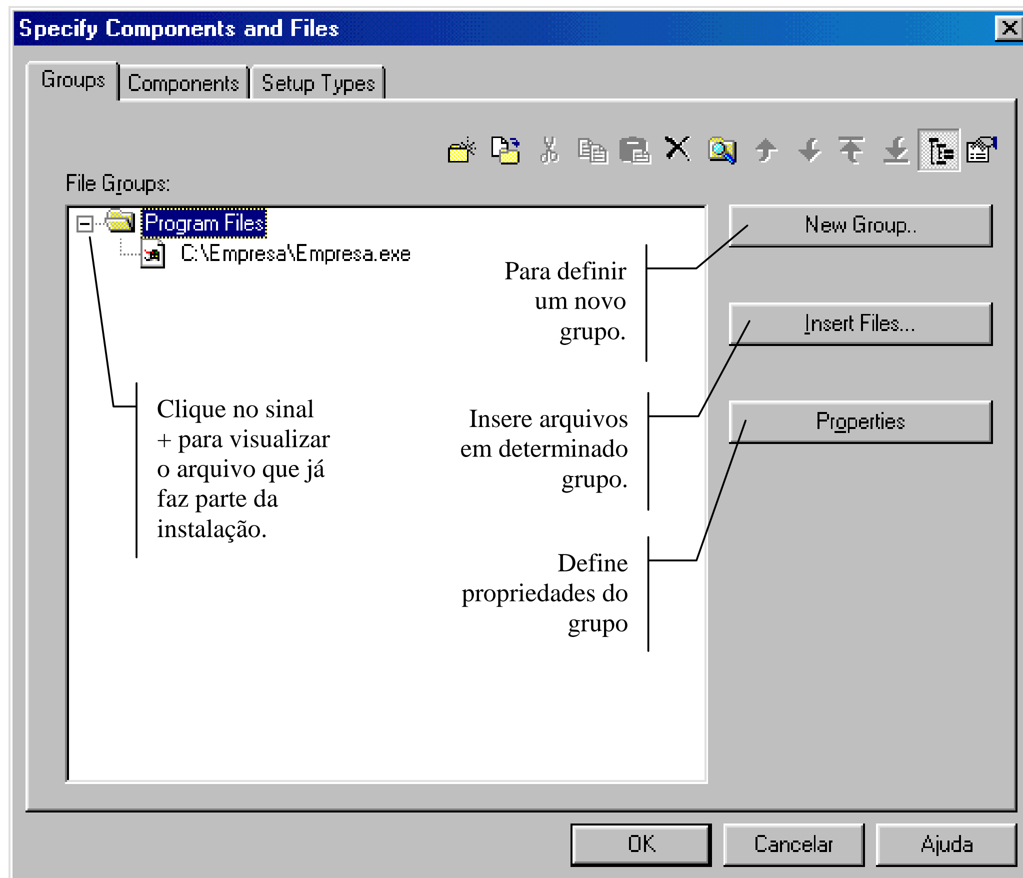
E na última guia: *Features* deixe ligado a opção *Automatic Uninstaller*.

Essa opção define o item de desinstalação no Windows através do **Painel de Controle | Adicionar ou Remover Programas**.

Confirme esta janela com o botão OK.

Specify Components and Files

Na janela principal selecione o item ***Groups and Files***



Já temos um grupo padrão chamado *Program Files*. Este grupo contém um único arquivo, o executável. Porém para que a aplicação Delphi (*que use banco de dados*) funcione corretamente, no micro do usuário deve haver além do executável, os arquivos do banco de dados e a camada BDE.

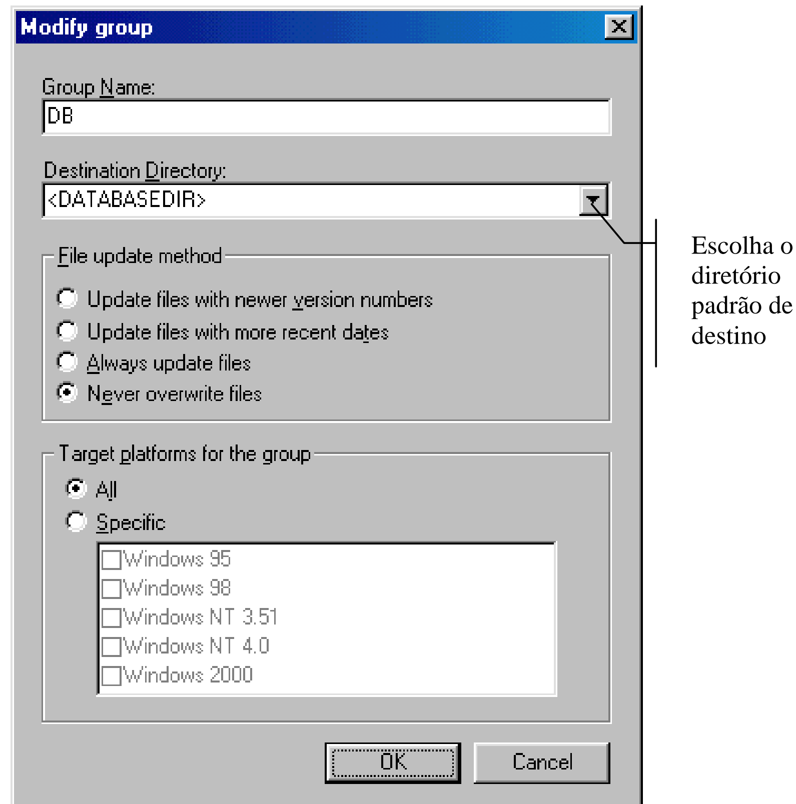
Vamos definir neste momento um grupo para o banco de dados.

Clique no botão New Group. A próxima figura vai ilustrar estas opções.

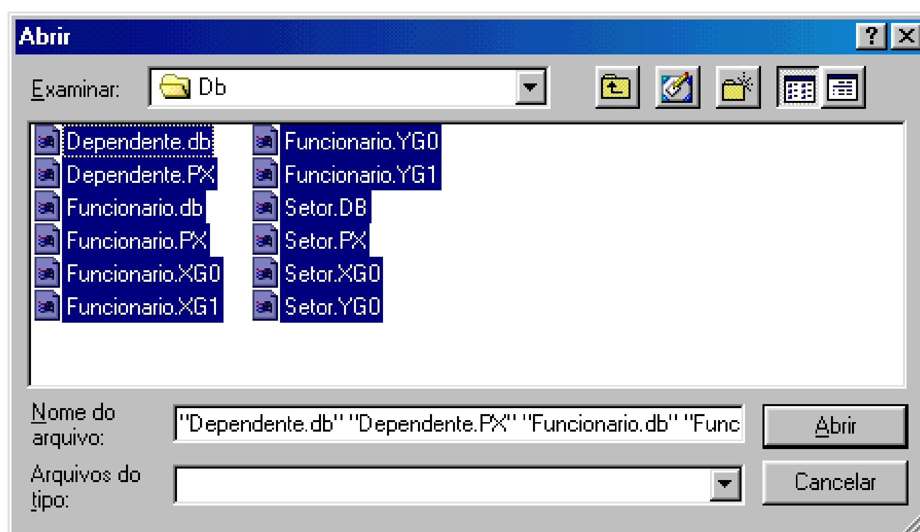
Digite um nome para o grupo, neste exemplo será **DB**

Defina a pasta de destino, neste exemplo será: **<DATABASEDIR>** ou seja, o usuário deverá escolher uma pasta, o InstallShield então utilizará a configuração do path os arquivos de dados. A definição deste path serão definidas adiante.

O método de atualização será o de *nunca atualizar*, pois em um *upgrade* do software, podemos enviar arquivos de dados vazios ou diferentes do que o usuário já está utilizando.

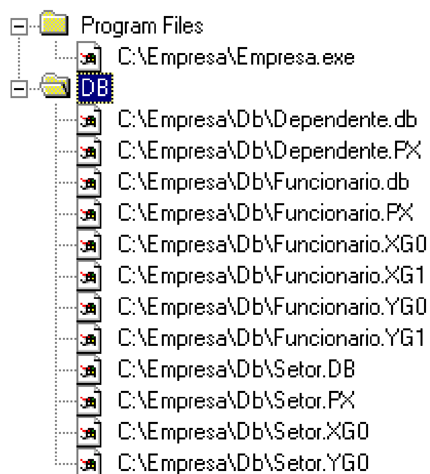


Confirme a janela acima com OK.
Uma nova pasta (grupo) deve ser exibido na janela anterior. Selecione o grupo DB e clique em *Insert Files*. Encontre a pasta onde estão os arquivos do banco de dados e selecione-os (pelo teclado pode-se usar CTRL+A)



Confirme a janela acima.

A janela anterior deve listar dois grupos com seus respectivos arquivos.

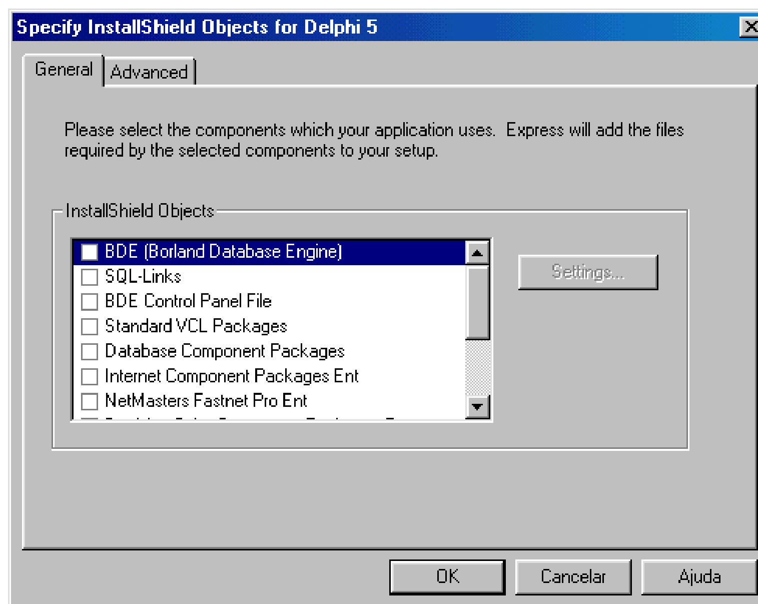


Confirme a janela com OK.

Dos arquivos básicos para a instalação temos o executável e o banco de dados já definidos em grupos. Falta configurar agora a camada BDE.

Specify InstallShield Objects for Delphi 5

Clique no item ***General Options*** da janela principal do InstallShield.



Assim que você selecionar o item BDE, uma nova janela deverá ser exibida.

Sempre utilize a opção Full BDE Installation para evitar conflitos de BDE local com BDE global. A registry do Windows suporta apenas uma configuração de BDE.

Avance para a próxima tela:

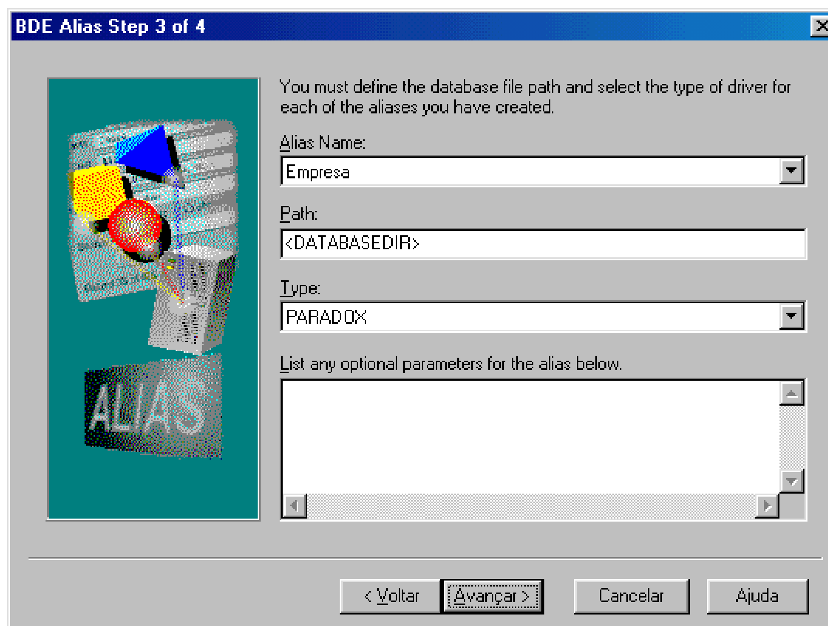
Vamos definir o nome do Alias da aplicação. Deve ser o mesmo nome utilizado pelo projeto em Delphi

Neste exemplo, o Alias tem onome de empresa. (O mesmo nome utilizado no BDE do projeto!)

Confirme o nome e clique em *OK* para confirmar. A próxima será exibida com apenas um item booleano para escolha:

Sempre confirme a gravação da configuração, ligando o checkbox disponível na janela.

Avance para a próxima etapa.



Clique em *Avançar* e em *Concluir* no último passo do processo.

Na janela onde selecionamos o BDE clique em *OK*.

Select User Interface Components

Na janela principal, clique em ***Dialog Boxes***.

Onde:

Item	*Objetivo*
Welcome Bitmap	Define uma figura no início do processo de instalação. Através da guia Settings pode ser escolhido um Bitmap de 16 cores.
Welcome Message	Define uma janela com informações padronizadas na lingua Inglesa.
Software Licence Agreement	Permite exibir uma mensagem padronizada de linceça através de uma arquivo texto (TXT). Configure o arquivo através da guia Settings.
Readme Information	Permite exibir uma mensagem genérica através de uma arquivo texto (TXT). Configure o arquivo através da guia Settings.
User Information	Permite exibir uma caixa de diálogo para preenchimento do usuário.
Choose Destination Location	Define a pasta onde será instalada a aplicação. A pasta padrão de *sugestão* para o usuário está definida na guia Settings.

Choose Database Location	Define a pasta onde será instalada a base de dados. A pasta *padrão de sugestão* deve ser definida através da guia Settings. Por exemplo **<INSTALLDIR>\DB** Dessa forma, o indicador **<DATABASEDIR>** instalará os arquivos no path que foi definido pelo usuário (**<INSTALLDIR>**) e no subdiretório **\DB.** O ALIAS será automaticamente criado e apontado para este subdiretório.
Setup Type	Define tipos de setup possíveis.
Custom Setup	Define quais os items podem ser instalados na situação de setup's personalizados.
Select Program Folder	Define em qual *grupo de programas* serão instalados os ícones.
Start Copying Files	Exibe um resumo do tipo de setup, pasta a ser utilizada e usuário.
Progress Indicator	Uma das principais janelas, define a progressão da instalação. Importante para que o usuário não pense que a máquina possa estar travada e tomar alguma medida para solucionar 'o problema'.
Billboards	Define uma figura no fim do processo de instalação. Através da guia Settings pode ser escolhido um Bitmap de 16 cores ou um WMF[15].
Setup Complete	Exibe uma janela de finalização de todo o processo. Na guia Settings pode-se escolher outras opções.

Specify Folders and Icons

Clique no item ***General Settings***.
Na guia *General* pode-se definir parâmetros para a aplicação (caso seje necessário) e o modo de exibição do ícone, se o programa será executado Normal, Maximizado ou Minimizado. Caso a opção Normal esteja selecionada, a opção WindowState do formulário principal da aplicação será usado.

Há a possibilidade de criar outros ícones para representar outros arquivos, além do executável (atual). Ao mudar o item *Description*, o ícone Add Icon ficará disponível.

Na guia *Advanced* pode-se definir uma tecla de atalho e qual o folder o ícone deve estar localizado no ambiente Windows, onde:

Item	***Objetivo***
Default Folder	Insere o ícone no grupo padrão de instação de programas. Ex: Iniciar | Programas | Empresa | Ícone Empresa.
Programs Menu Folder	Insere o ícone no grupo programas do próprio Windows. Ex: Iniciar | Programas | Ícone Empresa.
Start Menu Folder	Insere o ícone no menu do botão Iniciar.
Desktop Folder	Insere o ícone na área de trabalho. (Desktop)

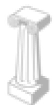
[15] Windows Metal Files. Arquivo padrão do recurso clip-art da Microsoft.

Startup Folder	Insere o ícone no grupo Iniciar. Ex: Iniciar | Programas | Iniciar | Ícone Empresa. **Obs.** Todo ícone contido neste grupo é executado na abertura do Windows.
Send To Folder	Insere o ícone no Menu Rápido do Windows, chamado através do botão direito Enviar para | Ícone Empresa

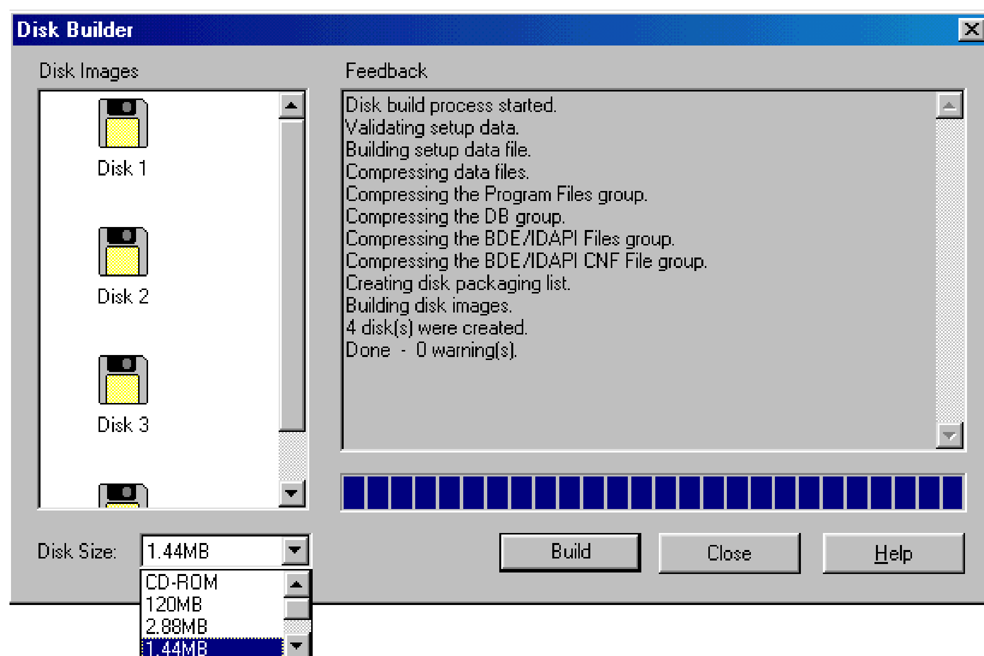
Run Disk Builder

Esta janela permite a criação dos arquivos *para* a mídia que pode ser um disquete de 3 ¼ , um zip drive ou um CD-ROM entre os mais populares. Há também opções para disquete de 720 Kb. É importante notar nos passos do processo se houve algum *warning* (cuidado) ou *error* .

O único trabalho é definir qual a mídia e clicar no botão Build. Neste exemplo, *arquivos para* quatro disquetes foram gerados.

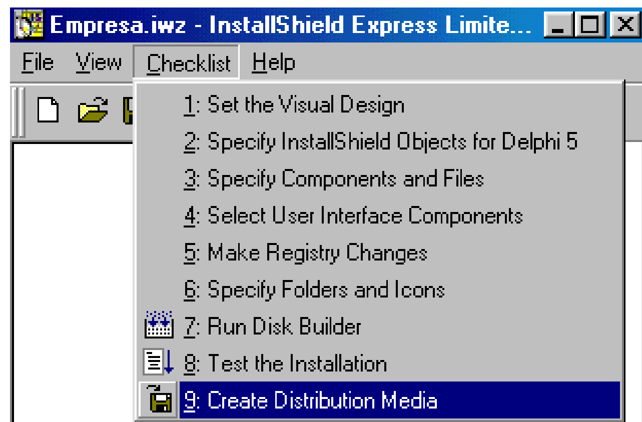


O BDE ocupa três disquetes. Subtende-se que a aplicação (EXE + Banco de dados) ocupa um disco.

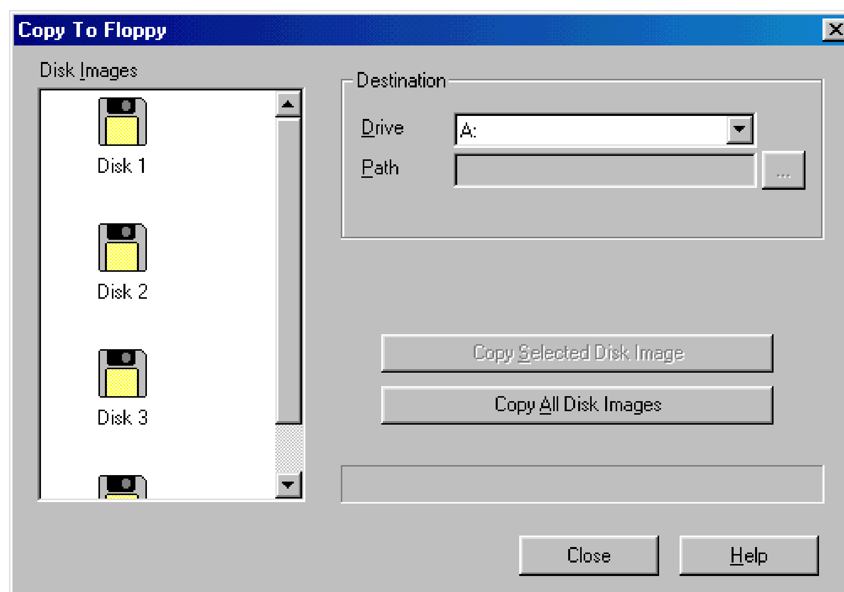


O último passo será copiar os *arquivos de instalação* gerados por todo o processo, para a mídia desejada. A opção 9 do menu **Chkcklist | Create Distribution Media** permitirá esta tarefa.

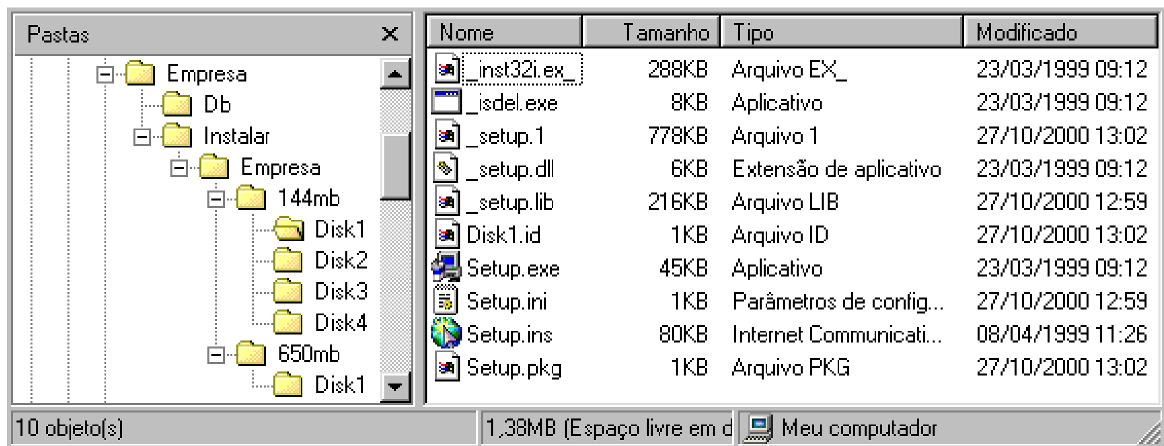
Como a seguir:



Na barra de ícones, há um ícone para o mesmo comando.



Se você não deseja gerar os discos a partir desta janela, pode-se abrir o gerenciador em: Iniciar | Programas | **Windows Explorer** e visualizar a estrutura de pastas criadas pelo InstallShield:



No exemplo acima, foram criados dois tipos de Disk Builder: Uma opção com disquetes de 1.44 e outra com CD-ROM. Note que foram criadas pastas separadas para cada opção. Os arquivos encontram-se separados e disponíveis para manipulação do desenvolvedor, podendo ser copiados para a mídia desejada.

Agora, vamos pensar na seguinte situação:
"Preciso enviar o projeto através da Internet ou de uma mídia, através de um único arquivo."

Poderíamos compactar os arquivos utilizando um compactador popular como zip ou arj mas veremos a seguir uma opção mais profissional.

INSTALLSHIELD PACKAGEFORTHEWEB

A empresa InstallShield possui um produto chamado InstallShield PackageForTheWeb que permite compactar nossos arquivos em um executável apenas.
Este software pode ser encontrado no site da empresa em: http://www.installshield.com/ no link produtos. Ele é descrito como **"PackageForTheWeb is available to all customers courtesy of InstallShield".** Ou seja, é só fazer o download e utilizar o software.

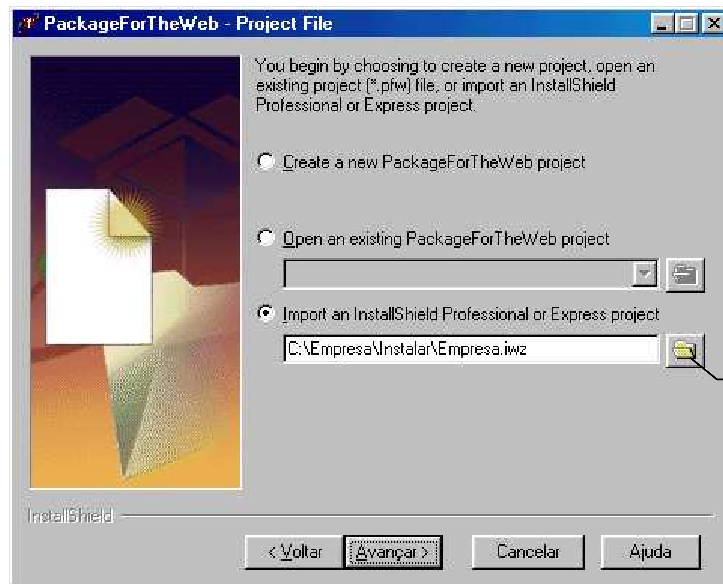
Para exemplificar sua utilização, vamos utilizar (aproveitar) o arquivo de projeto (.IWZ) anterior criado através do InstallShield Express.

Ao executar o software (PackageForTheWeb) uma tela inicial descrevendo seu objetivo e endereço na web é exibido. Clique em *Avançar*.

O arquivo de projeto do InstallShield PackageForTheWeb terá a extensão .PFW

Na próxima janela podemos escolher entre três opções:

Item	*Objetivo*
Create a new PackageForTheWeb project	Caso não tenhamos um projeto criado pelo InstallShield, podemos criar um projeto PFW independente através desta opção.
Open an existing PackageForTheWeb project	Permite editar um projeto já existente. (.PFW)
Import an InstallShield Professional or Express project	Permite importar o arquivo de projeto (.IWZ) do InstallShield Express criado anteriormente.



Permite escolher o arquivo IWZ existente.

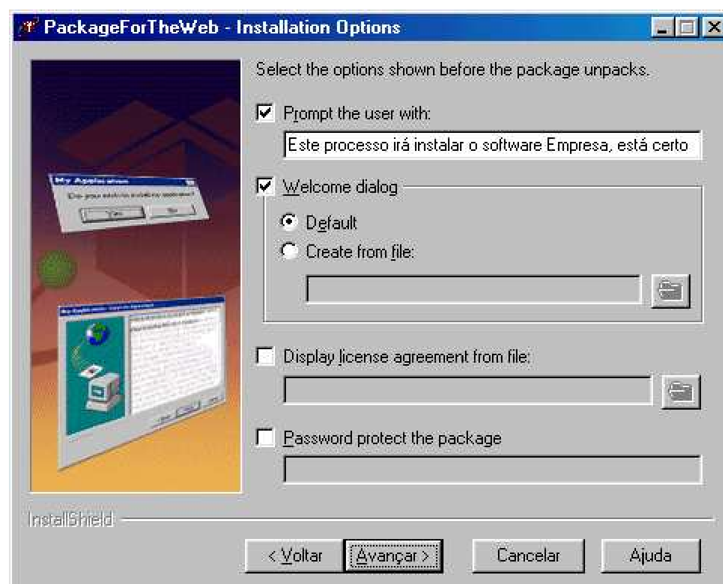
Escolha a opção de importar, localize o arquivo do projeto .IWZ e clique em *Avançar*.

Preencha a janela acima com informações que julgar necessárias e clique em *Avançar*.

Na próxima janela pode-se escolher entre a geração de um arquivo executável (*self-extract*) para disparar o processo ou apenas ‘empacotar’ os arquivos em um arquivo .CAB.

A escolha do idioma pode ser escolhido na combobox.

Clique em *Avançar*.



Nesta janela, podemos exibir uma caixa de diálogo (tipo MessageDlg) no início do processo para que o usuário possa abandonar ou prosseguir.

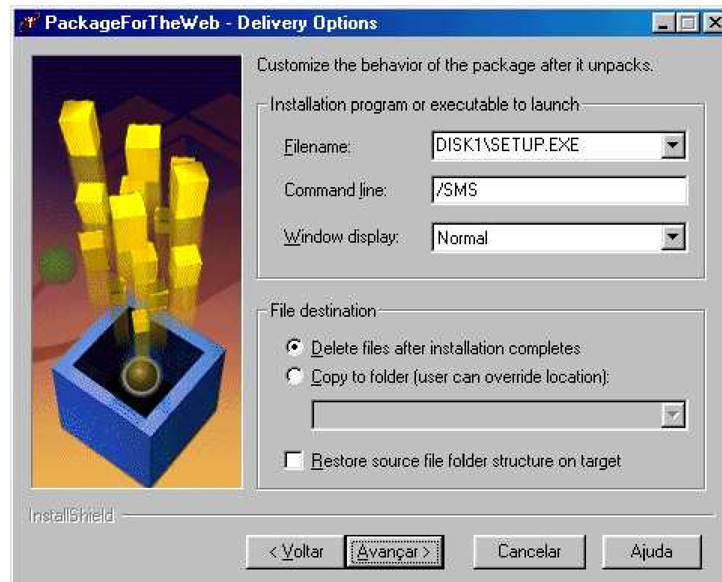
A caixa de 'bem vindo' padrão do software é uma opção que pode ser customizada através de um arquivo do tipo .RTF[16]

A opção Display licence também permite a inclusão de um arquivo de extensão RTF.

[16] Rich Text File – Padrão que permite formatos (negrito, itálico,font, etc...) em um arquivo gravado como ASCII

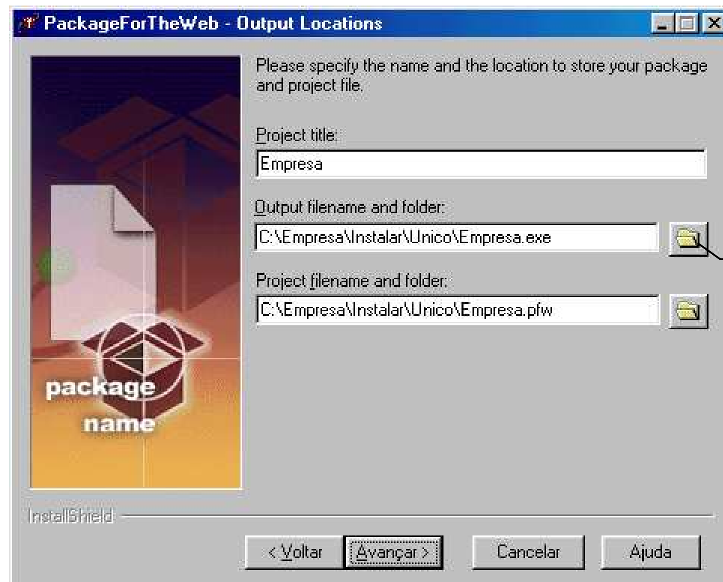
A senha (Password) pode ser definida nesta janela. Esta senha é apenas para autorizar o processo de descompactação e instalação.

Clique em *Avançar.* A próxima janela é uma das principais para entendermos a facilidade adicional deste programa com relação a outro processo de compactação.



Após a descompactação, o InstallShield vai disparar o processo de instalação através do arquivo SETUP.EXE já criado no InstallShield Express, e após a instalação os arquivos descompactados (temporários) devem ser deletados. Clicar em *Avançar.*

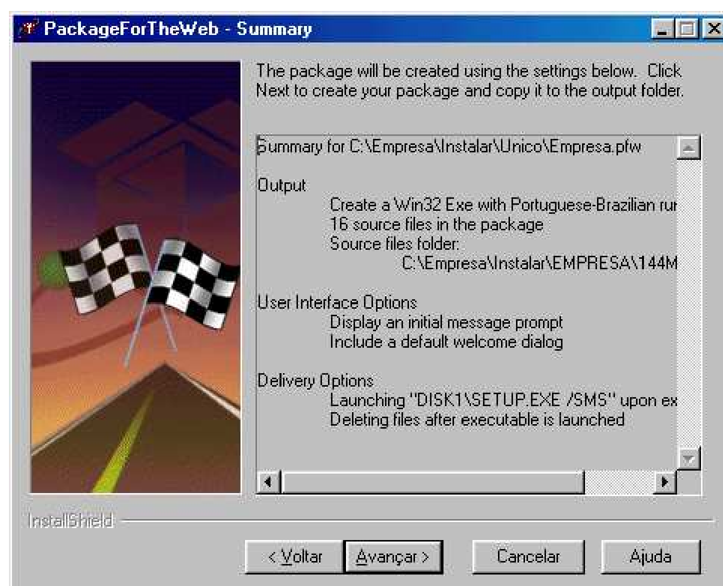
Há uma nova tecnologia de assinatura digital que pode ser obtida via Internet. Não vamos abordar este item. Clique em *Avançar.*



Pode-se definir uma pasta específica.

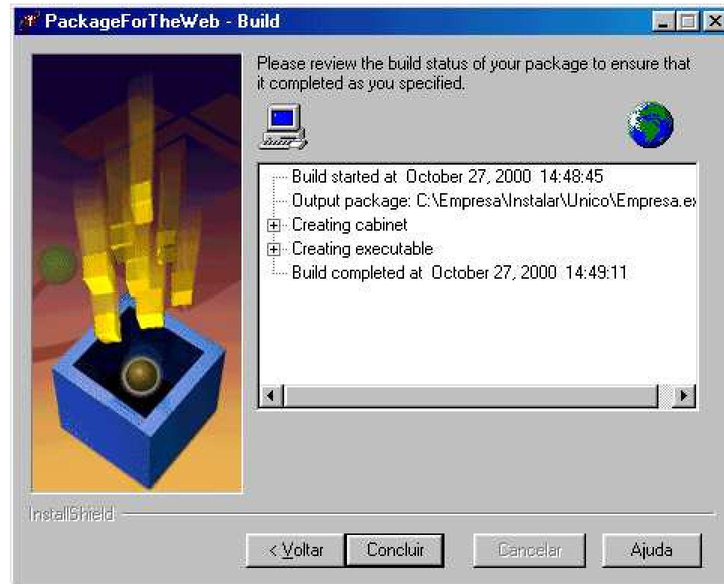
Nesta janela vamos definir o nome do projeto. Este nome será usado para o arquivo de projeto (PFW) e o arquivo executável *self-extract* (EXE).

O diretório para a geração destes arquivos pode ser definido pelos ícones de pastas. Neste exemplo, crie uma subpasta ‘Unico’ abaixo da pasta Instalar.
Clique em *Avançar.*



Esta é uma janela para conferir as opções que foram definidas anteriormente.

Clique em *Avançar.*

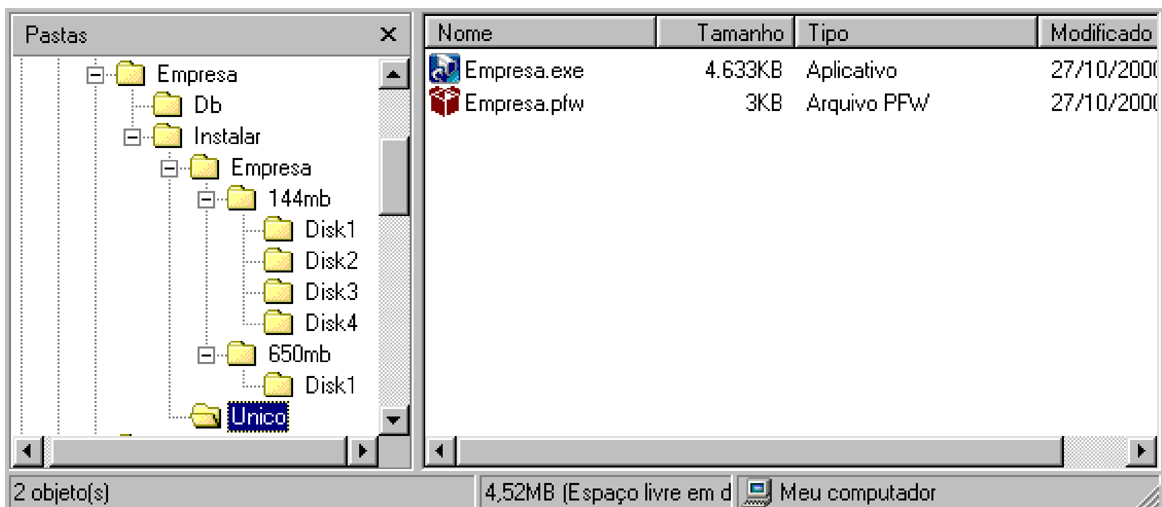


O processo vai construir o arquivo *self-extract* e o arquivo de projeto (PFW). Clique no sinal de + caso queira verificar os arquivos fazem parte do executável.

Clique em *Concluir.*

Para conferir o resultado abra o programa Windows Explorer e abra a pasta onde foram gerados os arquivos.

Podemos enviar apenas um arquivo (Empresa.exe) para realizar o processo de instalação de maneira mais simples possível.

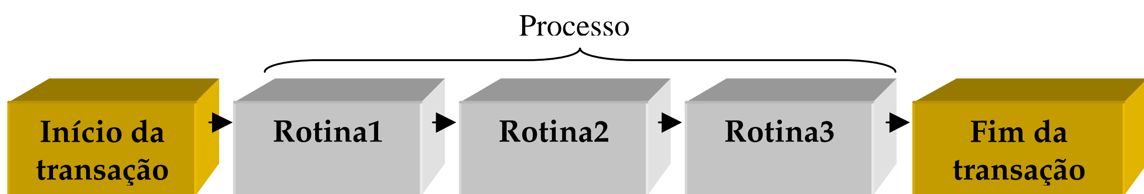


TRANSAÇÕES

Transações é um recurso muito utilizado em aplicações *client/server*. Nosso enfoque será introdutório, no sentido de despertar você como desenvolvedor, a utilizar esta tecnologia na construção de aplicações robustas.

A transação é uma definição que aborda o seguinte aspecto: Um processo iniciado com uma transação deve obrigatoriamente terminar, com ou sem sucesso. Este processo pode ser um conjunto rotinas que só terão valor se executadas em sequência.

O conceito básico pode ser ilustado como a seguir:



Caso alguma rotina tenha algum problema ou aconteça um imprevisto como queda de energia, a transação não será terminada.
Em bancos de dados que implementam este tipo de segurança[17], assim que o micro é ligado e o banco é aberto, através do arquivo de LOG a transação não finalizada terá que 'voltar atrás' na situação anterior ao início, mantendo o banco de dados consistente.

Vamos utilizar este conceito no Delphi através tabelas paradox lembrando que nem todos os recursos da transação (como arquivo de LOG) são implementados pelo banco.

DELPHI & TRANSAÇÕES

Os comandos para implementar a transação são métodos do **objeto DataBase**. Este componente representa o banco de dados, *sendo único na aplicação*.
Todas as tabelas devem se conectar ao DataBase.

Método	*Definição*
StartTransaction	Define o início da transação.
RollBack	Define o término sem sucesso da transação.
Commit	Define o término com sucesso da transação.

[17] SQL-Server, Oracle e Interbase são exemplos de bancos de dados cliente/servidor.

Inicie uma nova aplicação.

- ✓ Grave o projeto em uma pasta chamada **Trans**

Nome atual	*Novo nome*
Unit1	UFrmTrans
Project1	Transacao

Embora não seja obrigatório (ainda mais neste exemplo com uma tabela), vamos inserir um formulário DataModule, através do comando **File | New... (Data Module)**

Nome atual	*Novo nome*
Unit1	UDm

- ✓ No DataModule, insira os componentes DataBase, Table e DataSource.

Vamos utilizar as tabelas criadas no projeto empresa, bem como seu ALIAS.

Configure o componente DataBase nas propriedades indicadas:

```
object Database: TDatabase
  AliasName = 'Empresa'
  DatabaseName = 'BancoEmpresa'
  SessionName = 'Default'
  TransIsolation = tiDirtyRead
end
```

Onde:

Propriedade	*Definição*
AliasName	Define o nome do ALIAS no BDE.
DatabaseName	Define um nome para o ALIAS da aplicação.
SessionName	Define o nome da 'Sessão' utilizada para acesso ao banco.
TransIsolation	Define o nível de isolamento suportado pelo banco de dados.

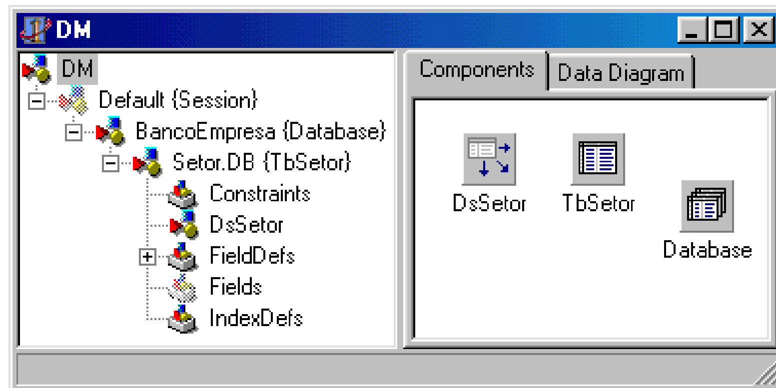
Configure os componentes Table e DataSource:

```
object DsSetor: TDataSource
  DataSet = TbSetor
end

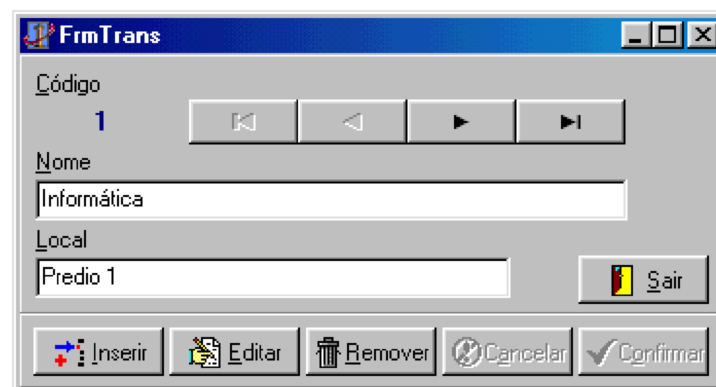
object TbSetor: TTable
  Active = True
  DatabaseName = 'BancoEmpresa'
  TableName = 'Setor.DB'
end
```

O DataModule permite confirmar a hierarquia dos componentes através da árvore de items do lado esquerdo da janela.

Note que o componente Table vai acessar o banco através do ALIAS (interno) *BancoEmpresa*.



Crie os campos persistentes com um duplo clique no componente Table.
Arraste-os para o formulário FrmTrans e crie uma interface básica com os componentes (Buttons, DBNavigator) já vistos anteriormente.



O código fonte com a implementação de transações está descrito a seguir:

```
unit UFrmTrans;

{A interface não foi impressa}

implementation

uses UDm;

{$R *.DFM}

procedure TFrmTrans.BbtSairClick(Sender: TObject);
begin
  FrmTrans.Close;
end;
```

```
procedure TFrmTrans.TrataBotoes;
begin
  BbtInserir.Enabled := not BbtInserir.Enabled;
  BbtEditar.Enabled := not BbtEditar.Enabled;
  BbtRemover.Enabled := not BbtRemover.Enabled;
  BbtCancelar.Enabled := not BbtCancelar.Enabled;
  BbtConfirmar.Enabled := not BbtConfirmar.Enabled;
  BbtSair.Enabled := not BbtSair.Enabled;
  DbNavigator1.Enabled := not DbNavigator1.Enabled;
end;

procedure TFrmTrans.FormCloseQuery(Sender: TObject;
  var CanClose: Boolean);
begin
  if Dm.TbSetor.State in [dsEdit, dsInsert] then
    if MessageDLG('Existem dados pendentes,'+#13+'deseja gravá-los?',
                  mtConfirmation, [mbYes,mbNo], 0) = mrYes then
       CanClose := False
    else
      begin
        Dm.TbSetor.Cancel;
        Dm.Database.Rollback;
        TrataBotoes;
        CanClose := True;
      end;
end;

procedure TFrmTrans.BbtInserirClick(Sender: TObject);
var ProxNum: Integer;
begin
  Dm.TbSetor.Last;
  ProxNum := Dm.TbSetor.FieldByName('SetCodigo').AsInteger + 1;
  Dm.Database.StartTransaction;
  Dm.TbSetor.Append;
  Dm.TbSetor.FieldByName('SetCodigo').AsInteger := ProxNum;
  DbEdit2.SetFocus;
  TrataBotoes;
end;

procedure TFrmTrans.BbtEditarClick(Sender: TObject);
begin
  Dm.Database.StartTransaction;
  Dm.TbSetor.Edit;
  TrataBotoes;
end;

procedure TFrmTrans.BbtRemoverClick(Sender: TObject);
begin
  if Dm.TbSetor.RecordCount = 0 then
     ShowMessage('Tabela vazia!')
    else if MessageDLG('Tem certeza que deseja remover o setor: '+#13+
                  Dm.TbSetor.FieldByName('SetNome').AsString +' ?',
                  mtConfirmation, [mbYes, mbNo], 0) = mrYes then
            Dm.TbSetor.Delete;
end;
```

```
procedure TFrmTrans.BbtCancelarClick(Sender: TObject);
begin
  Dm.TbSetor.Cancel;
  Dm.Database.Rollback;
  TrataBotoes;
end;

procedure TFrmTrans.BbtConfirmarClick(Sender: TObject);
begin
  Dm.TbSetor.Post;
  Dm.Database.Commit;
  TrataBotoes;
end;

end.
```

DELPHI & SQL

SQL é uma linguagem da mesma forma que Object Pascal, Basic ou C++, a grande diferença é que SQL é voltada totalmente para o Banco de Dados independente do Front End (Delphi, VB ou C++Builder). Obviamente, seus aspectos e particularidades devem ser estudados em um curso à parte. Nosso objetivo aqui é ter um entendimento de como o Delphi pode manipular o código SQL.

Através do SQL é possível manipularmos diferentes bancos de dados (Oracle, SQL-Server, DB2) utilizando uma única linguagem e não necessitando de aprender *todos*[18] os comandos ao mudar de banco de dados.

Um dos recursos mais utilizados da linguagem é o comando SELECT, sua utilização obedece uma estrutura definida como abaixo:

```
SELECT [DISTINCT] * | column_list
FROM table_reference
[WHERE predicates]
[ORDER BY order_list]
[GROUP BY group_list]
[HAVING having_condition]
```

Os items definidos entre [] são opcionais.

Exemplo:

SELECT * FROM employee WHERE HireDate < ‘01/01/1990’ ORDER BY HireDate	Selecione todos os campos Da tabela employee Onde a data é menor do que 01/01/1990 Ordenada pelo campo data

SELECT EmpNo,FirstName, LastName FROM employee WHERE (HireDate < ‘01/01/1990’) AND (Salary > 50000) ORDER BY FirstName	Selecione somente os 3 campos Da tabela employee Onde a data é menor do que 01/01/1990 E o salário é maior do que 50000 Ordenada pelo campo FirstName

As cláusulas Group By e Having não serão abordadas. Elas são utilizadas para estabelecer grupos e agregação.

O Delphi permite a manipulação de comandos em SQL através de um componente *DataSet* chamado *TQuery*. Vamos exemplificar uma aplicação utilizando consultas[19] em SQL.

---

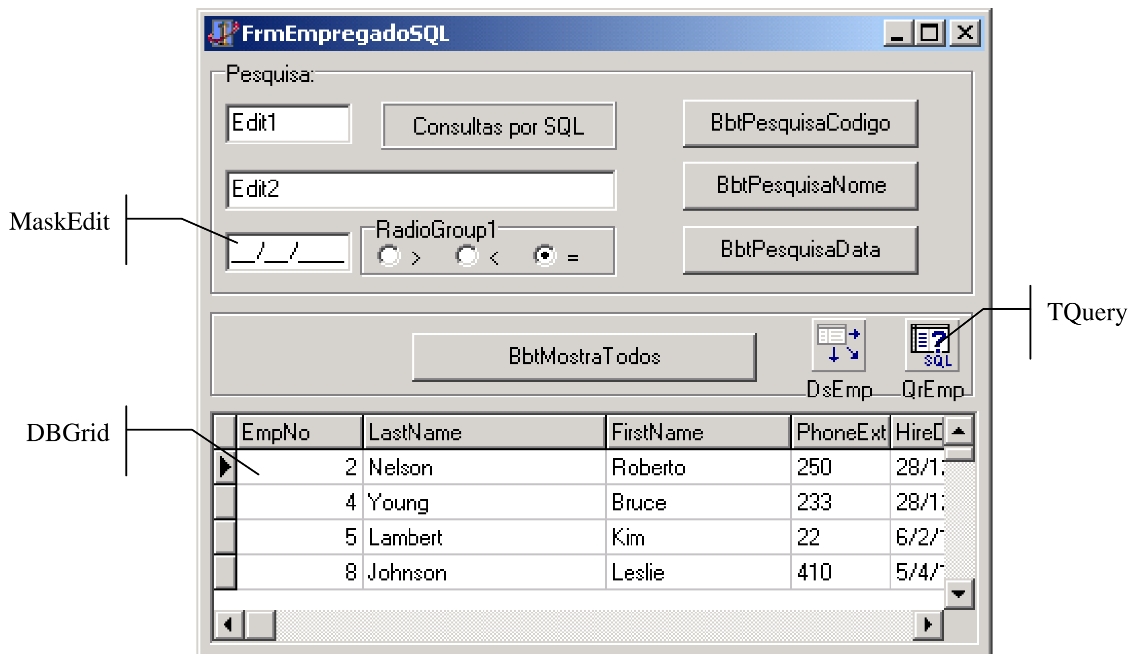
[18] Em cada banco de dados há uma extensão do SQL para manipular recursos específicos.
[19] Sinônimo de pesquisa.

Salve um novo projeto dentro de uma pasta com o nome de SQL.

Unit1	UFrmEmpregadosSQL
Project1	EmpregadoSQL

Conforme a figura a seguir, insira os componentes necessários.

Observe que os componentes utilizados (exceção - DBGrid) **não** são componentes *data-aware* (Paleta-Data Controls) porque não estamos inserindo ou editando os dados, mas passando *parâmetros* para a consulta.



O componente TQuery pode ser configurado como a seguir:

```
object QrEmp: TQuery
  DatabaseName = 'DBDEMOS'
  SQL.Strings =
     SELECT *
     FROM employee
  Active = True
  Name = QrEmp
end
```

O componente DataSource pode ser configurado como a seguir:

```
object DsEmp: TDataSource
  AutoEdit = False
  DataSet = QrEmp
  Name = DsEmp
end
```

Os componentes Buttons devem ter sua propriedade Name de acordo com a orientação na figura anterior.

TQuery - Propriedades

Active	Define se o dataset está ativo.
DatabaseName	Define o alias ou path fixo para o banco de dados.
SQL	Define a string SQL.

TQuery - Métodos

Close	Fecha o dataset.
ExecSQL	Executa o código SQL definido na propriedade SQL sem retornar um cursor. Utilizado em INSERT, UPDATE, DELETE, e CREATE TABLE.
Open	Executa o código SQL definido na propriedade SQL com retorno de cursor.
SQL.Add	Adiciona uma string à propriedade SQL.
SQL.Clear	Limpa a lista (TStrings) da propriedade SQL.

Após definido a disposição dos componentes defina os códigos para cada evento da unit como a seguir:

```
unit UFrmEmpregadoSQL;
interface
uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  StdCtrls, Grids, DBGrids, Db, DBTables, Mask, ExtCtrls, AppEvnts;

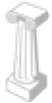
type
  TFrmEmpregadoSQL = class(TForm)
    DsEmp: TDataSource;
    QrEmp: TQuery;
    DBGrid1: TDBGrid;
    GroupBox1: TGroupBox;
    Edit1: TEdit;
    BtnPesquisaCodigo: TButton;
    Edit2: TEdit;
    BbtPesquisaNome: TButton;
    GroupBox2: TGroupBox;
    MaskEdit1: TMaskEdit;
    BbtMostraTodos: TButton;
    BbtPesquisaData: TButton;
    RadioGroup1: TRadioGroup;
    Panel1: TPanel;
    procedure BtnPesquisaCodigoClick(Sender: TObject);
    procedure BbtPesquisaNomeClick(Sender: TObject);
    procedure BbtMostraTodosClick(Sender: TObject);
    procedure BbtPesquisaDataClick(Sender: TObject);
    procedure Edit1KeyPress(Sender: TObject; var Key: Char);
  private
    { Private declarations }
  public
    { Public declarations }
  end;
```

```
var
  FrmEmpregadoSQL: TFrmEmpregadoSQL;

implementation

{$R *.DFM}

procedure TFrmEmpregadoSQL.BbtPesquisaCodigoClick(Sender: TObject);
begin
  if Edit1.Text <> '' then
  begin
  QrEmp.Close;
  QrEmp.SQL.Clear;
  QrEmp.SQL.Add('SELECT * FROM employee');
  QrEmp.SQL.Add('WHERE EmpNo = :Codigo');
  QrEmp.ParamByName('Codigo').AsInteger := StrToInt(Edit1.Text);
  QrEmp.Open;
  end;
end;
```



A passagem de parâmetros para o código em SQL será definido através do caracter **:**

```
procedure TFrmEmpregadoSQL.Edit1KeyPress(Sender: TObject; var Key: Char);
begin
  if ((Key in ['0'..'9'] = False) and (Word(Key) <> VK_BACK)) then
    Key := #0;
end;

procedure TFrmEmpregadoSQL.BbtPesquisaNomeClick(Sender: TObject);
begin
  if Edit2.Text <> '' then
  begin
    QrEmp.Close;
    QrEmp.SQL.Clear;
    QrEmp.SQL.Add('SELECT EmpNo,FirstName,LastName,Salary FROM employee');
    QrEmp.SQL.Add('WHERE UPPER(FirstName) LIKE :Nome');
    QrEmp.ParamByName('Nome').AsString := UpperCase(Edit2.Text) + '%';
    QrEmp.Open;
  end;
end;

procedure TFrmEmpregadoSQL.BbtMostraTodosClick(Sender: TObject);
begin
  QrEmp.Close;
  QrEmp.SQL.Clear;
  QrEmp.SQL.Add('SELECT * FROM employee');
  QrEmp.Open;
end;
```

```
procedure TFrmEmpregadoSQL.BbtPesquisaDataClick(Sender: TObject);
begin
    QrEmp.Close;
    QrEmp.SQL.Clear;
    QrEmp.SQL.Add('SELECT FirstName, LastName, HireDate FROM employee');
    case RadioGroup1.ItemIndex of
      0: QrEmp.SQL.Add('WHERE HireDate > :Data');
      1: QrEmp.SQL.Add('WHERE HireDate < :Data');
      2: QrEmp.SQL.Add('WHERE HireDate = :Data');
    end;//case
  try
    QrEmp.ParamByName('Data').AsDate := StrToDate(MaskEdit1.Text);
    QrEmp.Open;
  except
    ShowMessage('Dados inválidos!');
  end;//bloco try
end;

end.
```

TÉCNICAS DE INTERFACE

SPLASH SCREEN

As telas de apresentação (Splash Screen) são muito utilizadas em diversos programas do windows, exemplo Delphi, Word, Excel, etc... É uma janela de propaganda que é vista no inicio, durante o *carregamento* dos recursos daquele programa para a memória principal. Vamos criar uma tela de entrada para demonstrar este recurso:

Crie uma nova aplicação e grave os arquivos com os nomes:

- ✓ `Pasta – Splash`
- ✓ `Unit1 – UFrmPrincipal`
- ✓ `Project – Splash`

A propriedade Name do formulário deve ter o mesmo nome da Unit sem a letra 'U'.

Crie outro formulário (**File | New Form**) e grave-o com o nome `UFrmSegundo` .

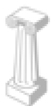
Crie agora, um terceiro formulário e grave-o com o nome `UFrmSplash` . Modifique as seguintes propriedades deste formulário:

```
object FrmSplash: TFrmSplash
  BorderStyle = bsNone
  Caption = 'FrmSplash'
  ClientHeight = 230
  ClientWidth = 265
  Color = clBtnFace
  Position = poScreenCenter
end;
```

Dentro deste formulário insira um componente image e modifique suas propriedades:

```
object Image1: TImage
  Align = alClient
  Picture = {insira uma figura a seu gosto}
end;
```

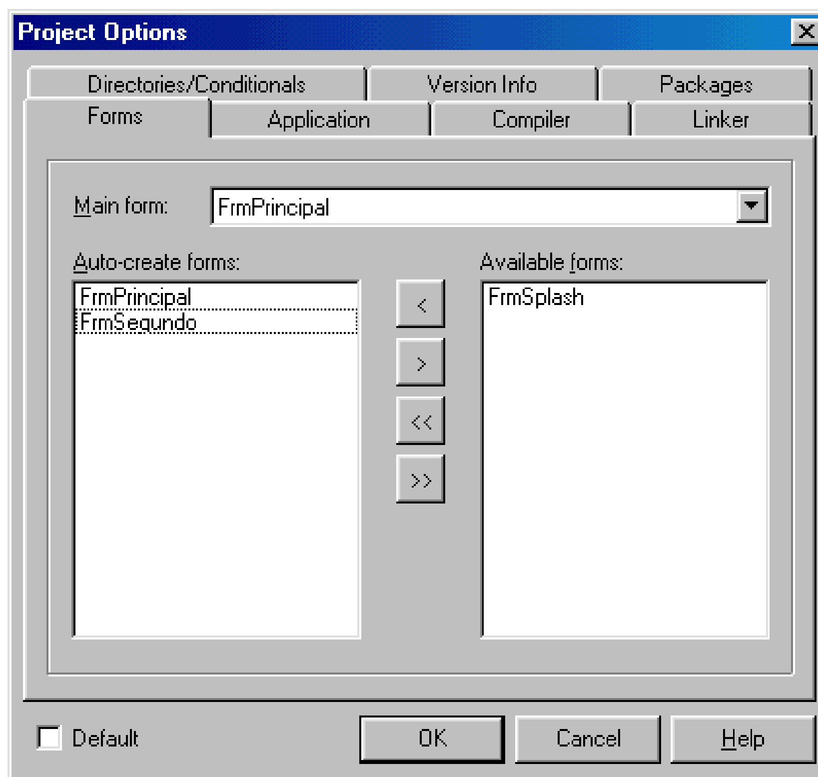
Pode-se inserir componentes Label's por cima do componente Image, basta ajustar a propriedade `Transparent` para true.



Nota-se que apesar do formulário FrmSplash ser exibido antes do formulário principal ele pode ser criado a qualquer momento, através da alteração na implementação de código a seguir.

Agora, clique no Menu **Project | Options** e na lista *Auto-Create* forms selecione o formulário FrmSplash. Através do ícone '**>**' mude o formulário para a lista *Available forms*.

Todo formulário criado pelo padrão do Delphi é carregado para a memória no momento da execução do programa, sendo definida a ordem através da lista *Auto-Create*. Isso não quer dizer que os formulário devem ser chamados nesta ordem, mas que *todos* os formulários referenciados em *Auto-Create* serão *criados* na memória principal automaticamente e todos que estão na lista *Available forms* estarão *disponíveis* para serem carregados em no momento definido pelo usuário.



- ✓ Confirme a janela e clique no Menu **Project – View Source**.
- ✓ A guia do arquivo de projeto (DPR) deverá ser exibida no Code Editor.

Vamos alterar o código do arquivo DPR para a seguinte disposição:
(Não é necessário digitar os comentários.)

```
program Splash;

uses
  Forms, Windows,
  UFrmPrincipal in 'UFrmPrincipal.pas' {FrmPrincipal},
  UFrmSegundo in 'UFrmSegundo.pas' {FrmSegundo},
  UFrmSplash in 'UFrmSplash.pas' {FrmSplash};

{$R *.RES}

begin
  Application.Initialize;
  {Cria o formulario atraves do metodo da propria classe}
  FrmSplash := TFrmSplash.Create(Application);
  //Exibe o formulario
  FrmSplash.Show;
  {Atualiza o form para exibir os componentes inseridos: Image, Labels, etc}
  FrmSplash.Refresh;

  Application.CreateForm(TFrmPrincipal, FrmPrincipal);
  Application.CreateForm(TFrmSegundo, FrmSegundo);

  {Caso queira exibir o formulario por um tempo minimo
   pode-se usar a função Sleep que 'congela o sistema'
   por um determinado tempo mensurado em milisegundos.
   Acrescentar a biblioteca Windows na clausula USES}
  Sleep(2000);
  {Libera a memoria destruindo o formulario de splash}
  FrmSplash.Free;
  {Disponibiliza a aplicação para a interação com o usuario}

  Application.Run;
end.
```

Com esse raciocínio, o formulário Splash será exibido inicialmente porém não será o *Main Form* (formulário principal) da aplicação, sendo destruído após os formulários serem carregados para a memória.

CRIANDO E DESTRUINDO FORMS

Como já foi visto neste exemplo anterior, os formulário são, por padrão, auto-criados pelo Delphi. Isso pode trazer alguns incovenientes em projetos maiores, imagine o Delphi ao iniciar o arquivo executável tentar carregar '50' formulários para a memória principal...

O primeiro passo é tirar o formulário da lista *Auto-Create* e passa-lo para a lista *Available forms* através do menu **Project | Options**.

Depois devemos definir o código de chamada do form e de liberação de memória do form chamado.

Supondo o seguinte exemplo: Dois formulários: um chamado UFrmPrincipal e outro UFrmSegundo, sendo que o principal faz uma chamada ao form secundário através de um button, o código seria o seguinte:

```
procedure TFrmPrincipal.Button1Click(Sender: TObject);
begin
  FrmSegundo := TFrmSegundo.Create(Application);
  FrmSegundo.ShowModal;//ou através do método Show
end;
```

No formulário chamado (FrmSegundo) deverá ser criado um código para liberar a memória *quando* este for fechado.

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

A variável Action define qual a ação o formulário deve tomar ao ser fechado. O *default* para formulários SDI é o valor caHide, ou seja, o formulário é apenas oculto permanecendo na memória principal, mudamos este comportamento através do valor caFree.
Uma outra técnica para o mesmo objetivo seria definir a liberação de memória em um bloco try como a seguir:

```
procedure TFrmPrincipal.Button1Click(Sender: TObject);
begin
  FrmSegundo := TFrmSegundo.Create(Application);
  try
    FrmSegundo.ShowModal;
  finally
    FrmSegundo.Free;
  end;//do bloco finally
end;
```

Este código acima só funciona com formulários Modais, pois o método Show permite a execução das próximas linhas de código após a sua chamada, trazendo um efeito indesejado neste exemplo.

MANIPULANDO CURSORES

É uma técnica comum no sistema operacional windows mudar o cursor dependendo da ação do usuário e da rotina a ser executada. Quando a rotina é passível de demora, é necessário trocar o cursor para o símbolo de ampulheta, para que o usuário não clique novamente na chamada da rotina ou pense que a máquina está travada, por exemplo.

Há no Delphi um objeto chamado `Screen`. Este objeto controla os cursores e é criado automaticamente pelo Delphi, o desenvolvedor não precisa cria-lo explicitamente.
Os cursores são são declarados como do tipo TCursor e podem ser manipulados como o exemplo à seguir:

```
procedure TFrmPrincipal.Button2Click(Sender: TObject);
var CursorAnterior : TCursor;
begin
  CursorAnterior := Sceen.Cursor;
  Screen.Cursor := crHourGlass;
  try
    Rotina.Demorada;
  finally
    Screen.Cursor := CursorAnterior;
  end;//do bloco finally
end;
```

O bloco try..finally foi utilizado para garantir que se algum código *(errado ou não)* for executado dentro do bloco try, o cursor deve voltar ao símbolo anterior ao símbolo da ampulheta (`crHourGlass`) que foi 'guardado' na variável `CursorAnterior`.

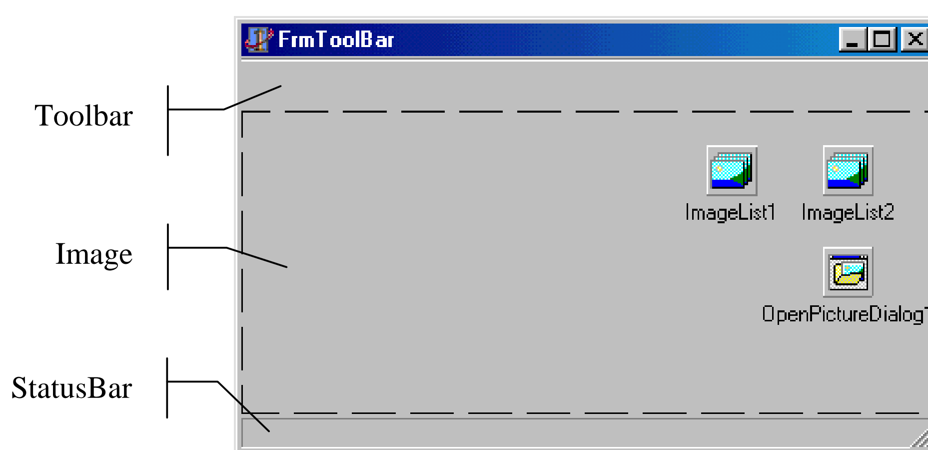
TOOLBAR

A Toolbar fornece uma barra de ferramentas com botões próprios e recursos personalizados. Os botões inseridos na Toolbar são do tipo TToolButton, considerados objetos internos da Toolbar. Vamos exemplificar sua utilização através de um exemplo:

Crie uma nova aplicação e grave os arquivos com os nomes:

- ✓ `Pasta – Toolbar`
- ✓ `Unit1 – UFrmToolBar`
- ✓ `Project – Toolbar`

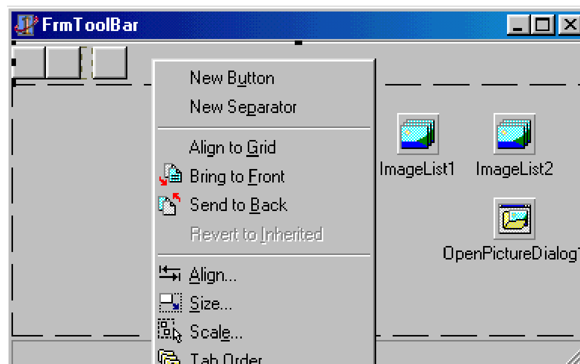
Os componentes inseridos serão:



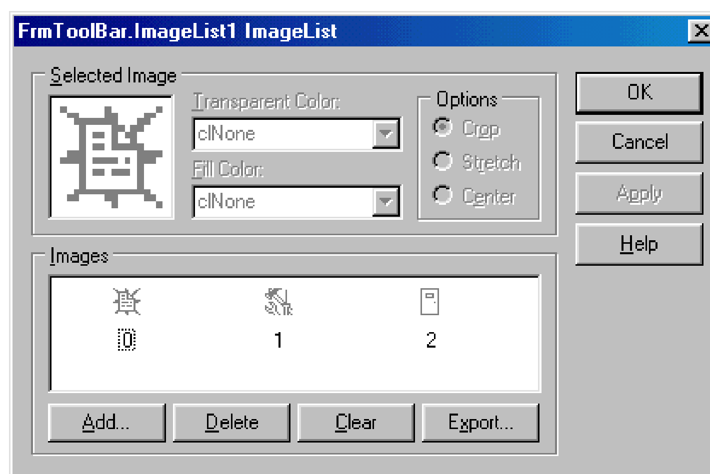
O componente Toolbar alinha-se automaticamente na parte superior do formulário da mesma maneira que a barra de Status alinha-se na parte inferior.
O componente Image pode ter a propriedade `Align` definida como `alClient` e a propriedade `Center` como `True`.

Para inserir botões na barra Toolbar, clique com o botão direito na barra e escolha New Button, para inserir um separador de botões escolha New Separator.

Insira 3 botões e separadores como no exemplo a seguir.



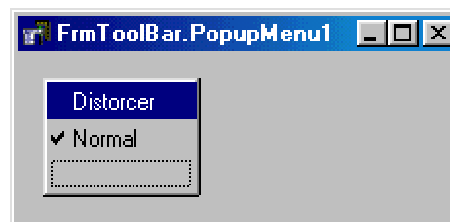
Para o objeto ImageList1 defina três figuras 'cinzas' clicando em Add... como abaixo:
As figuras podem ser – FileOpen.bmp, Tools.bmp, Doorshut.bmp – Encontradas no diretório padrão[20] de figuras do Delphi. Para excluir uma figura da lista, clique em Delete.



Para o objeto ImageList2 defina as mesmas figuras, porém coloridas.

Para o objeto PopUpMenu1 defina dois menus através do construtor:

Caption	*Name*
Distorcer	MnpDistorcer
Normal	MnpNormal



[20] C:\Arquivos de Programas\Arquivos Comuns\Borland Shared\Images\Buttons

Para o objeto ToolBar defina as propriedades:

```
object ToolBar1: TToolBar
  ButtonHeight = 25
  ButtonWidth = 24
  Flat = True
  HotImages = ImageList2
  Images = ImageList1
end;
```

`Images` define as figuras que serão atribuídas aos botões na visualização inicial do componente. `HotImages` define as figuras que serão atribuídas aos botões no momento em que o mouse 'passa' nestes componentes caso a propriedade `Flat` esteja ligada, gerando um efeito semelhante à barra de ícones do Internet Explorer.

O segundo botão da Toolbar (ToolButton2) deve ter as suas propriedade

```
object ToolButton2: TToolButton
  DropdownMenu = PopupMenu1
  Style = tbsDropDown
end
```

Defina a propriedade Hint para os três botões segundo o raciocínio:

Objeto	***Finalidade***	***Hint***
ToolButton1	Abrir um arquivo de imagem	Abre arquivo de imagem
ToolButton2	Distorcer ou não a figura	Permite distorcer a imagem
ToolButton3	Fechar o programa	Sair do aplicativo

Para a barra de status defina:

```
object StatusBar1: TStatusBar
  AutoHint = True
  SimplePanel = True
end
```

Defina os manipuladores de eventos necessários segundo a unit abaixo:

```
unit UFrmToolbar;
{A interface não foi impressa}
implementation

{$R *.DFM}

procedure TFrmToolBar.ToolButton1Click(Sender: TObject);
begin
  if OpenPictureDialog1.Execute then
    Image1.Picture.LoadFromFile(OpenPictureDialog1.FileName);
end;
```

```
procedure TFrmToolBar.ToolButton4Click(Sender: TObject);
begin
  FrmToolBar.Close;
end;

procedure TFrmToolBar.MnpDistorcerClick(Sender: TObject);
begin
  Image1.Stretch := True;
  MnpDistorcer.Checked := True;
  MnpNormal.Checked := False;
end;

procedure TFrmToolBar.MnpNormalClick(Sender: TObject);
begin
  Image1.Stretch := False;
  MnpNormal.Checked := True;
  MnpDistorcer.Checked := False;
end;

end.
```

APLICAÇÕES MDI

Uma interface MDI (Multiple Document Interface) possibilita a visualização de mais de um documento ou objeto simultaneamente.
Aplicações MDI são constituidas geralmente de dois ou mais formulários: o formulário principal (MDIform) que geralmente constitui a janela principal do programa e seus filhos (MDIchild) localizados dentro de sua área de trabalho.

Em aplicações MDI, você geralmente precisa mostrar múltiplas instâncias de um classe de formulário. Um formulário filho de MDI deve ser criado para ser exibido e deve ser *destruído* para não mais exibí-lo.

Com uma nova aplicação vamos iniciar uma aplicação MDI que possa abrir vários arquivos de figuras selecionados pelo usuário.

- ✓ Salve o projeto em uma *pasta* chamada ImagemMDI

Nome atual	***Novo nome***
Unit1	UFrmPai
Project1	ImagemMDI

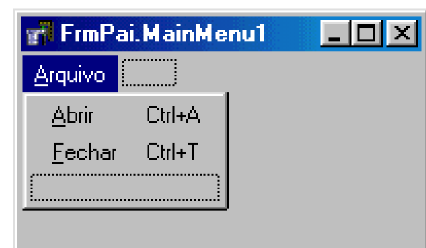
Altere as propriedades do form:

```
object FrmPai: TFrmPai
  Width = 447
  Height = 309
  Caption = 'FrmPai'
  FormStyle = fsMDIForm
  Position = poScreenCenter
end
```

Insira no formulário um componente OpenPictureDialog e um componente MainMenu

Contrua os seguintes Menus através do construtor do MainMenu:

Caption	*Name*	*ShortCut*
&Arquivo	MnuArquivo	
&Abrir	MnuAbrir	Ctrl+A
&Sair	MnuSair	Ctrl+T



Troque a propriedade Name do componente MainMenu para *MnuPai*

- ✓ Salve todo o projeto. (Atualizar)

Insira mais um formulário e salve-o:

Nome atual	*Novo nome*
Unit1	UFrmFilho

Modifique a seguinte propriedade:

```
object FrmFilho: TFrmFilho
  FormStyle = fsMDIChild
end
```

Insira um componente Image no formulário FrmFilho e modifique as seguintes propriedades:

```
  object Image1: TImage
    Align = alClient
    Center = True
  end
```

Salve todo o projeto. (Atualizar)

Execute a aplicação e visualize o formulário FrmFilho que já se encontra dentro de FrmPai. Note que formulários filhos de MDI não tem a opção de existir (ter sido criado) e não ser exibido (o que é possível para formulários comuns).

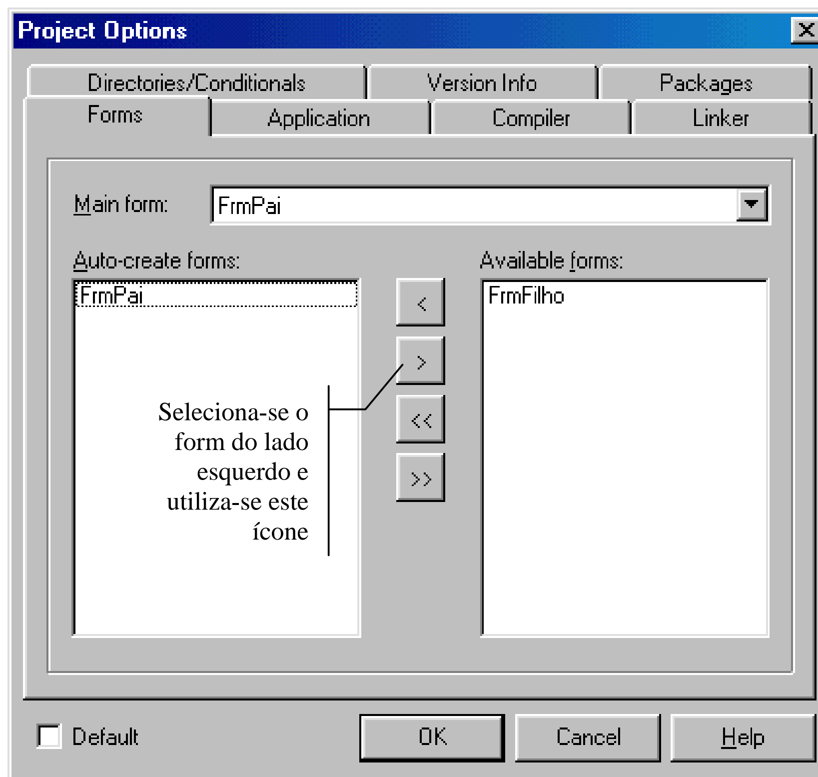
O *default* do ambiente Delphi é criar automaticamente uma instância de cada *form* do projeto. É por isto que existem as referências de criação no código dentro do arquivo de projeto (DPR). **Menu Project | View Source**

```
begin
  Application.Initialize;
  Application.CreateForm(TFrmPai, FrmPai);
  Application.CreateForm(TFrmFilho, FrmFilho);
  Application.Run;
end.
```

Como desejamos que a aplicação rode inicialmente sem nenhuma janela filha aberta, devemos alterar este comportamento. Para que o formulário MDIFilho não seja criado automaticamente com o projeto, devemos especificar no projeto que o mesmo não é "*auto-create*".
Para isso basta usar a opção de menu **Project | Options** e alterar para FrmFilho em *Avaliable forms*. Como na figura a seguir.

Desta maneira, o Form estará disponível, porém não será criado automaticamente pelo programa.

Após a confirmação, podemos notar que o código no arquivo de projeto (DPR) também foi alterado.



A maneira de criarmos o form FrmImagem deverá ser feita em tempo de execução através de código fazendo uma chamada ao método CreateForm do objeto Application.

Exemplo:

```
begin
  comando1;
  comando2;
  Application.CreateForm(TFrmFilho, FrmFilho);
end;
```

Desta maneira, defina o seguinte código para objeto MnuAbrir

```
procedure TFrmPai.MnuAbrirClick(Sender: TObject);
begin
  if OpenPictureDialog1.Execute then//abre a caixa de dialogo
    begin
      Application.CreateForm(TFrmFilho, FrmFilho);
        //carrega a figura para o componente ImgImagem
      FrmFilho.Image1.Picture.LoadFromFile(OpenPictureDialog1.FileName);
      FrmFilho.Caption := OpenPictureDialog1.FileName;//muda o caption
     end;
end;
```

Defina o seguinte código para MnuSair

```
procedure TFrmMain.MnuSairClick(Sender: TObject);
begin
  FrmPai.Close;
end;
```

Execute a aplicação e tente fechar as janelas filhas. É possível?

Ao contrário de outros tipos de forms, um form MDIChild não fecha automaticamente quando executamos seu método Close.
Entretanto no evento *OnClose* de qualquer *form* temos um parâmetro **Action** que nos permite definir uma ação especifica:

Action	***Comportamento***
CaNone	O form não fechará – nenhuma ação será tomada.
CaHide	O form é apenas "escondido", não sendo mais exibido – porém ainda existirá em memória. Este é o comportamento *default* de forms comuns.
caFree	O form será *destruído* – e não simplesmente escondido.
caMinimize	O form será minimizado, e não fechado. Este é o *default* para forms filhos de MDI.

Para forçar um MDIChild a fechar devemos alterar o parâmetro Action para caFree em seu evento OnClose.

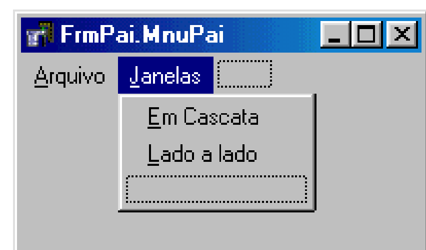
Vamos acrescentar o seguinte código no evento *OnClose* do form FrmImagem

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

(12) Vamos adicionar dois itens de menu ao MnuPai como no exemplo:

Items:

Caption	***Name***	***ShortCut***
&Janelas	MnuJanelas	
&Em Cascata	MnuEmCascata	
&Lado a Lado	MnuLadoaLado	



No evento *OnClick* do item MnuEmCascata definir o código:

```
procedure TFrmPai.MnuEmCascataClick(Sender: TObject);
begin
  Cascade;
end;
```

No evento *OnClick* do item MnuLadoaLado definir:

```
procedure TFrmPai.MnuLadoaLadoClick(Sender: TObject);
begin
  Tile;
end;
```

Criando um lista das janelas abertas

Altere a propriedade *WindowMenu* do form FrmPai para MnuJanelas.
Com isto faremos que uma lista das janelas filhas abertas fique disponível sob o Menu Janelas do formulário FrmPai.

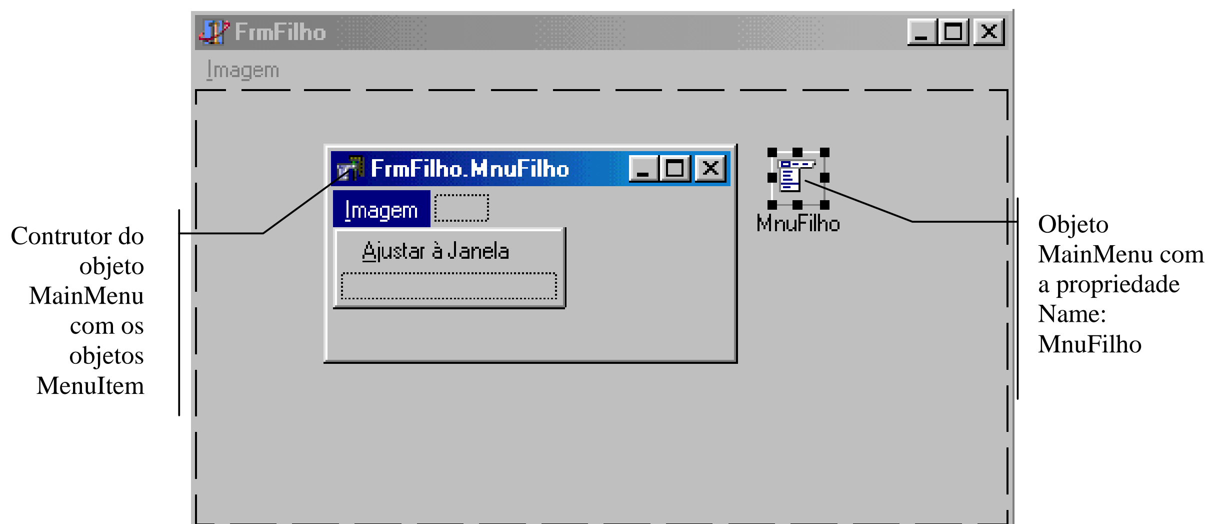
Mesclando Menus

Outro recurso interessante de aplicações MDI é a possibilidade de mesclar os menus da janela filha (FrmFilho) com a janela pai (FrmPai).

Vamos adicionar um objeto MainMenu no formulário FrmFilho, chamando-o de MnuFilho.

Items:

Caption	*Name*
&Imagem	MnuImagem
&Ajustar a janela	MnuAjustar



Para o evento *OnClick* de MnuAjustar definir:

```
procedure TFrmFilho.MnuAjustarClick(Sender: TObject);
begin
    //Checked mostra um 'v' (marcador) para controle lig/des.
  MnuAjustar.Checked := not MnuAjustar.Checked;
    //Permite distorcer a imagem para ocupar todo o espaço
  Image1.Stretch := not Image1.Stretch;
end;
```

Porém se executarmos a aplicação vamos notar que o Menu da janela filha sobrepôs o Menu da janela pai.

A posição do Menu das janelas filhas em relação aos da janela MDI é determinada pela propriedade GroupIndex de cada um.

Se desejamos que o Menu MnuFilho apareça *entre* os menus MnuArquivo e MnuJanela (do formulário FrmPai), devemos atribuir-lhes valores para a propriedade GroupIndex conforme abaixo:

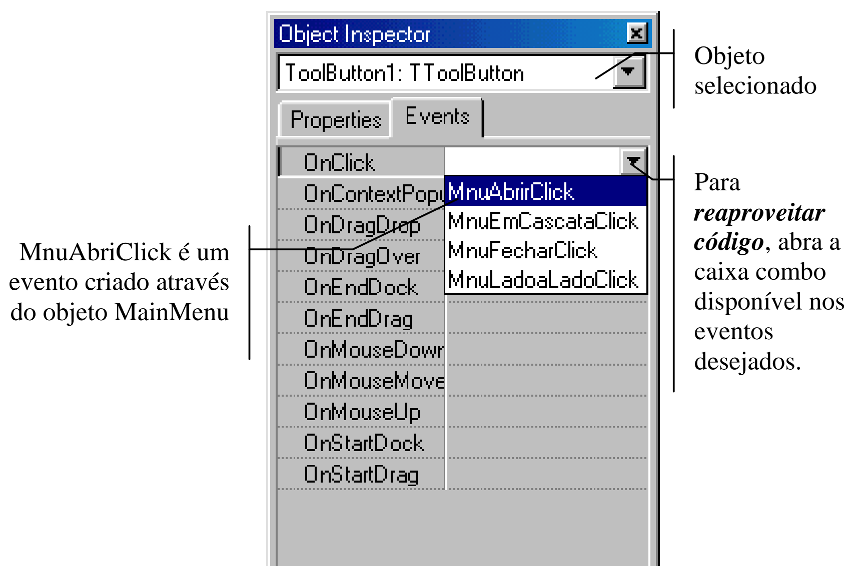
Menu	***GroupIndex***
MnuArquivo	0
MnuImagem	1
MnuJanelas	2

Reaproveitamento de código

Pode-se inserir uma ToolBar no formulário FrmPai para facilitar ainda mais a manipulação do aplicativo pelo usuário.

Se na ToolBar os botões terão os mesmos comandos dos items de Menu (Abri, Fechar), uma sugestão é que na ToolBar seja reaproveitado o código já escrito nos items de menu da seguinte maneira:

Crie a ToolBar, os botões desejados e selecione um dos botões. Na Object Inspector selecione o evento OnClick daquele botão e ***não*** dê um duplo clique, abra a caixa combo do evento OnClick e escolha um procedimento já pronto:



EXERCÍCIOS

EXERCÍCIO 1

Objetivo: Abordar a maneira com que o Delphi trabalha com os objetos e elementos básicos em Object Pascal. Introduzir a técnica de manipulação de propriedades e eventos.

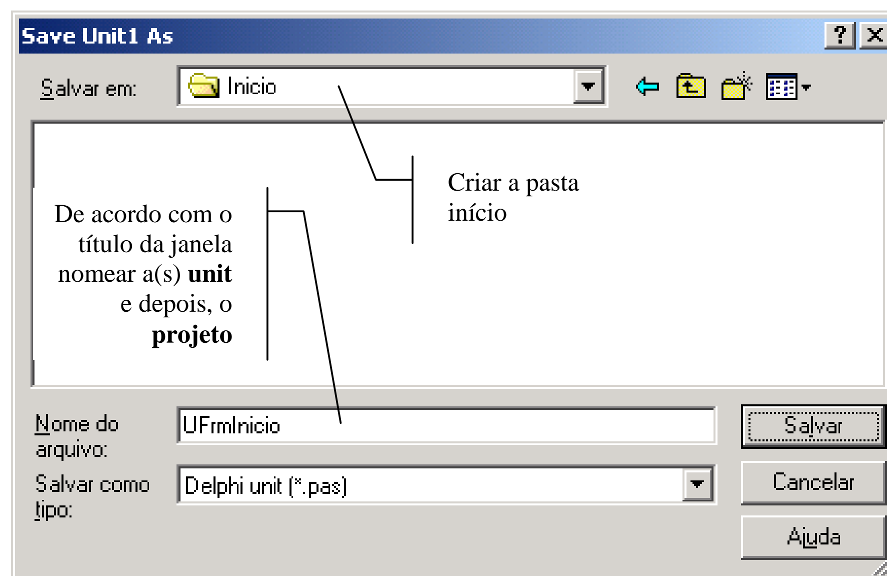
Componentes utilizados: Label, Edit e Button.

Enfoque: Quando o usuário digitar uma string, este texto deverá ser exibidono título formulário e em uma mensagem de caixa de diálogo.

Abra o Delphi ou se já aberto, clique no comando **File – New Application**. Perceba que inicialmente já temos um formulário a nossa disposição, seu nome (através da propriedade Name) é Form1.

- ✓ Salve o projeto antes de prosseguir criando uma *pasta* chamada Exercicio1 e dentro desta pasta, gravar a unidade e o arquivo de projeto.

Nome atual	*Novo nome*
Unit1	UFrmInicio
Project1	Inicio



Após ter gravado, insira os seguintes componentes no form: 1 Label, 1 Edit e 2 Buttons. Uma sugestão para a disposição dos objetos pode ser vista a seguir:

Modifique as propriedades dos componentes como a seguir:

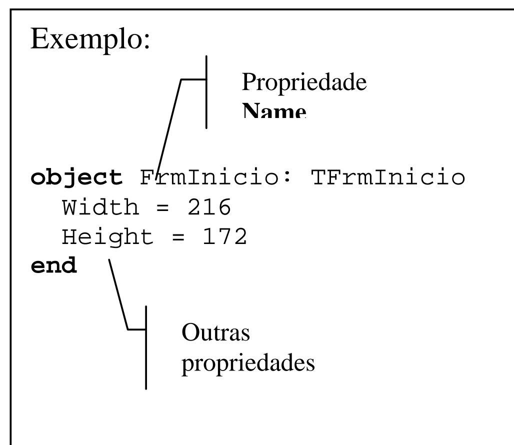
```
object FrmInicio: TFrmInicio
  Width = 216
  Height = 172
  ActiveControl = Edit1
  Caption = FrmInicio
  Position = poScreenCenter
  Name = FrmInicio
end

object Label1: TLabel
  Caption = &Digite:
  FocusControl = Edit1
end

object Edit1: TEdit
  MaxLength = 10
end

object Button1: TButton
  Cursor = crHandPoint
  Caption = &Confirma
  Default = True
  Hint = Clique aqui para executar a tarefa
  ShowHint = True
end

object Button2: TButton
  Cursor = crHandPoint
  Caption = &Sair
  Cancel = True
end
```



Para definirmos manipuladores para os objetos Buttons, faça o seguinte passo:

Selecione o Button1 (Caption – Confirma) e na janela Object Inspector selecione a *guia* Events. Identifique o evento: *OnClick* dando um duplo clique *no espaço em branco*.

Um procedimento no Editor de Código deverá ser exibido.

Insira o código entre o **`begin`** e o **`end`**. Como no exemplo abaixo:

```
procedure TFrmInicio.Button1Click(Sender: TObject);
begin
  {O título do formulário recebe uma frase concatenada
   com uma string digitada pelo usuário}
  FrmInicio.Caption := 'Projeto Início '+Edit1.Text;
  {Uma frase será exibida através de uma caixa de diálogo padrão}
  ShowMessage('Você digitou: '+Edit1.Text);
  Edit1.Clear; //O método Clear limpa a propriedade Text do objeto
end;
```

Para o Button2 repita o mesmo processo e insira o código:

```
procedure TFrmInicio.Button2Click(Sender: TObject);
begin
  FrmInicio.Close;
end;
```

Salve seu projeto novamente para ***atualizá-lo*** e através do comando **RUN** compile-o e faça os testes necessários.

EXERCÍCIO 2

Objetivo: Abordar a maneira com que o Delphi trabalha com os objetos e elementos básicos em Object Pascal. Revisar componentes vistos em sala e apresentar uma técnica de atribuição de código entre componentes bem como sua manipulação.

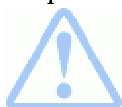
Componentes utilizados: Label, Edit, Button, CheckBox e RadioGroup.

Enfoque: O usuário deverá digitar uma string no componente Edit, este texto deverá ser exibido no Label(2). O CheckBox deverá habilitar ou não a digitação e o RadioGroup vai manipular o tamanho da fonte do componente Label.

Iniciar uma nova aplicação e salvá-la em uma pasta chamada **Exercicio2**.

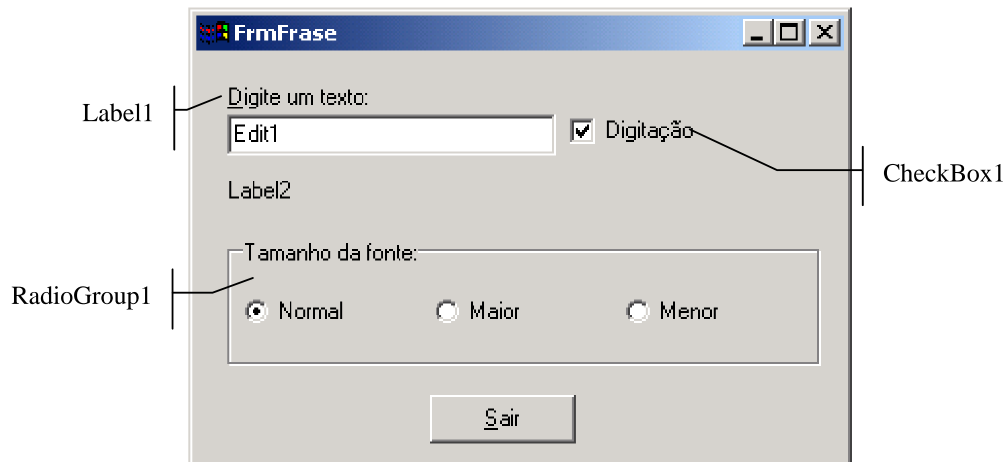
Nome atual	*Novo nome*
Unit1	UFrmFrase
Project1	Frase

Troque o nome do objeto form para FrmFrase.



Em nosso curso será uma ***regra***: O nome do *Form* será o nome da *Unit* sem a letra U.

Insira os objetos como o exemplo a seguir:



Os objetos podem ter as seguintes propriedades alteradas:

```
  object FrmFrase: TFrmFrase
    ActiveControl = Edit1
    BorderStyle = bsDialog
    Caption = Formulário Exercício 2
    Width = 341
    Height = 236
    Name = FrmFrase
..end

  object Label1: TLabel
    Caption = &Digite um texto:
    FocusControl = Edit1
  end

  object Label2: TLabel
    Width = 305
    Height = 25
    AutoSize = False
    Caption = {deixe vazio}
  end

  object Edit1: TEdit
    Text = {deixe vazio}
  end

  object CheckBox1: TCheckBox
    Hint = Permite habilitar o componente de entrada de dados
    Caption = Digitação
    Checked = True
    ShowHint = True
  end
```

```
  object RadioGroup1: TRadioGroup
    Caption = Tamanho da fonte:
    Columns = 3
    ItemIndex = 0
    Items.Strings =
      Normal
      Maior
      Menor
  end

  object Button1: TButton
    Hint = Clique para fechar o programa
    Cancel = True
    Caption = &Sair
    ShowHint = True
  end
```

Para definir os manipuladores de eventos, identifique os procedimentos abaixo:

NÃO digite a linha da **procedure**, apenas o código dentro do **begin** e **end**.

```
procedure TFrmFrase.CheckBox1Click(Sender: TObject);
begin
  Edit1.Enabled := CheckBox1.Checked;
end;

procedure TFrmFrase.Edit1Change(Sender: TObject);
begin
  Label2.Caption := Edit1.Text;
end;

procedure TFrmFrase.RadioGroup1Click(Sender: TObject);
begin
  if RadioGroup1.ItemIndex = 0 then
    Label2.Font.Size := 8
  else if  RadioGroup1.ItemIndex = 1 then
    Label2.Font.Size := 12
  else
    Label2.Font.Size := 6;
end;

procedure TFrmFrase.Button1Click(Sender: TObject);
begin
  FrmFrase.Close;
end;
```

Salve seu projeto novamente para ***atualizá-lo*** e através do comando **RUN** compile-o e faça os testes necessários.

EXERCÍCIO 3

Utilizando os conceitos vistos, crie um aplicativo que possa efetuar as quatro operações aritméticas básicas, somar, subtrair, dividir e multiplicar. Os dados serão números reais que o usuário deve digitar.
A quantidade de objetos, a disposição visual e a escolha dos manipuladores serão de escolha do desenvolvedor.

Objetivo: Abordar a maneira com que o Delphi trabalha com diversas funções de conversão e operações aritméticas internas.

Componentes utilizados: À escolha do desenvolvedor

Enfoque: O usuário deverá digitar dois valores e após escolher um operador aritmético realizar o cálculo.

Através de uma nova aplicação, grave o projeto dentro de uma pasta chamada Exercicio5

Nome atual	*Novo nome*
Unit1	UFrmMiniCalc
Project1	MiniCalc

EXERCÍCIO 4

Objetivo: Abordar a maneira com que o Delphi trabalha com objetos tendo um enfoque maior no código em Object Pascal utilizando recurso de *re*aproveitamento de código.

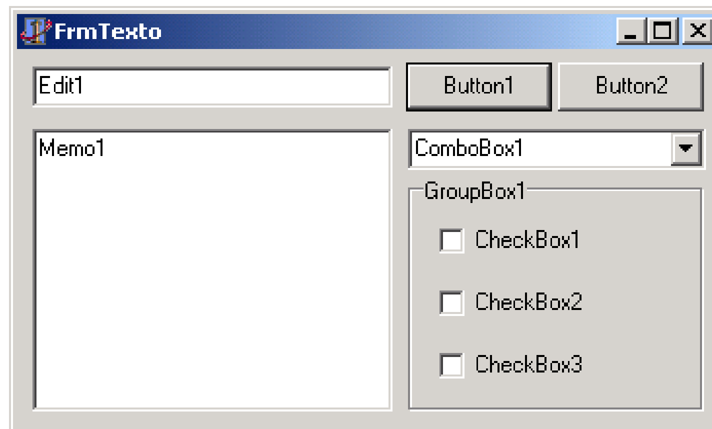
Componentes utilizados: Edit, Button, CheckBox, RadioGroup, ComboBox e Memo.

Enfoque: O usuário deverá digitar um caminho e arquivo válido no Edit, este texto deverá ser utilizado para carregar o arquivo (texto) no Memorando. Pode-se trocar a fonte pelo ComboBox e mudar o formato através do CheckBox.

Através de uma nova aplicação, grave o projeto dentro de uma pasta chamada Exercicio3

Nome atual	*Novo nome*
Unit1	UFrmTexto
Project1	Texto

Insira os componentes e alinhe-os.



Modifique as propriedades conforme a orientação abaixo:

```
object FrmTexto: TFrmTexto
  ActiveControl = Edit1
  BorderStyle = bsSingle
  Caption = Visualiza Texto ASCII
  Position = poScreenCenter
  Name = FrmTexto
end

object Edit1: TEdit
  Hint = Digite um caminho e nome de arquivo
  ShowHint = True
  Text = {deixe vazio}
end

object Memo1: TMemo
  Lines.Strings = {deixe vazio}
  Cursor = crNo
  ReadOnly = True
end

object ComboBox1: TComboBox
  Text = {deixe vazio}
end

object GroupBox1: TGroupBox
  Caption = Formato da fonte
end

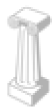
object CheckBox1: TCheckBox
  Caption = Negrito
end

object CheckBox2: TCheckBox
  Caption = Itálico
end
```

```
object CheckBox3: TCheckBox
  Caption = Sublinhado
end

object Button1: TButton
  Caption = &Carrega Arq.
  Default = True
end

object Button2: TButton
  Cancel = True
  Caption = &Sair
end
```



Ao declarar[21] o cabeçalho da procedure `FormataTexto` na seção `Interface`, pode-se utilizar na mesma linha, o atalho CTRL+SHIFT+C para declará-la na seção `Implementation`.

Defina os códigos para cada evento segundo o código fonte abaixo:

```
unit UFrmTexto;
interface
uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  StdCtrls, ExtCtrls;
                {Não digite as declarações da interface}
type
  TFrmTexto = class(TForm)
    Edit1: TEdit;
    Memo1: TMemo;
    ComboBox1: TComboBox;
    GroupBox1: TGroupBox;
    CheckBox1: TCheckBox;
    CheckBox2: TCheckBox;
    CheckBox3: TCheckBox;
    Button1: TButton;
    Button2: TButton;
    procedure Button1Click(Sender: TObject);
    procedure ComboBox1Change(Sender: TObject);
    procedure FormShow(Sender: TObject);
    procedure CheckBox1Click(Sender: TObject);
    procedure CheckBox2Click(Sender: TObject);
    procedure CheckBox3Click(Sender: TObject);
    procedure Button2Click(Sender: TObject);
    procedure FormCloseQuery(Sender: TObject; var CanClose: Boolean);
  private
    { Private declarations }
    procedure FormatoTexto;
  public
    { Public declarations }
  end;
```

[21] Somente devemos declarar procedures ou funções 'manuais'. As demais funções que possuem Sender como parâmetro são declaradas automaticamente pelo Delphi, através da Object Inspector ou DuploClick no objeto (Evento Padrão). A ordem *de construção* dos eventos não faz a menor diferença

```
var
  FrmTexto: TFrmTexto;

implementation

{$R *.DFM}

procedure TFrmTexto.Button1Click(Sender: TObject);
begin
  if Edit1.Text <> '' then
    Memo1.Lines.LoadFromFile(Edit1.Text)
  else
    ShowMessage('Digite um caminho e'+#13+
                'um nome de arquivo válido'+#13+
                'Exemplo: C:\autoexec.bat');
end;


procedure TFrmTexto.ComboBox1Change(Sender: TObject);
begin
  Memo1.Font.Name := ComboBox1.Text;
end;

{Para criar o procedimento FormatoTexto, digite sua declaração
(sem TFrmTexto) na cláusula private e utilize CTRL+SHIFT+C
Para todos os demais, selecione o objeto e utilize a object inspector}
procedure TFrmTexto.FormatoTexto;
begin
  if (CheckBox1.Checked and CheckBox2.Checked and CheckBox3.Checked) then
    Memo1.Font.Style := [fsBold, fsItalic, fsUnderline]
  else if (CheckBox1.Checked and CheckBox2.Checked) then
    Memo1.Font.Style := [fsBold, fsItalic]
  else if (CheckBox1.Checked and CheckBox3.Checked) then
    Memo1.Font.Style := [fsBold, fsUnderline]
  else if (CheckBox2.Checked and CheckBox3.Checked) then
    Memo1.Font.Style := [fsItalic, fsUnderline]
  else if (CheckBox1.Checked) then
    Memo1.Font.Style := [fsBold]
  else if (CheckBox2.Checked) then
    Memo1.Font.Style := [fsItalic]
  else if (CheckBox3.Checked) then
    Memo1.Font.Style := [fsUnderline]
  else Memo1.Font.Style := [];
end;

procedure TFrmTexto.FormShow(Sender: TObject);
begin
  ComboBox1.Items.AddStrings(Screen.Fonts);
  ComboBox1.Text := ComboBox1.Items.Strings[0];
  Memo1.Font.Name := ComboBox1.Text;
end;

procedure TFrmTexto.CheckBox1Click(Sender: TObject);
begin
  FrmTexto.FormatoTexto;
end;
```

```
procedure TFrmTexto.CheckBox2Click(Sender: TObject);
begin
  FrmTexto.FormatoTexto;
end;

procedure TFrmTexto.CheckBox3Click(Sender: TObject);
begin
  FrmTexto.FormatoTexto;
end;

procedure TFrmTexto.Button2Click(Sender: TObject);
begin
  FrmTexto.Close;
end;

procedure TFrmTexto.FormCloseQuery(Sender: TObject; var CanClose: Boolean);
begin
  if MessageDlg('Deseja realmente sair?',mtConfirmation,[mbYes,mbNo],0) =
mrYes then
    CanClose := True
  else CanClose := False;
end;
end.
```

Salve seu projeto novamente para ***atualizá-lo*** e através do comando **RUN** compile-o e faça os testes necessários.

EXERCÍCIO 5

Objetivo: Abordar a maneira com que o Delphi trabalha com objetos tendo um enfoque maior na diversidade de componentes.

Componentes utilizados: PageControl, BitBtn, Memo, Image, MainMenu, PopupMenu, OpenDialog, OpenPictureDialog, SpeedButton.

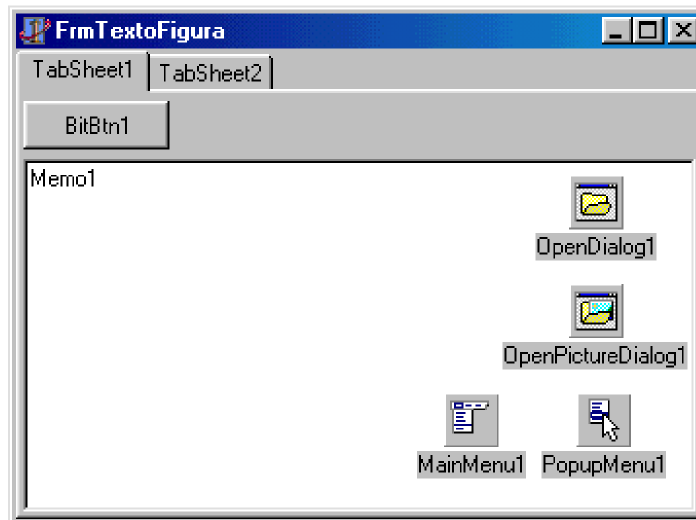
Enfoque: O usuário deverá escolher entre carregar um texto ou uma figura dependendo da guia de trabalho.

Através de uma nova aplicação, grave o projeto dentro de uma pasta chamada Exercicio4

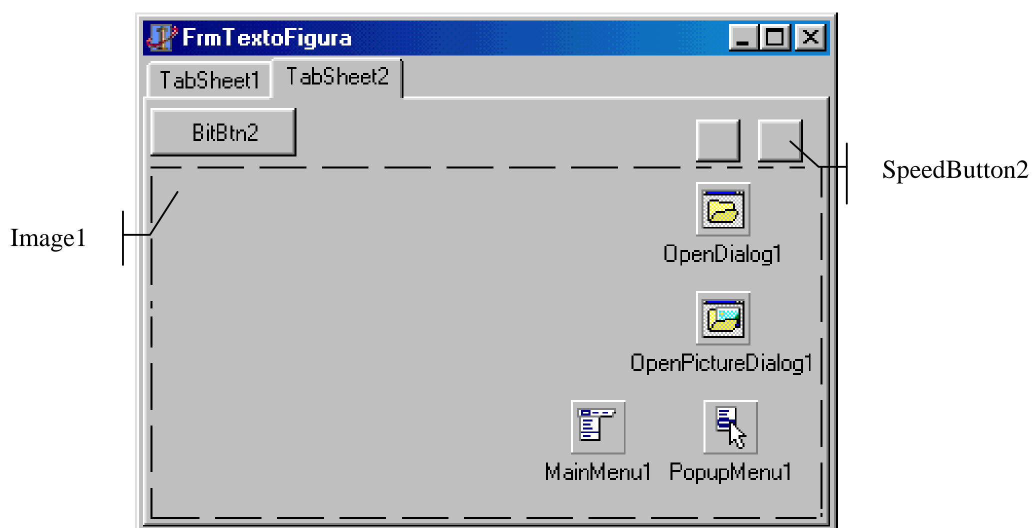
Nome atual	***Novo nome***
Unit1	UFrmTextoFigura
Project1	TextoFigura

- ✓ A propriedade Name do form deverá ser FrmTextoFigura

Insira os seguintes objetos na guia TabSheet1.

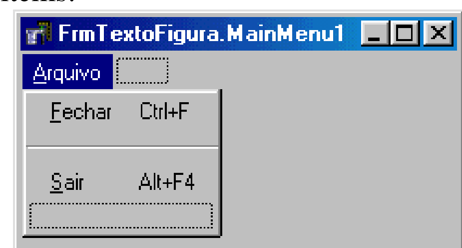


Insira os seguintes objetos na guia TabSheet2. ***Não*** insira os componentes invisíveis novamente.



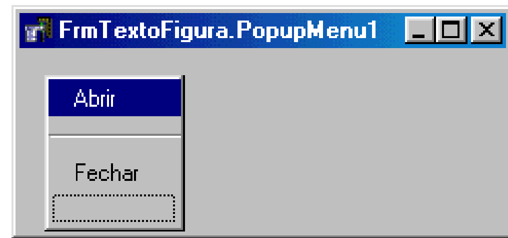
Construa o menu principal (MainMenu) com os seguintes items:

Caption	*Name*	*ShortCut*
&Arquivo	MnuArquivo	
&Fechar	MnuFechar	Ctrl+F
&Sair	MnuSair	Alt+F4



Contrua o menu rápido (PopupMenu) com os seguintes items:

Caption	*Name*
Abrir	MnpAbrir
Fechar	MnpFechar

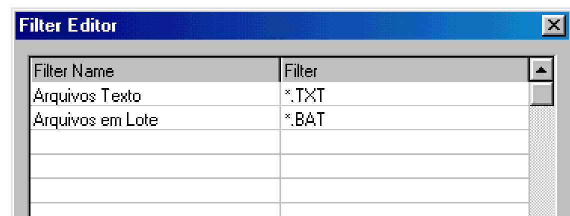


Selecione os dois componentes SpeedButton na TabSheet2 e defina a propriedade GroupIndex de ambos com o valor 1

Selecione o componente Memo1 e defina a propriedade PopupMenu com o valor PopupMenu1

Selecione o componente Image1 e defina a propriedade PopupMenu com o valor PopupMenu1

Selecione o componente OpenDialog1 e defina a propriedade Filter como na ilustração. As demais propriedades estão definidas como:




```
object OpenDialog1: TOpenDialog
  Filter =
  InitialDir = 'c:\'
  Title = 'Abrir arquivo texto'
end
```

Defina captions e figuras para os objetos BitBtn.

Defina figuras para os SpeedButtons.

Os demais detalhes de ‘acabamento’ são de escolha do desenvolvedor.

Agora, identifique os eventos abaixo e construa os códigos indicados, fazendo os comentários que julgar necessário.

```
implementation

{$R *.DFM}

procedure TFrmTextoFigura.BitBtn1Click(Sender: TObject);
begin
  if OpenDialog1.Execute then
    begin
      Memo1.Lines.LoadFromFile(OpenDialog1.FileName);
      TabSheet1.Caption := ExtractFileName(OpenDialog1.FileName);
    end;
end;
```

```
procedure TFrmTextoFigura.BitBtn2Click(Sender: TObject);
begin
  if OpenPictureDialog1.Execute then
    begin
      Image1.Picture.LoadFromFile(OpenPictureDialog1.FileName);
      TabSheet2.Caption := ExtractFileName(OpenPictureDialog1.FileName);
      SpeedButton1.Down := True;
      Image1.Stretch := False;
    end;
end;

procedure TFrmTextoFigura.SpeedButton1Click(Sender: TObject);
begin
  Image1.Stretch := False;
end;

procedure TFrmTextoFigura.SpeedButton2Click(Sender: TObject);
begin
  Image1.Stretch := True;
end;

procedure TFrmTextoFigura.MnuFecharClick(Sender: TObject);
begin
  if PageControl1.ActivePage = PageControl1.Pages[0] then
    Memo1.Lines.Clear
  else Image1.Picture := nil;
end;

procedure TFrmTextoFigura.MnpAbrirClick(Sender: TObject);
begin
  if PageControl1.ActivePage = PageControl1.Pages[0] then
    BitBtn1.Click
  else BitBtn2.Click;
end;

procedure TFrmTextoFigura.MnpFecharClick(Sender: TObject);
begin
  {Pode-se reaproveitar código através da object inspector tambem}
  MnuFechar.Click;
end;

procedure TFrmTextoFigura.MnuSairClick(Sender: TObject);
begin
  FrmTextoFigura.Close;

end;

procedure TFrmTextoFigura.FormCloseQuery(Sender: TObject;
  var CanClose: Boolean);
begin
  if MessageDlg('Deseja sair?',mtConfirmation,[mbYes,mbNo],0) = mrYes then
    CanClose := True
  else
    CanClose := False;
end;
```

EXERCÍCIO 6

Objetivo: Abordar a maneira com que o Delphi trabalha com objetos revisando conceitos já abordados.

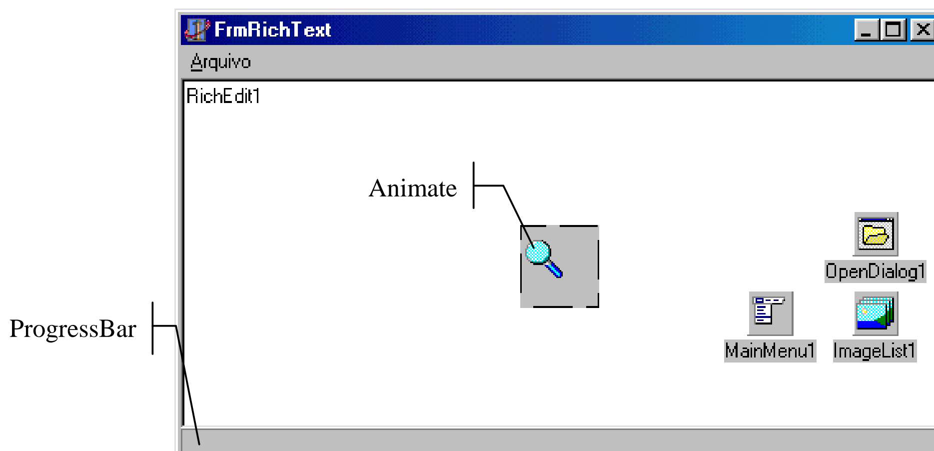
Componentes utilizados: MainMenu, OpenDialog, RichEdit, Animate, ProgressBar.

Enfoque: O usuário deverá escolher um arquivo .RTF e ao carregar o arquivo, o processo vai atualizar a ProgressBar de acordo com o número de caracteres.

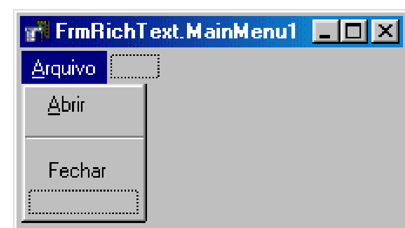
Através de uma nova aplicação, grave o projeto dentro de uma pasta chamada Exercicio6

Nome atual	*Novo nome*
Unit1	UFrmRichText
Project1	RichText

Insira os componentes e organize-os. A sugestão pode ser vista a seguir:



Insira duas figuras na ImageList através de um duplo clique. As figuras devem representar as funções (Abrir e Fechar)



No objeto MainMenu crie os seguintes items:

Caption	*Name*
Abrir	MnuAbrir
Fechar	MnuFechar

Para o objeto Animate, configure as seguintes propriedades:

```
object Animate1: TAnimate
  Active = False
  CommonAVI = aviFindFile
  Visible = False
end
```

Para inserir figuras nos items de Menu, selecione o MainMenu (1 click) e na propriedade Images defina o objeto ImageList correspondente. Depois, dê um duplo clique no MainMenu. Nos items de menu (dentro do construtor) defina na propriedade ImageIndex a figura desejada.

O aplicativo tem dois eventos basicamente. Implemente-os como a seguir.

```
implementation

{$R *.DFM}

procedure TFrmRichText.MnuAbrirClick(Sender: TObject);
var i:integer;
begin
  if OpenDialog1.Execute then
  begin
    RichEdit1.Visible := False;
    RichEdit1.Lines.LoadFromFile(OpenDialog1.FileName);
    ProgressBar1.Max := Length(RichEdit1.Lines.Text);
    Animate1.Visible := True;
    Animate1.Active := True;
    for i:=0 to Length(RichEdit1.Lines.Text) do
      ProgressBar1.Position := i;
    Animate1.Active := False;
    Animate1.Visible := False;
    ProgressBar1.Position := 0;
    RichEdit1.Visible := True;
    FrmRichText.Caption := ('O total de caracteres é: '+
                           IntToStr(Length(RichEdit1.Lines.Text)));
  end;
end;

procedure TFrmRichText.MnuFecharClick(Sender: TObject);
begin
  FrmRichText.Close;
end;
```

EXERCÍCIO 7

Objetivo: Abordar a maneira com que o Delphi trabalha com objetos revisando conceitos já abordados.

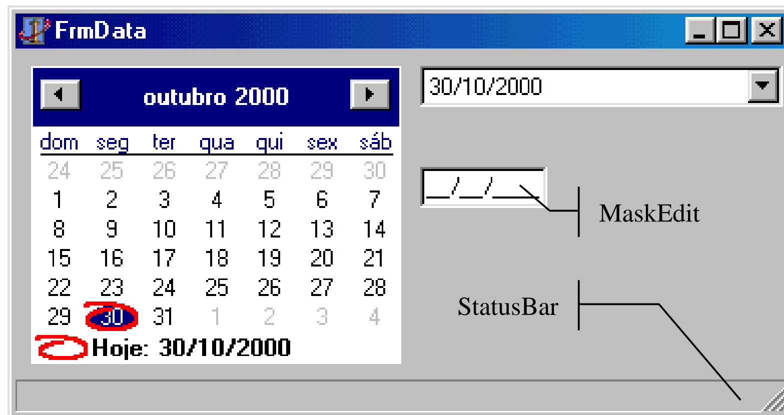
Componentes utilizados: MonthCalendar, DateTimePicker, MaskEdit, StatusBar.

Enfoque: O usuário deverá escolher uma data em qualquer um dos dois componentes, poderá também digitá-la no MaskEdit. A data será exibida na Barra de Status por extenso através da função FormatDateTime.

Através de uma nova aplicação, grave o projeto dentro de uma pasta chamada Exercicio7

Nome atual	*Novo nome*
Unit1	UFrmData
Project1	Data

Insira os componentes e identifique os eventos em seguida.




```
implementation

{$R *.DFM}

procedure TFrmData.MonthCalendar1Click(Sender: TObject);
begin
  StatusBar1.SimpleText :=
  FormatDateTime('dddd, dd "de" mmmm "de" yyyy',MonthCalendar1.Date);
  DateTimePicker1.Date := MonthCalendar1.Date;
  MaskEdit1.Text := DateToStr(MonthCalendar1.Date);
end;

procedure TFrmData.DateTimePicker1Change(Sender: TObject);
begin
  StatusBar1.SimpleText :=
  FormatDateTime('dddd, dd "de" mmmm "de" yyyy',DateTimePicker1.Date);
  MonthCalendar1.Date := DateTimePicker1.Date;
  MaskEdit1.Text := DateToStr(DateTimePicker1.Date);
end;

procedure TFrmData.MaskEdit1KeyDown(Sender: TObject; var Key: Word;
  Shift: TShiftState);
begin
  if Key = VK_RETURN then {Se o usuário teclar ENTER, então}
    begin
      StatusBar1.SimpleText :=
  FormatDateTime('dddd, dd "de" mmmm "de" yyyy',StrToDate(MaskEdit1.Text));
      MonthCalendar1.Date := StrToDate(MaskEdit1.Text);
      DateTimePicker1.Date := StrToDate(MaskEdit1.Text);
    end;
end;
```

EXERCÍCIO 8

Abra o exercício 5 e trate os possíveis erros através de tratamento local (try-except).

Abra o exercício 7 e trate os possíveis erros através do tratamento global (objeto ApplicationEvents).

EXERCÍCIO 9

Objetivo: Abordar a maneira com que o Delphi trabalha com objetos revisando conceitos já abordados.

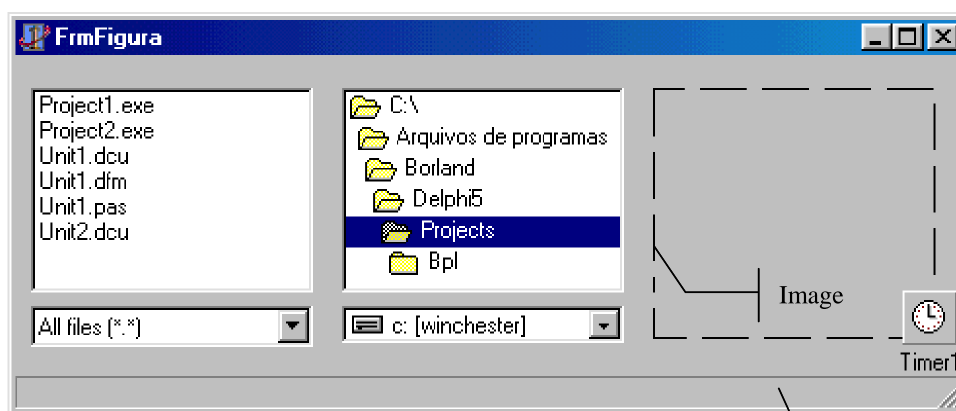
Componentes utilizados: StatusBar, FilterComboBox, FileListBox, DirectoryListBox, DriveComboBox, Image, Timer.

Enfoque: O usuário deverá escolher um arquivo BMP na caixa de listagem, sendo este exibido no componente Image. As horas serão exibidas em uma parte da StatusBar.

Através de uma nova aplicação, grave o projeto dentro de uma pasta chamada Exercicio9

Nome atual	*Novo nome*
Unit1	UFrmFigura
Project1	Figura

Insira os componentes e modifique suas propriedades em seguida.




```
object FilterComboBox1: TFilterComboBox
  Hint = 'Escolha um tipo de arquivo'
  FileList = FileListBox1
  Filter = 'Bitmap|*.BMP | Jpeg|*.JPG | Ícones|*.ICO'
end
```

```
  object DriveComboBox1: TDriveComboBox
    Hint = Escolha um drive
    DirList = DirectoryListBox1
  end

  object DirectoryListBox1: TDirectoryListBox
    Hint = Escolha uma pasta para visualizar mais arquivos
    FileList = FileListBox1
  end

  object FileListBox1: TFileListBox
    Cursor = crHandPoint
    Hint = Clique para visualizar um arquivo
  end

  object StatusBar1: TStatusBar
    AutoHint = True
    SimplePanel = False
  end
```

Para o objeto StatusBar, crie dois painéis na propriedade Panels. O primeiro deverá ter a largura (width) de 350.

Identifique os eventos e implemente o código a seguir:

```
implementation

{$R *.DFM}

procedure TFrmFigura.FormShow(Sender: TObject);
begin
  FileListBox1.Mask := FilterComboBox1.Mask;
end;

procedure TFrmFigura.FileListBox1Change(Sender: TObject);
begin
  if FileListBox1.FileName <> '' then
    Image1.Picture.LoadFromFile(FileListBox1.FileName);
end;

procedure TFrmFigura.Timer1Timer(Sender: TObject);
begin
  StatusBar1.Panels[1].Text := DateTimeToStr(Now);
end;
```

EXERCÍCIO 10

Objetivo: Manipular objetos com enfoque maior em operações internas.

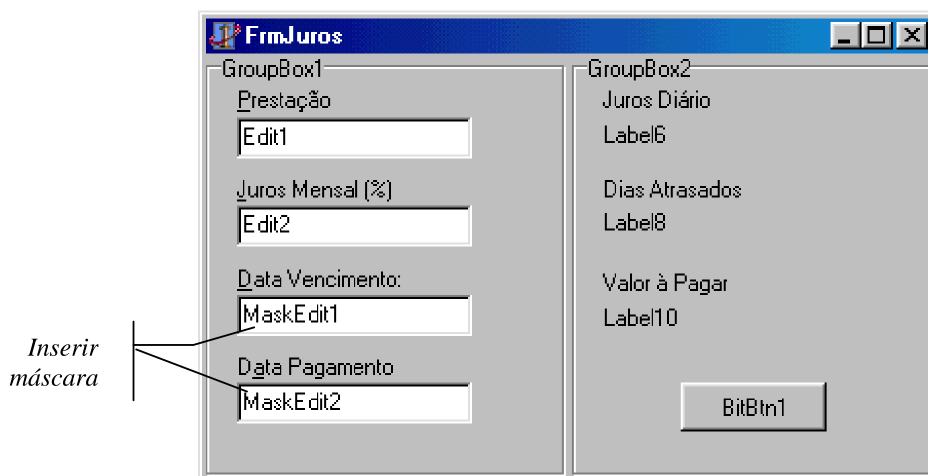
Componentes utilizados: GroupBox, Label, Edit, MaskEdit, BitBtn.

Enfoque: O usuário deverá preencher os dados básicos e ao clicar no botão, a aplicação deve calcular os items identificados do lado esquerdo. Eventuais exceções devem ser tratatadas para evitar 'mensagens técnicas' ou falta de orientação ao usuário.

Através de uma nova aplicação, grave o projeto dentro de uma pasta chamada Exercicio10

Nome atual	***Novo nome***
Unit1	UFrmJuros
Project1	Juros

Insira os seguintes componentes:



Pode-se trocar a sugestão do objeto MaskEdit para um DateTimePicker.

Crie agora uma rotina para calcular:

- Juros diários: É o valor dos juros mensal dividido por 30.
- Dias atrasados: É a diferença entre as datas de pagamento e vencimento.
- Valor à pagar: Prestação + (juros diário/100 * dias atrasados * prestação)

Identificar se o usuário pagou adiantado, se pagou, não haverá descontos, o valor a pagar será a prestação. Eventuais exceções devem ser tratatadas para evitar 'mensagens técnicas' ou falta de orientação ao usuário.

EXERCÍCIO 11

Objetivo: Manipular objetos com enfoque maior em operações internas.

Componentes utilizados: À escolha do desenvolvedor.

Enfoque: O usuário deverá preencher os dados básicos (altura atual e peso). Ao clicar no botão, a aplicação deve calcular o peso ideal informando quanto a pessoa deve engordar, emagrecer ou manter o peso.

Através de uma nova aplicação, grave o projeto dentro de uma pasta chamada Exercicio11

Nome atual	***Novo nome***
Unit1	UFrmPeso
Project1	Peso

Supondo uma regra, a implementação será a seguinte: `Peso_ideal = Altura_atual – 1,05`

O usuário deverá entrar com dois dados: Altura atual e peso atual. O programa deve calcular o peso ideal e exibir sua resposta de acordo com a diferença entre o Peso atual e o Peso ideal.

EXERCÍCIO 12

Objetivo: Manipular objetos com enfoque maior em operações internas.

Componentes utilizados: GroupBox, ListBox, Edit, MaskEdit, BitBtn, Label.

Enfoque: O usuário deverá preencher os dados básicos. Ao clicar no botão, a aplicação deve calcular o salário 'bruto'.

Através de uma nova aplicação, grave o projeto dentro de uma pasta chamada Exercicio12

Nome atual	***Novo nome***
Unit1	UFrmSalario
Project1	Salario

Orientações gerais:

- Permitir a entrada dos seguintes dados (GroupBox Dados Básicos): O nome do funcionário, o salário base, a data de admissão, o número de dependentes e selecionar o cargo do funcionário.

- Permitir a entrada dos seguintes dados (GroupBox Base de Cálculo): O valor em Real pago por cada dependente, o valor em porcentagem do Tempo de serviço do funcionário.

- Exibir os seguintes dados (GroupBox Valores Calculados): O cálculo dos dependentes do funcionário (Nº de dependentes * Valor em Real pago por dependente), o cálculo do tempo de serviço (Quantidade de anos na empresa * Valor em porcentagem do Tempo de serviço do funcionário em relação ao salário base), o cálculo do cargo (de acordo com o cargo selecionado, o valor em porcentagem do cargo em relação ao salário base).

Exemplo e sugestão visual a seguir:

DICAS

CRIAR UM HOTLINK

Podemos criar um link para a Internet dentro do Delphi através da função ShellExecute. Supondo que a propriedade Caption de um label seja um endereço web válido (http://www.fulanodetal.com.br) podemos inserir em seu evento OnClick o seguinte código:

```
ShellExecute(Handle, 'open', pchar(Label1.Caption), nil, '',SW_SHOWNORMAL);
```

É necessário incluir a biblioteca `ShellAPI` na cláusula uses da Inteface.

ENVIAR UM MAIL

Podemos utilizar a função ShellExecute para abrir o programa de mail do usuário e preparar seu envio.

```
ShellExecute(Handle, 'open', 'mailto:fulano@provedor.com.br', nil,
'',SW_SHOWNORMAL);
```

Caso queira colocar um subject padrão na construção do e-mail utilize a palavra subject da seguinte forma:

```
ShellExecute(Handle, 'open', 'mailto:fulano@provedor.com.br
?subject=Pergunta', nil, '',SW_SHOWNORMAL);
```

É necessário incluir a biblioteca `ShellAPI` na cláusula uses da Inteface.

EXECUTANDO UM PROGRAMA DE ACORDO COM A EXTENSÃO DE UM ARQUIVO

ShellExecute é um função API comparada a um 'canivete suiço', vamos utilizá-la novamente para chamar um programa através da extensão do arquivo passado como parâmetro.
Os parâmetros de ShellExecute são 6:

- Um manipulador de janela, que é o pai do processo disparado.
- Uma ação a executar, que pode ser "open", para abrir, "print", para imprimir, ou "explore", para abrir uma janela do Explorer no diretório desejado.
- Um ponteiro (tipo Pchar em Delphi) para o nome do programa ou diretório.
- Parâmetros para a aplicação.
- Diretório inicial.
- Modo de abertura do programa

Exemplo:

```
ShellExecute(Handle, 'open', 'exemplo.doc', nil, nil, SW_SHOWNORMAL);
```

É necessário incluir a biblioteca `ShellAPI` na cláusula uses da Inteface.

COMO SABER QUAL BIBLIOTECA INSERIR NA USES?

Quando o Delphi não declarar automaticamente determinada classa na sessão uses, um erro deverá ocorrer no processo de compilação. O nome do parâmetro ou função deverá estar em evidencia no rodapé do *Code Editor* em uma janela chamada *Message View*.
Clique em **Search | Browse Symbol** e digite a palavra procurada, a unit onde ela se encontra deverá aparecer. Digite este nome na cláusula uses.
Ou, pode-se para o cursor (piscando) na palavra e teclar F1, o help também exibe a unit.

DATAS

As datas são um tipo de dados TDateTime, internamente são armazenadas como números de ponto flutuante (Double). A parte inteira indica a data, em número de dias, contados a partir de 30/12/1899 e a fracionária, as horas. Pelo fato de serem armazenadas como Double, pode-se subtrair, somar e estabelecer comparações. Por exemplo:

```
var
  DataIni, DataFim : TDateTime;
  Dif : Single;
begin
  Dif := DataFim – DataIni; //diferença em dias entre as datas
  DataIni := DataIni+10; //acrescenta + 10 dias na data
end;
```

Para saber a hora de determinada data, multiplica-se a parte fracionária por 24:

```
  Frac(DataIni) * 24;
```

Podemos ainda comparar as datas com os operadores ‘<’ e ‘>’. Caso não seja necessário a comparação com a hora, basta converter a data para um inteiro e efetuar a comparação:

```
  if Trunc(Data1) < Trunc(Data2) then
```

A formatação de data pode ser feita através da função FormatDateTime.

```
function FormatDateTime(const Format: string; DateTime: TDateTime): string;
```

Exemplo:

```
Label1.Caption := FormatDateTime('dddd, d "de" mmmm "de" yyyy', date);
```

SAIR DE UMA APLICAÇÃO

O Delphi fornece diferentes métodos para sair de uma aplicação. O *ideal* é utilizar o método `Close`. Porém, há casos em que `Close` não atenderia à necessidade do desenvolvedor, nestes casos pode-se chamar `Halt` ou `Application.Terminate`.
`Halt` é um procedimento do Pascal e provoca o término da aplicação, não se importando com arquivos abertos. Já o método `Terminate` do objeto `Application` obriga o fechamento de todos os arquivos abertos e a destruição de todas a forms criadas.

REDUZINDO TESTES IF

A expressão:

```
if A = 10 then
  B := True
else
  B := False;
```

Pode ser reduzida a apenas uma linha:

```
  B := A = 10;
```

A = 10 é uma operação boleana, que retorna True se A é igual a 10 e False caso contrário. Neste caso estamos atribuindo à variável B o resultado da comparação, TRUE ou FALSE.

HINTS COM DUAS OU MAIS LINHAS

A object inspector permite Hint com apenas uma linha. Para gerar Hints com mais linhas pode-se em tempo de execução configurar a propriedade Hint da seguinte forma:

```
Edit1.Hint := 'Primeira linha do hint'#13'segunda linha do hint';
```

Pode-se também em tempo de projeto clicar na form com o botão direito e escolher a opção 'View as Text', procurar o objeto e a hint desejada e alterá-la:

```
Hint := 'Primeira linha do hint'#13'segunda linha do hint';
```

Para voltar ao Form clique com o botão direito e escolha 'View as Form'.

SUBSTITUIR A TECLA TAB POR ENTER NA MUDANÇA DE FOCO

- Setar a propriedade KeyPreview da Form para True
- Setar a propriedade Default de todos os botões da Form para False
- Criar um evento OnKeyPress para a Form como este:

```
procedure TForm1.FormKeyPress(Sender: TObject; var Key: Char);
begin
  if (Key = #13) and not (Activecontrol is TMemo) then begin
//desabilita o processamento posterior da tecla
   key := #0;
//simula o envio da tecla tab
   Perform(WM_NEXTDLGCTL,0,0);
  end;
end;
```

ÍCONES

O Delphi possui um conjunto de ícones e imagens em uma pasta padrão:

```
C:\Arquivos de programas\Arquivos comuns\Borland Shared\Images
```

EXECUTAR UM PROGRAMA

Para executar um programa use a API WinExec.

```
WinExec('calc.exe', SW_SHOWNORMAL);
```

LINKS

Alguns links interessantes:

Nacionais
http://www.clubedelphi.com.br/
http://www.marcosferreira.eti.br/oraculo.htm
http://www.delphibahia.eti.br/
http://www.drdelphi.com.br
http://www.inprise.com.br/

Internacionais
http://www.marcocantu.com/
http://www.drbob42.com/home.htm
http://www.delphi3000.com/
http://homepages.borland.com/torry/
http://delphi.about.com/compute/delphi/
http://delphi.icm.edu.pl/

VIRTUAL KEYS

Para manipular teclas diversas, pode-se utilizar os eventos `OnKeyDown` ou `OnKeyUp` testando o valor do parâmetro `Key` que representa a tecla pressionada.
Para gerar um evento global, configure a propriedade `KeyPreview` do Form para True e no manipulador de eventos (do form), crie o seguinte código:

```
procedure TForm1.FormKeyDown(Sender: TObject; var Key: Word;
  Shift: TShiftState);
begin
  case Key of
    VK_F2: ShowMessage('F2');
    VK_F3: ShowMessage('F3');
  end;
end;
```

Obviamente, no exemplo acima deve-se trocar ShowMessage pelo comando desejado. Os valores possíveis para teclas virtuais estão listadas abaixo.
Estas constantes estão na unit Windows.

```
  { Virtual Keys, Standard Set }
  {$EXTERNALSYM VK_LBUTTON}
  VK_LBUTTON = 1;
  {$EXTERNALSYM VK_RBUTTON}
  VK_RBUTTON = 2;
  {$EXTERNALSYM VK_CANCEL}
  VK_CANCEL = 3;
  {$EXTERNALSYM VK_MBUTTON}
  VK_MBUTTON = 4;  { NOT contiguous
with L & RBUTTON }
  {$EXTERNALSYM VK_BACK}
  VK_BACK = 8;
  {$EXTERNALSYM VK_TAB}
  VK_TAB = 9;
  {$EXTERNALSYM VK_CLEAR}
  VK_CLEAR = 12;
  {$EXTERNALSYM VK_RETURN}
  VK_RETURN = 13;
  {$EXTERNALSYM VK_SHIFT}
  VK_SHIFT = $10;
  {$EXTERNALSYM VK_CONTROL}
  VK_CONTROL = 17;
  {$EXTERNALSYM VK_MENU}
  VK_MENU = 18;
  {$EXTERNALSYM VK_PAUSE}
  VK_PAUSE = 19;
  {$EXTERNALSYM VK_CAPITAL}
  VK_CAPITAL = 20;
  {$EXTERNALSYM VK_KANA }
  VK_KANA = 21;
  {$EXTERNALSYM VK_HANGUL }
  VK_HANGUL = 21;
  {$EXTERNALSYM VK_JUNJA }
  VK_JUNJA = 23;
  {$EXTERNALSYM VK_FINAL }
  VK_FINAL = 24;
  {$EXTERNALSYM VK_HANJA }
  VK_HANJA = 25;
  {$EXTERNALSYM VK_KANJI }
  VK_KANJI = 25;
  {$EXTERNALSYM VK_CONVERT }
  VK_CONVERT = 28;
  {$EXTERNALSYM VK_NONCONVERT }
  VK_NONCONVERT = 29;
  {$EXTERNALSYM VK_ACCEPT }
  VK_ACCEPT = 30;
  {$EXTERNALSYM VK_MODECHANGE }
  VK_MODECHANGE = 31;
  {$EXTERNALSYM VK_ESCAPE}
  VK_ESCAPE = 27;
  {$EXTERNALSYM VK_SPACE}
  VK_SPACE = $20;
  {$EXTERNALSYM VK_PRIOR}
  VK_PRIOR = 33;
  {$EXTERNALSYM VK_NEXT}
  VK_NEXT = 34;
  {$EXTERNALSYM VK_END}
  VK_END = 35;
  {$EXTERNALSYM VK_HOME}
  VK_HOME = 36;
  {$EXTERNALSYM VK_LEFT}
  VK_LEFT = 37;
  {$EXTERNALSYM VK_UP}
  VK_UP = 38;
  {$EXTERNALSYM VK_RIGHT}
  VK_RIGHT = 39;
  {$EXTERNALSYM VK_DOWN}
  VK_DOWN = 40;
  {$EXTERNALSYM VK_SELECT}
  VK_SELECT = 41;
  {$EXTERNALSYM VK_PRINT}
  VK_PRINT = 42;
  {$EXTERNALSYM VK_EXECUTE}
  VK_EXECUTE = 43;
  {$EXTERNALSYM VK_SNAPSHOT}
  VK_SNAPSHOT = 44;
  {$EXTERNALSYM VK_INSERT}
  VK_INSERT = 45;
  {$EXTERNALSYM VK_DELETE}
  VK_DELETE = 46;
  {$EXTERNALSYM VK_HELP}
  VK_HELP = 47;
{ VK_0 thru VK_9 are the same as
ASCII '0' thru '9' ($30 - $39) }
{ VK_A thru VK_Z are the same as
ASCII 'A' thru 'Z' ($41 - $5A) }
  {$EXTERNALSYM VK_LWIN}
  VK_LWIN = 91;
  {$EXTERNALSYM VK_RWIN}
  VK_RWIN = 92;
  {$EXTERNALSYM VK_APPS}
  VK_APPS = 93;
  {$EXTERNALSYM VK_NUMPAD0}
  VK_NUMPAD0 = 96;
  {$EXTERNALSYM VK_NUMPAD1}
  VK_NUMPAD1 = 97;
  {$EXTERNALSYM VK_NUMPAD2}
  VK_NUMPAD2 = 98;
  {$EXTERNALSYM VK_NUMPAD3}
  VK_NUMPAD3 = 99;
  {$EXTERNALSYM VK_NUMPAD4}
  VK_NUMPAD4 = 100;
  {$EXTERNALSYM VK_NUMPAD5}
  VK_NUMPAD5 = 101;
```

```
{$EXTERNALSYM VK_NUMPAD6}
VK_NUMPAD6 = 102;
{$EXTERNALSYM VK_NUMPAD7}
VK_NUMPAD7 = 103;
{$EXTERNALSYM VK_NUMPAD8}
VK_NUMPAD8 = 104;
{$EXTERNALSYM VK_NUMPAD9}
VK_NUMPAD9 = 105;
{$EXTERNALSYM VK_MULTIPLY}
VK_MULTIPLY = 106;
{$EXTERNALSYM VK_ADD}
VK_ADD = 107;
{$EXTERNALSYM VK_SEPARATOR}
VK_SEPARATOR = 108;
{$EXTERNALSYM VK_SUBTRACT}
VK_SUBTRACT = 109;
{$EXTERNALSYM VK_DECIMAL}
VK_DECIMAL = 110;
{$EXTERNALSYM VK_DIVIDE}
VK_DIVIDE = 111;
{$EXTERNALSYM VK_F1}
VK_F1 = 112;
{$EXTERNALSYM VK_F2}
VK_F2 = 113;
{$EXTERNALSYM VK_F3}
VK_F3 = 114;
{$EXTERNALSYM VK_F4}
VK_F4 = 115;
{$EXTERNALSYM VK_F5}
VK_F5 = 116;
{$EXTERNALSYM VK_F6}
VK_F6 = 117;
{$EXTERNALSYM VK_F7}
VK_F7 = 118;
{$EXTERNALSYM VK_F8}
VK_F8 = 119;
{$EXTERNALSYM VK_F9}
VK_F9 = 120;
{$EXTERNALSYM VK_F10}
VK_F10 = 121;
{$EXTERNALSYM VK_F11}
VK_F11 = 122;
{$EXTERNALSYM VK_F12}
VK_F12 = 123;
{$EXTERNALSYM VK_F13}
VK_F13 = 124;
{$EXTERNALSYM VK_F14}
VK_F14 = 125;
{$EXTERNALSYM VK_F15}
VK_F15 = 126;
{$EXTERNALSYM VK_F16}
VK_F16 = 127;
{$EXTERNALSYM VK_F17}
VK_F17 = 128;
{$EXTERNALSYM VK_F18}
VK_F18 = 129;
  {$EXTERNALSYM VK_F19}
  VK_F19 = 130;
  {$EXTERNALSYM VK_F20}
  VK_F20 = 131;
  {$EXTERNALSYM VK_F21}
  VK_F21 = 132;
  {$EXTERNALSYM VK_F22}
  VK_F22 = 133;
  {$EXTERNALSYM VK_F23}
  VK_F23 = 134;
  {$EXTERNALSYM VK_F24}
  VK_F24 = 135;
  {$EXTERNALSYM VK_NUMLOCK}
  VK_NUMLOCK = 144;
  {$EXTERNALSYM VK_SCROLL}
  VK_SCROLL = 145;
{ VK_L & VK_R - left and right Alt,
Ctrl and Shift virtual keys.
  Used only as parameters to
GetAsyncKeyState() and
GetKeyState().
  No other API or message will
distinguish left and right keys in
this way. }
  {$EXTERNALSYM VK_LSHIFT}
  VK_LSHIFT = 160;
  {$EXTERNALSYM VK_RSHIFT}
  VK_RSHIFT = 161;
  {$EXTERNALSYM VK_LCONTROL}
  VK_LCONTROL = 162;
  {$EXTERNALSYM VK_RCONTROL}
  VK_RCONTROL = 163;
  {$EXTERNALSYM VK_LMENU}
  VK_LMENU = 164;
  {$EXTERNALSYM VK_RMENU}
  VK_RMENU = 165;
  {$EXTERNALSYM VK_PROCESSKEY}
  VK_PROCESSKEY = 229;
  {$EXTERNALSYM VK_ATTN}
  VK_ATTN = 246;
  {$EXTERNALSYM VK_CRSEL}
  VK_CRSEL = 247;
  {$EXTERNALSYM VK_EXSEL}
  VK_EXSEL = 248;
  {$EXTERNALSYM VK_EREOF}
  VK_EREOF = 249;
  {$EXTERNALSYM VK_PLAY}
  VK_PLAY = 250;
  {$EXTERNALSYM VK_ZOOM}
  VK_ZOOM = 251;
  {$EXTERNALSYM VK_NONAME}
  VK_NONAME = 252;
  {$EXTERNALSYM VK_PA1}
  VK_PA1 = 253;
  {$EXTERNALSYM VK_OEM_CLEAR}
  VK_OEM_CLEAR = 254;
```

CONFIRMAÇÃO DE GRAVAÇÃO DE REGISTROS EM PARADOX

O Paradox utiliza um buffer para a gravação dos registros, descarregando-os depois no disco. Para forçar a gravação diretamente utilize a API DbiSaveChanges

```
Table1.Post;
DbiSaveChanges(Table1.Handle);
```

É necessário incluir as bibliotecas `dbiTypes, dbiProcs` na cláusula uses da Inteface.

DISPARANDO SONS DO SISTEMA (MULTIMÍDIA)

Para utilizar os sons do sistema, utilize a função PlaySound.

```
PlaySound(pChar('SYSTEMSTART'),0,SND_ASYNC);
```

Outros sons:

SYSTEMSTART
SYSTEMEXIT
SYSTEMHAND
SYSTEMASTERISK
SYSTEMQUESTION
SYSTEMEXCLAMATION
SYSTEMWELCOME
SYSTEMDEFAULT

É necessário incluir as biblioteca `mmsystem` na cláusula uses da Inteface

LIMITE EM TABELAS PARADOX

Uma tabela em Paradox pode ter problemas acima de aproximadamente 750.000 registros. Com relação ao espaço em disco a tabela pode chegar à 131 megas. Acima disto, os problemas devem se agravar.

Fonte: http://www.undu.com

GARANTINDO UMA INSTÂNCIA DO APLICATIVO

Para que seu aplicativo não seja executado várias vezes, utilize o seguinte código no arquivo de projeto (DPR). A função FindWindow procura se há outra tela da classe passada como parâmetro, se houver, traz para frente e não cria a aplicação novamente.

```
program Project1;

uses
  Forms,Windows,
  Unit1 in 'Unit1.pas' {Form1};

{$R *.RES}

begin
if FindWindow('TForm1','Form1') <> 0 then
  SetForegroundWindow(FindWindow('TForm1','Form1'))
else
  begin
  Application.Initialize;
  Application.CreateForm(TForm1, Form1);
  Application.Run;
  end;
end.
```

É necessário incluir a biblioteca `windows` na cláusula uses do arquivo DPR.

PERMITIR APENAS NÚMEROS EM UM TEDIT

```
procedure TForm1.Edit1KeyPress(Sender: TObject; var Key: Char);
begin
  if ((Key in ['0'..'9'] = False) and (Word(Key) <> VK_BACK)) then
    Key := #0;
end;
```

Se alguém cuida saber alguma coisa, ainda não sabe como convém saber.
Ap. Paulo

APÊNDICE

INTRODUÇÃO À ORIENTAÇÃO A OBJETOS

Mais do que popular, a orientação a objetos é uma metodologia que ganhou espaço no mercado, e ao que tudo indica, veio para ficar.

Vamos abordar alguns conceitos importantes para o inicio do estudo da ferramenta Delphi, seria importante que na sua vida profissional você possa aprofundar-se mais nestes e outros conceitos que pelo enfoque, não serão abordados. Uma referencia bibliográfica no final da apostila irá ajuda-lo(a) a encontrar mais informações a respeito.

Independente da ferramenta (Delphi, C++Builder, PowerBuilder, entre outras), a orientação a objetos é uma teoria da década de 70 que veio ser implementada com grande destaque nas ferramentas visuais. Essa técnica é baseada no conceito de *classes*.

Classe é um 'tipo abstrado de dados', ou seja, um novo tipo criado pelo 'desenvolvedor'[22] que pode conter *métodos e propriedades*. É como ***compararmos*** um tipo *classe* com um tipo *record*. Com o tipo classe é possível implementar um conjunto de recursos OO[23], sendo um dos principais chamado *herança*, que é uma poderosa maneira de reaproveitamento de código.

Considerando simplesmente a títulode ***exemplo*** vamos criar a classe base `TSerVivo`
com as seguintes **características** e tipos de dados:

- *Tamanho : Single*
- *Cor : String*
- *Forma(Obeso, Magro) : String*

e que pode possuir as seguintes **ações**:

- Alimentar
- Respirar

Um exemplo de herança seria criar as classes `TSerHumano` e `TAnimais` como descendente de `TSerVivo`, desta forma, a classe descendente herda todas as características e ações criadas na classe ascendente podendo acrescentar particularidades à sua estrutura, exemplo:

[22] Na verdade, vamos trabalhar com classes prontas definidas pelo 'Delphi'.
[23] Orientação à Objeto

TSerHumano	TAnimais
• *Tamanho : Single* • *Cor : String* • *Forma : String* • Vestimenta : String • CorCabelo: String • Sexo : Char ➢ *Alimentar* ➢ *Respirar* ➢ Falar ➢ Caminhar	• *Tamanho : Single* • *Cor : String* • *Forma : String* • NºPatas : Integer • Cauda : Boolean ➢ *Alimentar* ➢ *Respirar* ➢ EmitirSom

Os *objetos* são variáveis do tipo *classe,* onde as *características* são consideradas *propriedades*, e as *rotinas* **chamadas pelos objetos**[24] são denominadas ***métodos***.

As **ações** disparadas pelo usuário ou pelo sistema no componente serão chamados ***eventos***. Eles são de extrema importância pois definem ***quando*** um conjunto de comandos será executado.

Ainda exemplificando, podemos utilizar as *propriedades* atribuindo valores de acordo com o tipo de dados definido (não estamos considerando a sintaxe *completa* em Object Pascal) e utilizar os *métodos* apenas chamando o nome da rotina.

```
    Edileumar,Jorgina   : TSerHumano;
    Cachorro            : TAnimais;

begin
  Edileumar.Tamanho := 1.70;
  Edileumar.Forma := 'Magro';
  Edileumar.Sexo := 'M';
  Jorgina.Tamanho := 1.60;
  Jorgina.Sexo := 'F';
  Jorgina.CorCabelo := 'Castanhos';
  Edileumar.Falar;
  Jorgina.Caminhar;
  Cachorro.Tamanho := 0.60;
  Cachorro.Cauda := True;
  Cachorro.Alimentar;
end;
```

Observe que as *propriedades* recebem valores e os *métodos* executam rotinas (procedimentos ou funções) **sobre o objeto** que o invocou.

Alguns métodos podem receber parâmetros.

[24] Ações disparadas pelo usuário ou sistema serão chamados de *eventos.*

O que acontece no Delphi é que existe uma gigantesca estrutura de classes onde `TObject` é a classe base das demais classes. Apesar de não ser necessário manipular este recurso diretamente, é importante saber seu funcionamento básico.

Você verá as declarações de classe dentro do Delphi como abaixo, um formulário com dois componentes (objetos) button e um componente edit será declarado assim:

```
type
  TForm1 = class(TForm)
    Button1: TButton;
    Button2: TButton;
    Edit1: TEdit;
  private
    { Private declarations }
  public
    { Public declarations }
  end;
```

Essa introdução vem apenas ilustrar os principais conceitos: Propriedades, eventos e métodos, sendo amplamente utilizados durante o curso.

A *herança* é um recurso importante implementado na OO, não sendo o único, obviamente. Polimorfismo, encapsulamento entre outros, são conceitos que a principio você não precisa saber para manipular a ferramenta (Delphi), mas em um futuro próximo será muito importante.

REFERÊNCIA BIBLIOGRÁFICA

Dominando o Delphi 5 “A Bíblia” – Marco Cantù
Makron Books, 2000

Programação Orientada a Objetos usando o Delphi 3 – Faiçal Farhat de Carvalho
Editora Érika

Delphi4 Curso Completo – Marcelo Leão
Axcel Books do Brasil

Delphi4 Senac – Adilson Resende
Editora SENAC Belo Horizonte

365 Dicas de Delphi – Bruno Sonnino
Makron Books