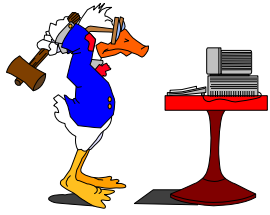
Apostila cedida por :

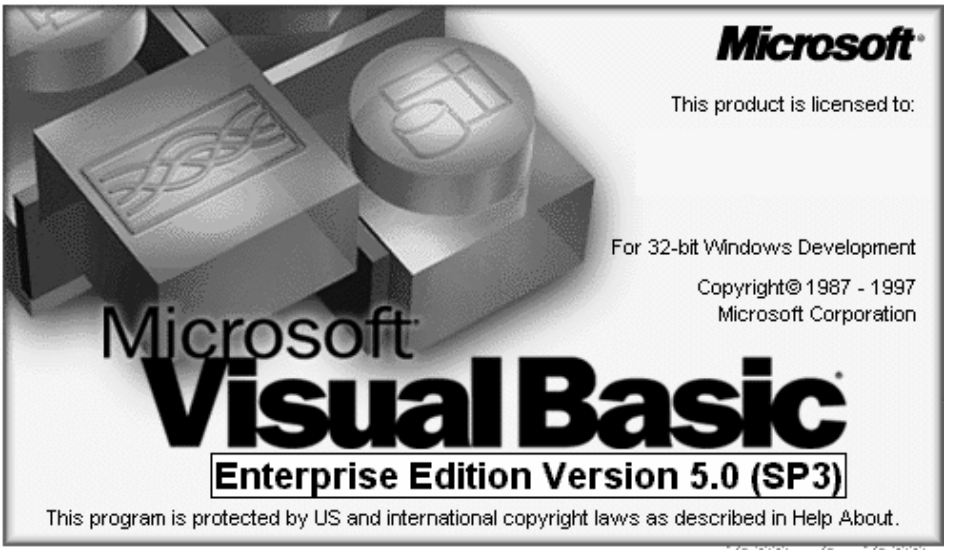
AACTEC

Computadores



Assistência Técnica de Microcomputadores
Rua Vinte Quatro N.º 06 Conj. M. Freire Guarulhos - SP
6480- 4288

Visual Basic 5.0



Passo a Passo

AACTEC
volcanic@ieg.com.br

Visual Basic 5.0

Passo a Passo

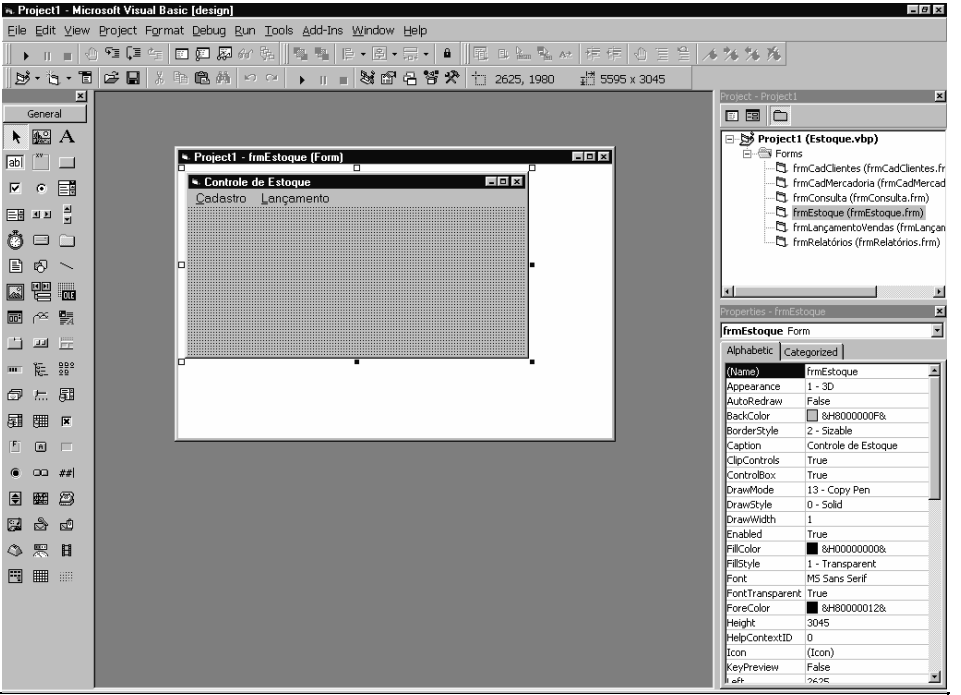
1ª Edição

Sumário

1 *APRESENTAÇÃO DO VISUAL BASIC 5*

- Apresentação das Janelas
- Salvando um Projeto

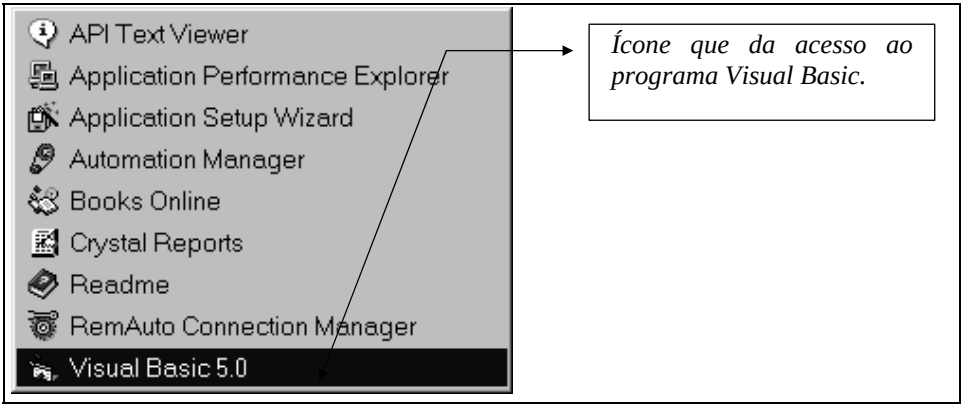
1.1 APRESENTAÇÃO DAS JANELAS



Disposição padrão das janelas do Visual Basic

O Visual Basic 5 se tornou a mais poderosa ferramenta de desenvolvimento de aplicações e o limite desta linguagem é a imaginação de cada um.

Para o programa ser apresentado para nós, precisamos dar um click no ícone “Visual Basic 5” encontrado na pasta “Microsoft Visual Basic 5.0”, como mostrado na figura 2.



A Versão 5 desta linguagem é destinado exclusivamente a criação de programas em 32 bits, ou seja, para Windows 95 ou superior ou Windows NT.

Nesta versão, quando abrimos o Visual Basic, ele nos dá uma séria de opções sobre o que desejamos criar: um executável, uma DLL, um ActiveX, etc. Por padrão usaremos somente "standard EXE" que é o usual para criação de programas executáveis.

1.1.1 Padronização em programas para Windows

Existe uma padronização que o programador deve obedecer quando estiver desenvolvendo sua aplicação, e com certeza o usuário irá agradecer e muito por esta lembrança. Veja alguns itens importante:

- Em menus de Edição sempre colocar as opções Recortar, Copiar e Colar.
- Sempre coloque um menu HELP.
- Não abuse das cores. Você conhece algum programa para Windows “sério” cujas janelas são apresentadas nas cores verde, azul, amarela? Com certeza não. As janelas são quase sempre brancas ou cinzas.

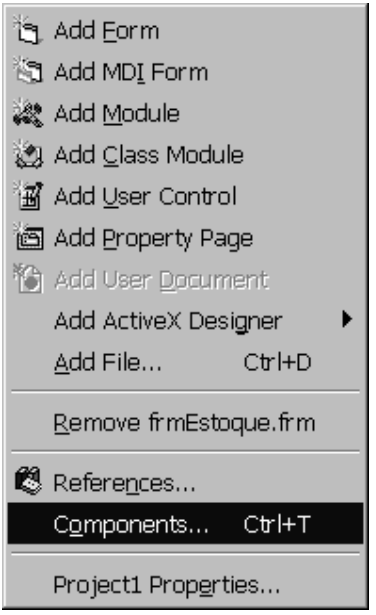
- Sempre use teclas de atalho para o usuário “cortar caminho”. Lembre-se que nem todos usuários gostam de usar o mouse intensivamente. Alguns preferem, sempre que podem, usar o teclado. Então dê esta opção.
- Sempre dê a possibilidade ao usuário de cancelar uma alteração feita em uma determinada janela. O botão “Cancelar” existe nos melhores programas existente no mercado.
- Quando um usuário escolhe uma aplicação para Windows para sua empresa e diversão, ele sabe que não vai encontrar dificuldades na operação do programa, pois todos são semelhantes em vários aspectos, então mantenha essa semelhança.

1.1.2 Barra de Menus e Botões

Usamos as opções do menu do Visual Basic para configurar nosso ambiente de trabalho, janelas, cores, inserir novos formulários, módulos, salvar o projeto, sair do Visual, etc. Não usamos os menus para construir nenhuma aplicação, suas opções são de assistência. Vamos no decorrer deste livro comentar as opções principais e mais usadas.

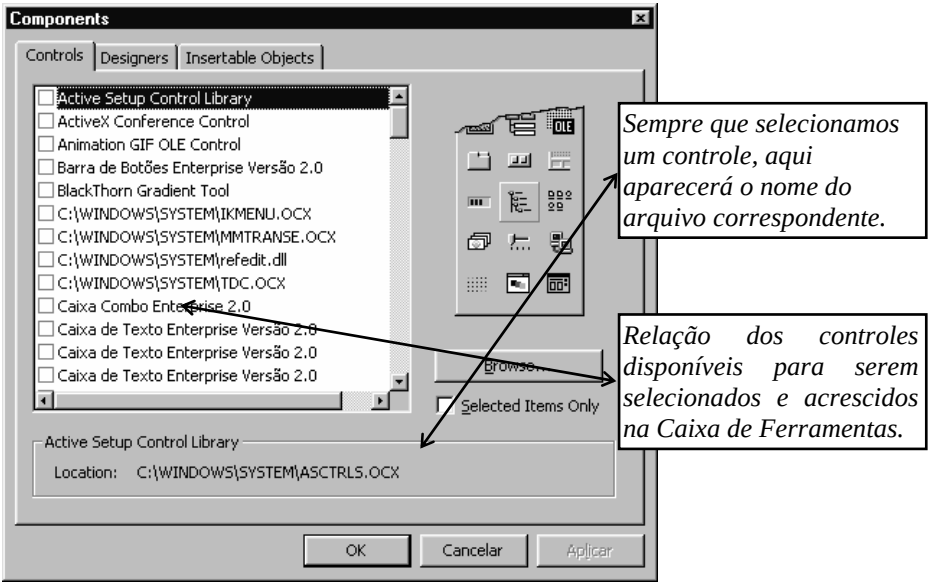
Como o acesso aos menus geralmente é um processo mais lento do que o acesso a botões, o Visual Basic dá a possibilidade de podermos “cortar caminho” através da barra de botões. As principais opções do Menu foram duplicadas em forma de botões.

1.1.3 Os Controles (Caixa de Ferramentas)



Os botões existente no lado esquerdo de nosso projeto, dentro de uma janela vertical, são chamados de controles, ou caixa de ferramentas, e serão esses controles que irão "montar" nossa aplicação. Através deles inserimos os objetos que precisamos para criar nossos formulários.

Quando entramos no Visual Basic pela primeira vez, nem sempre todos os controles estão disponíveis. Para podermos configurar quais controles queremos que apareça em nosso projeto, precisamos acessar o menu Project, opção Components. Será mostrado uma lista de Controles que podem ser anexados a Caixa de Ferramentas para podermos usar em nosso projeto. Com um click do mouse sobre a caixa de seleção



dos controles, pode-se marcar ou desmarcar quais controles necessitamos.

Para termos acesso a todos os controles padrão do Visual Basic, selecionamos todos que possuem a extensão OCX. Confira no campo “Active Setup Control Library” o nome do arquivo que possui o controle.

***Barra de Ferramentas**: Dentro dela esta todos os controles (objetos) que podemos usar em nosso projeto. Para usar um deles precisamos selecionar (clicando nele com o mouse) e inseri-lo dentro de nosso Formulário. Cada Objeto possui uma série de propriedades que podemos personalizar para usar de acordo com nossa necessidade.*

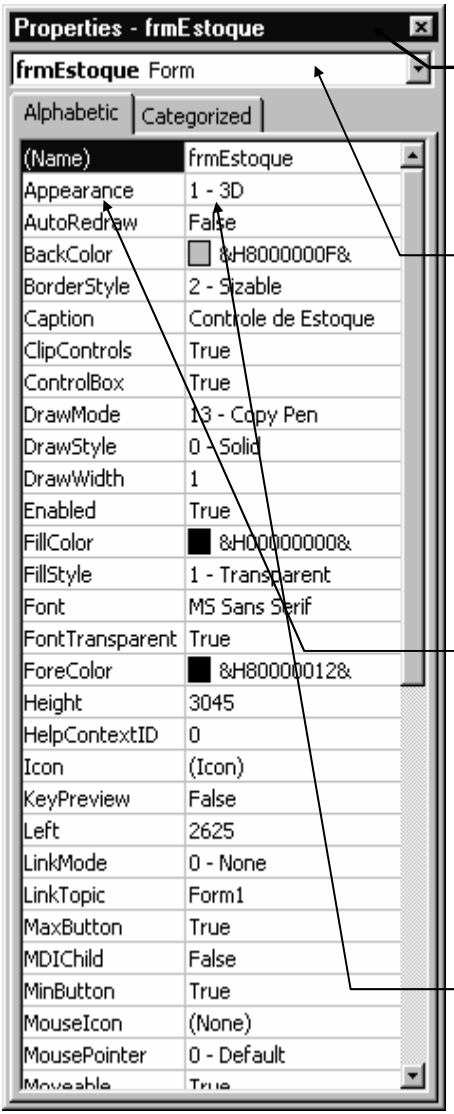
Temos aqui Objetos para colocar rótulo, caixa de edição, manipular Banco de Dados, caixa de Opção, barra de status, inserir figura, botão de comando, etc. Toda interface de nosso formulário pode ser montado a partir desses controles.

Uma vez inserido um desses Objetos dentro de nosso formulário, podemos manipular sua aparência e sua função através das propriedades que cada Objeto desse possui.

Dependendo da versão do Windows que estiver usando, alguns Controles podem não aparecer, pois existe alguns que são especifico do Windows 95.

1.1.4 Propriedades dos Objetos

Sempre que selecionamos um Objeto qualquer aparecerá no canto direito da tela uma janela de propriedades daquele objeto selecionado. Podemos então modificar as propriedades da forma que precisamos, e algumas delas veremos o resultado ainda em tempo de projeto, outros só veremos o resultado em tempo de execução.



Titulo "Properties" e o nome dado para o formulário selecionado no projeto.

Nesta caixa aparece no canto direito o nome dado pelo usuário ao Objeto selecionado, e no canto direito o tipo de Objeto. No exemplo mostramos o objeto de nome ***Form1*** *tipo* ***Form*** *(formulário). Podemos também selecionar outro objeto que existe em nosso formulário nesta caixa, clicando na seta para baixo, e selecionando outro objeto que ali esteja disponível para seleção.*

Nesta coluna aparece o nome das Propriedades. Cada Objeto possui uma quantidade diversa de propriedade para vários aspectos diferente. Como por exemplo, a Propriedade "BorderStyler" que especifica o tipo de borda que o Objeto terá. A Propriedade "Caption" que da um título ao Objeto, Propriedade "BackColor" onde se escolhe a cor de fundo que o objeto terá. Lembrando que essas propriedades atuam somente no Objeto selecionado. Se queremos mudar uma determinada propriedade de um Botão de Comando por exemplo, precisamos selecioná-lo na forma, e depois levar o mouse até a propriedade que se quer mudar.

Esta coluna é onde modificamos a propriedade escolhida, seja através de uma lista de opções (ComboBox), ou digitada diretamente.

Vamos por ora ressalvar duas propriedades que a maioria absoluta dos objetos possuem: **Caption** e **Name.**

A propriedade **Caption** é responsável pela legenda (título) que o objeto terá quando aparecer em nosso formulário. Por exemplo, num objeto **CommandButton** (Botão de Comando), o **Caption** é quem coloca o título que irá aparecer dentro do botão.

A propriedade **Name** nomeia um objeto, e sempre que formos nos referenciar a este objeto em qualquer etapa de nosso projeto, usamos o nome dado na propriedade **Name** dele. Se um botão "Cancelar" termos o nome de

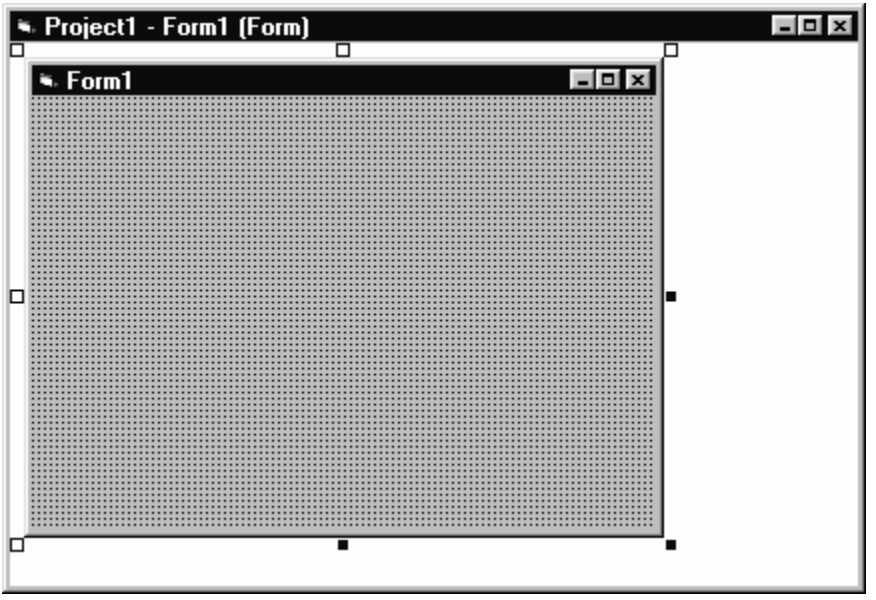
cmdCancelar, é este o nome que vamos usar na codificação deste objeto. Este nome fica atribuído ao objeto.

Para se dar nomes ao alguns objetos algumas regras tem que serem seguidas, como por exemplo, não pode conter espaços em branco, não se pode usar sinais como " #$&+-^ ". Como padrão acostuma-se a usar as três primeiras letras do nome para abreviar o tipo do Objeto, para durante a codificação do programa facilitar na identificação de qual objeto estamos usando. Por exemplo: Um botão de comando é do tipo **CommandButton**, e abreviamos como "cmd", um formulário é do tipo **Form**, e abreviamos como "frm", uma máscara de edição é do tipo **MaskEdBox**, e abreviamos como "msk". Essas abreviações ficam a critério do programador.

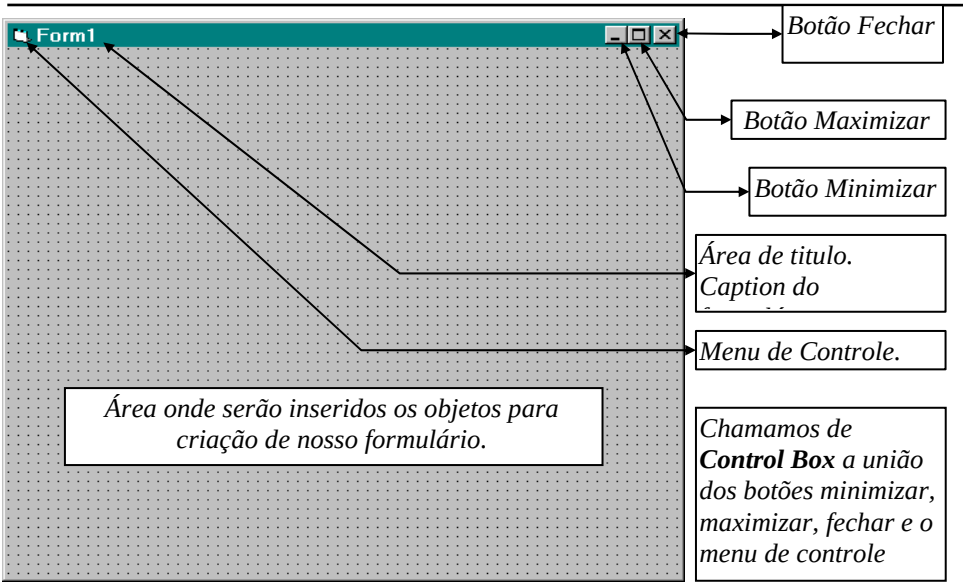
Mas para compor a propriedade **Name**, não colocamos somente abreviações do tipo de objeto que ele representa, mas também o que se trata o Objeto. Por exemplo: temos duas caixa de opções (**OptionButton**), onde temos a opção para o usuário escolher: Masculino e Feminino. Abreviamos este tipo de objeto como "opt", e o nome destes OptionButton ficaria assim: optMasculino, e o outro optFeminino.

Para se conhecer o tipo de cada Objeto existente na caixa de ferramentas é só ficar com o mouse em cima de qualquer objeto, e o tipo aparecerá dentro de uma caixinha amarela.

1.1.5 O Formulário (Form)

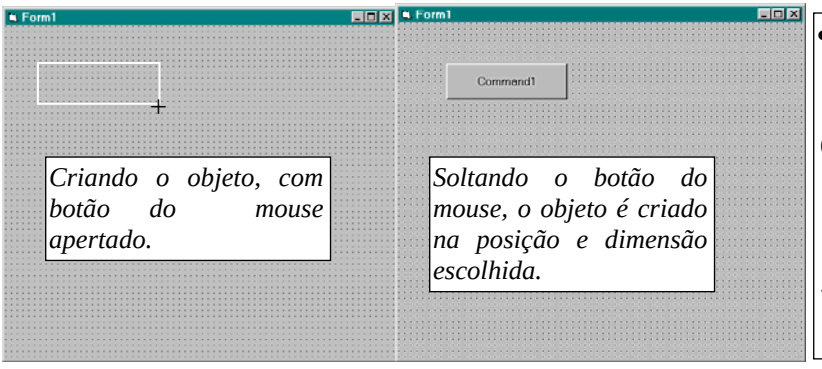


Nesta janela é onde colocamos todos os Objetos necessários para montar nosso projeto. Um projeto pode ter vários formulários. Chamamos de formulário, mas na verdade se trata de uma janela que nosso programa vai usar, seja uma janela de cadastramento, janela principal, janela de aviso. Não importa. Sempre que uma janela for aparecer em nosso programa, temos que criar um Formulário (**Form**). Quando abrimos o Visual Basic aparece um formulário vazio, somente com o título “Form1” o Menu de Controle com o ícone padrão do Visual Basic e os botões de Minimizar, Maximizar e Fechar. No corpo do formulário existe somente uma série de pontos pequenos espalhados. Esses pontos não aparece quando nossa aplicação é executada. Eles têm a função única de auxiliar no alinhamento dos objetos que são inseridos dentro do formulário. Se não existisse esses pontos ficaria mais difícil colocar os **OptionButton** (optMasculino e optFeminino) um abaixo do outro na mesma posição.



Para se inserir um Objeto da caixa de ferramentas para o formulário existem dois procedimentos: Escolha o objeto que se quer usar, e de dois clicks com o mouse em cima dele, e logo o objeto irá aparecer no centro do formulário. Esse método não é muito aconselhado pois o Controle escolhido não vai para onde você precisa dele, e sim sempre para o centro do formulário, obrigando-o a ir com o mouse até o objeto e arrastá-lo para onde realmente necessitamos dele.

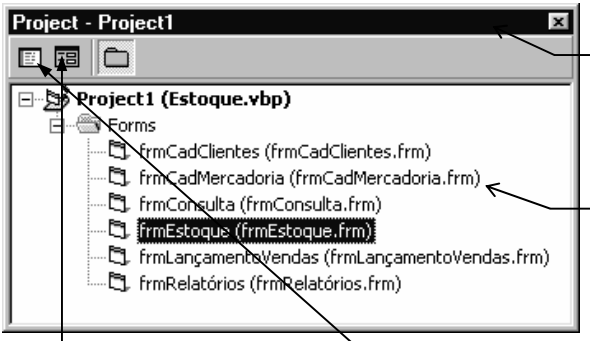
O modo mais aconselhado para inserção de objetos no formulário, é dando um click com o mouse no objeto, e depois leve o ponteiro do mouse para o formulário (dentro do formulário o ponteiro do mouse irá se transformar numa cruz), e então clicando no botão esquerdo do mouse faça um quadrado. O objeto irá aparecer dentro de seu quadrado. Isto é útil pois podemos inserir o objeto exatamente onde precisamos.



- escolha o objeto do tipo CommandButtom e coloque-o dentro do formulário.

1.1.6 Caixa de Projetos

Dentro da caixa de projetos existe uma lista dos formulários, módulos, bibliotecas e classes criados dentro de nosso projeto. Imagine um projeto que contenha cerca de 8 janelas (consequentemente 8 formulários), para acessarmos o formulário pretendido para alterar ou acrescentar algum objeto, basta clicar no formulário escolhido dentro da caixa de projetos.



Nome dado ao projeto

No lado direito, entre parentese, esta o nome do arquivo físico gravado no seu disco. No lado esquerdo esta o nome (Name) dado para o formulário.

Ao selecionar o formulário pretendido, aperte este botão e o formulário irá aparecer no centro da tela. Caso o componente não seja um formulário, este botão estará desabilitado.

Selecionando um formulário, aperte este botão para ver sua codificação. Nesta janela de código aparecerá os procedimentos, funções e eventos. Se o componente escolhido não tiver opção de codificação, então este botão estará desabilitado.

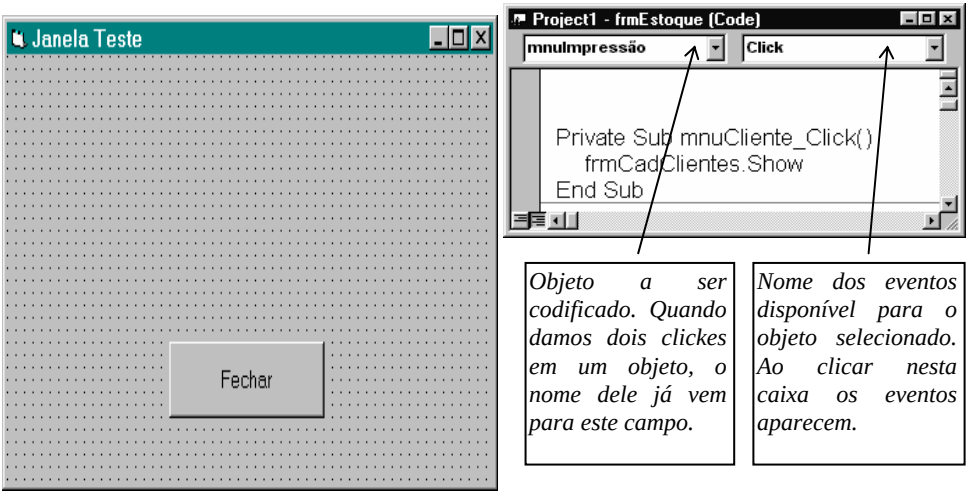
1.1.7 Janela de Codificação

Aqui é onde a programação no sentido da palavra começa. A criação da interface com o usuário não usa nenhuma linha de código, mas quando precisamos começar a manipular os dados digitados pelo usuário, então é necessário algum conhecimento dos procedimentos, comandos e funções do Visual Basic.

Cada objeto colocado no formulário possui eventos, e baseados nestes eventos vamos criar nosso programa. Por exemplo: Se queremos que ao apertar o botão “Fechar”, o programa feche a janela, então precisamos, dentro da janela de codificação criar um evento para o objeto **Botão** chamado **Click.** Dentro deste evento colocamos tudo que queremos que aconteça ao darmos um click neste botão criado.

Para acessar a janela de codificação podemos dar dois clickes em qualquer objeto ou apertar o botão “View Code” na Caixa de Projetos. O mais prático é sempre dar dois clickes no objetos que queremos codificar, pois assim a janela de codificação será aberta já pronta para codificar o objeto selecionado.

É bom lembrarmos que cada objeto possui vários eventos diferentes, e nossa tarefa básica é codificar cada evento que nosso programa necessitar. Somente ocorre eventos durante a execução do programa. Por exemplo, existe um evento chamado **Click**, que é para quando damos um click com o mouse no objeto. Evento **MouseMove** para quando o mouse se movimentar dentro do objeto. Evento **KeyPress** para quando um tecla for pressionada dentro do objeto. Estudaremos mais detalhado esses eventos. Por ora é importante entender a utilidade deles, e qual a finalidade deles em nossa codificação.



- Crie um Projeto novo, e no objeto Form modifique as propriedades:
  Caption = “Janela Teste”
  Name = “frmJanelaTeste”
- Em seguida insira um CommandButton no formulário e mude as propriedades também:
  Caption = “Fechar”
  Name = “cmdFechar”
- Dê agora dois clickes no botão “Fechar” dentro do formulário. A janela de código será aberta e será mostrado a seguinte codificação:

```
Private Sub cmdFechar_Click()
```

```
End Sub
```

- O Cursor ficará piscando acima do comando “End Sub”, esperando você digitar algo ali.

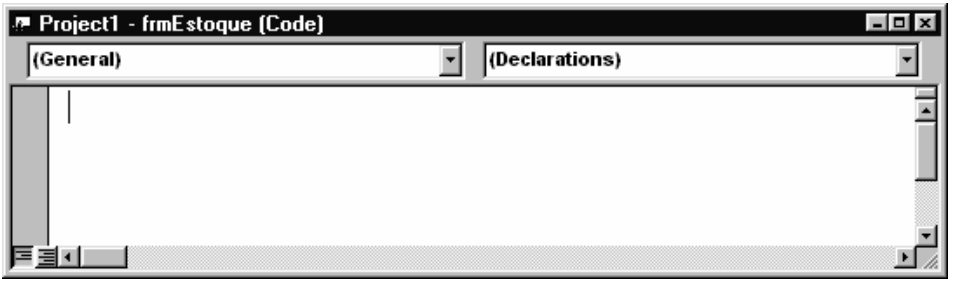
☞ Sempre que criamos um evento aparece o nome “Private Sub” sinalizando o inicio do procedimento. Depois vem o nome do procedimento, que no nosso caso é “cmdFechar_Click”. Note que é o nome que damos para o Objeto selecionado quando chamados a janela de código. Após o sinal de “sublinhado” aparece o nome do evento, igual o que aparece na caixa de combinação acima.

- Note na janela de codificação que o campo **Object** será o cmdFechar, pois este objeto é quem estava selecionado quando chamados a janela de codificação. Ao lado esta uma lista de Eventos, e o evento Click já estará pronto para ser codificado. Podemos trocar com o mouse o evento (**Procedure**) que queremos codificar ou até o objeto (**Object**). Mas vamos manter no nosso exemplo.
- Digite o comando: Unload frmJanelaTeste.

☞ O comando Unload serve para retirarmos da memória a janela que esta aberta.

- Agora aperte o botão Executar para rodar a aplicação. Ao apertar o botão fechar a janela desaparecerá o voltaremos para o Tempo de Projeto.

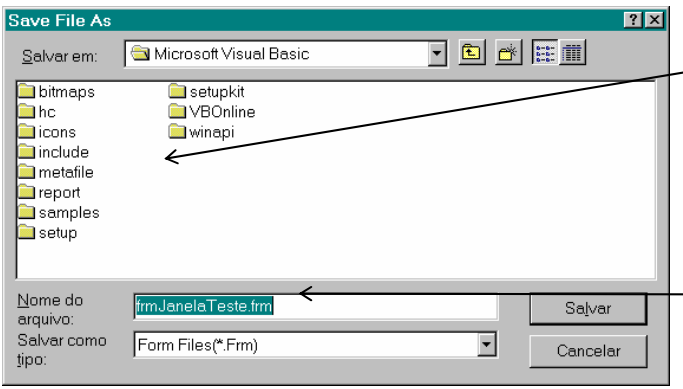
Para se entender melhor em que momento do programa esses eventos são chamados, é só imaginar o seguinte: quando executamos uma aplicação Windows qualquer, o usuário pode fazer várias tarefas (ou eventos) como apertar uma tecla, o botão do mouse, movimentar o mouse, dimensionar o tamanho de uma janela, etc. Pois bem, quando qualquer um desses possíveis eventos são executados o Windows verifica se existe alguma codificação para aquele evento, se existir ele executa todas as linhas até encontrar a expressão “End Sub”



Quando precisarmos criar uma codificação dentro de nosso formulário mas que tenha seu alcance estendido para todos os procedimentos e eventos do formulário, e não para algum determinado, escolhemos o **Object** "General", que significa Geral (na caixa de eventos aparecerá o nome "declaration"). Se declararmos ali uma variável ela terá validade para todo o formulário. Veremos isto melhor no Capítulo sobre variáveis.

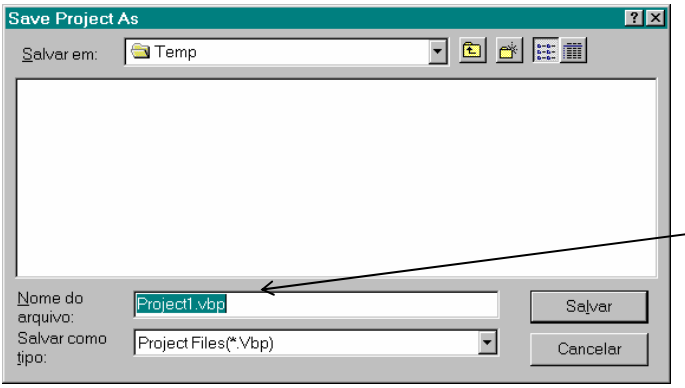
1.2 SALVANDO UM PROJETO

Para salvamos um projeto, primeiro precisamos entender que um projeto é um conjunto de formulários, módulos, classes, etc., e que quando vamos salvar o projeto que estamos trabalhando, o Visual Basic pedirá o nome do arquivo dos formulários ligado a ele e depois o nome do arquivo do projeto.



Escolhemos aqui o diretório (pasta) onde armazenaremos o arquivo de Formulário.

Aqui colocamos o nome do arquivo. Por "Default" irá aparecer o nome dado ao formulário na propriedade ***Name*** *acrescido da extensão .frm*



Nome do arquivo do Projeto. Por "Default" aparece o nome Project1, mas sempre é bom modificar para um nome que sinalize melhor o nome que nosso projeto (programa) terá.

Sempre coloque cada projeto dentro de uma pasta diferente para não ter problema de localização.

EXERCÍCIOS PROPOSTOS

1 - Descreva 5 botões da barra de botões que você acha mais importante:

Nome	Descrição

2 - Qual opção do menu nos dá a possibilidade de incluir mais Objetos dentro da Caixa de Ferramentas?

3 - Explique com suas palavras o que você entendeu sobre:

Caixa de Ferramentas:

Objetos:

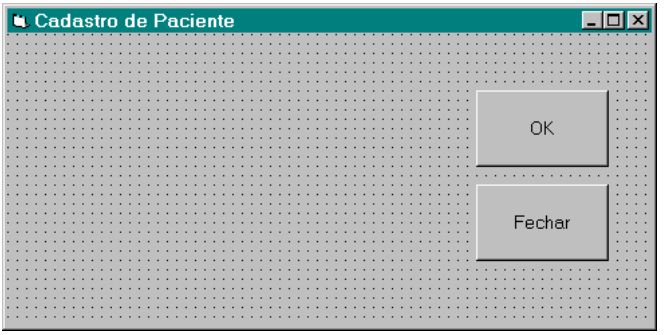
Propriedades:

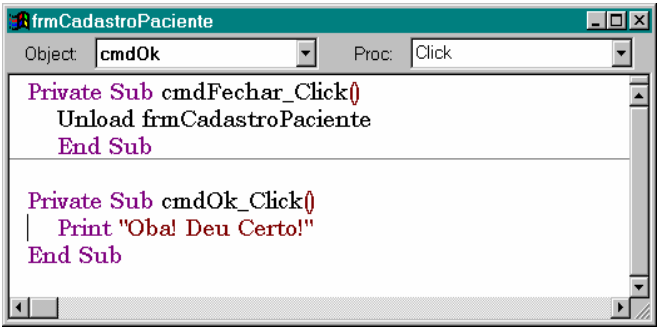
Formulário:

Janela de Projetos:

Codificação:

4 - Crie um projeto da seguinte forma:





No Objeto Formulário coloque dois Command Buttom no lado direito da janela como no exemplo. Depois muda as propriedades **Caption** do formulário para “Cadastro de Paciente” e **Name** para “frmCadastroPaciente”. No primeiro botão criado passe o **Caption** para “OK” e **Name** para cmdOk. O Outro botão terá o **Caption** “Fechar” e o **Name** “cmdFechar”. Pronto! A interface esta pronta. Vamos a codificação.

Dê dois clickes no botão “OK” e na janela de codificação escreva Print “Oba! Deu Certo!”, como esta na janela acima. Feche a janela, e voltando ao formulário dê agora dois clickes no botão “Fechar”. A janela de codificação volta a aparecer e nos digitaremos Unload frmCadastroPaciente.
Execute o programa e veja que fizemos com que o botão “OK” mostre na janela a frase “Oba! Deu Certo!” e o botão “Fechar” termina o programa.

☞ O Comando Print tem a finalidade de mostrar na janela aberta a expressão que esta na sua frente entre aspas. Na verdade numa programação profissional ele não é usado, pois não definimos sua localização na tela. Ele sempre imprime a expressão no canto superior esquerda. Estamos usando ele aqui somente para fins didáticos. Vamos substitui-lo depois pelo Objeto Label.

5 - Qual a diferença dos arquivos com extensão .vbp dos com extensão .frm

2 OS CONTROLES BÁSICOS

(A interface)

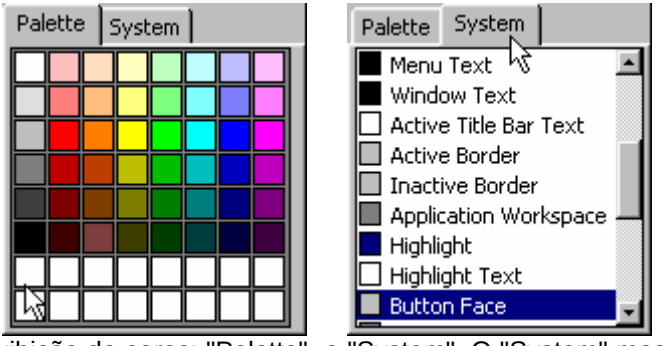
- Form
- CommandButton
- Label
- TextBox
- CheckBox
- OptionButton
- Frame

2.1 FORMULÁRIO E SUAS PROPRIEDADES

A principal propriedade do **Form** é a propriedde **Name**, pois através dela que o Visual Basic irá sugerir o nome que sera dado para o arquivo do Formulário (*.frm), mas essa propriedade **Name** não é importante somente no objeto **Form**, mas em todos os objetos. O Visual Basic automaticamente nomeia todos os objetos que são inseridos no formulário, mas nem sempre serve para nós. Por exemplo, quando inserimos um **CommandButton** no **Form** ele por *Default* já irá possuir o **Name** como “Command1”. SEMPRE é bom mudarmos esse padrão. A importância do **Name** nos objetos é fundamental para quando fazemos alguma referencia ao objeto. Por exemplo, se na codificação do programa, por algum motivo, precisarmos nos referenciar a um botão colocado no formulário cuja finalidade é fechar a janela, seria mais prático que o **Name** dele seja "cmdFechar", em vez de "Command1". Desta forma, visualmente, saberemos que se trata de um botão de comando cuja finalidade é fechar algo. Então é importante sempre termos o cuidado de nomearmos TODOS os objetos que colocamos no formulário com nomes bem descritivos e que nos dê uma pista do que se trata o objeto em questão. As outras propriedades vamos estudá-las no decorrer deste livro.

2.1.1 Propriedades Principais do Form

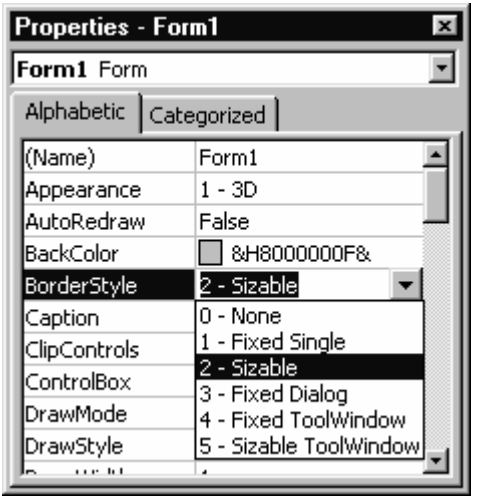
Appearance: Define se o modo de exibição será 3D ou normal.
BackColor: Muda a cor de fundo do formulário. Existe dois modos de



exibição de cores: "Palette" e "System". O "System" mostra as cores definidas no sistema de cores do Windows. Muito útil, pois sempre existe aquela situação que o usuário entra nas configuração do Windows, altera a aparência das cores e depois o seu programa não acompanha a mudança. Se precisar

que o programa desenvolvido tenha as cores sempre adaptadas de acordo com o padrão de cores escolhido na aparência do Windows então deve-se usar sempre as cores mostradas no "System" dessa propriedade.

BorderStyle: Muda o tipo de borda do formulário. None: sem borda, sem barra de titulo, sem menu de controle, sem botão maximizar e sem botão minimizar, e não pode redimensionar a janela.



Fixed Single: Borda fixa. Aceita o Menu de Controle, Maximizar, Minimizar, barra de título, mas não pode ser redimensionado.

(Default) Sizable: Borda comum. Possui o Menu de Controle, Maximizar, Minimizar, barra de título e pode ser redimensionada.

Fixed Dialog: Muito usada para janelas que vão manter um dialogo com o usuário. Ela pode ter Menu de controle e a barra de título, mas não terá os botões de maximizar e Minimizar. Não pode ser redimensionada.

Fixed ToolWindow: Não possui o menu de controle, o botão Maximizar e o botão minimizar. Também não pode ser redimensionada. Aparecerá somente a barra de título e o botão Fechar próprio do Windows 95. Este tipo de formulário não aparece na barra de tarefas do Windows.

Sizable ToolWindow: Não possui o menu de controle, o botão Maximizar e o botão minimizar. Pode ser redimensionada. Aparecerá somente a barra de título e o botão Fechar próprio do Windows 95. Este tipo de formulário não aparece na barra de tarefas do Windows.

Caption: Será o texto mostrado na barra de título do formulário (da janela).

ControlBox: Retira ou coloca o menu de controle e os botões maximizar, minimizar e fechar da janela.

Enabled: Se esta opção estiver com False, ou seja, estando o Enabled desabilitado, isto indicará que nenhum objeto desta janela, e nem a própria

janela, poderá responder a eventos gerados pelo usuário, como clicar do mouse, pressionamento de teclas, etc.

Font: Escolhe a fonte de letra padrão que será usada para todos objetos inseridos neste formulário.

Icon: Nesta propriedade escolhemos um arquivo de ícone para associar a esse formulário. Esse ícone é o que vai aparecer quando a janela for minimizada e no lado esquerda da barra de título (Menu de Controle).

KeyPreview: determina se os eventos do teclado no formulário serão executados antes dos eventos correspondentes aos Objetos inseridos no formulário.

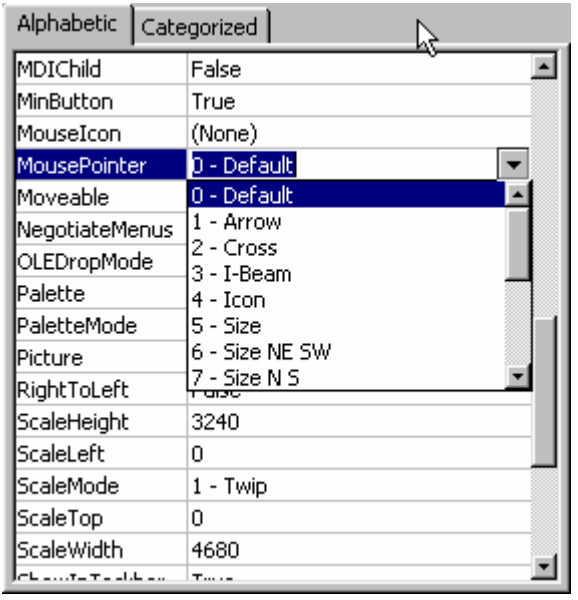
MaxButton: Habilita ou não o botão de maximização.

MDIChild: Determina se a janela será uma janela filha. Ou seja, terá uma outra janela mestre que agrupara o **Form** que se esta criando.

MinButton: Habilita ou não o botão de minimizar.

MouseIcon: Sempre que o mouse for movido em cima do formulário, o ícone associado a esta propriedade aparecerá (desde que a propriedade MousePointer esteja customizada).

MousePointer: Nesta propriedade especificamos o tipo de ponteiro que o mouse terá quando se mover sobre o formulário. Veja os tipos existentes:



	0	(Default) Ponteiro padrão.
	1	Ponteiro em forma de seta
	2	Ponteiro de seleção exata

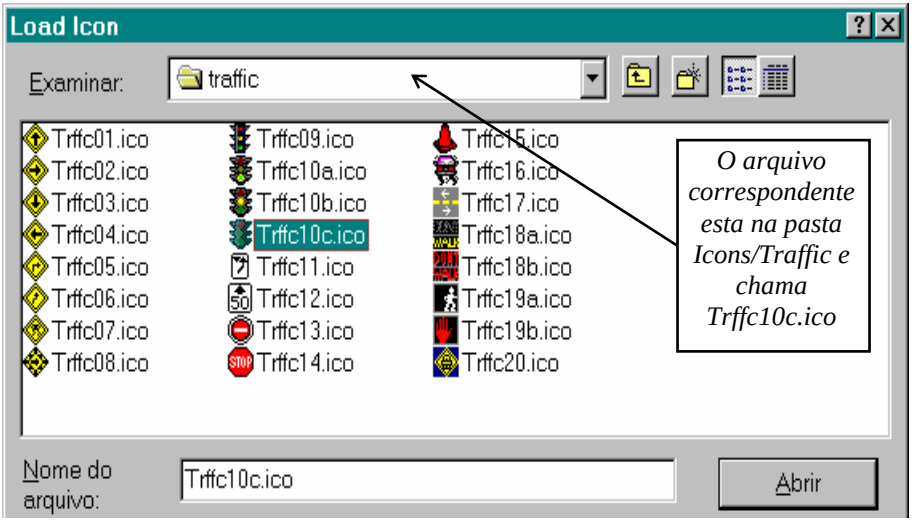
	3	Seleção de escrita
	4	Ícone
	5	Seleção de Mover Objetos
	6	Redimensionamento na diagonal
	7	Redimensionamento na vertical
	8	Redimensionamento na diagonal
	9	Redimensionamento na horizontal
	10	Seleção alternada
	11	Sinal de ocupado.
	12	Não disponível (Ocupado)
	13	Trabalhando em segundo plano (Somente 32-bit Visual Basic.)
	14	Seleção de Ajuda. (Somente 32-bit Visual Basic.)
	15	Todos os Tamanhos. (Somente 32-bit Visual Basic.)
	99	Aparece o ícone escolhido na propriedade MouseIcon

Name: Nome definido para o Objeto formulário.
Picture: Inseri uma figura em nosso formulário como papel de parede.
ShowInTaskbar: Habilita ou não a possibilidade da janela aparecer na barra de tarefas do Windows.
Visible: Determina se quando executarmos o programa essa janela irá ficar visivel ou invisível.
WindowState: Determina se, quando executarmos o programa, a janela irá aparecer na tela do computador Normal, Maximizada ou Minimizada.

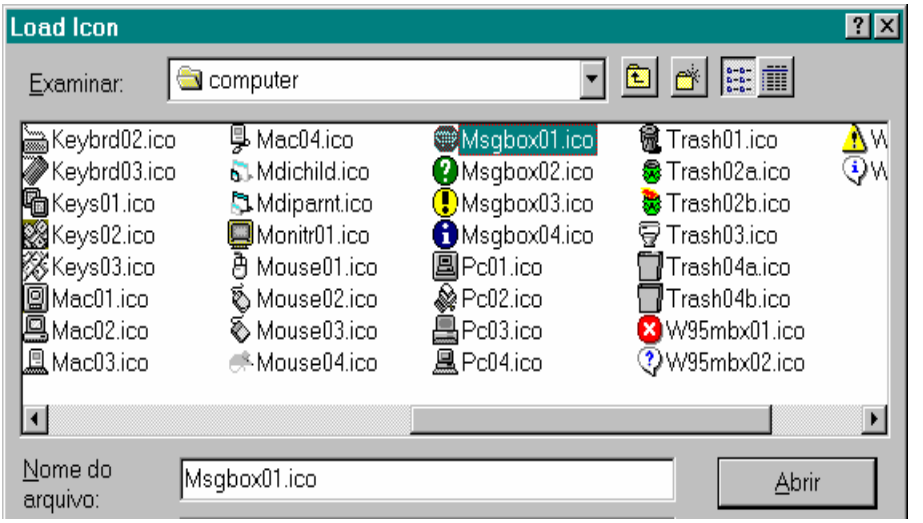
- Crie um formulário Novo (opção File / New Project no menu).
- No Formulário criado mude as propriedades:
  Caption = "Lançamentos dos Exames"

Name = “frmLançamentoExames”

- Configure a janela para não aparecer o botão de Maximizar. Para isto mude a Propriedade MaxButton para False.
- Faça com que o ponteiro do mouse mude para o desenho de um sinaleiro com a luz vermelha acesa toda vez que o mouse mover no formulário. Para isto a propriedade MousePointer deve ser “Custom” e na propriedade MouseIcon identificamos a pasta (diretório) onde esta o arquivo de ícone que contém o sinaleiro. Veja a figura:



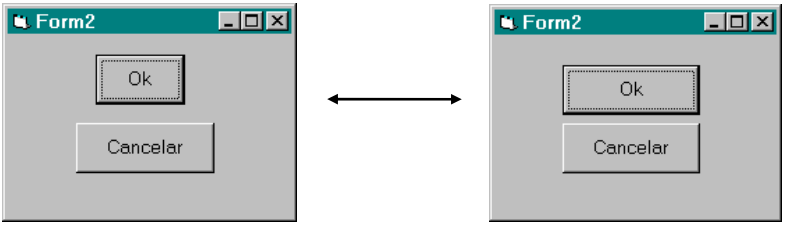
- Escolha agora o ícone que será associado ao formulário (janela). A propriedade que usaremos para este fim será “Icon”. Procure o arquivo que contem o desenho de uma placa de trânsito “Stop” contida na pasta “Computer”. Veja a figura:

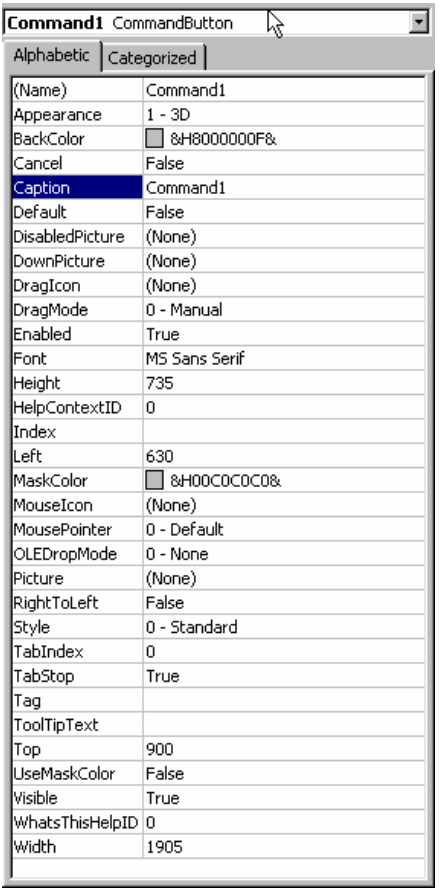


- `Pronto. Rode sua aplicação de veja o que acontece.`

2.1.2 As Propriedades Principais do CommandButton

O Botão de Comando executa determinadas ações estabelecidas pelo programador. Um conselho: Não coloque botões de vários tamanhos dentro do formulário. Se temos dois botões: "Cancelar" e "OK", não será porque o botão "OK" tem um título menor que seu tamanho será menor. Nivele-o pelo nome maior.





Cancel: Se esta opção for ajustada como verdadeira, o botão será associado a tecla ESC, e sempre que pressionarmos esta tecla será como se tivéssemos apertado o botão.

Caption: O título que será exibido dentro do botão.

Default: Estando o botão com essa propriedade como True indica que este botão será o Padrão da janela, e sempre que apertarmos a tecla Enter será como se tivéssemos apertado o botão.

Enabled: Determina se o botão será habilitado para pressionamento por parte do usuário ou não.

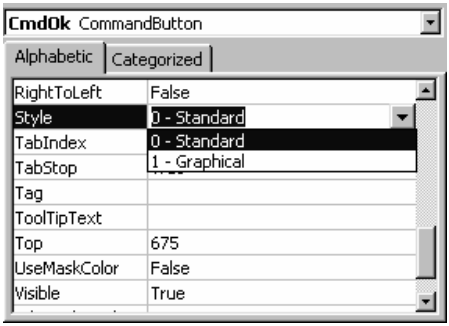
Font: Escolhe a fonte de letra que o *Caption* terá.

Name: O nome que daremos para o Objeto botão de comando.

Visible: Determina se o botão será visível para o usuário quando o programa estiver em execução.

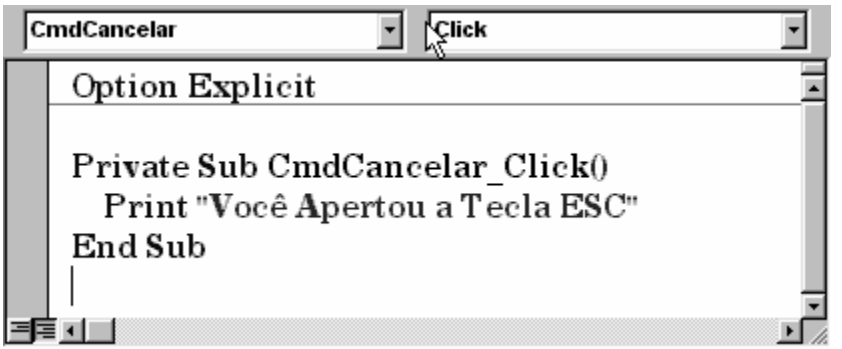
Style: Escolhe se o botão terá elemento

gráfico em seu interior. Se escolher Style do tipo "Graphical" o CommandButton aceitará figuras, e elas devem ser inseridas na propriedade **Picture** do objeto CommandButton



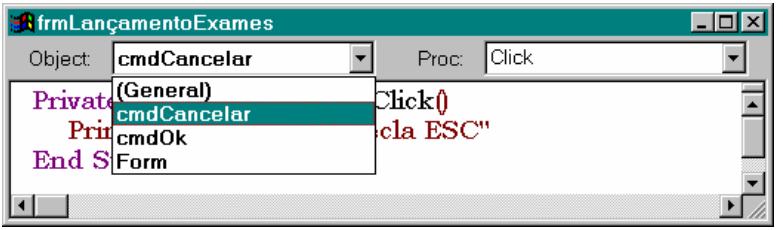
- Em um formulário novo acrescente dois botões de comando. Mude as propriedades deles da seguinte forma:

Propriedade	**botão1**	**Botão2**	***Explicação***
Caption	OK	Cancelar	*título para o botão*
Name	cmdOk	cmdCancelar	*nomear o botão*
Cancel	False	True	*habilitar ESC*
Default	True	False	*habilitar ENTER*

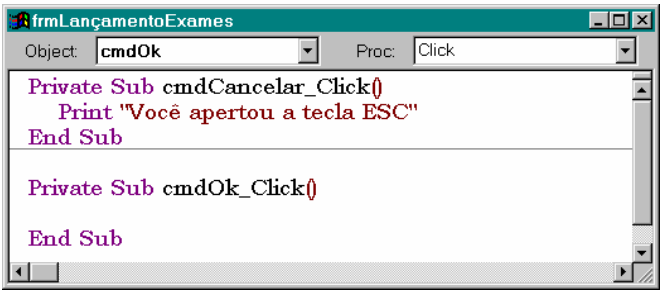


- Dê dois clickes no botão Cancelar. Irá aparecer a janela de codificação para o botão Cancelar. Digite: Print “Você apertou a tecla ESC”.

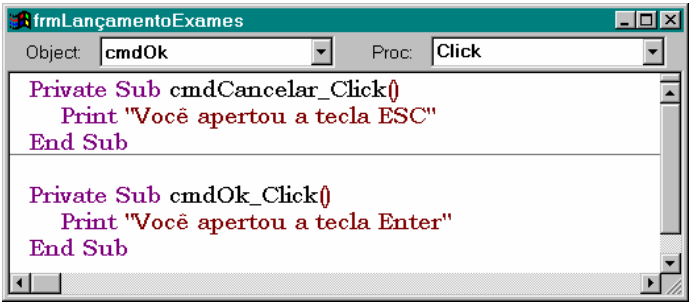
- Estando ainda na janela de codificação, leve o ponteiro do mouse até o topo desta janela onde esta o nome: **Object.** Na frente estará o nome do botão que estávamos trabalhando na codificação. Aperte o botão. Irá aparecer o nome dos objetos que temos: cmdCancelar, cmdOk, Form, e General.



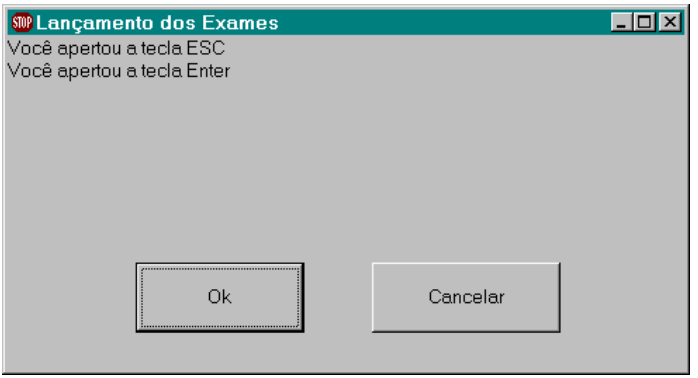
- Nesta caixa de combinação dê um click no objeto cmdOK. Com isto será criado automaticamente um evento Click para este objeto.



- Digite: Print “Você apertou a tecla Enter”



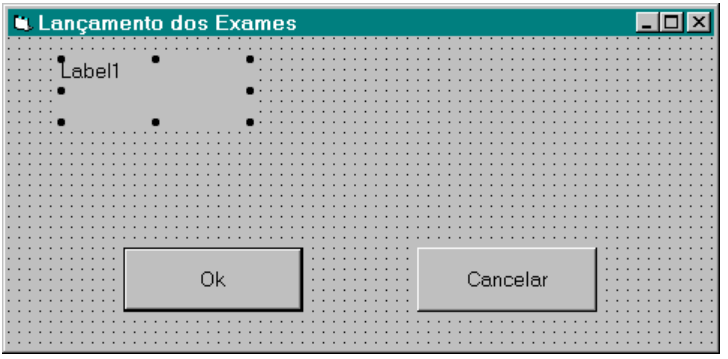
- Com isto, criamos um evento para o botão OK e para o Botão cancelar. Execute agora a aplicação e aperte a tecla ENTER e depois a tecla ESC.



- O resultado será a mesma coisa que apertarmos o botão OK e o botão cancelar.

2.1.3 *Propriedades Principais do Label*

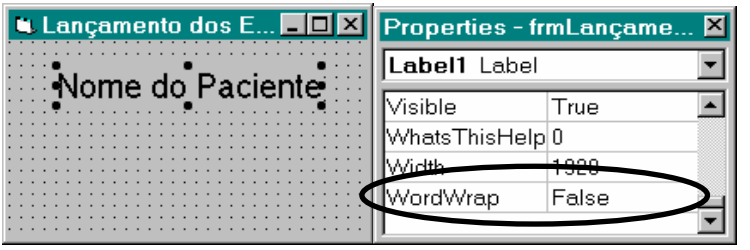
Esse objeto é um texto, não editável, que usamos geralmente para rotular outros Objetos. A Finalidade básica dele é de colocar um texto na forma.



Alignment: Determina o alinhamento que o texto terá dentro do label.

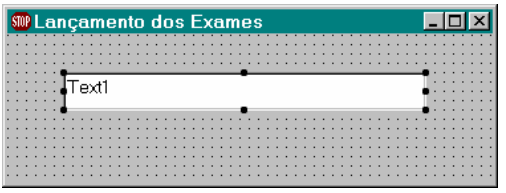
AutoSize: Com esta propriedade habilitada indicamos ao Controle para automaticamente dimensionar seu tamanho ao tamanho do texto do label.
Note na figura que, quando o AutoSize foi passado para True, as marcas de seleção do label foi ajustado para o tamanho do texto “label1”.
BackColor: Escolhe a cor de fundo que envolverá o label.

BackStyle: Escolhe entre o fundo Transparente ou Opaco para o label.
BorderStyle: Escolhe entre colocar uma moldura envolvendo o label ou não.
Caption: A propriedade principal do label. Determina o texto que será exibido dentro do Objeto.
Enabled: Habilita ou desabilita o objeto. Quando esta em False fica com a cor de seu conteúdo acinzentada.
Font: Escolhe a fonte de letra que terá o texto digitado na propriedade Caption.
ForeColor: Escolhe a cor da fonte de letra
Name: Nomeia o Objeto label. Como já foi dito é importante que todos os objetos seja nomeado. A inicial do label é "lbl".
Visible: Indica se o objeto será visível ou não para o usuário.
WordWrap: Quando o AutoSize esta em true, não é possível expandir o texto digitado na propriedade Caption em outras linhas. Mas se passarmos essa propriedade WordWrap para True isto poderá ser feito, bastando para isto dimensionarmos o label.

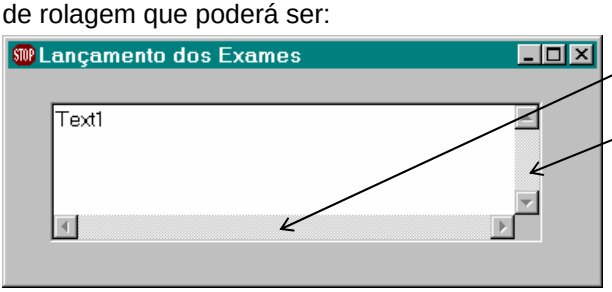


2.1.4 Propriedades Principais do TextBox

Exibe uma caixa de texto onde o usuário irá digitar dados diversos. Quando inserimos esse Objeto, ele automaticamente coloca a expressão "Text1" dentro da área de digitação. Logicamente, sempre teremos que remover essa expressão, pois não faz sentido deixar uma caixa de texto com "Text1" em seu interior. O usuário de seu programa ficará sem entender a finalidade.



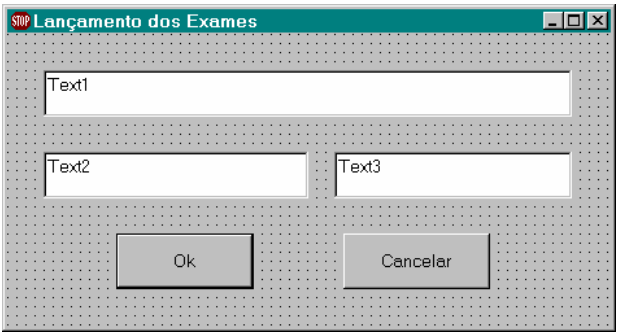
BackColor: Escolhe a cor de fundo da Caixa de Texto. Geralmente é branco.
BorderSytle: Tipo da borda: Fixa e simples, ou sem borda.
Enabled: Estando False o objeto não estará habilitado para interagir com o usuário.
Font: Escolhe a fonte de letra que será mostrada dentro da caixa de texto.
ForeColor: Escolhe a cor da fonte de letra.
Locked: Estando em false trava qualquer digitação na caixa de texto
MaxLength: Quantidade máxima de caracteres dentro da caixa de texto.
MultiLine: Habilita a possibilidade de se digitar mais de uma linha na caixa de texto.
Name: Nomeia o objeto TextBox. Geralmente com a inicial txt. Exemplo: Se formos usar essa caixa de texto para que o usuário digite o nome do paciente, poderíamos abreviar assim: txtNomePaciente.
PasswordChar: Se durante a digitação de qualquer dado na caixa de texto, quisermos que o Visual Basic mostre outro caractere no lugar do caractere digitado, é só especificarmos aqui qual queremos que seja mostrado. Muito usado para digitação de senhas.
ScrollBars: Estando a propriedade MultiLine habilitada, é interessante colocarmos um ScrollBars na caixa de texto, pois ele acrescentará uma barra de rolagem que poderá ser:



1 - Horizontal
2 - Vertical
3 - Both: Horizontal e Vertical juntos.

Text: A propriedade Text é a mais importante deste Objeto. Todo o texto digitado pelo usuário dentro da caixa de texto é incorporado nesta propriedade.

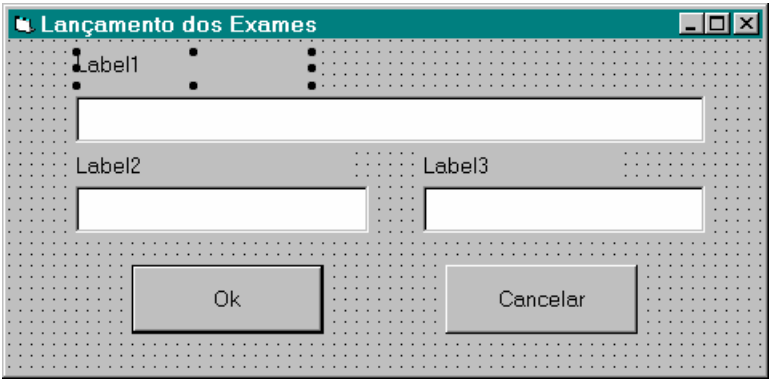
- Coloque 3 caixa de texto em nosso formulário, de forma que a disposição das mesmas fiquem como na figura abaixo.



- Usaremos a primeira caixa de texto para digitarmos o nome do paciente. A segunda para digitarmos o sexo dele (Masculino ou Feminino) e a terceira para ser digitado o nome do exame que ele vai fazer.
- Para tanto, vamos mudar algumas propriedades destas 3 caixas de texto.

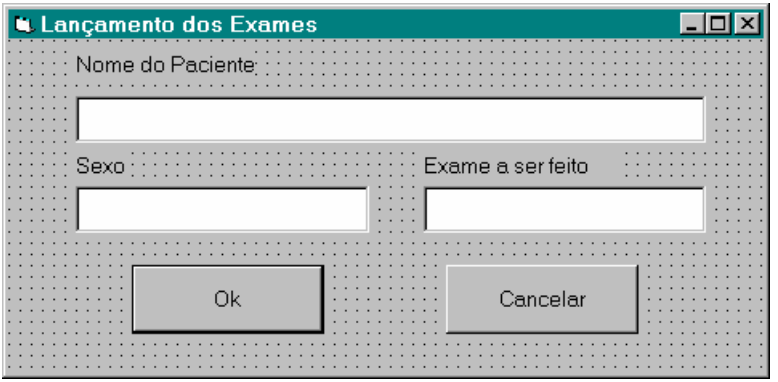
Propriedade	**Text1**	**Text2**	**Text3**
Name	TxtNome	txtSexo	TxtExame
Text	“deixar vazio”	“deixar vazio”	“deixar vazio”
MaxLength	30	9	15
ForeColor	Vermelho	Preto	Preto

- Coloque agora 3 Objetos Label no formulário na seguinte disposição:

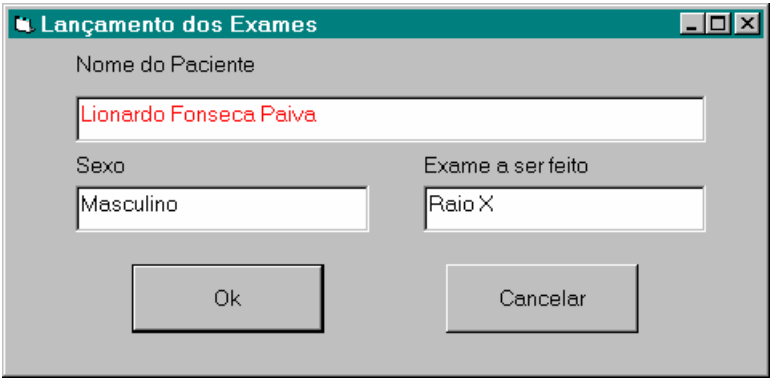


Propriedade	**Label1**	**Label2**	**Label3**
Name	lblNomePaciente	lblSexo	LblExame

Caption	Nome do Paciente	Sexo	Exame a ser feito
AutoSize	True	True	True
Appearance	3D	3D	3D



- O resultado será uma janela como a figura acima.



- Pratique nesta janela e veja se o nome realmente esta com o limite de 30 caracteres.

2.1.5 Eventos relacionados ao Objeto TextBox

Irei citar aqui alguns dos principais eventos relacionados a caixa de texto. Lembramos que todos os Eventos ocorre quando o usuário faz alguma ação. Por exemplo: se ele aperta um botão aciona o Evento **Click**, se ele digita algo numa caixa de texto, aciona o evento **Change** e **KeyPress**, se ele movimento o mouse sobre um objeto, aciona o evento **MouseMove.**

Vamos explicar esses eventos aqui, mas antes é importante entendermos que cada movimento ou ação do usuário dispara um evento no programa. Mas esses eventos somente terão alguma reação se programarmos isto na janela de codificação.

Change: Ocorre sempre que o usuário altera o valor contido na caixa de texto.
Click: Ocorre quando o usuário pressiona o botão esquerdo do mouse sobre a caixa de texto.
DlbClick: Se o usuário apertar duas vezes o botão esquerdo do mouse sobre a caixa de texto
KeyDown: Ocorre quando o usuário aperta uma tecla no teclado.
KeyUp: Ocorre quando o usuário solta a tecla apertada.
KeyPress: Ocorre quando uma tecla é pressionada e solta. Como argumento possui uma variável cujo conteúdo é a tecla pressionada (código correspondente na tabela ASCII).

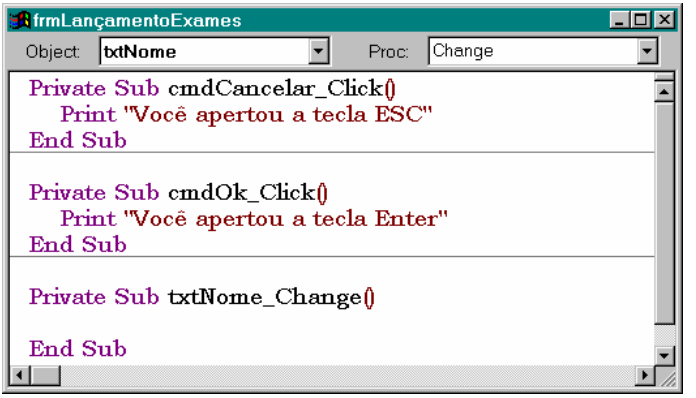
Cod.		Cod.		Cod.		Cod.	
0	·	32	Espaço	64	@	96	`
1	·	33	!	65	A	97	a
2	·	34	"	66	B	98	b
3	·	35	#	67	C	99	c
4	·	36	$	68	D	100	d
5	·	37	%	69	E	101	e
6	·	38	&	70	F	102	f
7	·	39	'	71	G	103	g
8	Backspace	40	(	72	H	104	h
9	Tab	41	)	73	I	105	i
10		42	*	74	J	106	j
11	·	43	+	75	K	107	k
12	·	44	,	76	L	108	l
13	Enter	45	-	77	M	109	m
14	·	46	.	78	N	110	n
15	·	47	/	79	O	111	o
16	·	48	0	80	P	112	p
17	·	49	1	81	Q	113	q
18	·	50	2	82	R	114	r
19	·	51	3	83	S	115	s
20	·	52	4	84	T	116	t
21	·	53	5	85	U	117	u
22	·	54	6	86	V	118	v
23	·	55	7	87	W	119	w
24	·	56	8	88	X	120	x
25	·	57	9	89	Y	121	y
26	·	58	:	90	Z	122	z
27	Esc·	59	;	91	[	123	{
28	·	60	<	92	\	124	|

29	·	61	=	93	]	125	}
30	·	62	>	94	^	126	~
31	·	63	?	95	_	127	·

· Estes caracteres não são aceitos pelo Microsoft Windows.

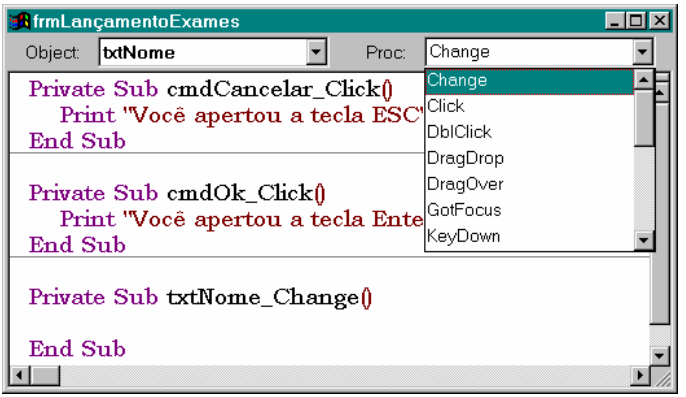
MouseMove: Ocorre quando movemos o ponteiro do mouse sobre a caixa de texto.
MouseDown: Ocorre quando o usuário aperta o botão do mouse (seja da direita ou da esquerda) sobre a caixa de texto.
MouseUp: Ocorre quando o usuário solta o botão do mouse (seja da direita ou da esquerda) sobre a caixa de texto.,

• Conhecendo os eventos e sabendo para que eles servem vamos começar a usá-los. No formulário **frmLançamentoExames** que já criamos posicione o cursor sobre a caixa de texto "txtNome" e dê dois clickes com o mouse. Irá aparecer uma janela assim:

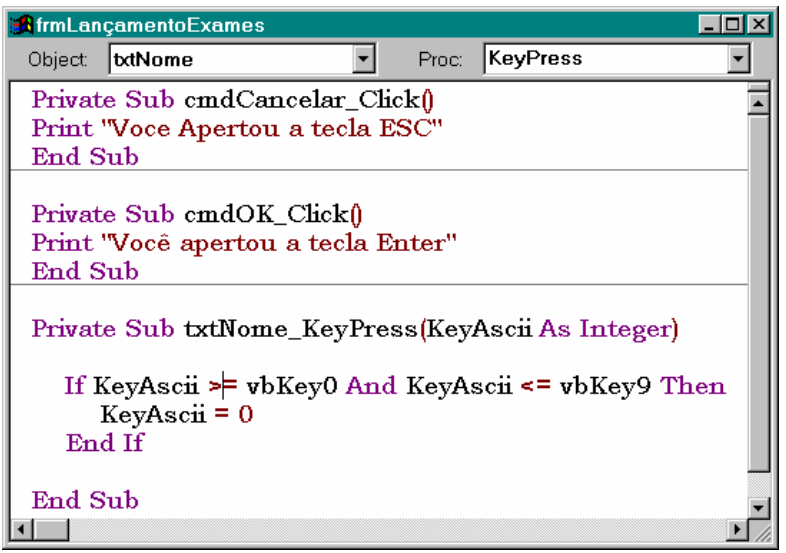


• O Objeto ativo será o "txtNome" e o evento será o "Change". Para caixa de texto esse evento é o que primeiro aparece. Vamos mudá-lo passando para o evento "KeyPress". Para fazer isto, leve o ponteiro do mouse até a caixa de combinação que esta o nome Change e clique no botão que

esta do lado direito da janela. Aparecerá a lista de Eventos disponíveis para a caixa de texto. Click sobre o nome “KeyPress”. Ele irá agora criar um outro procedimento de evento chamado “Private Sub txtNome_KeyPress(KeyAscii As Integer)”. Vamos usar esse evento no objeto txtNome para não aceitar o usuário digitar números nesta caixa de texto. A linha de comando ficará assim:



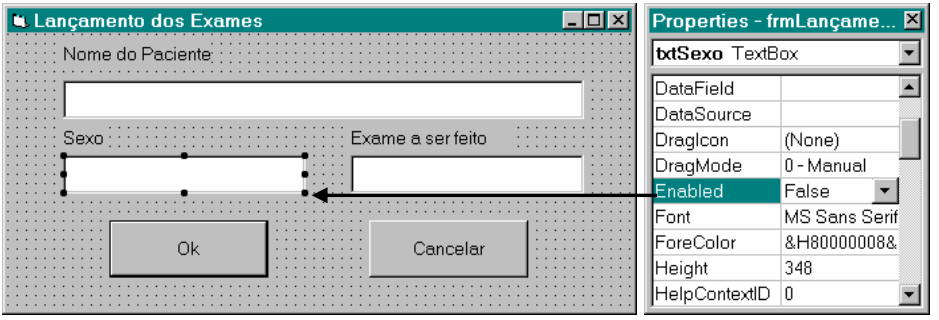

```
Private Sub txtNome_KeyPress(KeyAscii As Integer)
if Keyascii >= vbKey0 and Keyascii <= vbKey9 then
      Keyascii = 0
endif
EndSub
```



- Pronto! Quando o usuário digitar um dígito numérico no teclado ele será ignorado pela caixa de texto.
- Existem algumas expressões novas como “KeyAscii”, “If”, “vbKey0” e “vbKey9”. Vamos falar melhor deles nos próximos

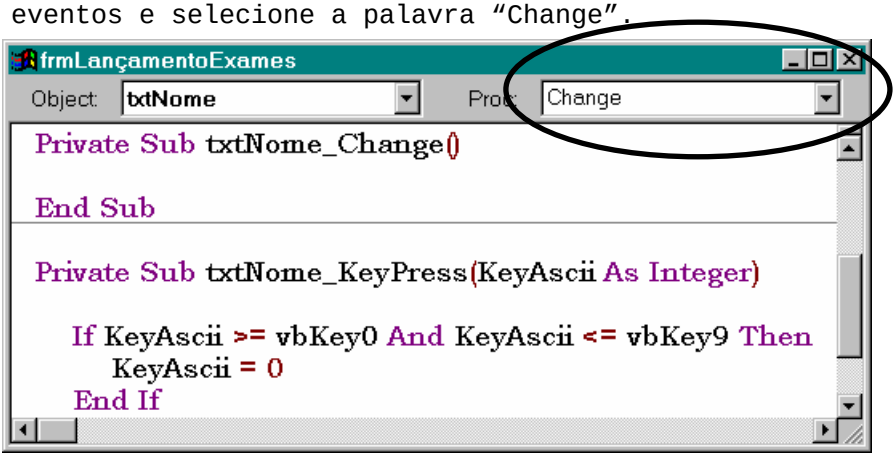
capítulos. Por ora, para se ter um entendimento melhor do que ocorreu durante a execução deste evento vamos explicar rapidamente essas novas expressões:

- KeyAscii : é uma variável criada pelo evento "KeyPress" cujo conteúdo será o valor (na tabela ASCII) correspondente a tecla digitada.
- VbKey0 : Uma constante que simboliza o número Zero.
- vbKey9 : Uma constante que simboliza o número nove.
- If : Comando condicional. Significa "Se". Ou seja, se a tecla digitado for maior ou igual a zero e também menor e igual a nove então faça algo.
- Rode o programa e verifique que a caixa de texto já esta devidamente codificada para não aceitar números.
- Para melhorarmos ainda mais nossa aplicação, vamos analisar o seguinte: Não podemos deixar o usuário digitar o sexo do paciente antes de digitarmos o nome dele! Temos que fazer uma codificação que verifique se o usuário digitou o nome, e somente depois liberar a digitação do sexo. OK? Vamos lá!
- Selecione o Objeto lblSexo e mude a propriedade "Enabled" dele para False, e o Objeto txtSexo.Enabled também para False.

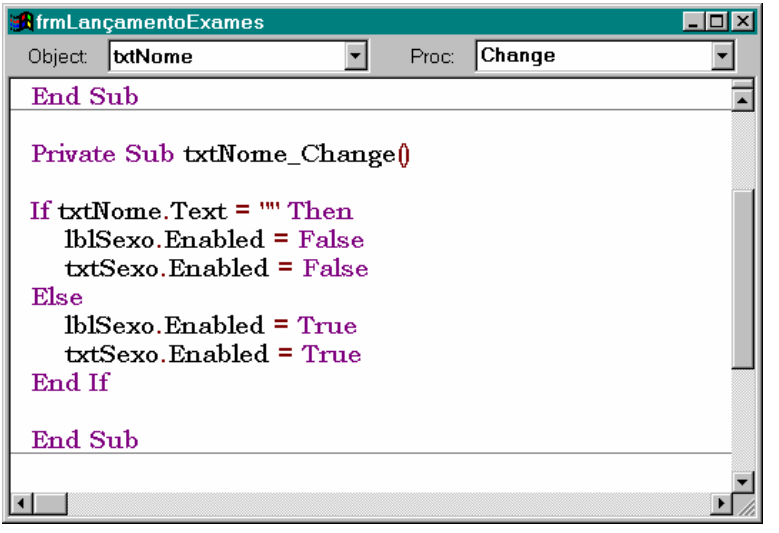


- Note que quando executarmos o programa o label "Sexo" estará na cor cinza e a caixa de texto abaixo não aceitará digitação.
- Então vamos agora colocar uma codificação que verifique se na caixa de texto txtNome existe algo digitado. Se existir, libera a digitação do Sexo, se não existir continua sem liberação.
- Dê dois clickes com o mouse no Objeto "txtNome" e veja se o evento Change está ativo para ser configurado. Se não

tiver leve o mouse até a caixa de combinação com a lista dos eventos e selecione a palavra “Change”.



- Como foi explicado acima, esse evento é acionado sempre que o usuário altera um valor digitado na caixa de texto. Mas isto não significa que na caixa de texto obrigatoriamente **tem** que existir um valor para ser alterado. Na verdade quando está vazia e digitamos algo ali estamos alterando seu conteúdo.
- Com isto, concluímos que sempre que digitarmos algo ali (letra, número, barra de espaço, delete, Backspace, etc.) o evento “Change” será acionado.
- Digite o seguinte código na janela de codificação aberta:



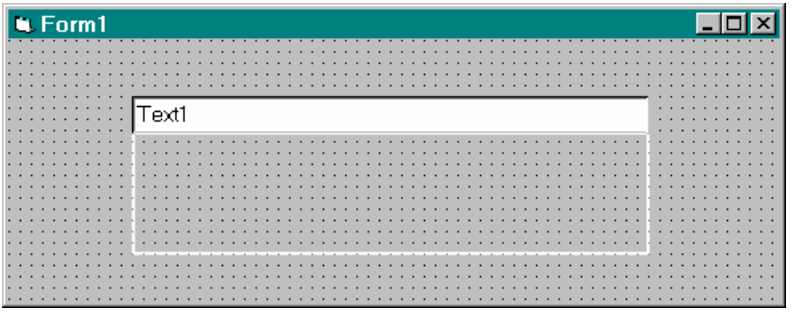
- O programa irá policiar a caixa de texto “txtNome”. Se a sua propriedade “Text” (que contém o que o usuário

digitou), for igual a vazio (“”), então as propriedades “Enabled” do lblSexo e do txtSexo serão desabilitadas, caso contrario (Else), essas propriedades serão habilitadas.

- Note que podemos mudar a propriedade de um Objeto tanto a nível de codificação em tempo de execução, como em tempo de projeto.
- Execute o programa, o digite seu nome na caixa de texto “Nome do Paciente”. Observe que quando você começar a digitar automaticamente a caixa de texto “Sexo” será liberada.
- Agora use a tecla “Backspace” para voltar a digitação e apagar o nome digitado. Veja que quando se apaga o ultimo caractere e a caixa de texto fica vazia as propriedades “Enabled” do “Sexo” são desabilitadas automaticamente.

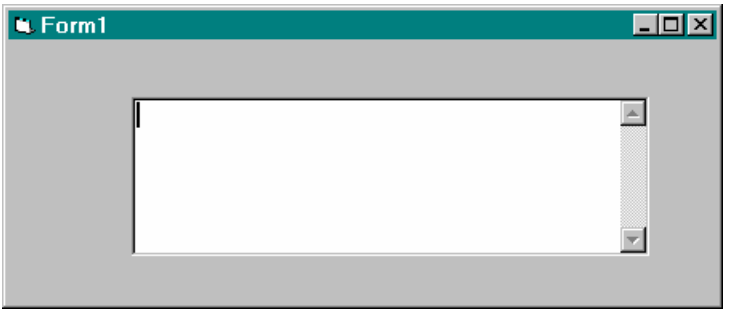
2.1.6 Caixa de Texto para várias linhas

Se quisermos configurar uma caixa de texto para receber várias linhas de digitação, e não uma linha como no exemplo acima, temos que fazer as seguintes mudança no objeto:



Aumente o tamanho da caixa de texto na vertical, para darmos espaços para as novas linha que serão inseridas. Mude as propriedades:

Propriedade	**Text1**	**Descrição**
MultiLine	True	Habilita múltiplas linhas
Text	“Vazio”	Tira o texto “Text1”
ScrollBars	2 - Vertical	Coloca barra de rolagem vertical



No fim de cada linha o cursor passará automaticamente para a linha de baixo.

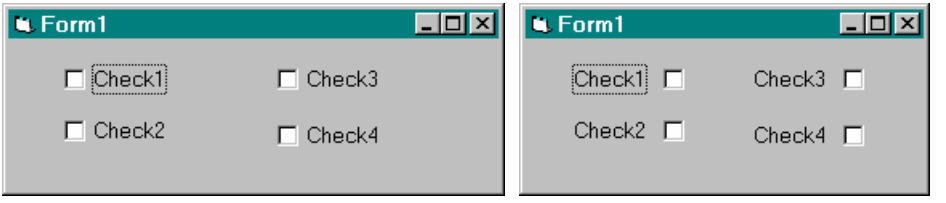
2.1.7 Propriedades Principais do CheckBox

Exibe uma caixa de checagem (ou verificação) onde o usuário poderá ligar ou desligar uma determinada opção.

Usamos o CheckBox para mostrar várias opções ao usuário para ele escolher uma, algumas, todas ou nenhuma.

No nosso exemplo de “Lançamento de Exames” esse objeto poderia substituir com vantagens a caixa de texto onde digitaremos o Exame que o paciente fará. Sabendo de antemão que o laboratório possui somente exames de *Raio X, Sangue, gravidez e espermograma* podemos criar 4 caixas de checagem, sendo cada uma para um exame diferente. Com isto, poderíamos selecionar ou não o exame em vez de digitá-lo.

Alignment: Especifica se o texto ficara do lado esquerdo da caixa.



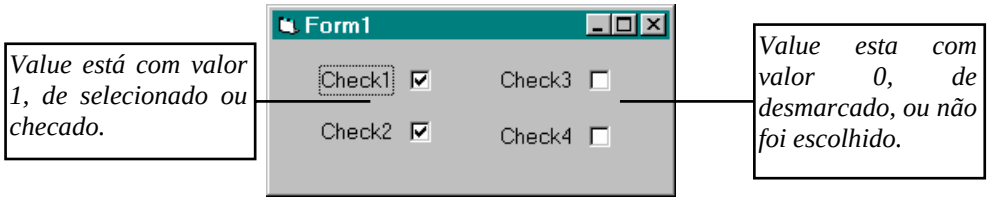
Caption: O texto anexado ao Objeto CheckBox.

Enabled: Habilita ou não esse Objeto. Estando desabilitado não aceitará que o usuário faça evento com a caixa de checagem.

Font: Escolhe uma fonte de letra para o texto digitado no Caption

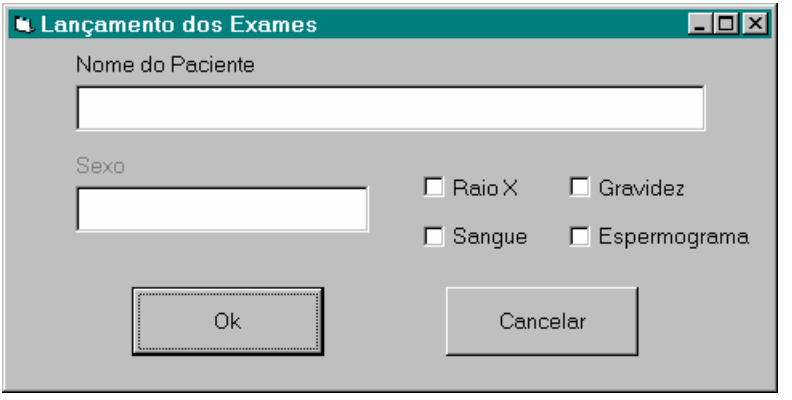
Name: Nomeia o Objeto. A inicial abreviada para este tipo de controle é “chk”.

Value: Determina o estado o objeto: 0 - Unchecked, 1 - Checked e 2 - Grayed.



- No exemplo "Lançamento de Exames", Elimine a caixa de texto "txtExames" e o label "lblExames". Em seu lugar coloque 4 caixas de checagem. Mude as seguintes propriedades:

Propriedade	**Check1**	**Check2**	**Check3**	**Check4**
Name	chkRaioX	chkSangue	chkGravidez	chkEspermo grama
Caption	Raio X	Sangue	Gravidez	Espermogr ama



- Pronto! Agora ficou mais fácil escolher o exame que o paciente quer fazer. Na codificação do programa, sempre que quisermos saber se algum desses exames foi escolhido é só verificarmos se a propriedade "Value" de cada um deles esta "Checked", ou seja, selecionado.
- Note que podemos marcar um exame, alguns, todos ou nenhum. Fica a critério do usuário. Mas no nosso exemplo vai ser difícil um paciente entrar em nosso laboratório e não fazer NENHUM exame. Se o nome dele esta sendo

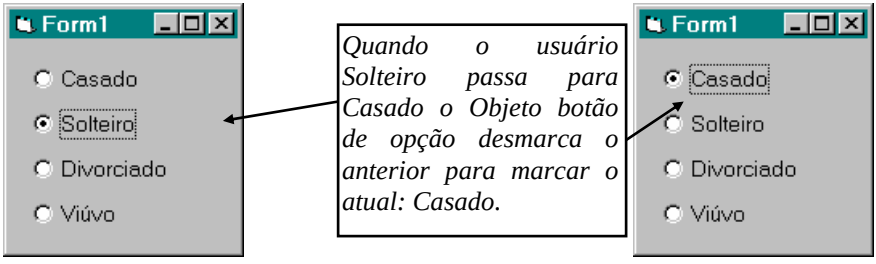
processado no computador é porque pelo menos um exame ele vai fazer. Depois vamos fazer uma rotina que verifica isto.

- Algumas outras alteração teremos que fazer também, como por exemplo não deixar o programa aceitar que um paciente do sexo masculino faça exame de Gravidez, ou do sexo feminino fazer Espermograma. Mas antes de trabalharmos nisto é necessário conhecermos mais um objeto importante.

2.1.8 Propriedades Principais do OptionButton

Este botão de opção é usado para o usuário poder escolher entre uma opção ou outra. Quando ele escolhe uma a outra é desmarcada. Diferentemente da Caixa de Checagem onde poderíamos ter várias marcadas ou nenhuma, o Objeto OptionButton exibe que somente uma opção seja marcada.

Por exemplo, precisamos criar um objeto em que o usuário coloque se uma determinada pessoa é Casada, Solteira, Divorciada ou Viúva.



Seja lá qual estado civil seja, quando mudar a outra deixa de existir. Uma pessoa não pode ser Casado e Solteiro ao mesmo tempo. Então neste caso o usual é o Botão de Opção, pois se colocássemos a Caixa de Checagem, o Objeto aceitaria que escolhêssemos dois ou mais estado civil.

Alignment: Determina se o texto ficará do lado direito ou esquerdo do botão de opção.

Caption: O texto que será anexado ao Objeto.

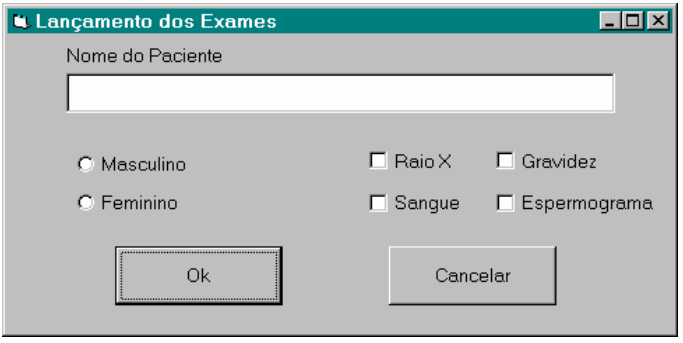
Enabled: Habilita ou não o objeto. Estando desabilitado o usuário não poderá selecionar a opção, e a cor do texto será acinzentado.

Name: O nomeia o Objeto. A inicial abreviada que se usa é “opt”.

Value: True para quando a opção esta selecionada e False para quando a opção não esta selecionada.

- Para nosso programa de "Lançamento de Exames" vamos trocar a caixa de edição onde o usuário digita o Sexo do paciente por dois botão de opção. Como só existe dois tipos de sexo e uma pessoa não pode ter mais de um sexo (pelo menos teoricamente!), nossa aplicação mais prática se usarmos esse objeto.
- Apague o "lblSexo" e o "txtSexo" e coloque em seu lugar dois Controles OptionButton.

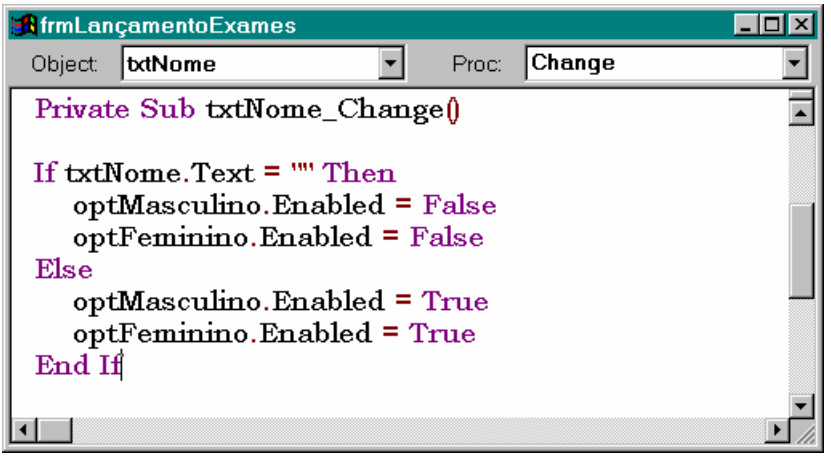
Propriedade	**Option1**	**Option2**
Name	optMasculino	OptFeminino
Caption	Masculino	Feminino



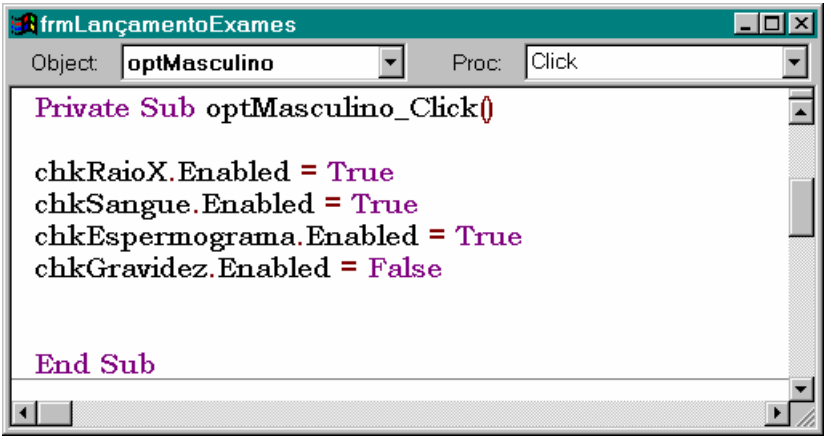
Os Botões de Opção quando são inseridos no formulário automaticamente são conjugados, de maneira que quando selecionamos um o outro é desmarcado. Não importa quantos sejam.

- Vamos agora ponderar alguns pontos: Se o usuário não digitar o nome do paciente primeiro, ele não poderão escolher o sexo, e se não escolher o sexo ele não poderá escolher a exame que o paciente fará, pois o tipo de exame necessita do sexo do paciente, para evitar que alguém do sexo masculino faça exame de gravidez.
- Então para corrigir esses problemas, vamos primeiro passar todas as propriedades "Enabled" dos Objetos OptionBottum e CheckBox para "False". Com isto, sempre que se entrarmos no formulário pela primeira vez esses objetos estarão indisponível para o usuário manipular até que ele cumpra algumas determinações.
- Vamos agora liberar a escolha do Sexo depois que um nome tiver sido digitado na Caixa de Texto "txtNome".

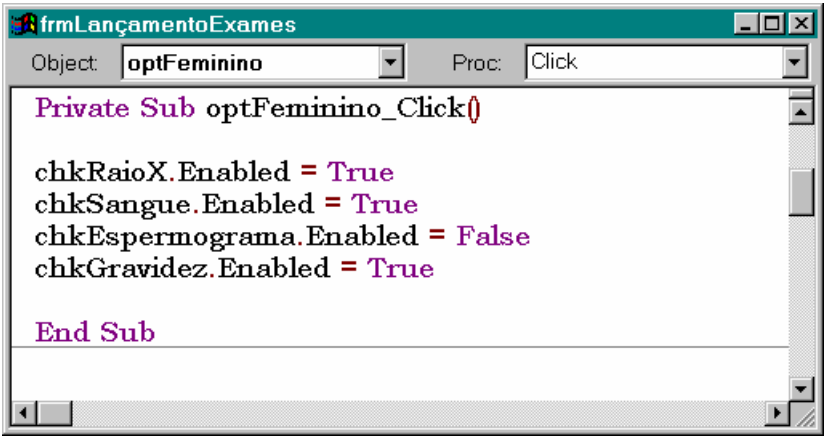
- Chame a janela de Codificação e na rotina "txtNome_Change()" altere para ter as seguintes linhas de comando:



- Rode o programa e teste para verificar se esta tudo correndo bem. Todas os objetos de escolha estão desabilitados até que um nome seja digitado. Neste momento a escolha do Sexo do paciente é liberado.
- Vamos agora fazer o mesmo com os Objetos "OptionButton". Vamos criar um evento Click para eles, de maneira que quando o usuário escolher um determinado sexo para o paciente seja liberado os exames correspondente.
- Dê dois clickes com o mouse no objeto "optMasculino". A janela de codificação será aberta e o evento Click já terá sido criado.
- Vamos então habilitar os exames para esse Objeto:



- Colocamos "Enabled = True" para os exames liberados para o objeto "optMasculino" e "Enabled = False" para os exames que não estão liberados.
- Faça a mesma coisa para o "optFeminino" agora.

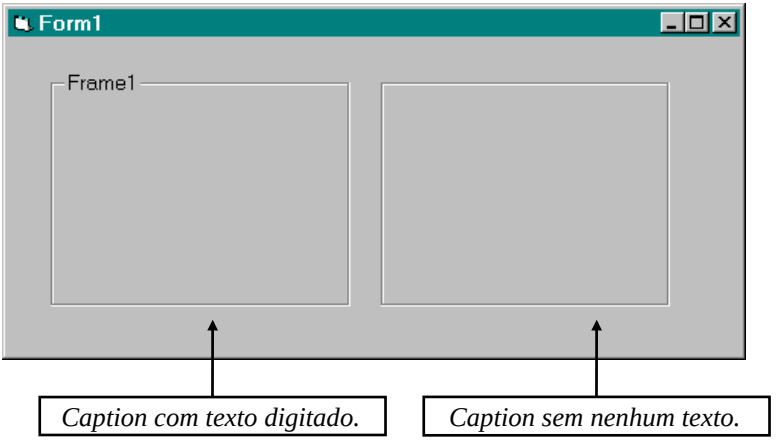


- Execute o programa. Esta funcionando perfeitamente. Tentamos tirar o máximo de possibilidades de erro que um usuário pode cometer.

2.1.9 Propriedades Principais do Frame

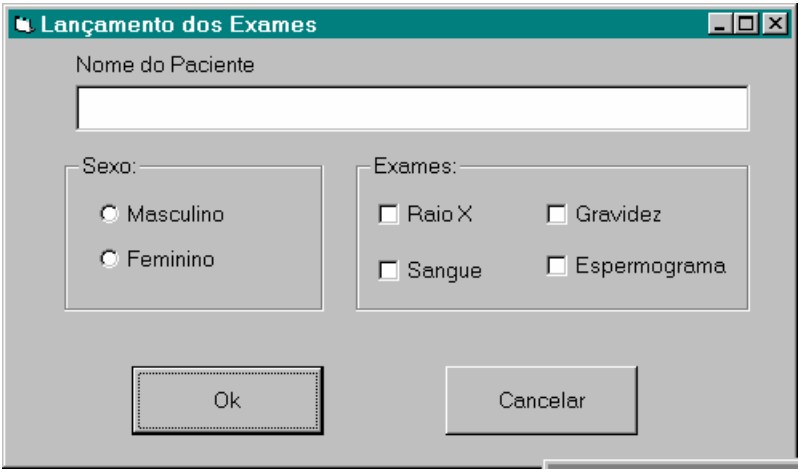
Esse objeto é muito útil para nos ajudar na criação de uma interface mais limpa para o usuário. Podemos criar molduras que agruparão outros objetos. Essas molduras possuem formato 3D embelezando ainda mais nossa janela.

Caption: Coloca um texto no topo da moldura. Não é obrigatório.



Enabled: Determina se todos todos os objetos colocados no interior da moldura estará ou não disponível para o usuário.
Name: Nome dado ao objeto. A inicial abreviada é “fra”. Cuidado para não colocar “frm” é a abreviatura do Objeto Form.

- Vamos fazer uma ultima alteração em nosso programa de “Lançamento de Exames”. Nossa interface está precisando de uns retoques, e para fazer esse incremento vamos colocar um objeto “Frame” envolvendo os “OptionButton” e os “CheckBox”.



- Perceba que a aparência de nosso programa melhorou muito agora. E ainda usamos a propriedade “Caption” do “Frame” para rotular as opções de escolha do sexo e do exame.
- Vale comentar que para que os objetos que estarão dentro do “Frame” seja realmente **anexados** a ele, é necessário que esses objetos sejam criados **dentro** dele. Assim, sempre que movermos a moldura, os Objetos acompanharam.
- Então, para que no nosso exemplo as coisas funcionem com perfeição, devemos apagar os objetos “OptionButton” e “CheckBox”, inserir as duas molduras em seus respectivos lugares e somente depois criar (dentro do “Frame”) os objetos apagados.

EXERCÍCIOS PROPOSTOS

1 - Explique a finalidade dos Objetos:

Form:

CommandButton:

Label:

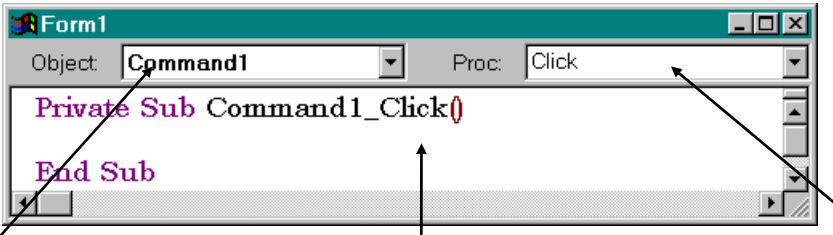
TextBox:

CheckBox:

OptionButton

Frame:

2 - Explique:



3 - Quais as abreviaturas que usamos para os Objetos:

Form:	
CommandButton:	
Label:	
TextBox:	
CheckBox	
OptionButton	

Frame:	

4 - Pela aparência descreva que tipo de Borda possui os formulários abaixo:

5 - Explique a finalidade das Propriedades do Formulário MouseIcon e MousePointer:

6 - Qual a importância da propriedade AutoSize do objeto Label?

7 - Descreva as propriedades do objeto TextBox relacionadas abaixo:

Font
Locked
Name
Text

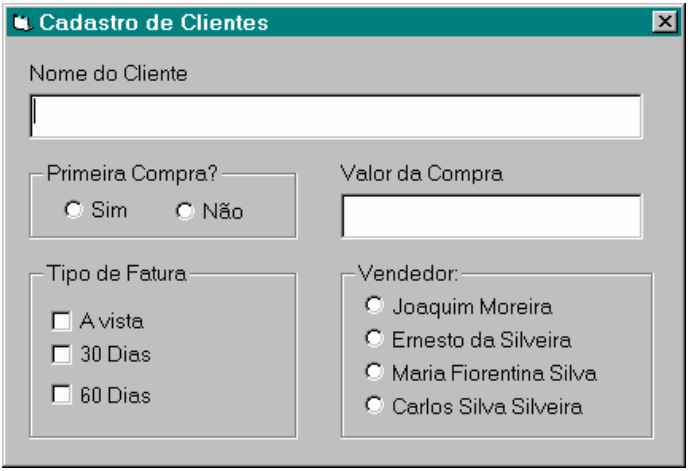
8 - O que é um Evento?

9 - Explique a diferença entre o Evento Change e o KeyPress do objeto TextBox:

10 - Em que situação podemos usar uma Caixa de Checagem e um Botão de Opção? Exemplifique.

11 - A propriedade Enabled existe em quase todos Objetos, e sua finalidade é:
( ) Deixar o Objeto invisível
( ) Mudar a cor do Objeto
( ) Fazer com que o Objeto não aceite interferência do Usuário.

12 - Crie um novo formulário que contenha as seguintes especificações:



Alguns pontos deste nosso simples programa tem que ser levado em consideração: Se for primeira compra o cliente pode comprar somente a vista. Se o valor da compra for maior que 200 reais ele poderá comprar

com 60 dias. Logicamente que no campo valor não poderá aparecer letras.

ANOTAÇÕES PARA NÃO ESQUECER

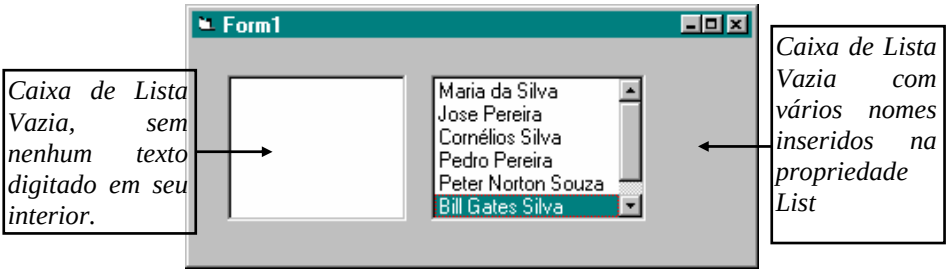
3 SELECIONANDO ITENS

- ListBox
- ComboBox

ANOTAÇÕES PARA NÃO ESQUECER

3.1 O OBJETO LISTBOX

Caixa de Lista é um objeto onde pode-se adicionar ou remover vários itens, e o usuário pode selecionar um ou vários destes itens.

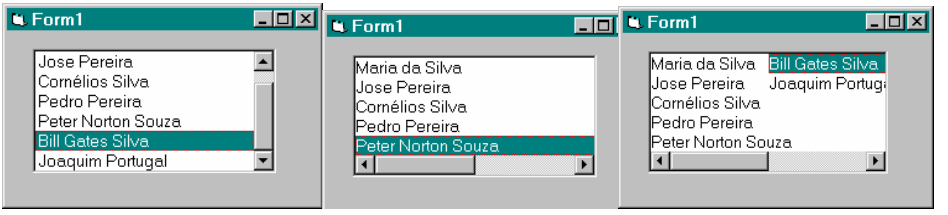


Este tipo de objeto é muito útil para mostrarmos uma relação de nomes ou códigos para se um (ou vários) seja selecionado, e sempre que a quantidade desses itens superar o tamanho do ListBox um ScroolBar Vertical automaticamente irá aparecer.

É bom ressalvar que este objeto, internamente, cria um índice começando com 0 (zero) até na quantidade existente de itens. Exemplo: se temos 4 nomes dentro do ListBox, “Maria, José, Cornélios, Pedro e Peter”, então o índice criado será 0 para Maria, 1 para José, e assim sucessivamente. Este índice será usado sempre que for referenciar aos nomes contido dentro do ListBox.

3.1.1 Propriedades Principais do ListBox

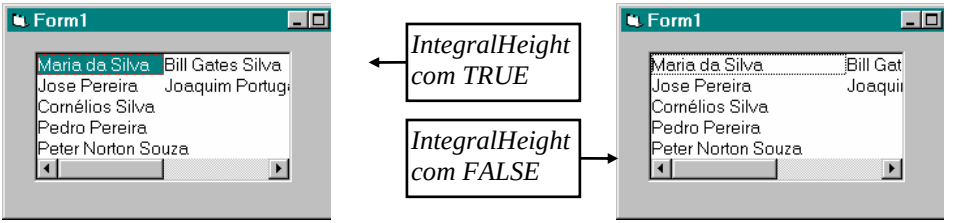
Columns: Determina a quantidade de colunas que a caixa de lista terá. Quando esta com 0 (zero) significa que terá somente uma coluna e a barra de rolagem será vertical. Se o valor desta propriedade for 1 será formado também somente uma coluna mas a barra de rolagem será horizontal. Valor 2 significa que os itens inseridos no ListBox serão ajustados em 2 colunas, e assim por diante.



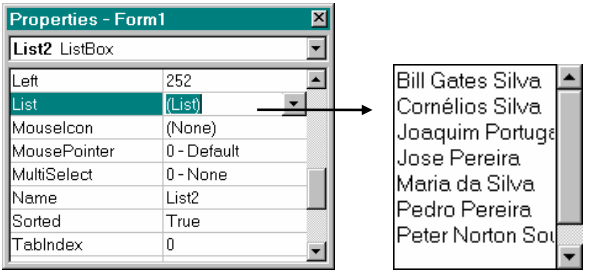
Columns ajustado para 0. Os nomes ficam em 1 coluna de forma vertical.	*Columns ajustado para 1. Os nomes ficam e 1 coluna de forma horizontal.*	*Columns igual a 2. Os nomes ficam dispostos em duas colunas de forma*

Enabled: Habilita ou não o ListBox para o usuário selecionar algum item no objeto.

IntegralHeight: Determina a possibilidade dos itens dentro da caixa de lista ser exibido de forma parcial.



List: é o Local onde digitamos os itens que estarão dentro do ListBox.

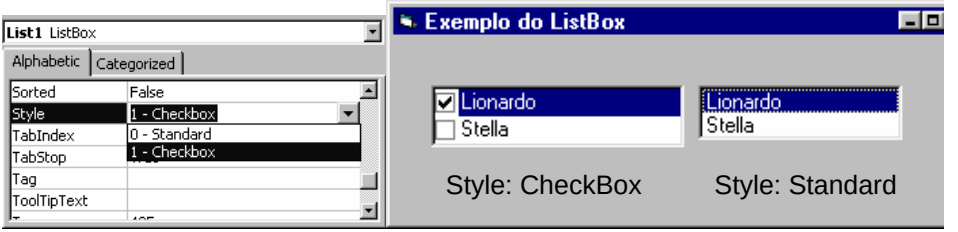


Os nomes digitados nesta propriedade em tempo de projeto são automaticamente inseridos dentro do ListBox, mas existe outro processo de se incluir dados no objeto em tempo de execução.

MultiSelect: Quando esta propriedade esta habilitada significa que a caixa de lista aceitará múltiplas seleções, ou seja, poderá ser selecionado mais de um ítem. As opções são 0 - None para seleção somente de 1 ítem. 1 - Simple para seleção de vários itens usando apenas o clicar do mouse ou barra de espaço. 2 - Extended é o padrão do Windows para multiplas seleções. Para selecionar mais de 1 item usa-se a combinação de tecla CTRL + Click do mouse ou barra de espaço.

Name: Nome que o Objeto ListBox terá. A abreviação padrão é "lst".

Sorted: Classifica os itens existente dentro do ListBox em ordem alfabética ou numérica ascendente. Esta propriedade em tempo de execução tem a finalidade de informar o estado que se encontra o Sorted.



Style: O Estilo Standard é o padrão, e o Visual Basic 5 acrescentou o Estilo

CheckBox, onde os itens existentes no ListBox são acompanhados de um quadradinho do lado esquerdo para se fazer a seleção dos itens. Para multiplas seleções esse tipo de ListBox é mais intuitivo para o usuário.

- Coloque um Objeto ListBox no formulário, e dê um nome para o objeto de “lstNomes”, e na propriedade “List” Digite os nomes contido na figura abaixo.

- Execute o programa e veja como ficará a disposição dos nomes digitados dentro do Objeto.
- Note que podemos selecionar somente um nome. Vamos tentar selecionar vários nomes. Habilite a propriedade MultiSelect com 2 - Extended. Rode o programa novamente e veja que agora vários nomes podem ser selecionado.
- Vamos agora colocar esses nomes em 2 colunas. Para isto a propriedade Columns deve conter o valor 2. Mude o IntegralHeight para False para melhorar a aparência. Em tempo de execução você verá o resultado.
- Mude a propriedade Sorted para True e todos os itens ficará na ordem alfabética.

3.1.2 Propriedades em tempo de execução

Existe algumas propriedades que não aparecem em tempo de projeto na janela de Propriedades, e podemos nos referenciar a elas somente através da janela de codificação, e serão executadas em tempo de execução.

ListCount: Retorna a quantidade de itens existente dentro de um ListBox.
ListIndex: Retorna o número correspondente ao índice do item selecionado.
NewIndex: Retorna o número correspondente ao índice do ultimo item inserido no ListBox.

SelCount: Quando a propriedade MultiSelect esta ativada, possibilitando a seleção de vários itens dentro da caixa de lista, o SelCount retorna a quantidade de itens que foi selecionado.

Selected: Retorna True ou False sinalizando se algum item foi selecionado. É necessário informar o índice correspondente. Exemplo: Selected(2): Se o item que possui o índice 2 for selecionado retornará True, se qualquer outro for selecionado retornará False.

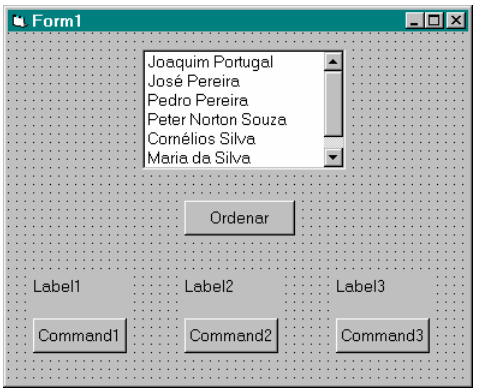
Text: Retorna o texto do item selecionado. Não necessita de índice.

List: Retorna o texto do item especificado no índice. Exemplo: List(2) irá mostrar o texto existendo na lista referente ao índice 2.

- No exemplo que criamos acima, um ListBox com nomes inseridos nele, vamos fazer algumas mudanças. Acrescente mais 3 botões no formulário.

Propriedadades	**Botão1**	**Botão2**	**Botão3**
Caption	Quantidade	Índice	Text
Name	cmdQuantidade	CmdÍndice	cmdText

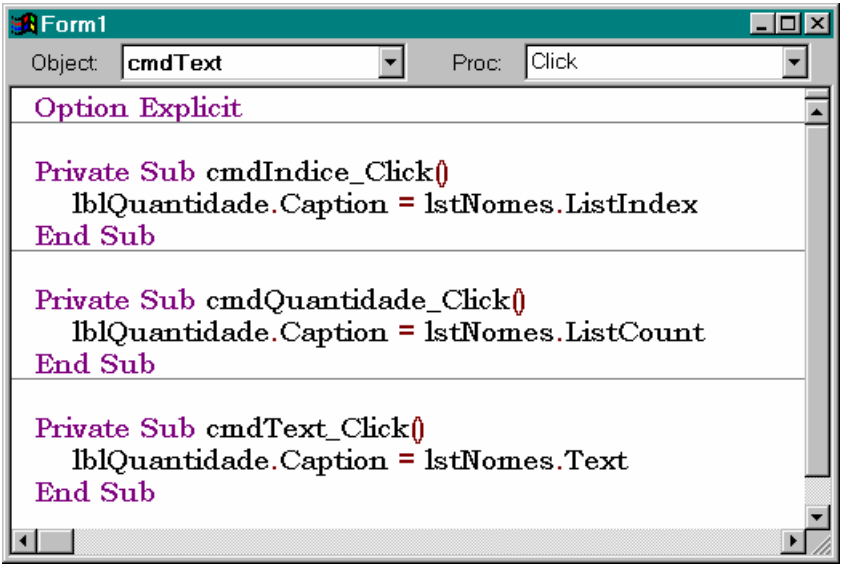
- Vamos Colocar 3 objetos Label no formulário. A disposição do formulário ficaria assim:



Propriedadades	**Label1**	**Label2**	**Label3**
AutoSize	True	True	True
Caption	“deixar vazio”	“deixar vazio”	“deixar vazio”
Name	lblQuantidade	LblÍndice	lblText

- Vamos agora codificar o programa para quando o usuário apertar o botão “Quantidade” aparecer a quantidade de

itens dentro da caixa de Lista. Quando apertar o botão "Índice" mostrar o índice correspondente ao item selecionado. O botão Texto irá mostrar o texto selecionado dentro do ListBox. Veja os Códigos a serem usados:



- Note que as propriedades estão retornado valores que estão sendo inseridos no Objeto lblQuantidade e na sua propriedade Caption em tempo de execução.

3.1.3 Eventos do ListBox

Somente um evento do ListBox veremos agora: O DblClick. Este evento executará alguma ação quando dermos duplo click em algum item de nossa caixa de Lista.

Não use o Click! Para este tipo de objeto é mais intuitivo usar o Duplo click que o click. Enquanto o usuário esta escolhendo qual item selecionar, ele ficará "passeando" com o mouse entre os itens existente, dando click em algum deles e clicando no ScrollBar.

3.1.4 Métodos AddItem, RemoveItem e Clear

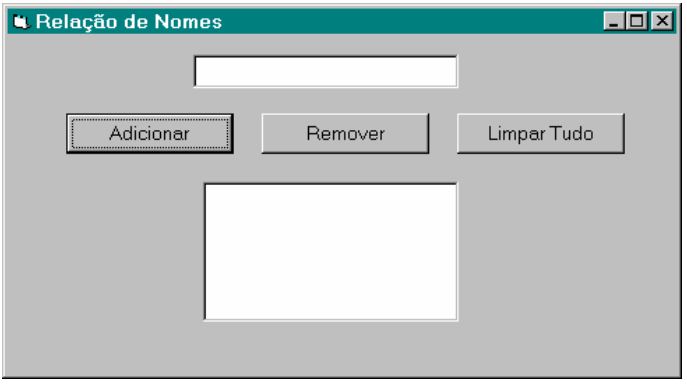
No exemplo dado anteriormente digitamos os itens que iriam compor o ListBox através da propriedade List em tempo de projeto. Mas nem sempre acontece de sabermos quais os itens que farão parte do objeto antes de sua execução. O mais natural é que esses itens sejam inseridos durante a execução. Para que isto aconteça temos que usar o Método AddItem.

O Método RemoveItem retira do ListBox o item selecionado, e o Clear limpa todo o Objeto.

- Vamos criar um formulário novo (Menu File/New Project).
- Dentro do formulário inserir três botões de comando, uma caixa de lista e uma caixa de texto.

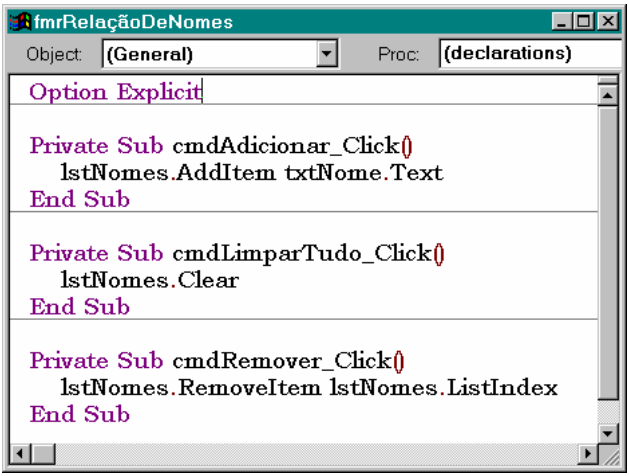
Propriedadades	Botão1	Botão2	Botão3
Caption	Adicionar	Remover	Limpa Tudo
Name	CmdAdicionar	cmdRemover	CmdLimpaTudo

Propriedadades	Caixa de Lista	Caixa de Texto
Name	LstNomes	txtNome
Text		"Deixar Vazio"



As propriedades do formulário será "Relação de Nomes" para o Caption e frmRelaçãoDeNomes para o Name. Veja ao lado como ficará a janela.

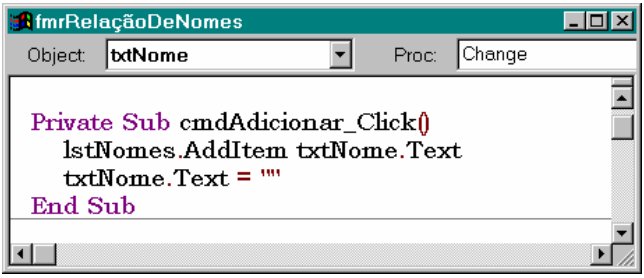
- O programa funcionará da seguinte forma: Digita-se um nome na Caixa de Texto e ao teclar o botão "Adicionar" o conteúdo será transferido para a Caixa de Lista. Quando quisermos remover algum item, selecionamos e apertamos o botão "Remover". O botão "Limpar Tudo" remove todo o conteúdo do ListBox.



O método AddItem requer na sua sintaxe o texto que será adicionado ao ListBox. Chamamos então o objeto txtNome e sua propriedade Text que contém o texto.

O método RemoveItem necessita do número do índice que o item existente dentro ListBox possui. Para se obter este índice usamos a propriedade ListIndex.

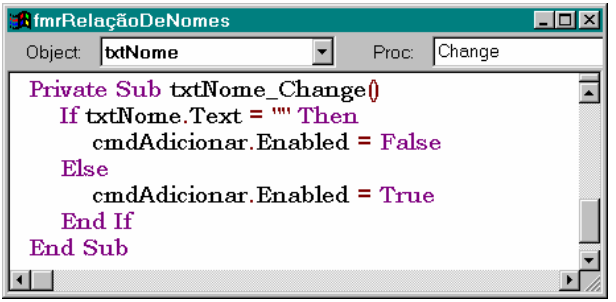
- Execute o programa e digite alguns nomes, adicionando, removendo alguns e depois limpando tudo.
- Alguns problemas apareceram: ❶ Sempre que adicionamos algo o ultimo nome digitado na caixa de texto continua lá. ❷ Do jeito que o programa esta ele aceita adicionarmos nomes em branco. ❸ Se apertarmos o botão "Remover" sem ter um nomes no ListBox selecionado ele retorna um erro de execução, pois a propriedade ListIndex retorna o índice do item selecionado. Ou seja, tem que haver uma seleção do item que se quer apagar. ❹ A tecla ENTER não é aceita na confirmação dos nomes digitados.
- Nossa! Quantos problemas. Mas vamos então resolver todos.



- O Objeto txtNome tem sua propriedade Text adicionada ao lstNomes, e em seguida é igualada a um conteúdo vazio.

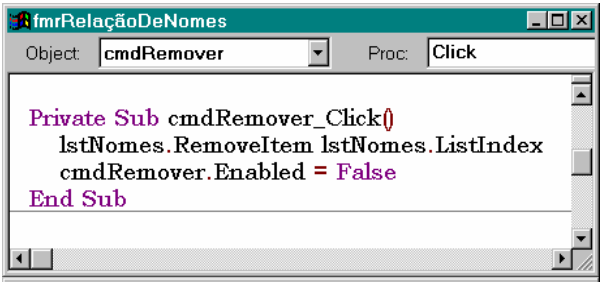
- Resolvemos o primeiro problema. Rode o programa é comprove isto! Agora temos que desabilitar o botão "Adicionar" e somente deixá-lo habilitado quando o usuário digitar algo na caixa de texto (objeto txtNome). Sendo assim não corremos o risco de algo vazio ser inserido na caixa de lista.

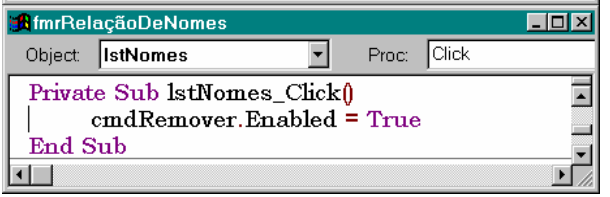
- Primeiro passo será selecionarmos em tempo de projeto o Objeto "cmdAdicionar". Vá na propriedade "Enabled" e passe para "False".
- Crie agora um evento "Change" para a caixa de texto "txtNome":



- Veja que na codificação verificamos se o txtNome esta vazio. Se tiver desabilita o botão Adicionar, e não tiver vazio habilita o botão.

- Usamos o Evento "Change" porque ele é o responsável pela verificação constante do que foi digitado na caixa de texto. Sempre que digitamos, alteramos ou apagamos algo numa caixa de texto o evento "Change" é chamado.
- Note agora que se começarmos a digitar um nome qualquer na caixa de texto o botão aparece. Se apagarmos o conteúdo digitado, com delete ou backspace, o botão é desabilitado.
- Para resolver o terceiro problema, teremos que primeiro somente aceitar o método "RemoveItem" se houver um item selecionado. Se não houver ele não executa esse método. Então vamos a luta!
- Em tempo de projeto mude a propriedade "Enabled" do botão "Remover" para False. Iremos habilita-lo somente quando um item for selecionado usando o evento click do objeto lstNomes.



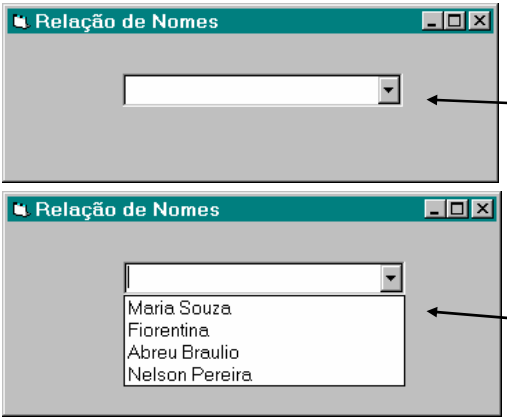


- No evento Click do objeto lstNomes é solicitado ao programa para habilitar o botão remover. Depois, quando o usuário apertão o botão e remover o item pretendido, colocamos o botão remover novamente desabilitado.

- Rode o programa e veja o que acontece.
- Para o usuário que esta digitando é desconfortaste ter que digitar um nome e ter que usar o mouse (tirar a mão do teclado) para inserir um nome e depois clicar novamente na caixa de texto para digitar outro nome. Para resolver esse problema é simples. Selecione o botão "Adicionar" e mude a propriedade "Default" dele para "True". Assim sempre que o usuário digitar um nome e teclar ENTER será como se tivesse apertado o botão "Adicionar".
- Agora o programa está pronto! Pode vendê-lo!

3.2 O OBJETO COMBOBOX

Este objeto é na verdade uma combinação da caixa de texto e da caixa de lista. Uma diferença que este Objeto possui em relação a caixa de lista é que ele não aceita multiseleção. Somente é acessado um item de cada vez.



Objeto ComboBox com sua caixa de Lista recolhida. Ela somente é acessada ao darmos um click com o mouse na seta para baixo do lado direito. Da forma em que esta podemos digitar na caixa de texto apresentada.

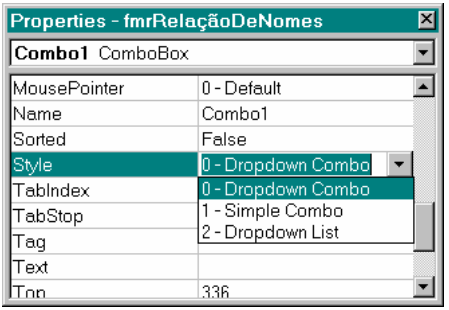
ComboBox com a caixa de lista expandida. Pode-se escolher um nome qualquer na lista que ele será automaticamente levado até a caixa de texto.

Uma das grandes vantagens da Caixa de Combinação é a economia de espaços na tela. Podemos colocar uma relação enorme de itens dentro de uma caixa de combinação que ela será apresentada somente com uma caixa de texto, e a relação aparece somente se o usuário necessitar, digitando assim na seta para baixo do Objeto.

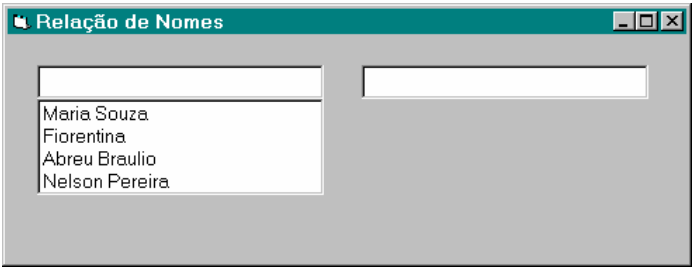
3.2.1 Propriedades Principais do ComboBox

O ComboBox usa as mesmas propriedades que aprendemos para o ListBox. A diferença esta somente em duas que veremos agora.

Style: Aqui escolhemos o tipo de Caixa de Combinação iremos colocar no formulário:



0 - Dropdown Combo: é a opção padrão do Objeto. Aqui pode-se digitar qualquer nome na área de digitação, clicar a seta para baixo e escolher qualquer um dos itens que ele será automaticamente inserido na área de texto.
1 - Simple Combo: Caixa de Combinação simples. Aparece em destaque a área de edição de texto, onde podemos digitar algum item; ou selecionar qualquer um que esteja na caixa de lista, que será inserido na área de texto. O botão em forma de seta para baixo não existe neste tipo de ComboBox. Se aumentarmos o tamanho da Caixa de Combinação na vertical, aparecerá a lista e esta ficará fixa. Caso deixamos ela somente do tamanho da área de texto, então a lista não aparecerá, e se quisermos saber quais nomes existe teremos que apertar no teclado a seta para baixo ou para cima, para que os itens existentes dentro da Lista apareçam.

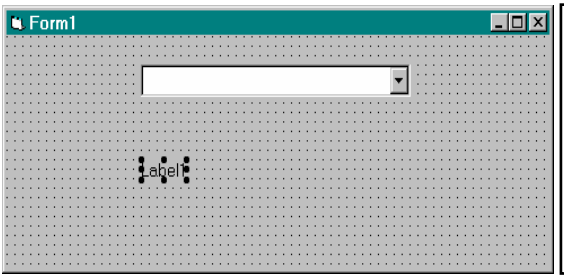


2 - Dropdown List: Neste tipo de Caixa de Combinação o usuário pode somente escolher um item relacionado na lista, não podendo digitar nada. A área de texto não aceitará digitação.
Text: Nesta propriedade digita-se um texto que ficará, como padrão, fixo na área de texto na caixa de combinação. Geralmente deixa-se em branco. Podemos usar essa propriedade também, em tempo de execução, para saber qual texto o usuário digitou ou selecionou.

3.2.2 Os Métodos

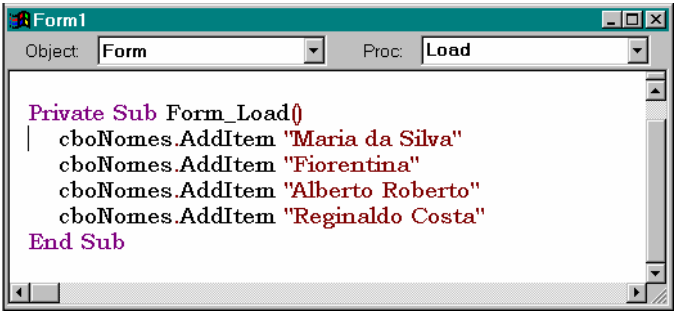
Usamos para o ComboBox os mesmos métodos usados para o ListBox: AddItem, RemoveItem e Clear. O modo de manipulá-los na janela de codificação também é o mesmo.

- Crie um formulário novo e coloque nele um ComboBox e um label conforme figura abaixo:

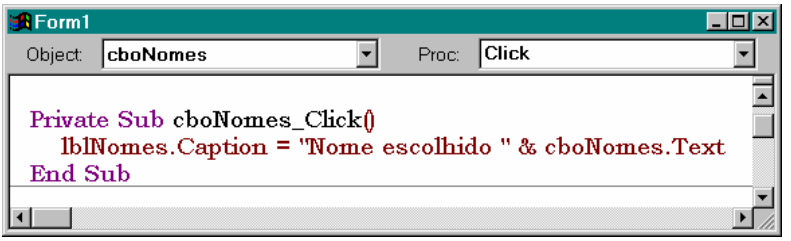


- nomeie a Caixa de Combinação para cboNomes e deixe a propriedade Text vazia. O label nomei-o para lblNomes, o Caption deixe vazio e o AutoSize como True.

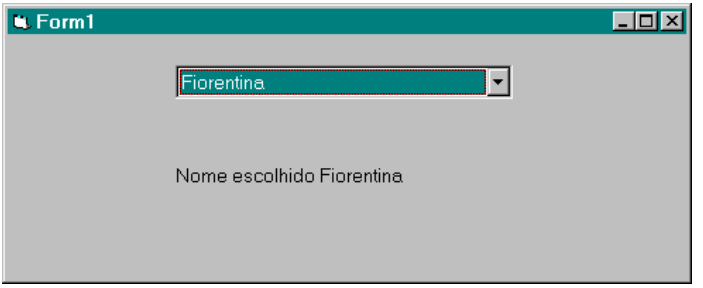
- Crie um evento "Load" para o formulário da seguinte forma:



- O Evento "Form_Load" do formulário é o primeiro que é lido pelo programa assim que a janela aparece na tela em tempo de execução. Tudo que for codificado para este evento será, então, executado antes da leitura pelo programa de qualquer objeto inserido no formulário.
- Defina a propriedade "Style" do cboNomes como 2. Assim o usuário não poderá digitar nenhum nome, somente escolher na lista.
- Crie um evento "Click" para o cboNomes:



- Estamos solicitando ao programa para quando o usuário der um click na lista de nomes aparecer a mensagem que ele escolheu determinado nome. O sinal de "&" que aparece na codificação é o responsável pela concatenação (junção, emenda)da expressão "Nome escolhido" com o conteúdo de "cboNomes.Text".



EXERCÍCIOS PROPOSTOS

1 - Descreva as seguintes propriedades do ListBox:
List:

Sorted:

MultiSelect:

Selected:

ListCount

ListIndex:

2 - Qual a diferença entre as propriedades em tempo de execução e as em tempo de projeto.

3 - Marque Verdadeiro ou Falso para as afirmações:
a) As propriedades existentes em tempo de projeto não é possível usá-las em tempo de execução. ( )
b) A propriedade NewIndex retorna o ultimo índice entrado na caixa de Lista. ( )
c) A propriedade Name e Caption possuem o mesmo valor. ( )

4 - Explique os métodos:
AddItem:

RemoveItem:

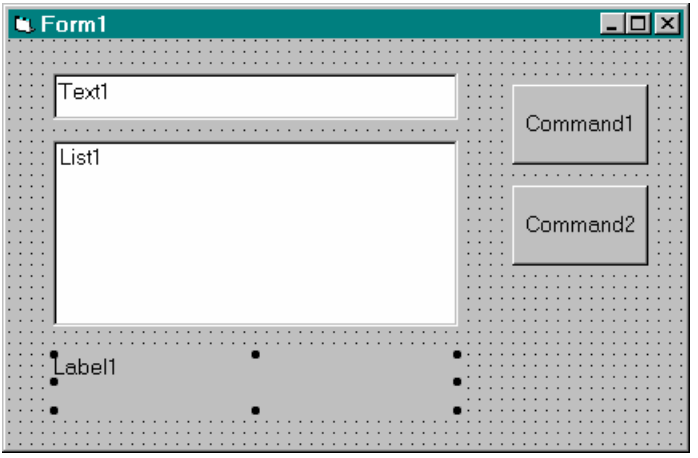
Clear:

5 - Quais são as diferenças entre o ListBox e o ComboBox?

6 - Na propriedade Style quais os tipos existentes:

7 - Explique o evento Form_Load:

8 - Desenvolva um cadastro de mercadoria, onde o usuário digitará a descrição das mesmas (TextBox) e elas serão incluídas dentro de uma lista (ListBox). Crie os botões Adicionar e Remover. Na mesma janela, quando o usuário selecionar qualquer das mercadorias dentro da Caixa de Lista, dando duplo click, a descrição da mercadorias será mostrada num label. Veja a disposição da janela:



4 O FOCO

- TabIndex
- GotFocus e LostFocus
- SetFocus
- Mnemônico

ANOTAÇÕES PARA NÃO ESQUECER

4.1 O Foco

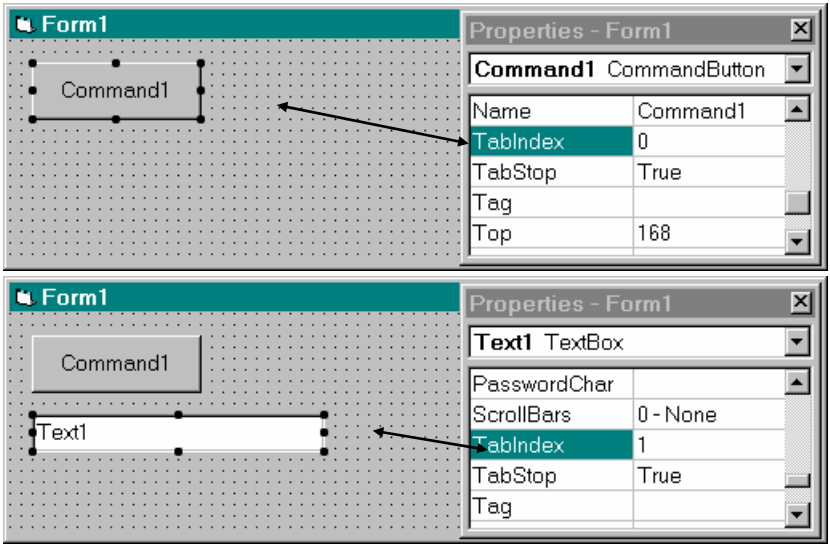
Foco é o termo usado para descrever o objeto que esta em destaque no momento. O objeto que esta sendo usado pelo usuário. Um objeto pode perder ou ganhar o foco. Quando estamos manipulando-o ele ganha o foco, se apertamos a tecla TAB ou clicarmos com o mouse em outro objeto o foco é transferido para outro objeto, ou seja, um ganha o foco e outro perde o foco.

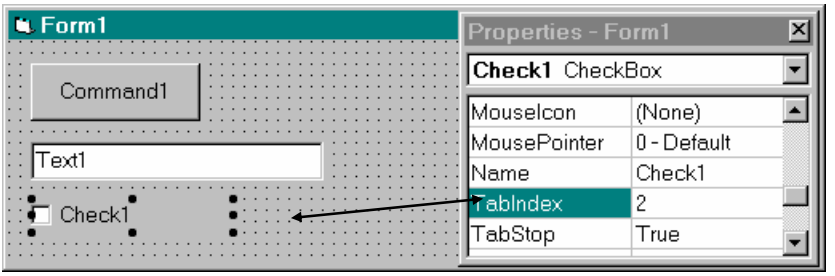
A seqüência que a tecla TAB usa para dar foco aos objetos no formulário é a ordem que eles foram inseridos no formulário. Se você inseri um botão de comando e depois insere uma caixa de texto, então quando executarmos o programa o primeiro a receber o foco será o botão de comando. Quando apertar a tecla TAB o foco será então transferido para a caixa de texto. Podemos também passar o foco para um determinado objeto usando o mouse.

4.1.1 Propriedades TabIndex e TabStop

Estas duas propriedades existem na maioria absoluta dos objetos, e com elas podemos manipular a ordem de tabulação (passagem do foco) entre os controles existentes no nosso formulário.

A propriedade TabIndex possui uma seqüência numérica que representa a ordem de tabulação dentro do formulário. O primeiro objeto inserido possui sempre o numero 0, e assim que outros são também inseridos este número vai sendo incrementado em mais um, independente do objeto inserido.



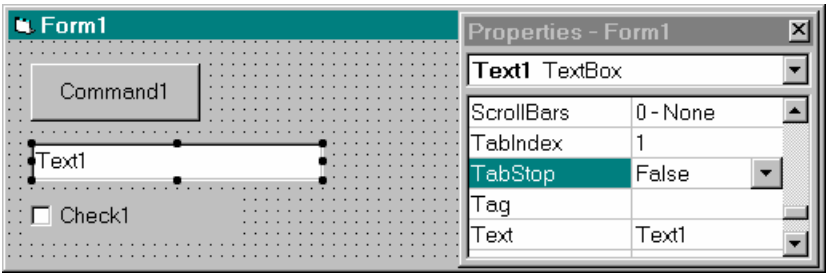


Neste exemplo, quando o programa é executado e apertamos a tecla TAB o foco é passado do botão de comando para a caixa de texto e depois para a caixa de checagem. Mas se quisermos alterar esta ordem é só mudar o numero existente na propriedade TabIndex.

Vamos passar a ordem de tabulação destes objetos para: Botão de comando, caixa de checagem e depois caixa de texto.

Para fazer isto basta alterar a propriedade da caixa de texto para 1.

A propriedade TabStop possui a finalidade de fazer a tecla Tab ignorar o controle na tabulação. Ou seja, o objeto que tiver TabStop = False, não receberá o foco via teclado, somente se o usuário clicar com o mouse no objeto.



Quando o objeto esta com a propriedade Enabled = False ou Visible = False, o foco não passa por eles, nem via teclado nem via mouse.

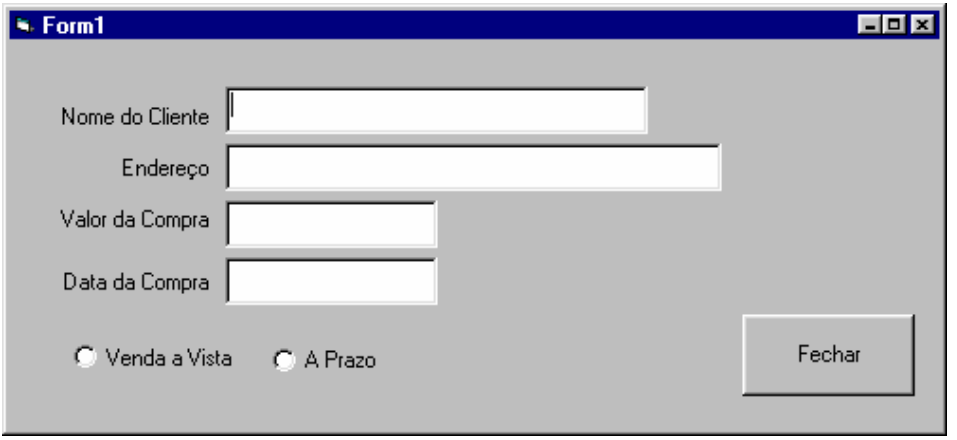
O Objeto Label não possui foco, pois ele não interage com o usuário.

4.1.2 A Tecla Enter

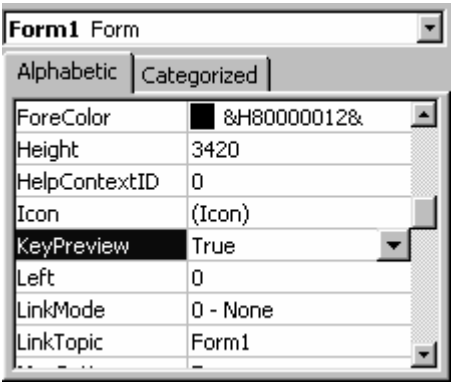
O Windows possui a tecla TAB para mudar o posicionamento do Foco, porém a grande massa de usuários que sempre acostumou usar a tecla ENTER para passar o foco de um objeto para outro não se adaptou ainda. Se você quizer fazer um programa que habilite a funcionalidade da tecla ENTER para mudança de foco, aqui vai a dica:

PRESTE ATENÇÃO

- Crie um formulário e acrescente nele algumas caixas de texto como no exemplo a seguir:



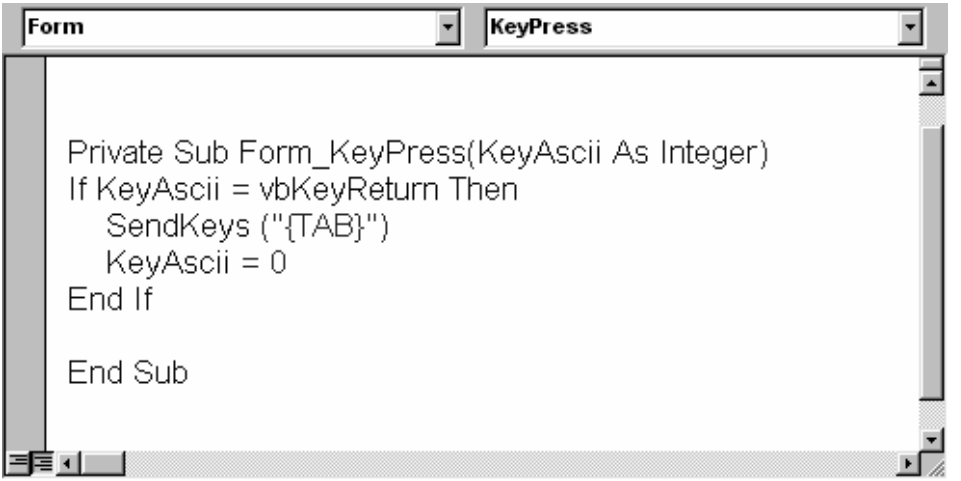
- A propriedade TabIndex desses objetos devem estar na ordem crescente. O foco ira acompanhar exatamente o que for inserido nesta propriedade. Se colocar o botão Fechar com o TabIndex 1, ele sera o segundo a receber o foco.
- Defina a propriedade KeyPreview do formulário para verdadeiro.



- Com isto, o programa dara prioridade aos eventos de teclado do formulário como um todo antes de verificar os eventos de teclado de cada objeto inserido no **form**.

- Uma vez feito isto, precisamos agora criar o evento de teclado para o formulario. Entre na janela de codificação e chame o evento **KeyPress** do **Form.** Repetindo: Do **Form**!

- Os codigos digitados neste evento sera avaliado a cada pressionamento de tecla. Criamos então a seguinte codificação:



O Programa verifica se foi pressionado a tecla ENTER que é referenciada aqui pela constante VbKeyReturn. Se foi pressionado é chamado a função SendKeys que força o pressionamento da tecla TAB, ou seja, seria como se o programa dissesse para a máquina: Se o usuário apertar a tecla ENTER troque pela tecla TAB.

- Por fim usamos "KeyAscii = 0" para cancelar o Beep que é emitido sempre que a tecla ENTER é pressionado.

4.1.3 Método SetFocus

O método SetFocus move o foco para o objeto especificado. Veja sua sintaxe:

NomeDoObjeto.SetFocus

NomeDoObjeto: Um objeto Form que representa um formulário, ou um objeto Control que representa um controle no formulário ou folha de dados ativo

Use o método SetFocus quando você quiser que um determinado campo ou controle tenha o foco, para que toda a entrada do usuário seja direcionada para esse objeto. De modo a ler algumas das propriedades de um controle, você precisa assegurar que o controle tenha o foco. Algumas propriedades só podem ser definidas quando o controle não tem o foco. Por exemplo, você não pode definir as propriedades Visible ou Locked de um controle como False quando esse controle tem o foco.

Você não pode mover o foco para um controle se sua propriedade Enabled estiver definida como False. Você precisa definir a propriedade Enabled de um controle como True antes de poder mover o foco para esse controle. Você pode, porém, mover o foco para um controle se sua propriedade Locked estiver definida como True.

4.1.4 Eventos GotFocus e LostFocus

GotFocus : Ocorre quando um formulário ou controle recebe o foco.
LostFocus : Ocorre quando um formulário ou controle perde o foco.

Esses eventos ocorrem quando o foco é movido em resposta a uma ação do usuário, como o pressionamento da tecla TAB ou o clicar no objeto, ou quando você usa o método SetFocus no Visual Basic.
Um controle só pode receber o foco se suas propriedades Visible e enabled estiverem definidas como True. Um formulário só pode receber o foco se não tiver controles ou se todos os controles visíveis estiverem desativados.

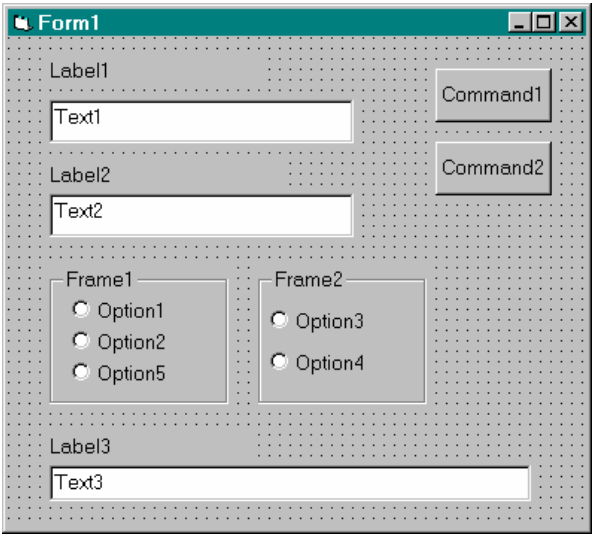
Você pode especificar o que acontece quando um formulário ou controle recebe o foco executando um procedimento de evento quando o evento GotFocus ocorrer. Por exemplo, anexando um procedimento de evento GotFocus a cada controle de um formulário, você pode guiar o usuário através de seu aplicativo exibindo breves instruções ou mensagens em uma caixa de texto. Você também pode oferecer indicações visuais ativando, desativando ou exibindo controles que dependam do controle que tem o foco.

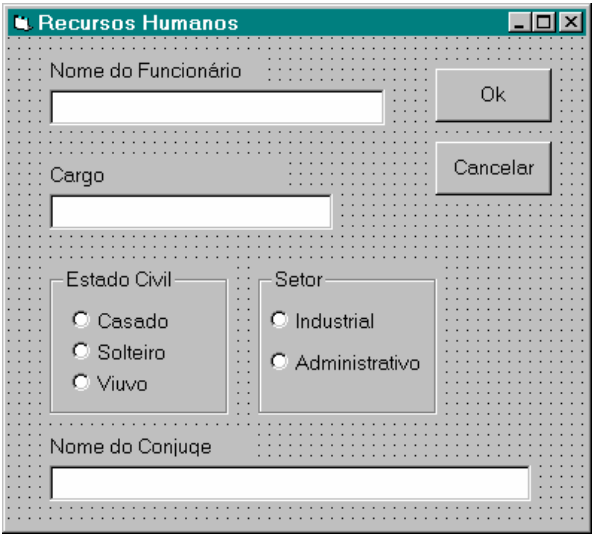
Também Pode-se usar um procedimento de evento LostFocus para validar dados inseridos conforme o usuário move o foco de um controle. Você

também pode reverter ou alterar as condições que você define no procedimento de evento GotFocus do objeto. Outros usos para os procedimentos de evento LostFocus e GotFocus são ativar, desativar, ocultar e exibir outros objetos.

PRESTE ATENÇÃO

- Crie um novo projeto e coloque no formulário os objetos relacionados abaixo:



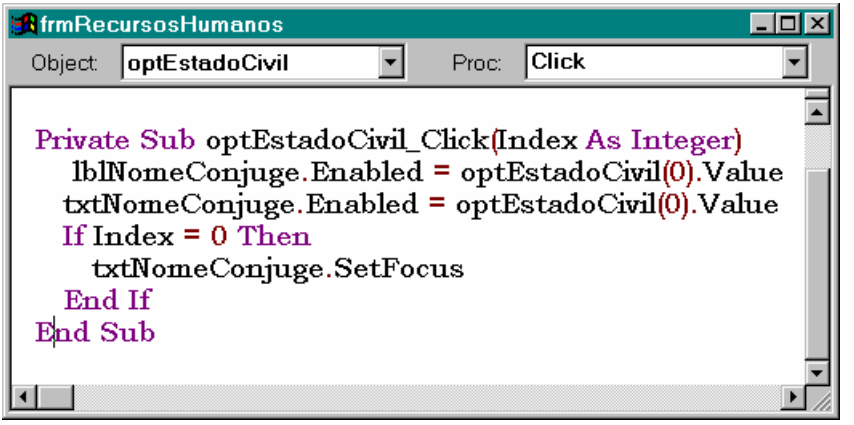


Formulário:
Caption:Recursos Humanos
Name:frmRecursosHumanos
Caixa de Texto 1:
Text: <deixar vazio>
Name:txtNomeFuncionário
Caixa de Texto 2:
Text: <deixar vazio>
Name:txtCargo
Caixa de Texto 3:
Text: <deixar vazio>
Name:txtNomeConjuge
Enabled = False
Label 1:
Caption:Nome do Funcionário
Label 2:
Caption:Cargo
Label 3:
Caption: Nome do Cônjuge
Enabled: False
Name: lblNomeConjuge
Frame 1:
Caption:Estado Civil
Frame 2:
Caption: Setor

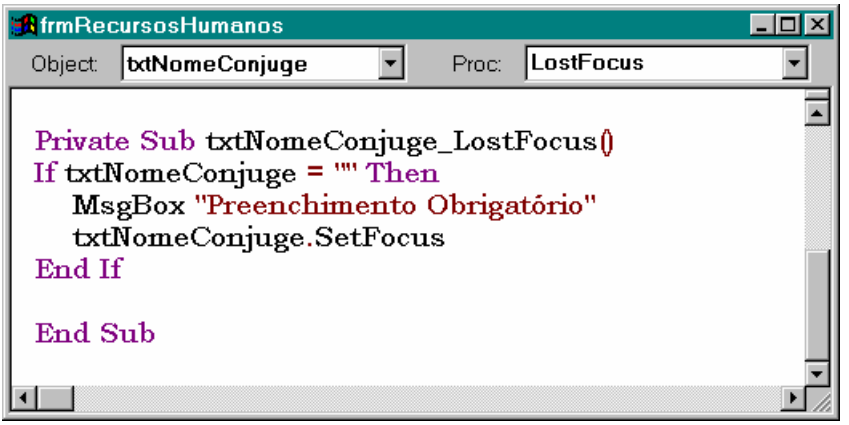
- Os botões de opção OptionButton para estado civil, dê o nome de optEstadoCivil para todas as três opções. Com isto

o Visual Basic perguntará se você esta criando um array de controle, e pode responder que sim. Fazendo isto, criamos uma matriz de nome optEstadoCivil com 3 elementos, sendo que o elemento 1 representa a opção casado, elemento 2 representa opção Solteiro e elemento 3 opção Viuvo.

- A ordem de tabulação é a seqüência dos objetos dispostos no formulário. Os dois botão de comando deve ter o TabStop = False, pois não serão incluídos na tabulação.
- A caixa de texto txtNomeConjuge esta desabilitada, e somente poderá ser habilitada se o estado civil do funcionário for casado. Para isto a seguinte codificação para o objeto optEstadoCivil deve ser feito:



- Note que usamos o SetFocus para, no caso do usuário escolher a opção Casado, o foco ir para a caixa de texto txtNomeConjuge.
- Para entender melhor esse evento, veja que o Visual Basic criou automaticamente um argumento chamado Index do tipo integer. O conteúdo deste argumento é o número do índice correspondente a opção da matriz optEstadoCivil.
- Vamos criar um evento LostFocus para txtNomeConjuge:



- Este evento é acionado sempre que o objeto perde o foco, ou seja, codificamos para no momento que tiver perdendo o foco verificar se o nome do cônjuge foi preenchido. Se não foi da uma mensagem avisando que é obrigatório o preenchimento e retorna o foco para a caixa de texto para que o nome seja então digitado.
- Rode o programa e veja o resultado.
- Colocamos o método SetFocus dentro de um evento LostFocus para demostrar sua utilização, mas esta é uma prática não recomendada, pois imagine se você clica na opção "casado" por engano, e quer voltar para a opção solteiro (que seria o correto). O programa não deixaria, pois exigiria a digitação do nome do cônjuge.

4.1.5 Mnemônico (Tecla de Acesso)

Mnemônico é a letra sublinhada onde, via teclado, podemos fazer um acesso diretamente ao objeto, apertando a tecla ALT e a letra correspondente.

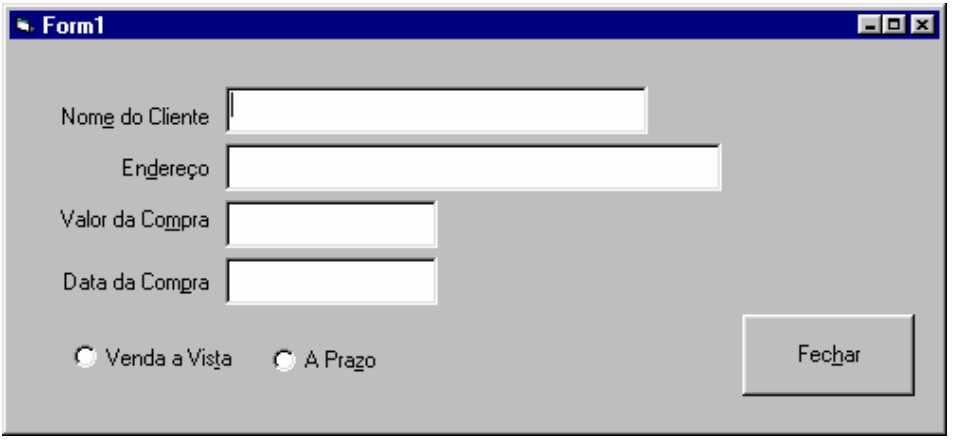
Para criar um Mnemônico usamos a propriedade Caption dos objetos, e anexado a letra escolhida digitamos o símbolo "&" (sempre do lado direito da letra). Por exemplo, se temos um botão de comando "Cancelar" e queremos que a letra "n" seja a Mnemônica, então no Caption alteramos para "Ca&ncelar". Assim aparecerá no vídeo como : "Cancelar", e se o usuário usar a combinação de tecla ALT+N passará o foco para este objeto e o acessará.

Este recurso vale para todos os objetos que possuem a propriedade Caption, mas é bom lembrar que alguns objetos não faz sentido colocar o Mnemônico, como por exemplo para um formulário.

Sempre que possível use as mesmas teclas de acesso em todo seu programa. Se numa janela você tem o botão “Cancelar” com o mnemônico na letra “n”, então sempre que este botão aparecer em outras janelas tente repetir a tecla de acesso usada.

Outra dica: Não use tecla de acesso para opções perigosas, como por exemplo um botão “Apaga Tudo”. Se colocarmos uma tecla de acesso ALT+P nele corremos o risco de do usuário apertar esta combinação por engano e limpar tudo. Estas opções delicadas sempre é bom obrigar o usuário a usar o mouse ou a tecla TAB para ele ter certeza do que esta fazendo.

- Usando o exemplo anterior, vamos colocar teclas de acesso nos objetos:



- Perceba que agora é só apertar a tecla ALT e a letra sublinhada para que o objeto seja acessado. Entretanto para que o programa combine o objeto **label** no objeto mais próximo, que no nosso exemplo são as caixas de texto, é necessário que a ordem do TabIndex esteja rigorosamente crescente.
-

5 CONTROLES ESPECIAIS

- MaskEdBox
- CommonDialog

Alguns controles do Visual Basic necessitam de uma explicação mais profunda devida a sua grande utilidade dentro da linguagem de programação. Habitualmente estes controles são pouco tratados em literaturas sobre o Visual Basic, e devido a isto vamos aprofundar mais sobre eles aqui.

5.1 MaskEdBox

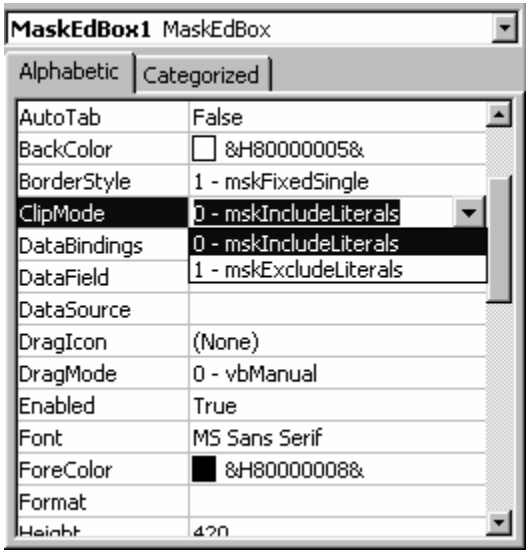
Este objeto é semelhante a uma caixa de texto, entretanto ele possui alguns recursos adicionais, como a possibilidade de colocar uma máscara para o texto que irá ser digitado e validar a digitação automaticamente.

Vejas as principais propriedades:

AllowPrompt : Determina se o caractere informado como prompt é válido durante a digitação.

AutoTab : Determina se quando o usuário terminar de preencher a mascara do objeto o foco é automaticamente passado para o objeto seguinte, sem necessidade do usuário apertar TAB ou o mouse.

ClipMode : Determina se, diante de um evento de copiar ou recortar dados do objeto Maskedit para a área de transferência, devem ser enviados os dados digitados com os caracteres que compõem a máscara ou não.



* **ClipText** : Retorna o texto digitado no objeto sem os caracteres que compõem a máscara.

Format : Determina o formato que os dados serão exibidos. Seque os mesmos padrões estabelecidos para a função Format (veja no capítulo Funções Auxiliares)

Use a propriedade Format para exibir dados em um formato consistente, ou seja, os dados serão exibidos neste formato, mesmo que o usuário digite os dados diferentemente do formato. Por exemplo, se você definir a propriedade Format para “dd/mmm/yyyy”, todas as datas digitadas serão exibidas no formato 18/Set/1995. Se o usuário digitar a data como 18/09/95 (ou qualquer outro formato de data válido), o Visual Basic converterá a exibição para o formato estabelecido, que é dia / mês-por-extenso-abreviado / ano-com-4-digitos.

A propriedade Format afeta apenas a maneira como um valor é exibido e não como ele é armazenado. Da mesma forma, um formato de exibição não é aplicado até que o usuário termine a digitação e o controle perca o foco.

Nada é exibido no campo para sugerir ou controlar o formato no qual os dados são inseridos.

Se você precisar controlar a maneira como os dados são digitados, use uma máscara de entrada além de ou ao invés de um formato de exibição de dados. Se você quiser que os dados sejam exibidos exatamente como foram inseridos, não defina a propriedade Format.

*** FormattedText** : Retorna o texto digitado, incluindo os caracteres que compõem a máscara.

Mask : Máscara que moldará o controle.

O Visual Basic fornece duas propriedades que produzem resultados parecidos: a propriedade Format e o Mask.

Use a propriedade Mask para exibir caracteres de exibição literais no campo com espaços em branco a serem preenchidos. Por exemplo, se todos os números de telefones que inserir em um campo tiverem o mesmo formato, você poderá criar uma máscara de entrada:

(###) ###-#### ➲ (___) ___ -____ ➲ (062) 621-3862

Uma máscara de entrada garante que os dados se ajustem ao formato definido e você poderá especificar os tipos de valores que poderão ser inseridos em cada espaço em branco. Por exemplo, a máscara de entrada anterior solicita que todas as entradas contenham exatamente os dígitos necessários para completar um código de área e número de telefone, e que somente dígitos possam ser inseridos em cada espaço em branco.

Você pode definir uma máscara de entrada usando os seguintes caracteres.

0	Dígito (de 0 a 9, entrada requerida, sinais de mais (+) e menos (-) não permitidos).
9	Dígito ou espaço (entrada não requerida, sinais de (+) e menos (-) não permitidos).
#	Dígito ou espaço (entrada não requerida, os espaços são exibidos como vazios enquanto os dados são editados, mas são removidos quando perde o foco, sinais de mais e menos permitidos).
L	Letra (de A a Z, entrada requerida).
?	Letra (de A a Z, entrada opcional).
A	Letra ou dígito (entrada requerida).
a	Letra ou dígito (entrada opcional).
&	Qualquer caractere ou espaço (entrada requerida).
C	Qualquer caractere ou um espaço (entrada opcional).
, . : ; - /	Marcador de posição decimal e separadores de milhares, de data e de hora. (O caractere realmente usado depende das configurações do Painel de Controle do Windows).
<	Faz com que todos os caracteres sejam convertidos para minúsculos.
>	Faz com que todos os caracteres sejam convertidos para maiúsculos.
****	Faz com que o caractere seguinte seja exibido literalmente (por exemplo, \A é exibido simplesmente como A).

Quando você define uma máscara de entrada e a propriedade Format para o mesmo objeto, a propriedade Format tem precedência quando os dados são exibidos. Isso significa que mesmo você tendo salvo uma máscara de entrada, ela é ignorada quando os dados são formatados. O dado original como foi digitado não é alterado; a propriedade Format só afeta a maneira como os dados são exibidos.

MaxLength : Determina a quantidade máxima de caracteres que o MaskEdBox pode ter.

Name: Nomeia o objeto. Geralmente inicia o nome com “msk”

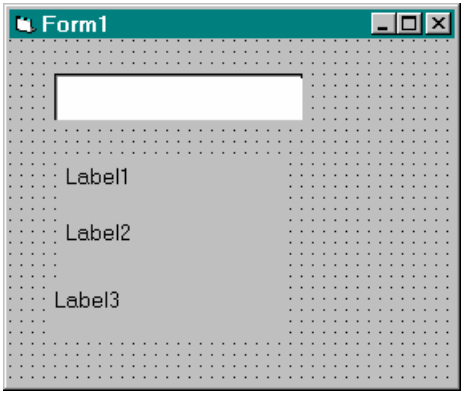
PromptChar : Escolhe o caractere padrão que será exibido simbolizando o estado vazio. Por default possui o caractere “_”, e aconselho a substituir pelo caractere de espaço. Esta propriedade não aceita vazio.

PromptInclude : Determina se o caractere inserido na propriedade PromptChar será incluído na propriedade Text.

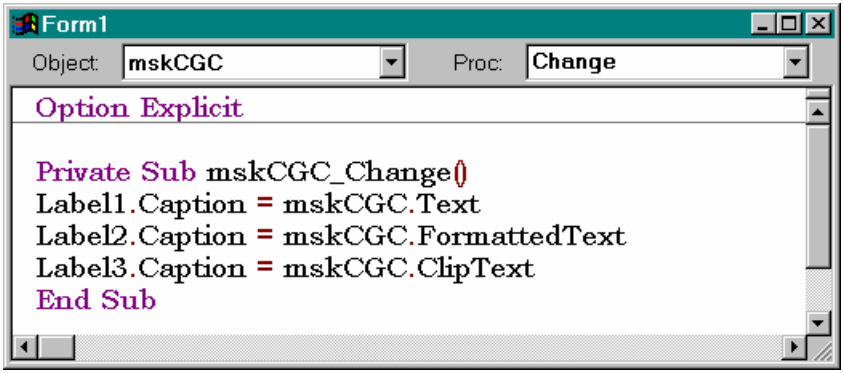
* **Text** : Contém o texto digitado pelo usuário no objeto.

Evento ValidationError : Este evento ocorre sempre que o usuário digita alguma entrada que não corresponde a máscara estabelecida.

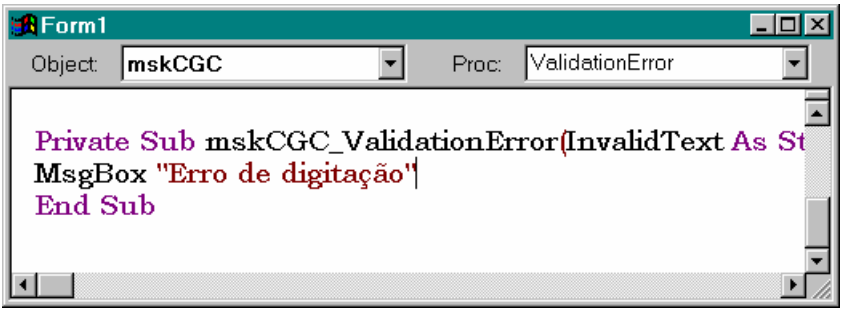
- Crie um novo projeto e insira 1 MaskEdBox e 3 labels:



- Nomeie o maskedit para “mskCGC” e coloque altere a propriedade Mask para “##.###.###/###-##”.
- Rode o programa e veja o resultado.
- Vamos ver agora como algumas propriedades recebem o conteúdo digitado. Abra a janela de codificação e digite:



- Cada propriedade recebe o conteúdo de uma maneira. Quando precisar armazenar ou apresentar esse conteúdo é só escolher a forma que necessitar.
- Perceba que os pontos e a barra esta fixa, mas aparece um inconveniente sinal de “_”. Para retirá-lo mude a propriedade PromptChat para “ “(uma barra de espaço). Agora sim, aparece somente nossa formatação.
- Vamos agora criar um evento ValidationError para quando o usuário digitar alguma tecla inválida o Visual Basic dar um alerta:



- Numa máscara onde se exige a digitação de números, se digitar uma letra do alfabeto ou tentar digitar mais digitos que a mascara suporta, a mensagem de advertência irá aparecer.

5.2 COMMONDIALOG

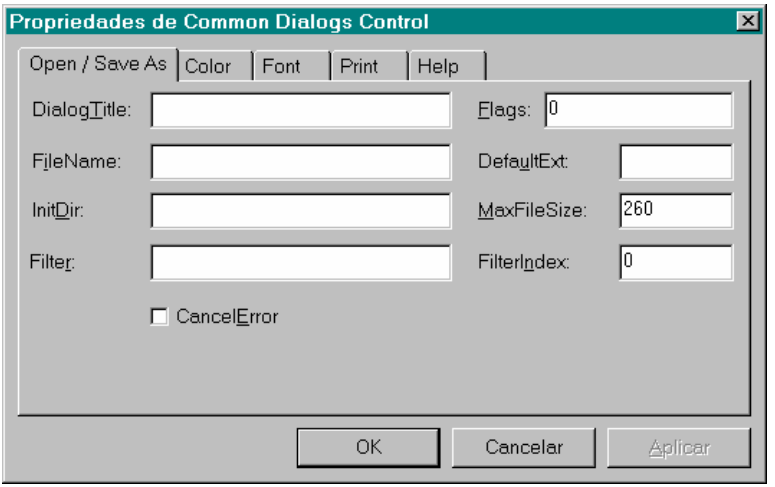
O Visual Basic traz algumas caixas de diálogos prontas para Abrir arquivo, Salvar, Imprimir, escolher Cor, ou escolher fontes de letra. Sua utilização é simples e basta inserirmos o objeto CommonDialog no formulário para ativarmos os diálogos. Na codificação do programa usamos a propriedade Action para o Visual Basic abrir o diálogo pretendido:

* **Action** : Determina o tipo de diálogo que será exibido:

0 Nenhum diálogo.
1 Mostra caixa de diálogo Abrir Arquivo
2 Mostra caixa de diálogo Salvar Arquivo
3 Mostra caixa de diálogo Escolher Cor
4 Mostra caixa de diálogo Escolher Fonte de Letra
5 Mostra caixa de diálogo de Impressão.
6 Executa o WINHELP.EXE.

As principais propriedades deste objeto podem ser acessadas selecionando (Custom) ou clicando no botão direito do mouse e selecionando “Propriedades” no menu:

Open/SaveAs: Aqui definimos as características do arquivo que será aberto ou salvo.



DialogTitle: Determina o texto que irá aparecer na barra de título.

FileName: Nome e Caminho padrão para um determinado arquivo.

InitDir: Diretório padrão onde sempre será iniciado o diálogo

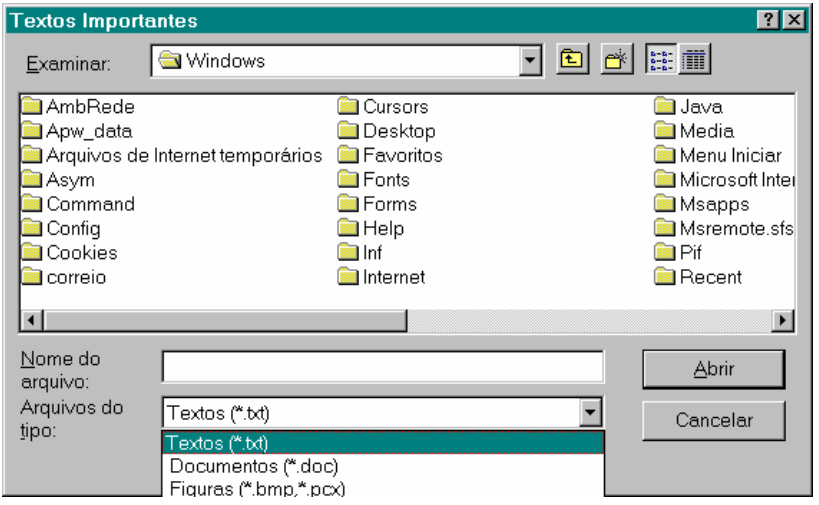
Filter: Especifica os tipos de arquivos que poderão ser selecionados. Veja a regra de uso desta propriedade:

Descrição | Tipo de arquivo |

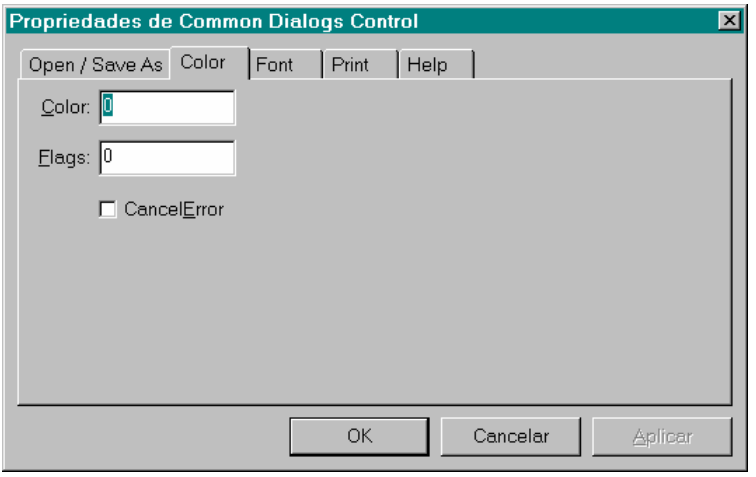
Primeiro usamos um texto descritivo para o arquivo que será selecionado. Depois uma barra vertical (|), e a extensão que este arquivo usa e finaliza com outra barra.

Textos (.txt) | *.txt | Documentos (*.Doc) | Figuras (*.bmp,*.pcx) | *.bmp;*.ico |*

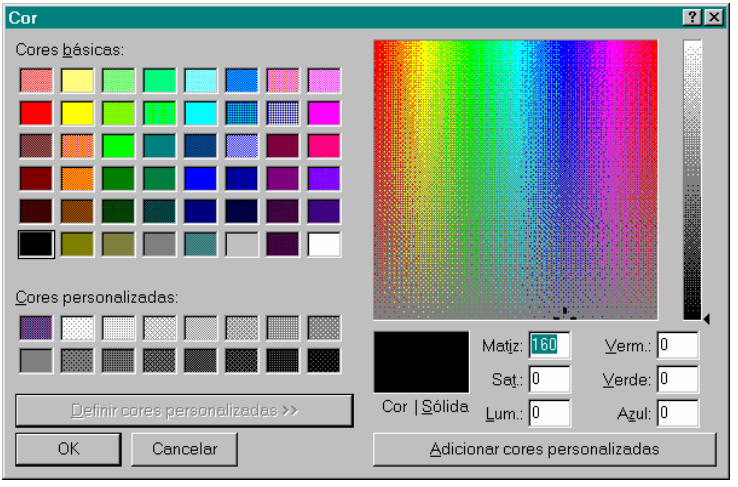
Colocando desta forma veja como ficará a caixa de diálogo:



Color: Definimos as características da janela para escolha de cores.



Color: Especifica a cor selecionada no diálogo. Essa propriedade serve não só para capturar a cor escolhida como para levar para a janela Color a cor em uso.



Font: Janela para definição de fontes de letra.

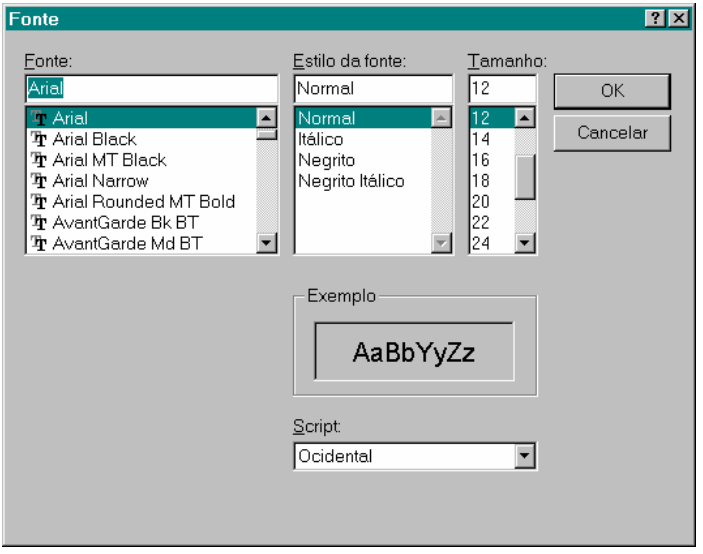
FontName: Nome da fonte corrente, usado no momento em que a janela de escolha de fonte foi chamada. Também informa qual o nome da nova fonte escolhida.

FontSize: Determina o tamanho da fonte corrente e o novo tamanho escolhido.

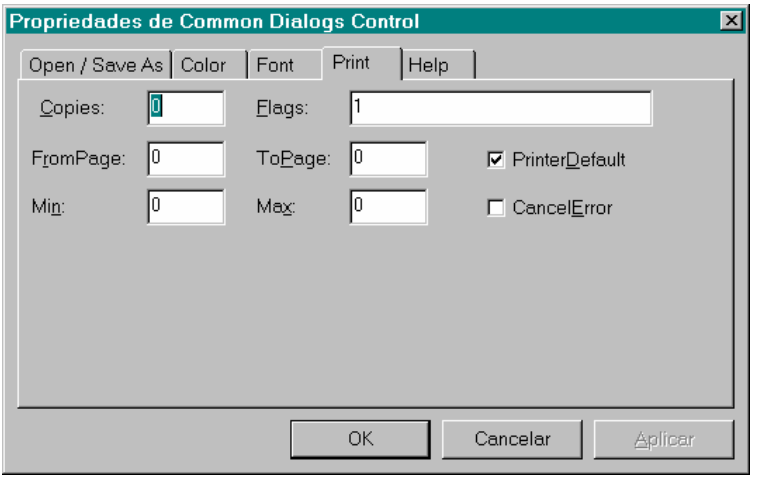
Min e Max: Tamanho mínimo e máximo que um usuário pode escolher para uma determinada fonte de letra.

Flags: Determina algumas opções para a caixa de díálogo. Coloque 1.

Style: Determina como será o tipo da fonte de letra.



Print: Janela de configuração de impressora e impressora. Obrigatoriamente temos que usar quando enviamos algum relatório para ser impresso.



Copies: Determina a quantidade de cópias padrão para serem impressas.

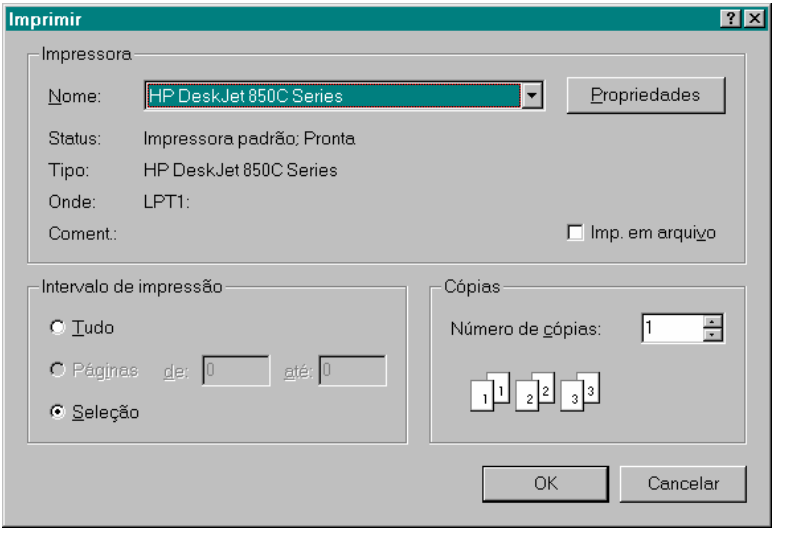
FromPage: Página inicial a ser impressa.

ToPage: Página final a ser impressa.

Min: Limite mínimo de um intervalo de impressão

Max: Limite máximo de intervalo de impressão.

PrinterDefault: Determina se o usuário poderá alterar a impressora padrão do Windows.



Métodos Aplicáveis ao Controle: Podemos usar métodos para exibir as caixas de diálogo.

NomeDoObjeto.ShowColor Exibe janela para escolha de cores.

NomeDoObjeto.ShowFont Exibe janela para escolha de fonte de letra.

NomeDoObjeto.ShowOpen Exibe janela com opções de abrir arquivo.

NomeDoObjeto.ShowSave Exibe janela com opções para salvar arquivo.

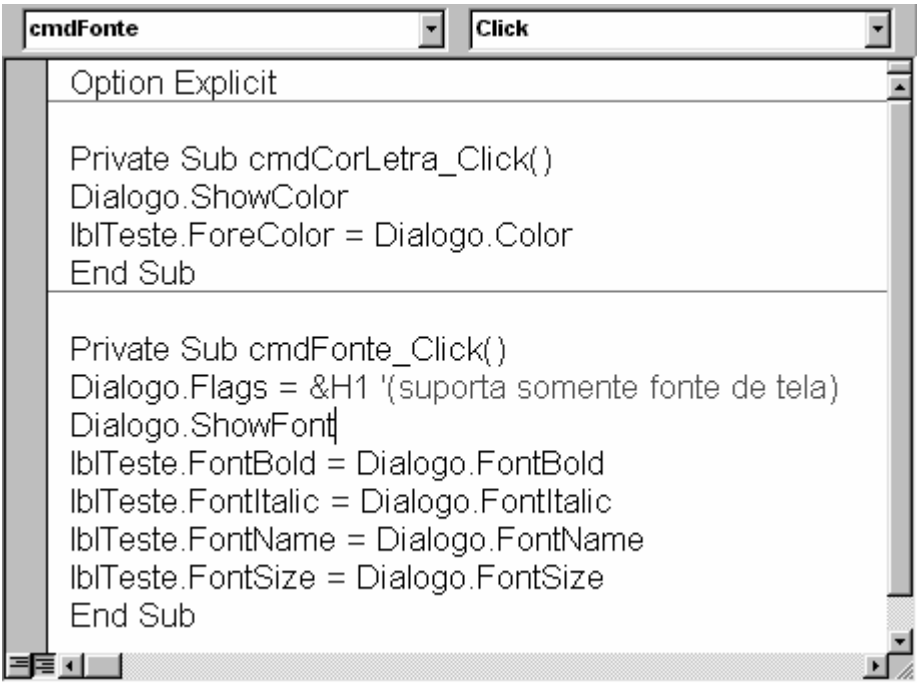
NomeDoObjeto.ShowPrinter Exibe janela com opções de impressão.

- Crie um formulário da seguinte forma:

- Nomeie os objetos:

Tipo de Objeto	**Name**
Label	LblTeste
Command1	CmdCorLetra
Command2	cmdFonte
CommonDialog	Dialogo

- Crie a seguinte codificação para os botões:



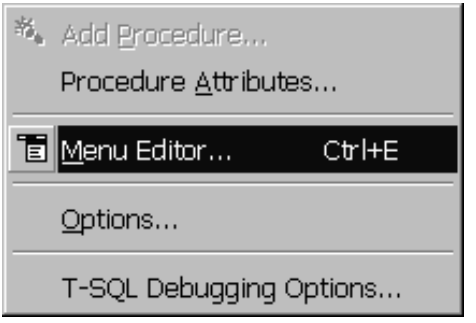
- Com isto o usuário poderá escolher a cor da letra que para o label assim como mudar a fonte de letra e o tamanho. Repare que usamos o "Flags = &H1" para forçar o Visual Basic a ler somente as fontes de tela. Consulte o Help para ver os outros Flags possíveis.

6 MENUS

- Criando Menus
- Menus Instantâneos

ANOTAÇÕES PARA NÃO ESQUECER

6.1 MENUS

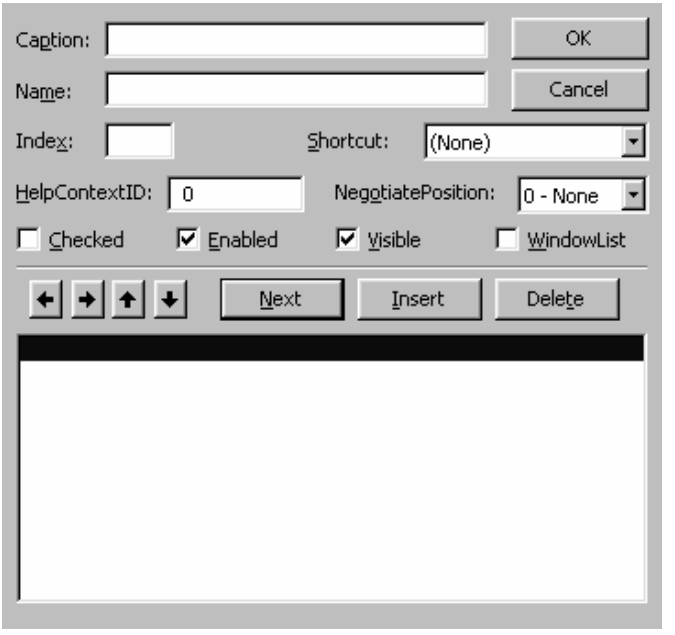


A maioria das aplicações possuem menus para facilitar o usuário na localização de todas as janelas que compõem um programa.

Para criar menus, usamos o menu TOOLS opção Menu Editor.

Sempre que criarmos menu temos que seguir algumas regras como colocar acesso Mnemônico em todos os itens e seguir um padrão, como, pôr exemplo, o menu Arquivo colocar o sublinhado na letra “A”, colocar “...” quando este menu abre uma janela de diálogo, estabelecer uma Tecla de Atalho para as principais opções do menu e colocar um traço de separação para separar seções dentro de um mesmo menu, como pôr exemplo, no menu ao lado, usou este recurso para separar algumas opções.

6.1.1 Criando Menus



Veja as principais propriedades do objeto menu:

Caption : Nesta propriedade definimos o titulo do item que irá aparecer no menu. Nesta propriedade prefixamos um letra com o símbolo "&" para que o Visual Basic destaque a letra com sublinhado. Se for um item da barra de menus, ao teclarmos ALT e a letra destacada, o menu desse item será aberto automaticamente. Se for um item de uma menu já aberto, basta acionarmos somente a letra destacada para executarmos a função vinculada ao mesmo.

Name : Usamos esta propriedade para definirmos o identificador para o item. Nomeamos cada item de um menu para podemos manipulá-los através de codificaçõa. Para menus usamos abreviação "mnu"

Index : Permite criar uma matriz para identificar os itens do menu. Para que a matriz seja estabelecida o name deve ser igual para os que vão compor a matriz.

ShortCut : Determinamos uma combinação de tecla para que o menu seja acessado de forma mais rápida. Quando definimos tecla de atalho para um item do menu, a rotina anexada é chamada independente do local onde o foco esteja.

WindowList : Quando esta propriedade esta habilitada, numa aplicação com várias janelas, o Visual Basic cria uma lista das ultimas janelas que foram acessadas.

Checked : Sempre que o usuário clica no item que possui esta propriedade habitada para True o Visual Basic coloca um sinal de marcado "✓" ao lado do nome. Se clicar novamente esta marca é retirada. Usamos esta propriedade quando queremos mostrar para o usuário o estado Ligado ou Desligado de algum item do menu.

Enabled: Habilita ou não um item do menu. Quando esta desabilitada o Visual Basic coloca o texto do Caption acinzentado.

Visible : Quando necessitamos que um determinado item do menu exista mas que fique invisível para o usuário até que determinada condição seja estabelecida, usamos esta propriedade.

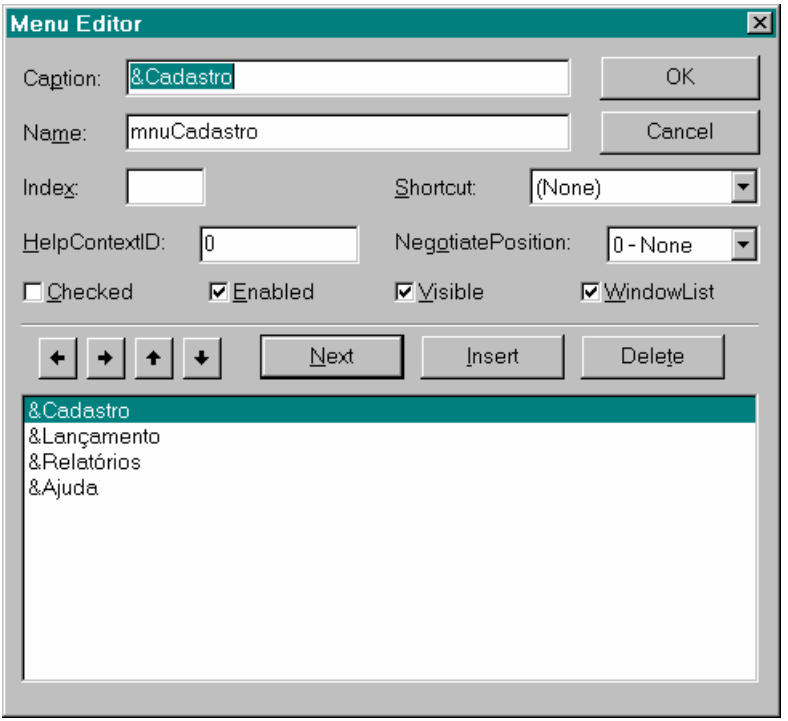
Usamos os botões de setas para **Baixo** de para **Cima** para movermos um item de lugar depois de criado. O botão seta para **Direita** transforma o item selecionado em submenu item imediatamente acima. O botão seta para **esquerda** volta o item um nível.

O botão **Next** é o default, ou seja, ele é acionado sempre que digitamos ENTER. Ele passa para o próximo item de menu a ser digitado ou na seqüência.

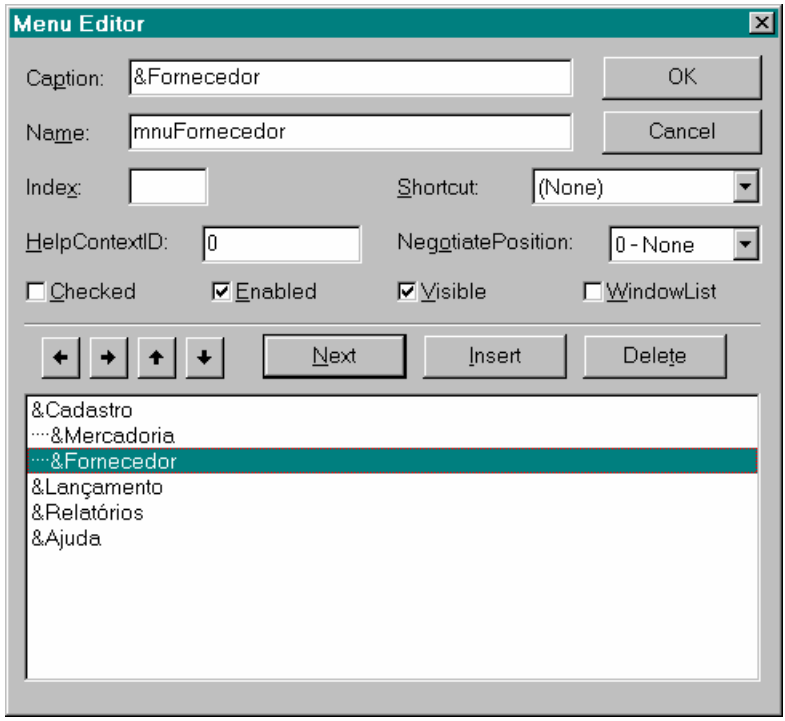
Insert é usado para inserir um item de menu entre outros já criados e **Delete** apaga o item selecionado.

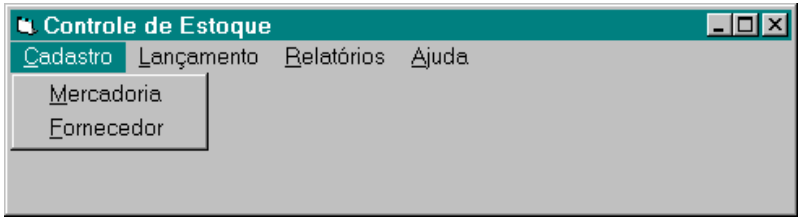
- Num projeto novo vamos criar um menu para um programa de controle de estoque. Vamos primeiramente definir as opções mestre que vão ficar no topo do menu.

- Para criarmos estes itens basta abrirmos a janela de criação de menu e fazer os lançamentos conforme estabelecido abaixo:

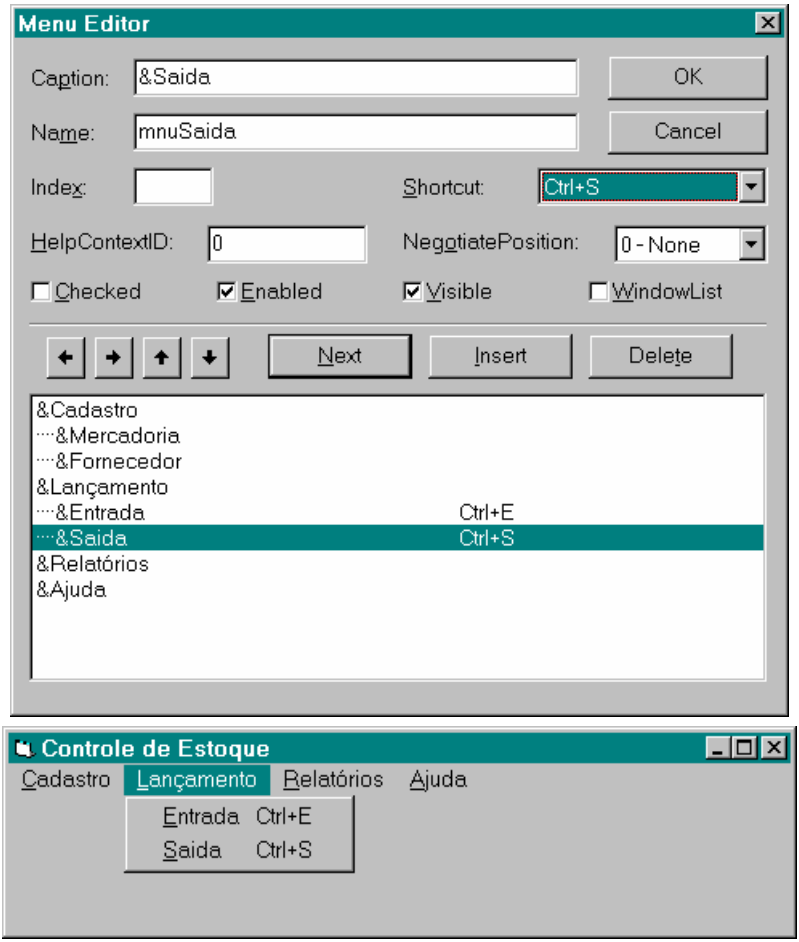


- Pronto. Execute o programa e veja o menu já criado automaticamente. Mas logicamente precisamos agora criar subitens em cada uma dessas opções. Selecione “&Lançamento” e aperte a tecla “Insert” e depois o botão seta para a direita. Fazendo estaremos criando subitens para o item “Cadastro”. Vamos criar as opções “Mercadoria” e “Fornecedor”. Note que quando apertamos ENTER para o próximo ele não insere um novo item. Temos que repetir a operação “Insert” e “Seta para Direita”.

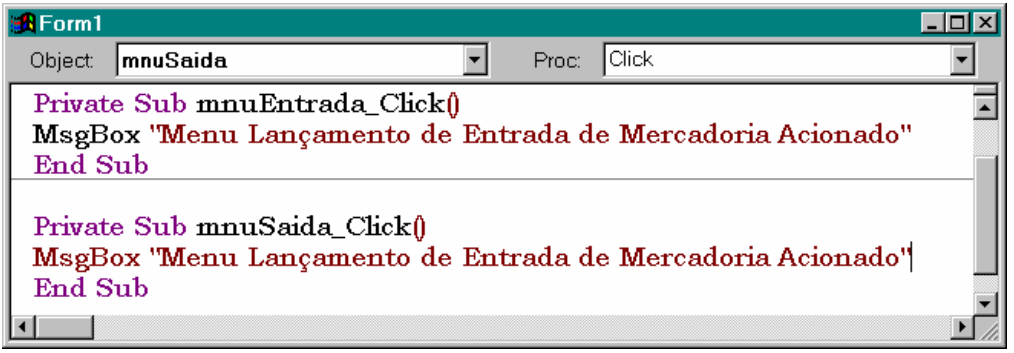




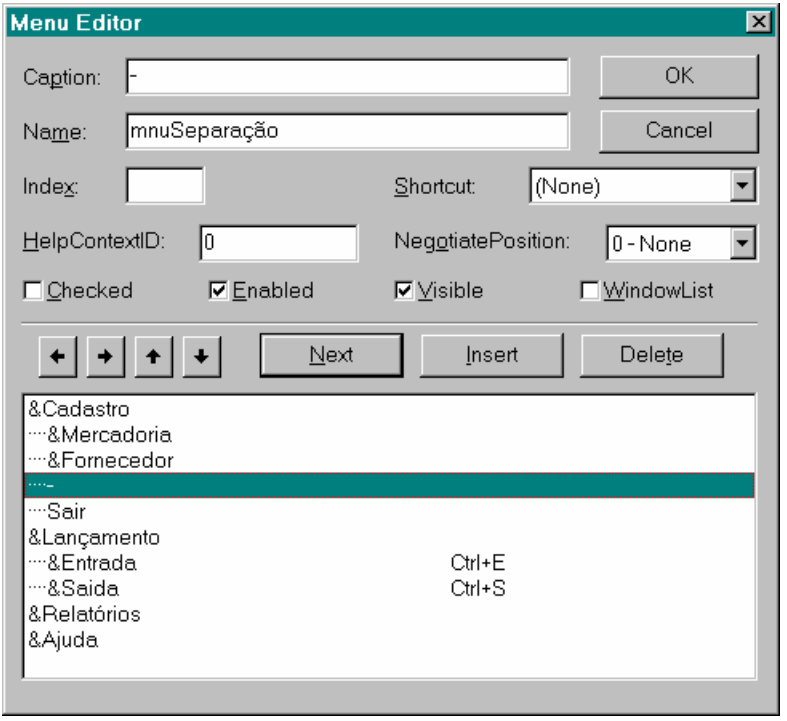
- Agora repita o processo para criar subitens para "Lançamento" de nomes "Entrada" e outro "Saída" e coloque uma tecla de atalho para cada um conforme modelo:

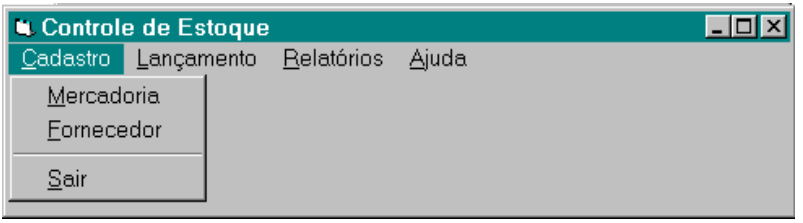


- Criar um menu é simples, e o único evento que existe para os menus é o “Click” que é exatamente igual ao evento “Click” usado para CommandButton.
- Se em tempo de projeto dermos um click num item de menu o Visual Basic abre a janela de codificação para digitarmos alguma rotina de programação para o menu. Afinal, um menu não é somente enfeite. Cada item seu teve efetuar um determinado evento.
- Vamos criar um evento Click para as opção “Entrada” e “Saida“ bastante simples:



- Execute o programa e veja que clicando nas opções com o mouse ou apertando as teclas de atalho correspondentes o MsgBox é acionado.
- Vamos agora voltar ao Editor de Menu e na opção "Cadastro" inclua no final o item "Sair". Vamos colocar um item "Separador" para esteticamente ficar mais apresentável. Este item é colocando inserindo o caractere "-" no caption.





- Crie um evento para o item "Sair":

- O Comando "END" encerra a aplicação.

6.1.2 Menus Instantâneos

O Windows 95 usa com muita freqüência este tipo de menu. São menus que se abrem sempre que um apertamos o botão direito do mouse sobre algum objeto. Para esses menus surgirem na tela usamos o comando PopupMenu no evento MouseUp de qualquer objeto. Veja como usar essas rotinas.

PopupMenu : Neste comando especificamos o nome dado ao Menu que contém submenus e que irá aparecer ao acionarmos este comando.

Exemplo: *PopUpMenu mnuLançamento*

Irá aparecer os subitens do menu Lançamento, ou seja, Entrada e Saida.

MouseUp (Button as Integer, Shift As Integer, X as Single, Y as Single): Sempre que o usuário aperta um botão do mouse este evento é chamado. Usamos os argumentos que o evento oferece para sabermos qual dos botões foram apertados, se foi em conjunto com alguma tecla e qual coordenada estava o ponteiro do mouse no momento que o evento foi chamado.

Button : retorna um número inteiro que informa qual dos botões foram apertados:

1	Botão esquerdo pressionado.	vbLeftButton
2	Botão direito pressionado.	vbRightButton
4	Botão do centro pressionado.	vbMiddleButton

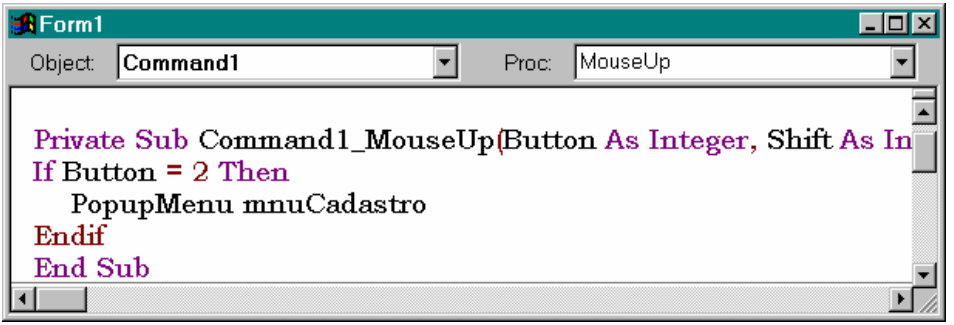
Shift : retorna um número inteiro que informa qual tecla estava pressionada no momento em que algum botão do mouse também foi pressionado.

1	SHIFT pressionado.	vbShiftMask
2	CTRL pressionado.	vbCtrlMask
4	ALT pressionado	vbAltMask

X : Coordenada para linha

Y : Coordenada para Coluna

- Na aplicação anterior coloque no formulário um CommandButton e crie um evento MouseUp para ele com a seguinte codificação:



- Como não especificamos as coordenadas, como padrão o Visual Basic estabelece a posição atual do mouse.
- Note que pedimos para o menu “mnuCadastro” ser carregado sempre que apertamos o botão direito do mouse sobre o objeto “Command1”. Se chamarmos este evento sobre o formulário ou qualquer outro objeto nada acontecerá.

7 VARIÁVEIS E MATRIZES

-
- Numérica
- Texto
- Data
- Byte
- Boolean
- Object
- Variant
- Matrizes

7.1 AS CARACTERÍSTICAS DE UMA VARIÁVEL

As variáveis possuem uma importância fundamental dentro de qualquer linguagem de programação. Elas armazenam os dados que vão para a memória do computador, e sempre que referenciamos a elas, esses dados retornam da memória. E quando estão na memória podem fazer cálculos e executar determinadas funções.

Para uma variável receber um conteúdo qualquer é necessário anexar esse conteúdo ao nome que daremos para a variável. Exemplo:

Aluno = "João Batista da Silva"

Note que usamos o sinal de igual para que o nome do aluno fosse inserido na variável *Aluno*, que estará agora na memória do computador, e sempre que nos referenciarmos a *Aluno* será como se tivéssemos referenciado ao "João Batista da Silva".

Para darmos nomes a uma variável é necessário seguir algumas regras básicas:

1. Sempre iniciar com letras (A até Z). Não use números ou símbolos para iniciar o nome de uma variável.
2. Pode-se usar até 255 caracteres para compor o nome da variável, então batize-as com nomes bastante claros, para que, apenas lendo seu nome, saibamos do que se trata e para que serve. Exemplo: "NomeDependenteFuncionário". Parece grande demais, mas só de lê-lo já saberemos que se trata de uma variável que irá armazenar o nome do dependente de um funcionário.
3. Na composição do nome não podemos usar espaço em branco, sinal de hífen, símbolos como @#*&^$% ou sinais de pontuação. Pode-se usar somente Letras do alfabeto, números e o sublinhado.
4. Cuidado para não usar nomes reservados pela linguagem. Exemplo: If, Caption, Name, Sub, End, etc.

7.1.1 O Comando Dim

Sintaxe: Dim *NomeDaVariável* As *TipoDeVariável*

NomeDaVariável: Representa um nome válido como descrito acima que usamos sempre que precisarmos nos referenciar ao seu conteúdo.

As TipoDeVariável: Cada variável tem um tipo predefinido, e na criação dela, usando o comando DIM, aconselha-se a já definir para o programa que tipo de dados essa variável irá guardar. Exemplo: Números, Dados, Textos, etc.

O comando DIM é usado para declararmos para o programa a existência de uma variável e o tipo que ela é.

Este comando é obrigatório e podemos declarar as variáveis no momento em que estamos necessitando delas, já em uso. Exemplo:

```
Function CalculaFormula()
        x = 10
        y = 13
        CalculaFormula = x*y+30
End Function
```

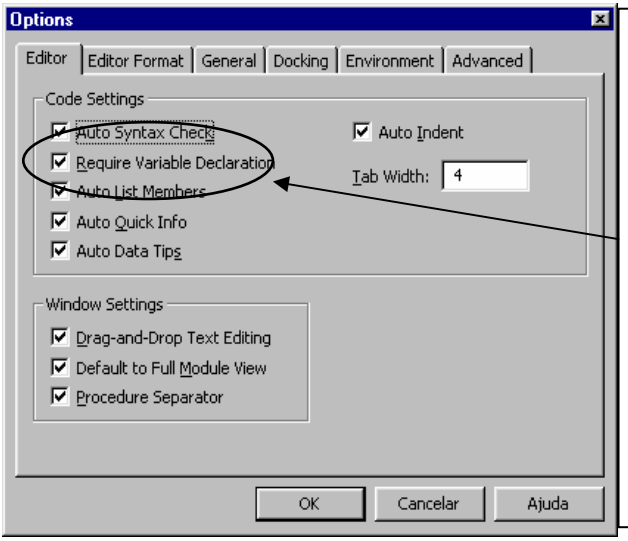
Este exemplo bem simples é somente para mostrar que usamos duas variávels chamadas x e y e não as declaramos. Precisamos delas e usamos. Caso fosse usar o comando Dim, esta mesma função ficaria assim:

```
Function CalculaFormula()
        Dim x As Integer
        Dim y As Integer
        x = 10
        y = 13
        CalculaFormula = x*y+30
End Function
```

A expressão Integer significa tipo de variável numérica que poderá aceitar somente números inteiros. Veremos melhor esses tipos adiante.

Este método é extremamente útil, pois imagine um programa onde existe uma imensa quantidade de variáveis. Se todas foram previamente declaradas saberemos de antemão os nomes dela e para que servem.

Podemos forçar o Visual Basic a aceitar em nosso programa somente variáveis que foram declaradas. Para tanto temos que acessar o TOOLS no menu principal, opção OPTIONS:



No menu options, janela Environment, habilite a opção "Require Variable Declaration" (Requer Variável Declarada). Assim sempre que usar uma variável que não foi declarada pelo comando Dim e não tem seu tipo definido o Visual Basic apresentará uma janela de alerta. Faça isto agora!

Usando isto, o Visual Basic irá exigir a declaração das variáveis, e não corremos o risco de, quando digitarmos o nome de alguma variável errada, ele aceitar. Exemplo: Criamos uma variável de nome: ClienteCadastrado. E em algum canto de nossa codificação digitamos: ClienteCadastado. Faltou a letra "r". Se a opção "Require Variable Declaration" estiver habilitada o Visual Basic nos avisará que este nome não existe como variável. Caso contrário, o Visual Basic irá imaginá-la como uma nova variável e aceitará o erro. Não corra risco!

7.1.2 Os Tipos de Variável

Nome	Descrição	Abrangência
Integer	Numérico Inteiro	-32.768 a 32.767
Long	Numérico Inteiro	-2.147.483.648 a 2.147.483.648
Byte	Numérico Inteiro	0 a 255
Single	Numérico real	-3,402823E38 a -1,401298E-45 1,401298E-45 a 3,402823E38
Double	Numérico real	-1,79769313486232E308 a 4,94065645841247E-324 4,94065645841247E-324 a 1,79769313486232E308
Currency	Numérico - Valores monetários	-922.337.203.685.477,5808 a 922.337.203.685.477,5807
String	Texto	2.147.483.648 de caracteres nos sistemas 32 bits 65.536 de caracteres nos sistemas 16 bits
Date	Data e Hora	entre 01/01/100 a 31/12/9999
Boolean		True (verdadeiro) False (falso)

Object		Contém um objeto
Variant		Pode ser numérico, string, objeto ou valor nulo.

7.1.3 As Variáveis Númericas

As variáveis Integer, Long, Byte, Single, Double e Currency armazém em eu conteúdo somente números. A diferença entre elas estão na abrangência de cada uma e se aceitam ou não casas decimais. A variável Currency é mais especifica para tratar valores monetários, e ela já converte os valores no formato estabelecido no Painel de Controle (Configurações Regionais) para moeda.

Sempre é bom verificarmos o tipo de número que irá conter a variável numérica que estamos criando. Se for conter somente números inteiros, e no máximo 10.000 registros, então a Integer esta de bom tamanho. Se vamos criar um contador que irá contar até 100, então a variável mais cabível é a byte.

A escolha correta da variável numérica é importante no consumo de recursos de processador de seu computador. Se você escolhe o tipo Double essa variável irá consumir os recursos de sua máquina para que possa trabalhar com números dos tipo "4,94065645841247E-324". Mas se você vai somente armazenar na variável números na casa do milhar, então a máquina estará absorvendo do processador mais recursos do que vai usar.

Acostume a definir uma determinada variável como numérica somente quando esta variável será objeto de cálculos.

7.1.4 Variável String

Esta variável armazena textos. Esses textos podem ser caracteres alfanuméricos, símbolos, etc. Sempre que formos nos referenciar a expressão que está contida na variável string temos que colocar "" (aspas). Exemplo: NomeEscola="Modelo Informática".

Neste tipo de variável podemos fazer comparações, concatenação de duas variáveis string, ou até extrair uma parte do conteúdo.

O que é concatenação?

É a junção de duas variáveis ou expressões texto. Exemplo: "12" + "13" resultará em "1213". "Editora" + "Terra" resultará em "Editora Terra".

Entretanto, apesar de podermos usar o operador "+" para fazer concatenação, vamos usar em seu lugar sempre o operador "&". Esse operador, em detrimento ao outro, força a concatenação mesmo quando os dados são incompatíveis. Exemplo: "12" + 13 resultará em 25. Observe que uma expressão era String e a outra numérica, e o operador "+" forçou a soma.

Se usássemos neste exemplo “12” & 13 resultaria em “1213” apesar da expressão 13 ser numérica.

Quando criamos uma variável do tipo String podemos delimitar previamente o tamanho que ela terá. Veja a sintaxe:

*Dim NomeEscola as String * 30*

Criamos uma variável de nome “NomeEscola” do tipo String (texto) e que terá no máximo 30 caracteres. Se colocarmos mais que isto em seu conteúdo o excedente será ignorado. Se colocarmos menos que isto será preenchido com espaços em branco.

7.1.5 Variável Boolean

Este tipo de variável contém somente dois valores: True ou False.

Indicado como resultado de comparações. Exemplo:

Dim Comparação As Boolean

Dim NomeEscola As String

NomeEscola = “Modelo Informática”

Comparação = (NomeEscola = “Informática”)

A variável Comparação resultará em False.

As propriedades dos objetos que temos que informar True ou False (como por exemplo Enabled) são do tipo Boolean.

7.1.6 Variável Date

Como o próprio nome diz, são variáveis que armazenam Datas. Entretanto podemos também armazenar horas. Para sinalizar para o programa que vamos armazenar uma data numa determinada variável usamos o #. Exemplo: Hoje=#15/07/96#. Veja o que acontece se usarmos de forma diferente:

Hoje = 15/07/96

O programa irá imaginar que estamos na verdade dividindo 15 por 7 e depois dividindo por 96.

Hoje = “15/07/96”

O programa irá ter seu conteúdo como uma string (texto) no formato de uma data.

Uma variável data aceita alguns tipos de cálculo como por exemplo:

a = #15/07/96#

b = #30/07/96#

b - a = 15 (Uma data menos outra retorna o intervalo em dias)

a + 3 = #18/07/96# (uma data mais um valor numérico soma a quantidade de dias na data)

a - 3 = #12/07/96# (uma data menos um valor numérico subtrai a quantidade de dias na data.

7.1.7 Variável Object

O Visual Basic trabalha com objetos. Nossa caixa de ferramentas está cheia deles. Mas temos também objetos Database, Fonts, etc.

Se precisarmos inserir esses objetos em uma variável usamos um comando chamado Set, e não o sinal de igual. Exemplo:

Dim BancoDeDados As Object
Set BancoDeDados = OpenDatabase("C:\Visual Basic\BIBLIO.MDB")

7.1.8 Variável Variant

Este tipo de variável não possui um tipo definido. Ele se transforma em qualquer tipo dependendo de seu conteúdo.

Usamos uma Variant quando não sabemos que tipo de dados terá a variável ou se já sabemos de antemão que o conteúdo vai necessitar passar por transformações nos tipos de dados armazenados.

7.1.9 Null

Quando o conteúdo da variável é um dado inválido ela retorna a palavra-chave Null.

7.2 Abrangência e Tempo de Vida de uma Variável

Variáveis criadas dentro de uma rotina valem somente para a rotina que a criou. Por exemplo, quando criamos um evento para um botão de comando, estamos criando uma rotina (procedure) chamada Private Sub:

Private Sub Command1_Click()
Dim Nome as String
NomeTeste = "Fechar"
Command1.Caption = NomeTeste
End Sub

A variável NomeTeste que criamos dentro do Private Sub possui validade somente dentro desta rotina de código. Chamamos de variável Local. Quando encerrar (comando End Sub) a variável será retirada da memória do computador e não mais existirá. Outro Exemplo:

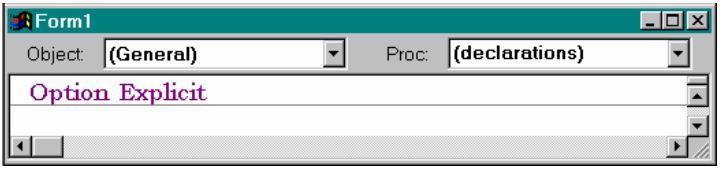
Private Sub Command1_Click()
Dim NomeTeste as String

NomeTeste = “Fechar”
End Sub

Private Sub Command2_Click()
Command2.Caption = NomeTeste
End Sub

Repare, a variável NomeTeste foi criada dentro da rotina Private Sub Command1 e na outra rotina Private Sub Command2 fazemos referencia a ela. Como ela foi criada dentro da Private Sub Command1 ela tem validade somente lá. Este exemplo nosso retornará um erro, pois NomeTeste não existe para Private Sub Command2.

Entretanto, se queremos que uma variável tenha uma abrangência em todas as rotinas criadas dentro do formulário, então na janela de código do formulário na parte superior mude o Object para “General” e o Proc para “declarations). As variáveis que forem criadas ali valerá para todas as rotinas criadas para o formulário e seus objetos anexados a ele.



Exemplo:
Option Explicit
Dim teste As String

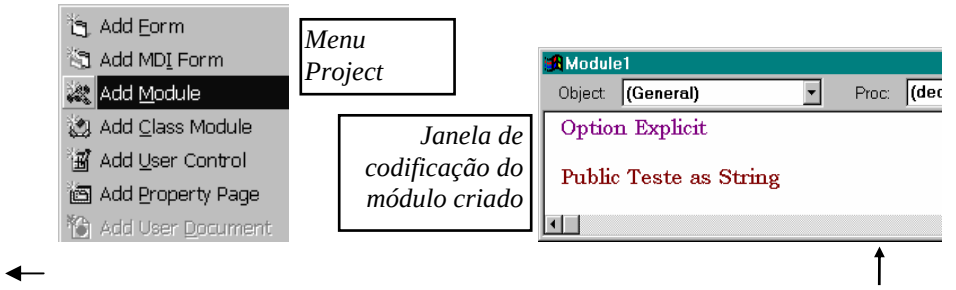
Private Sub Command1_Click()
teste = "Testando a Variavel"
Print teste
End Sub

Veja que a variável teste foi criada fora da rotina Private Sub, mas possui abrangência dentro dela.

Se criarmos um outro formulário e pedirmos para imprimir a variável teste irá ocasionar um erro, pois a validade desta variável é somente dentro do formulário em que ela foi criada.

Caso precisamos de uma variável de caráter publico, que tenha validade Global, em todos os formulários e rotinas de nossa aplicação, será necessário inserir um módulo (Menu Insert opção Module) e dentro dele criar a variável com o comando PUBLIC no lugar de DIM. Veja exemplo:

Public Teste as String

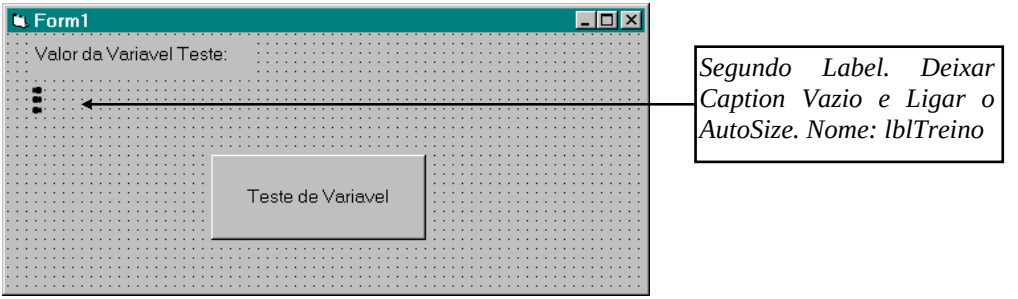


Resumindo:

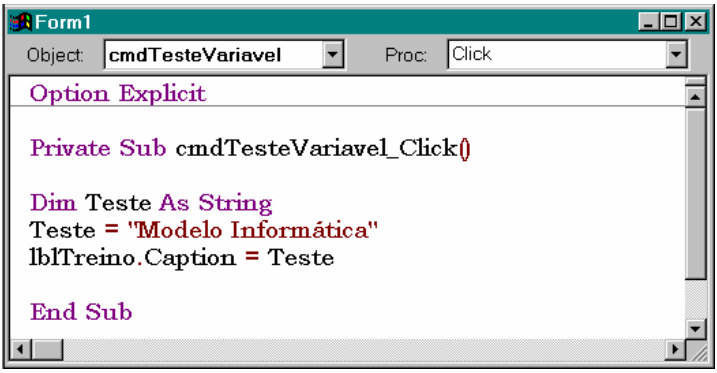
Veja então as formas de se declarar uma variável:

- Variável pública à todo o programa: podemos declarar uma variável pública, especificando-a com o comando **Public** na seção **Declarations** de qualquer módulo.
- Variável pública ao módulo: através do comando de declaração **Dim** na seção **Declarations** do módulo desejado, especificamos que uma variável será pública apenas ao módulo onde fora declarada.
- variável local à rotina (**Sub** ou **Function**): declarando uma variável com o comando **Dim** dentro de qualquer procedimento **Sub** ou **Function** a mesma será reconhecida apenas pelo procedimento.
-

- Crie um novo projeto e coloque nele um botão de comando e dois Labels. Não esqueça de habilitar no Menu Tools/Options a opção "Require Variable Declaration".
- Codifique o botão da seguinte forma:

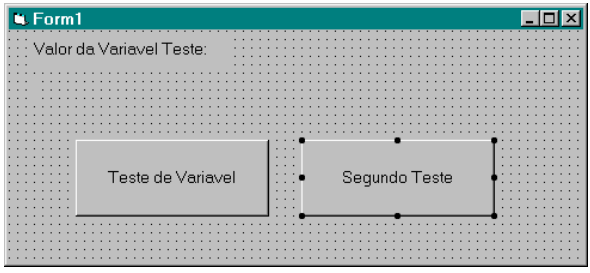


- Vamos agora codificar o botão. Dê dois clickes nele e digite os seguinte comandos:

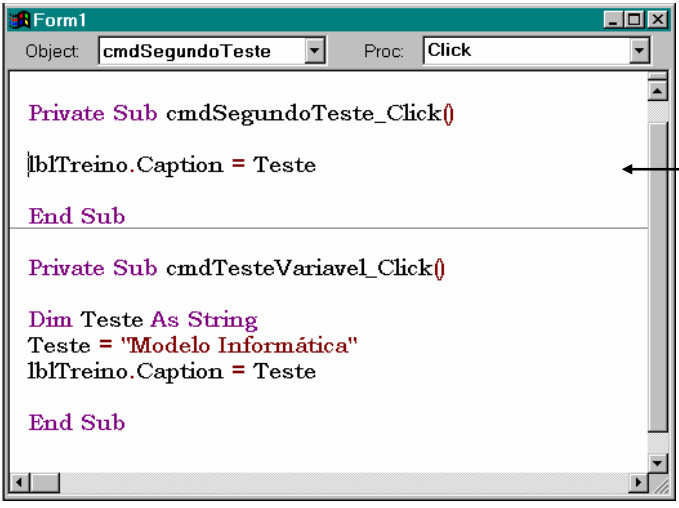


Foi criado uma variável de nome Teste, e seu conteúdo é "Modelo Informática". Assim, sempre que fizermos referencia a Teste irá aparecer o valor contido nela.

- Execute o programa e aperte o botão. Onde estava o Label "lblTreino" irá aparecer o conteúdo da variável.
- Como criamos a variável dentro da rotina "Private Sub cmdTesteVariavel_Click" ela terá validade somente ali dentro.
- Para comprovar isto, crie outro botão:

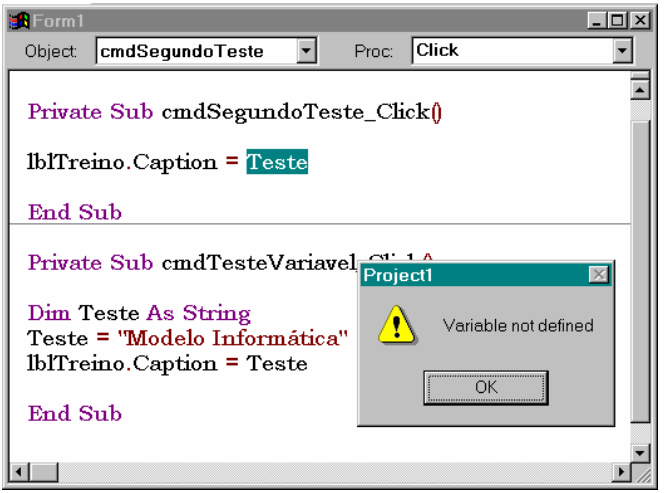


- Crie a seguinte codificação para este botão:

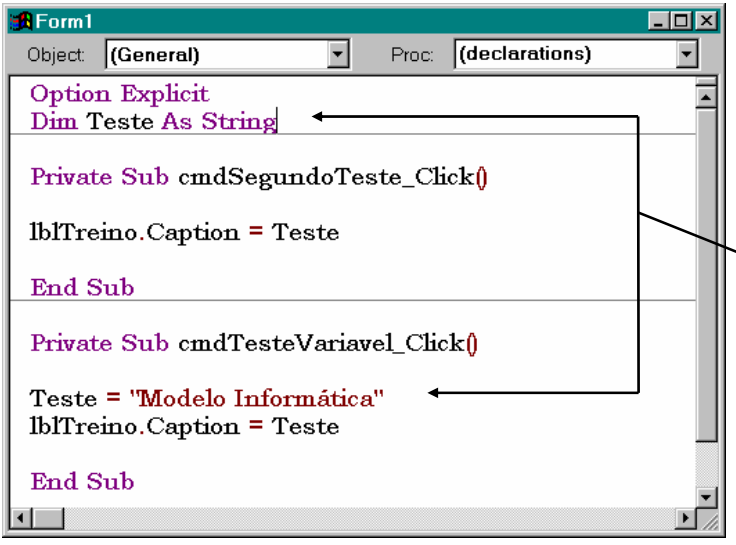


Nesta codificação do segundo botão apenas mandamos inserir o conteúdo da variável Teste para o Label

- Rode o programa e aperte o segundo botão. Irá aparecer uma mensagem de erro alertando que a variável "Teste" não existe:



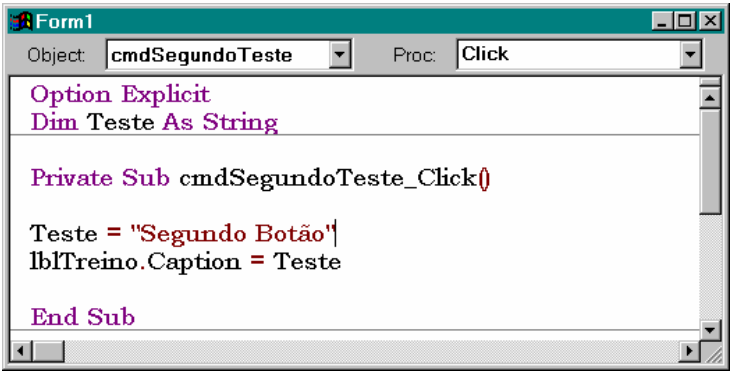
- Agora apague a linha "Dim Teste as String". Vá até o "Object" (Caixa de Combinação na parte superior), chame a opção "General", e digite:



Colocando o comando Dim no object "General" todas as variáveis criadas ali tem abrangência em toda a janela de codificação do formulário (todas as rotinas existentes).

- Rode o programa e aperte o segundo botão "cmdSegundoTeste" e veja o que acontece: A mensagem de erro que aparecia antes não aparece mais, pois agora a variável abrange esta rotina também.
- Mas não existe nenhum conteúdo para a variável Teste até o momento em que o botão "cmdTesteVariável" seja pressionado. A variável Teste terá um conteúdo somente quando o botão "cmdTesteVariável" for pressionado. Até então, a variável existe mas seu conteúdo é vazio.

- Para entender melhor, mude a codificação do botão "cmdSegundoTeste" para:



7.3 MATRIZES

Nas matrizes podemos armazenar vários dados (seja texto, números ou datas) em uma única variável. O que diferencia essencialmente uma matriz de uma variável é que a primeira possui níveis dentro de si, e em cada nível podemos armazenar dados. Veja o exemplo de uma declaração e inicialização de uma matriz:

Dim NomeCliente(3) as String
NomeCliente(0) = "Fernando Henrique"
NomeCliente(1) = "Fernando Collor"
NomeCliente(2) = "José Sarney"

Criamos um matriz de nome NomeCliente e definimos como sendo uma String. Entretanto, definimos também que essa matriz terá 3 níveis somente. Em cada nível podemos colocar um elemento (um conteúdo) dentro dele.

Todos os elementos da matriz precisam ser do mesmo tipo estabelecido no comando **Dim**. Porém, se declararmos um matriz do tipo **Variant**, cada elemento poderá ser de um tipo de dado diferente.

Para acessarmos um determinado conteúdo temos que especificar o número que acompanha o nome da matriz. Chamamos esses números de índice, e são estes índices que localiza os elementos dentro da matriz. Exemplo: Se desejamos imprimir a matriz que contém o nome "José Sarney" temos que digitar:

Print NomeCliente(2)

Se digitar somente NomeCliente o Visual Basic não irá retornar um erro em tempo de execução:

Este erro acontece porque uma vez definido NomeCliente como matriz, ela poderá somente ser chamada como uma matriz (com seu índice).

A abrangência e tempo de vida de uma matriz segue as mesmas regras estabelecida para as variáveis.

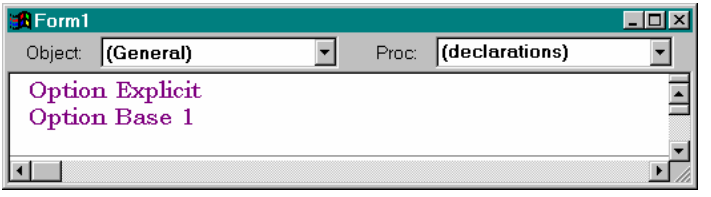
As regras de nomeação de uma matriz também são as mesmas aplicadas as variáveis.

7.3.1 Matrizes Unidimensional

Quando vamos criar uma matriz colocamos logo no início a quantidade de elementos que ela poderá conter. Essa quantidade colocamos entre parênteses e assim é criado na memória do computador espaços reservados a todos os elementos que iremos usar.
Veja o exemplo abaixo, onde temos uma matriz de números inteiros contendo 10 elementos:

Dim Contador(10) as Integer

O primeiro elemento será referenciado com o índice 0, ou seja, Contador(0); o segundo elemento como Contador(1), e assim por diante. O índice do último elemento será sempre um a menos que a quantidade especificada entre os parênteses na declaração. Isto acontece porque como padrão o Visual Basic inicia sua contagem para índices em 0. No exemplo, o índice varia de 0 a 9. Mas isto não quer dizer que não podemos mudar esse padrão. Para isto usamos o comando **Option Base,** adicionado à seção **Declarations**. Option Base 0 faz com que o índice das matrizes inicie em 0, por outro lado, como Option Base 1, o valor inicial para para 1.



Se quiser personalizar as matrizes para algumas começarem a partir de um determinado numero de outras a partir de outro, podemos especificar na criação o menor e o maior índice que a matriz terá:

Dim Contador (1 to 10) as Integer
Dim NomeCliente(5 to 10) as String

O primeiro exemplo terá um índice de 1 a 10 e no outro um índice de 5 a 10. Qualquer número informado fora desse intervalo retornará um erro.

7.3.2 Matrizes Multidimensional

Imagine matriz multidimensional como sendo uma matriz dentro de outra. Por Exemplo:

Option Base 1
Dim Contador(3,2) as String.

Significa que temos um matriz de nome Contador, possui 3 elementos e cada elemento possui 2 elementos. Ou seja, no total, essa matriz terá 6 elementos. Veja como ficaria:

Contador(1,1) = "Janeiro"
Contador(2,1) = "Fevereiro"
Contador(3,1) = "Março"
Contador(1,2) = "Venda Ruim"
Contador(2,2) = "Venda Regular"
Contador(3,2) = "Venda Boa"

Veja como ficaria a representação gráfica dessa matriz

	Coluna 1		Coluna 2	
Linha 1	Janeiro	(1,1)	Venda Ruim	(1,2)
Linha 2	Fevereiro	(2,1)	Venda Regular	(2,2)
Linha 3	Março	(3,1)	Venda Boa	(3,2)

Perceba que esses tipos de matriz são multidimensional porque criam subdivisões dentro de cada elemento. Poderíamos ter matrizes do tipo:

Dim Contador(3,5,8) as Integer

Esse monstro que criamos significa essa matriz possui 3 elementos. Cada Elemento se subdivide em outros 5 elementos. Cada elemento desses cinco elementos se subdividem em outros 8 elementos, formando no total 120 elementos (3X5X8). Confesso que a primeira impressão é horrível. Mas a possibilidade existe e o bicho não é tão feio. Podemos usar matrizes multidimensional como uma planilha eletrônica de cálculo.

7.3.3 Matrizes Dinâmicas

As vezes necessitamos de criar uma matriz mas não sabemos de antemão quantos elementos ela terá, assim, não poderíamos especificar no comando **Dim** o valor máximo do índice.

Para isto, o Visual Basic criou as Matrizes Dinâmicas. Elas não possuem tamanho fixo. Podemos variar seu tamanho de acordo com nossa necessidade na execução do programa. A criação de uma matriz assim usa a seguinte sintaxe:

Dim NomeCliente() as String

Veja que entre parênteses deixamos vazio para durante o programa preenchermos.

Usamos o comando **ReDim** para fazer esse Redimensionamento da matriz. Exemplo: Se precisarmos incluir mais 1 elemento na matriz NomeCliente usariamos o comando:

ReDim NomeCliente(1)
NomeCliente(1) = "João Figueiredo"

Se depois precisarmos colocar mais 2 elementos usariamos:

ReDim NomeCliente(3)
NomeCliente(1) = "João Figueiredo"
NomeCliente(2) = "José Sarney"
NomeCliente(3) = "Fernando Collor"

Neste exemplo quando redimensionamos pela segunda vez tivemos que colocar um conteúdo novamente para o NomeCliente(1) pois o comando **ReDim** sempre apaga os dados armazenados anteriormente.

Quando não queremos que isto aconteça usamos a cláusula **Preserve** Juntamento com o ReDim. Assim os conteúdos anteriores são preservados. Veja:

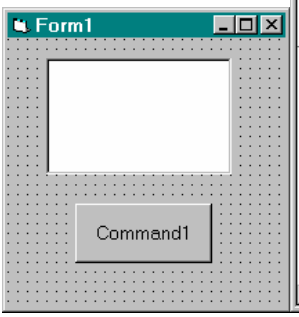
ReDim NomeCliente(1)
NomeCliente(1) = "João Figueiredo"

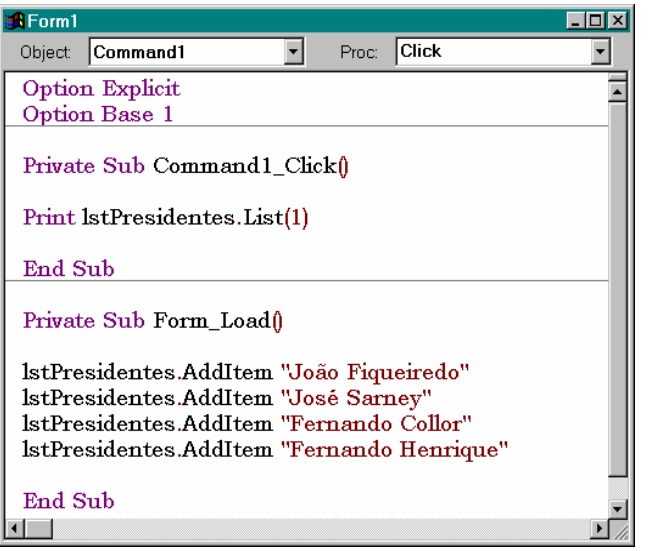
Se depois precisarmos colocar mais 2 elementos usaríamos:

ReDim Preserve NomeCliente(3)
NomeCliente(2) = "José Sarney"
NomeCliente(3) = "Fernando Collor"

7.3.4 Matrizes no ComboBox e no ListBox

Insira no formulário um ListBox e nomeie-o para lstPresidentes. Coloque também um botão de comando. Veja ao lado como ficará a codificação.





Objetos como o **ComboBox** e o **ListBox** usam matrizes para fazer referencia ao seu conteúdo através da propriedade **List.** Veja este exemplo:

Note que na propriedade **List** existe um índice que usamos para referenciar aos dados contido no Objeto ListBox. Este índice sempre inicia com 0 e vai até o ultimo existente dentro da Caixa de Lista. A propriedade **NewIndex** do objeto retorna o índice do ultimo item que entrou no ListBox, e a propriedade **ListCount** retorna a quantidade de itens existentes. No nosso exemplo, o NewIndex seria igual a 3 e o ListCount igual a 4.

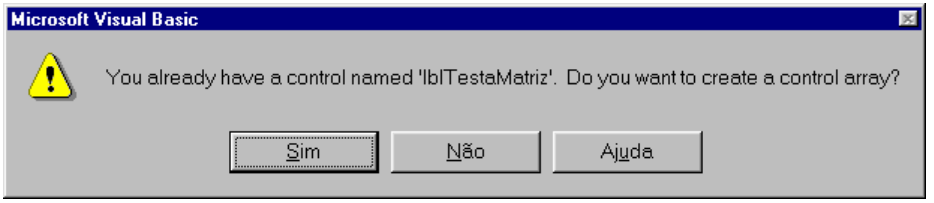
LstPresidentes.List(3) retorna a expressão “Fernando Henrique”.

Print lstPresidentes.List(lstPresidentes.NewIndex) retorna a expressão “Fernando Henrique” também.

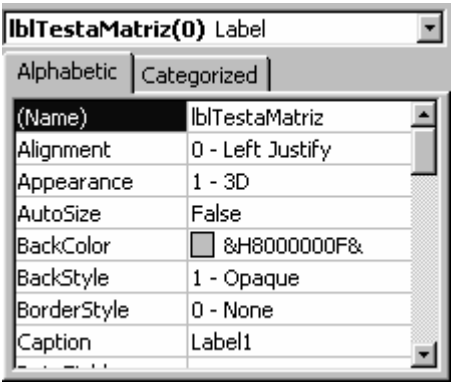
7.3.5 A Propriedade Index dos Objetos

Todos objetos que são inseridos no formulário, e possuem o mesmo nome, o Visual Basic automaticamente os converte para matriz, e na propriedade Index é inserido o número correspondente ao seu índice. Isto acontece porque não podemos ter dois ou mais objetos com o mesmo nome (propriedade **Name**) em nosso projeto.

Por exemplo, vamos colocar dois **Labels** no formulário e mudar sua propriedade **Name** para “lblTestaMatriz”. No primeiro o Visual Basic aceitará o nome sem problemas, no segundo **Label** quando tentarmos colocar o mesmo nome já usado anteriormente por outro **Label**, o Visual Basic avisará que o nome já existe e pergunta se não queremos usar uma matriz:

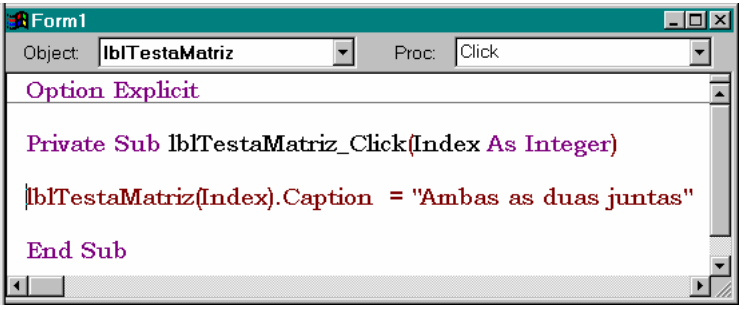


Se optarmos pelo sim ele criará uma matriz e renomeará o objeto para lblTestaMatriz(0) e lblTestaMatriz(1). A propriedade index de cada um possuirá os números 0 e 1 respectivamente.



Usando uma matriz o nosso objeto agora terá que ser referenciada com o seu índice para podermos saber qual objeto estamos querendo usar:

Se formos fazer alguma codificação para este objeto, teríamos que sempre colocar o índice correspondente ao objeto que estamos trabalhando.



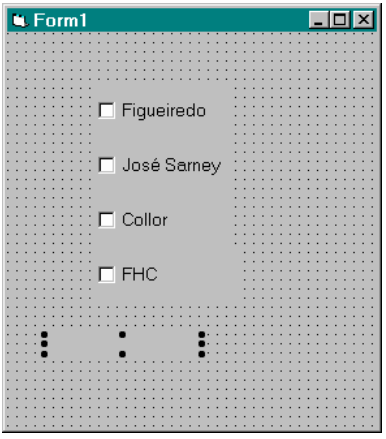
Note que o evento click que criamos para o Label possui um argumento “Index As Integer”. A variável index terá em seu conteúdo o número

do índice correspondente ao Objeto clicado. Ou seja, se chamarmos o evento click através do Label1 o número do índice será 0. Se usarmos o Label2 o índice será 1.

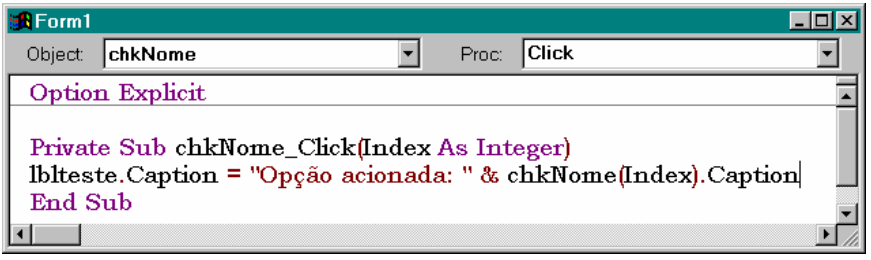
- Crie um novo formulário e nele coloque 4 Objetos CheckBox com as seguintes propriedades:

Propriedades	**Check1**	**Check2**	**Check3**	**Check4**
Name	chkNome	chkNome	chkNome	chkNome
Caption	Figueiredo	José Sarney	Collor	FHC

- Coloque também no formulário um Label. Deixe o Caption vazio, o Autosize em True e nomeio "lblTeste".



- Crie uma codificação para o objeto CheckBox da seguinte forma:



- Para conseguirmos o mesmo efeito sem usar matriz teríamos que criar um evento click para cada opção do CheckBox.

EXERCÍCIOS PROPOSTOS

1 - Cite três regra que temos que respeitar na criação de uma variável:

2 - Explique o comando DIM:

3 - Quais os tipos de variáveis existente?

4 - O que você entendeu de variável LOCAL?

5 - A seção “declarations” da janela de codificação tem qual finalidade?

6 - O que você entendeu de Matriz?

7 - Qual a diferença de Matriz Unidimensional e Multidimensional?

8 - O que é uma matriz Dinâmica?

9 - Qual a diferença da propriedade NewIndex e ListCount?

10 - Explique a propriedade Index dos objetos.

8 OPERADORES

- Matemático
- Relacional
- Lógico
- String

8.1 OPERADORES

A finalidade básica dos operadores são para comparar, calcular, igualar, concatenar... enfim, fazer operações envolvendo variáveis ou campos de Banco de Dados.

8.1.1 Operadores Matemáticos

Estes tipos de operadores possuem a finalidade de efetuar cálculos e já são bastante conhecidos, principalmente para quem usa calculadora.

Operador	Descrição	Exemplo
+	Soma	Var = 18+5
-	Subtração	Var = 18-5
*	Multiplicação	Var = 18*5
/	Divisão	Var = 18/5
\	Divisão. Resultado será um número inteiro	Var = 18\5
^	Exponenciação	Var = 18^5
Mod	Resto da Divisão	Var = 18 Mod 5

Esses operadores respeitam uma ordem de precedência universal:

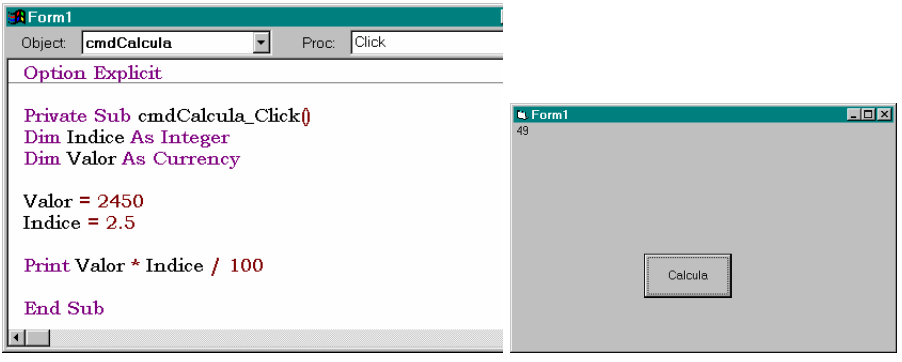
1ª. Exponenciação

2ª. Multiplicação e Divisão

3ª. Adição e Subtração

Caso precisamos de efetuar um calculo: 3+4*2^2/4-1+5 resultará em 11. Se por algum motivo precisarmos que o programa efetue a soma primeiro, então devemos coloca-la entre parênteses: (3+4)*2^2/4-(1+5) = 1. Usamos os parênteses para mudar a ordem de precedência dos operadores.

Os operadores atuam não somente em números, mas também nas variáveis cujo conteúdo seja valores numéricos. Exemplo:.



8.1.2 Operadores Relacionais

Estes operadores fazem comparações entre variáveis ou expressões. O retorno desta comparação será sempre True (Verdadeiro) ou False (Falso). Nos exemplos abaixo todas as comparações serão verdadeiras.

Operador	Descrição	Exemplo
>	Menor	3 < 5
<	Maior	5 > 3
<=	Menor ou Igual	3 <= 5
>=	Maior ou Igual	5 >= 3
=	Igual	5 = 5
<>	Diferente	3 <> 5

8.1.3 Operadores Lógicos

Enquanto que os operadores matemáticos retornam o resultado de um cálculo, os operadores lógicos sempre retornam True quando a lógica é exata, ou False quando a lógica não é exata.

And: Usamos quando precisamos comparar duas relações e ambas tem que ser verdadeira.

20 > 10 and 5 = 5

O Resultado será verdadeiro (True). 20 é maior que 10 **e** 5 é igual a 5.

20 > 10 and 5 = 4

O Resultado será falso (False). 20 é maior que 10 mas 5 não é igual a 4. Somente 1 das duas expressões deram certo.

Or: Usamos quando precisamos comparar dois relacionamentos e pelo menos um deve ser verdadeiro (ou um ou outro).

20 < 10 Or 5 > 4

O resultado será verdadeiro. 20 não é menor que 10, mas 5 é maior que 4.

20 < 10 Or 4 > 5

O resultado será falso. 20 não é menor que 10 e nem 4 é maior que 5.

Not: É um operador que inverte o resultado, passando algo que seria verdadeiro para falso e vice-versa.

Not(10 > 5)

O resultado será falso. 10 é maior que 5. Seria Verdadeira a comparação, mas como possui o Not, e ele inverte o resultado, o retorno é falso.

Xor: Este operador exige precisão. Não admite confusão na comparação dos relacionamentos. Ou é uma coisa ou outra. Nunca as duas.

8 > 10 Xor 8 > 6

Retorna verdadeiro, pois o resultado das duas comparações foram diferentes. A primeira é falsa e a segunda é verdadeira.

8 > 10 Xor 6 > 8

Retorna falso, pois o resultado das duas comparações foram semelhantes. A primeira é falsa e a segunda também.

10 > 8 Xor 8 > 6

Retorna falso, pois o resultado das duas comparações foram semelhantes. A primeira é verdadeira e a segunda também.

Eqv: Abreviatura de Equivalente. Faz uma equivalência lógica entre duas expressões.

10 > 8 Eqv 8 > 6

Retorna verdadeiro, pois ambas comparações são iguais.

8 > 10 Eqv 8 > 6

Retorna falso, pois as comparações retornam valores diferentes.

Imp: Abreviatura de Implicação (uma coisa implica em outra). O resultado será False somente se a primeira comparação for True e a segunda for False. Nos outros casos o resultado será sempre True. Este operador é o único onde a ordem dos operandos fazem diferença.

10 > 8 Imp 6 > 8

Retorna Falso, pois a primeira comparação é Verdadeira e a segunda é falsa.

10 > 8 Imp 8 > 6

Retorna Verdadeiro.

8 > 10 Imp 6 > 8

Retorna Verdadeiro.

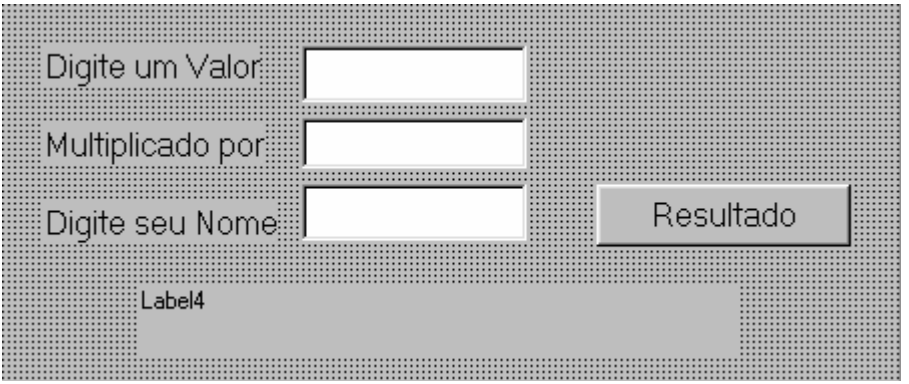
8.1.4 Operador de String

O mais simples de todos. É o símbolo “&” que serve para juntar duas expressões ou variáveis texto. Exemplo:

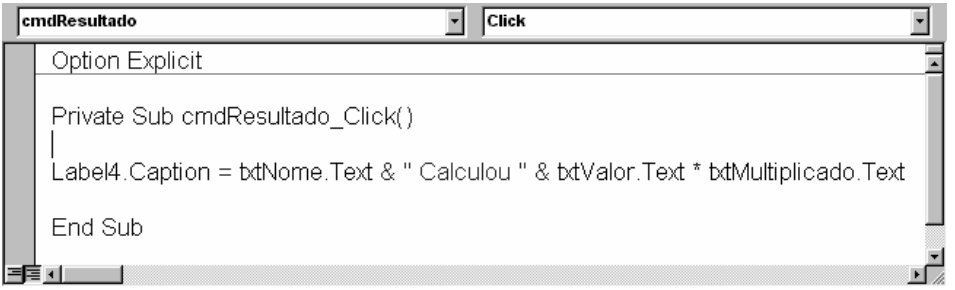
“Lionardo “ & “Fonseca Paiva”

Resulta em “Lionardo Fonseca Paiva”. O operador “+” também pode ser usado, mas não é aconselhado, uma vez que o “&” possui uma abrangência maior, convertendo expressões de outros formatos para o string para forçar a concatenação.

- Crie um formulário conforme modelo abaixo.

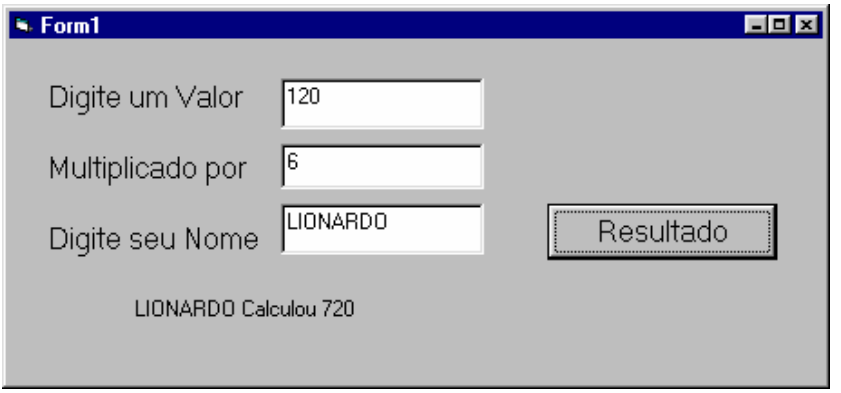


- Vamos criar um programa em que o usuário irá digitar um valor qualquer, ele valor será multiplicado pelo numero informado e o resultado era apresentado no "Label4" juntamente com o nome do usuário.
- Nomeie as caixas de texto para txtValor, txtMultiplicado e txtNome

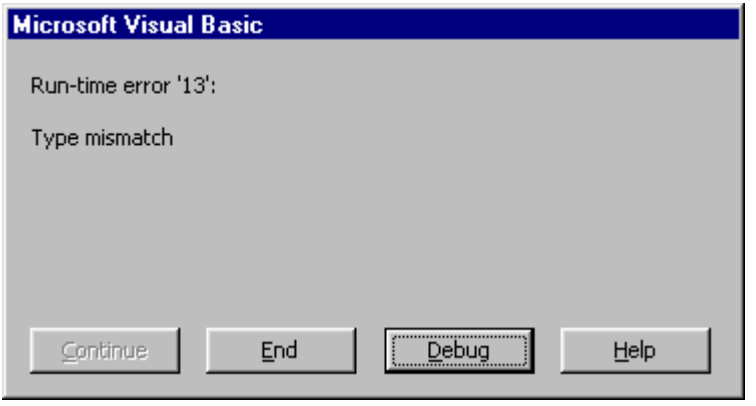


Vamos codificar o botão "cmdResultado":

- Isto fará com que o programa calcule o valor informado e concatene este valor com o nome do usuário.



- Importante o programador tentar cercar o máximo de erros possiveis que o usuário pode cometer, como por exemplo, se for digitado caracteres do alfabeto onde se espera numeros ou deixar um dos itens em branco, o programa irá dar a seguinte mensagem de erro:



- Para se evitar isto o programa deve verificar se os dados digitados são válidos para depois efetuar o cálculo.

EXERCÍCIOS PROPOSTOS

1 - Qual a diferença entre o operador / e o operador \ ?

2 - Quando que um operador retorna True ou False como resultado?

3 - Qual o resultado do operador Mod?

4 - Explique os operadores lógicos e exemplifque:

And:

Exemplo:

Or:

Exemplo:

Xor:

Exemplo:

Not:

Exemplo:

5 - Dê um exemplo de cada operador de comparação relacionado abaixo:

>

>=

<

<=

<>

=

9 COMANDOS CONDICIONAIS E DE LAÇO

- Condicionais
- Laço

9.1 COMANDOS CONDICIONAIS

Como o próprio nome já indica, estes comandos executam determinadas ações desde que algumas condições sejam favoráveis (verdadeiras).

IF <<Condição>> THEN... END IF

Traduzindo: IF = Se, THEN = Então, END IF = Fim do Comando Se.

Quando colocamos uma estrutura condicional IF dentro do nosso programa estamos querendo que o computador avalie uma condição e conforme for o resultado siga determinado caminho. Veja um exemplo:

```
Dim Contador as Integer
Contador = 20
IF Contador < 20 Then
        Print “Contador é menor que 20”
End If
Print “Fim do Programa”
```

O programa verifica: Se contador for menor que 20 então imprima no formulário “Contador é menor que 20”. Quando terminar imprima “Fim do Programa.

Quando uma determinada condição que colocamos no comando IF é verdadeira ele começa a executar todas as linhas que existem entre IF e END IF. Se a condição não for verdadeira então o programa passa a execução para a linha após o END IF e ignora tudo que houver dentro da estrutura IF... END IF.

```
Dim Contador as Integer
Contador = 20
IF Contador > 20 Then
        Print “Contador é maior que 20”
End If
Print “Fim do Programa”
```

Agora o programa irá verificar se Contador é maior que 20. Ele não é maior que 20. Esta condição é falsa. Então, sendo assim, ele não executará as linhas existentes dentro da estrutura IF...END IF, e só irá imprimir no formulário a frase “Fim do Programa”.

IF <<Condição>> THEN... ELSE... END IF

Traduzindo: IF = Se, THEN = Então, ELSE = Senão, END IF = Fim do Comando Se.

Acrescentamos na estrutura o comando ELSE, que é sempre executado quando a condição não é verdadeira.

```
Dim Contador as Integer
Contador = 20
IF Contador < 20 Then
        Print “Contador é menor que 20”
ELSE
        Print “Contador é maior ou igual a 20”
End If
Print “Fim do Programa”
```

O programa verifica: Se o Contador for menor que 20 imprima a frase: “Contador é menor que 20”, senão, imprima “Contador é maior ou igual a 20”.

Perceba que agora o programa irá executar um comando da estrutura, seja um ou outro, mas pelo menor um será executado. Se a condição for verdadeira o comando executado é sempre o que esta após o comando IF, se for falsa ele ignora as instruções que existem após o comando IF e passa a execução para a linha após o ELSE e vai até o final.

Lembre-se: Sempre que usar o comando IF é necessário usar o comando THEN após a condição, e depois encerrar a estrutura condicional IF com END IF.

IF <<Condição>> THEN... ELSEIF... ENDIF

Traduzindo: IF = Se, THEN = Então, ELSEIF = Senão se, END IF = Fim do Comando Se.

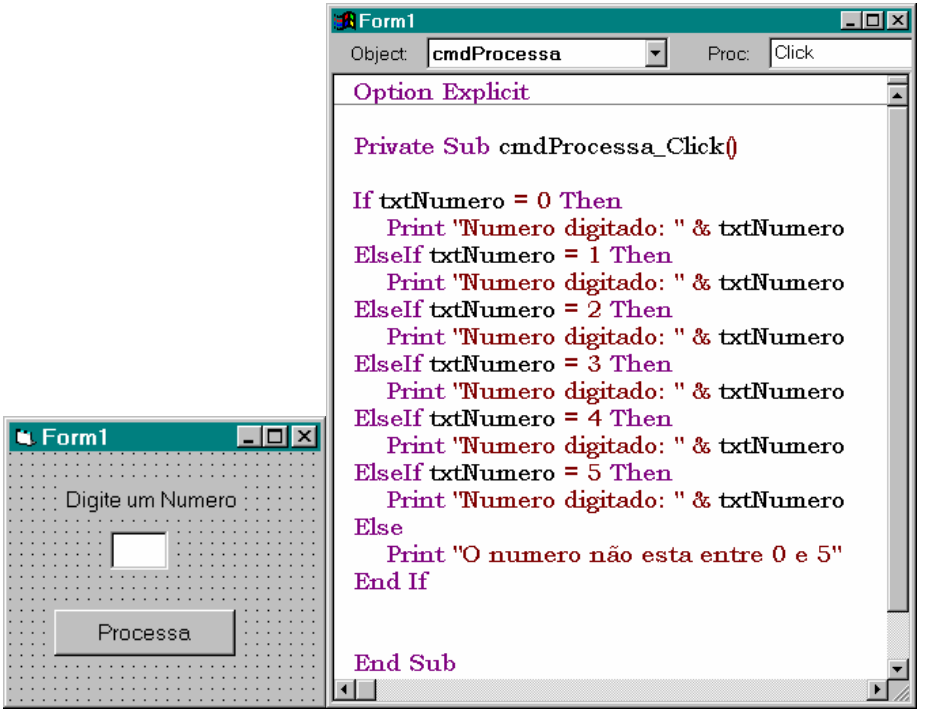
Modificamos comando ELSE para ELSE IF, que é sempre executado quando a condição não é verdadeira, mas ainda dependendo de uma outra condição.

```
Dim Contador as Integer
Contador = 20
IF Contador < 20 Then
        Print “Contador é menor que 20”
```

ELSEIF Contador = 20
Print "Contador é igual a 20"
ELSE
Print "Contador é maior que 20"
End If
Print "Fim do Programa"

O programa verifica se o contador é menor que 20. Se não for verifica se é então igual a 20, senão irá imprimir que contador é maior que 20.

Veja outro exemplo:



SELECT CASE

Significa "Seleção de Casos". Ou seja, colocamos várias possibilidades (varios casos) para o Visual Basic e ele escolhe um.

Veja a estrutura do comando:

Opção = 3
Select Case Opção
Case 1
Print "Opção 1 acionada"
Case 2

Print “Opção 2 acionada”
Case 3
Print “Opção 3 acionada”
End Select

Veja que este comando verifica se a Opção é 1, depois verifica se é 2 e depois se é 3. Se a variável Opção for 3 ele executará as instruções contidas na próxima linha. Caso a variável Opção não seja 1,2 ou 3 então o comando SELECT CASE é encerrado.

Opção = 3
Select Case Opção
Case 1
Print “Opção 1 acionada”
Case 2
Print “Opção 2 acionada”
Case 3
Print “Opção 3 acionada”
Case Else
Print “Opção não é 1,2 ou 3”
End Select

Acrescentando na estrutura cláusula CASE ELSE (caso contrário), o Visual Basic irá verificar se a variável é 1,2 ou 3, não sendo então será CASE ELSE, e a linha seguinte a esta cláusula será executada.

No comando Case podemos usar também intervalos, como por exemplo:

Case 2 ***to*** *4*

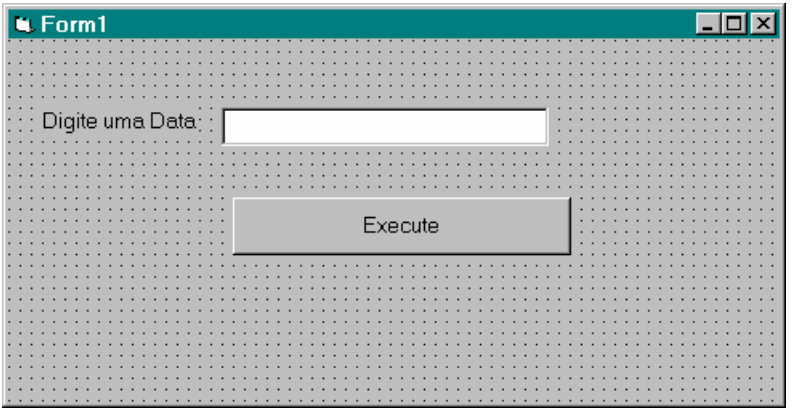
Ou seja, caso a condição esteja entre 2 e 4.

Caso queira que na verificação da condição seja avaliado se a variável é maior ou menor que determinada expressão, usamos:

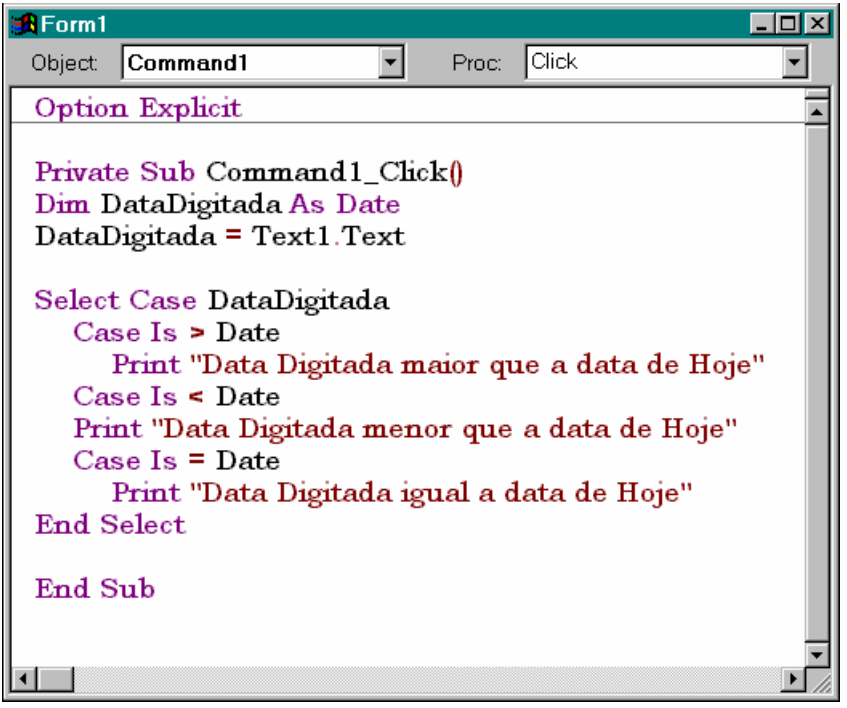
Case **Is** > 50

]

- Em um novo formulário crie uma caixa de texto para digitarmos uma data e um botão como esta abaixo:



- Vamos codificar o botão para quando digitarmos uma data o programa verificar se é igual a data de hoje ou não. Para obtermos da data de hoje do sistema temos que usar a função **Date.**



- O programa analizara a data digitada, e mostrará no vídeo como é a data em comparação com a data de hoje.

9.2 COMANDOS DE LAÇO

Estes comandos criam círculos viciosos, ou seja, criam um estrutura condicional que se repetem até a que condição seja satisfeita.

DO WHILE <<Condição>> LOOP (Faça enquanto)

Executa todos os comandos que existem entre DO WHILE e o LOOP. Quando a execução do programa encontra o comando LOOP, o DO WHILE é reavaliado e a rotina continua até que a condição estabelecida esteja satisfeita.

```
Contador = 0
Do While Contador < 10
        Contador = Contador + 1
        Print Contador
Loop
Print "Fim da execução"
```

Faça enquanto Contador for menor que 10.

Inicializamos uma variável de nome Contador com Zero, e pedimos para o programa: "Repita as instruções abaixo enquanto Contador for menor que 10". O comando LOOP faz com que o programa volte para a linha do DO WHILE e teste a condição de novo. Somente quando a condição for verdadeira, ou seja, quando Contador for maior ou igual a 10, que o programa executará a linha após o LOOP.

Perceba que dependendo do resultado da condição os comandos existentes dentro da estrutura podem não serem executados, passando direto para a linha após o comando LOOP.

DO ... LOOP WHILE <<Condição>> (Repita enquanto)

Neste caso o Comando Do abre a seqüência de repetição, mas não faz nenhum teste de condição. Este teste é feito no final da estrutura com o comando LOOP WHILE, ou seja, obrigatoriamente as instruções contidas após o comando DO serão executadas.

```
Contador = 0
Do
    Contador = Contador + 1
    Print Contador
Loop While contador < 10
Print "Fim da execução"
```

Retorne para repetir os comandos enquanto o Contador for menor que 10.

DO UNTIL <<Condição>> LOOP (Faça até que)

Executa todos os comandos que existem entre DO UNTIL e o LOOP. Quando a execução do programa encontra o comando LOOP, o DO UNTIL é reavaliado e a rotina continua até que a condição estabelecida esteja satisfeita.

```
Contador = 0
Do Until contador >= 10
        Contador = Contador + 1
        Print Contador
Loop
Print "Fim da execução"
```

Faça até que Contador seja maior ou igual a 10.

Inicializamos uma variável de nome Contador com Zero, e pedimos para o programa: "Repita as instruções abaixo até que Contador seja maior ou igual que 10". O comando LOOP faz com que o programa volte para a linha do DO UNTIL e teste a condição de novo. Somente quando a condição for verdadeira, ou seja, quando Contador for realmente maior ou igual a 10, que o programa executará a linha após o LOOP.

Semelhante ao DO WHILE, dependendo do resultado da condição os comandos existentes dentro da estrutura podem não serem executados, passando direto para a linha após o comando LOOP.

DO ... LOOP UNTIL <<Condição>> (Repita até que)

O Comando DO abre a seqüência de repetição, mas não faz nenhum teste de condição. Este teste é feito no final da estrutura com o comando LOOP UNTIL, ou seja, obrigatoriamente as instruções contidas após o comando DO serão executadas.

```
Contador = 0
Do
    Contador = Contador + 1
    Print Contador
Loop Until Contador >= 10
Print "Fim da execução"
```

Retorne para repetir os comandos até que o Contador seja maior ou igual a 10.

FOR <<Intervalo>> ... NEXT (Conte de número inicial até numero final)

O comando FOR faz uma contagem de um determinado intervalo de números. Sempre que essa contagem encontra com o comando NEXT (próximo) a execução do programa retorna ao comando FOR até que o número final seja alcançado. Exemplo:

Conte de um até 10

```
Contador = 0
For Contador  = 1 to 10
    Print “A variável Contador agora vale “ &Contador
Next
Print “Fim da execução”
```

Neste programa o Visual Basic inicializa a variável Contador em zero, e o comando FOR avisa “vamos contar até 10, começando pelo 1”. As instruções contidas na linha abaixo são executadas, e quando o comando NEXT é encontrado a execução volta para o FOR, e a variável Contador é incrementada em mais um e assim por diante. Quando Contador for igual a 10 a estrutura FOR/NEXT é desfeita.

STEP

Usamos o STEP em conjunto com o FOR para fazer com que a contagem seja incrementada. Exemplo: Se queremos que o comando conte de 2 em 2 colocamos *FOR Variável = 1 TO 10 STEP 2*

EXIT <<Comando>>

- **Exit Sub**: Força a saída da Sub rotina. Quando a execução do programa encontra este comando, o Visual Basic transfere o controle do programa para a linha seguinte a aquela que chamou a rotina.
- **Exit Function**: Força a saída da função. Quando a execução do programa encontra este comando, o Visual Basic transfere o controle do programa para a linha seguinte a aquela que chamou a rotina.
- **Exit Do**: Força a saída de um LOOP, seja WHILE ou UNTIL, mesmo que a condição estabelecida não seja verdadeira:

```
Contador = 0
Do While Contador < 10
        Contador = Contador + 1
        Print Contador
        if Contador = 5 then
          Exit Do
        end if
```

```
Loop
Print “Fim da execução”
```

- **Exit For:** Força a saída de um FOR..NEXT, mesmo que o número final não tenha sido alcançado.

```
Contador = 0
For Contador  = 1 to 10
    Print “A variável Contador agora vale “ &Contador
    If Contador = 6
        Exit For
    end if
Next
Print “Fim da execução”
```

expressões de outros formatos para o string para forçar a concatenação.

EXERCÍCIOS PROPOSTOS

1 - Cite duas diferenças entre um laço DO WHILE e um laço DO - LOOP WHILE?

2 - Qual intrução que verifica sua condição antes da execução das rotinas que fazem parte do laço?

3 -Se necessitassemos executar um vários comandos e funções repetidamente até que uma determinada condição fosse verdadeira, qual loop usuariamos?

4 -Explique quando usamos um loop DO UNTIL e quando usamos um DO - LOOP UNTIL:

5 - De um exemplo de utilização do loop FOR..NEXT

10 FUNÇÕES DE AUXILIO

- Data e Hora
- Conversão
- String
- Matemática
- Entrada de Dados
- Identificação
- Manipulação de Matriz

ANOTAÇÕES PARA NÃO ESQUECER

10.1 FUNÇÕES

Funções nada mais é que rotinas prontas para executar determinadas ações.

Existem no Visual Basic diversos tipos de funções que nos auxiliam na programação em todos os aspectos. São funções que fazem cálculos, funções que exibem resultados, funções para todos os gostos. Vamos começar a ver algumas dessas funções e com isto descobriremos o poder que elas exercem dentro da linguagem e sua utilidade.

Existem funções que precisam de um argumento para executar uma tarefa, outras somente retorna algum valor sem necessidade do argumento, como visto nos exemplos abaixo:

10.1.1 Funções matemáticas

ABS: Retorna sempre um valor positivo (absoluto).
VariávelNumérica = Abs(Expressão numérica)
Print Abs(-45)
Valor Impresso: 45

ATN: Retorna o arco-tangente de um número.
VariávelNumérica = Atn(Expressão numérica)

COS: Calcula o cosseno de um ângulo
VariávelNumérica = Cos(<expressãoNumérica>)

FIX: Retorna a parte inteira de um número, ignorando as casas decimais, se houver. Não faz arredondamento
VariávelNumérica = Fix(<expressãoNumérica>)
Print Fix(145.87)
Valor Impresso: 145
Print Fix(-145.87)
Valor Impresso: -145

HEX: Retorna a representação hexadecimal de um número decimal.
VariávelNumérica = Hex(<expressãoNumérica>)

INT: Retorna a parte inteira de um número, ignorando as casas decimais, se houver. Não faz arredondamento. Se o argumento for um número negativo será incrementado em um.
VariávelNumérica = INT(<expressãoNumérica>)

Print Int(145.87)
Valor Impresso: 145
Print Int(-145.87)
Valor Impresso: -146

LOG: Calcula o logaritmo natural de um número
VariávelNumérica = LOG(<expressãoNumérica>)

RND: Retorna um número randômico, ou seja, escolhe um número aleatoriamente.
VariávelNumérica = Rnd[(<expressãoNumérica>)]

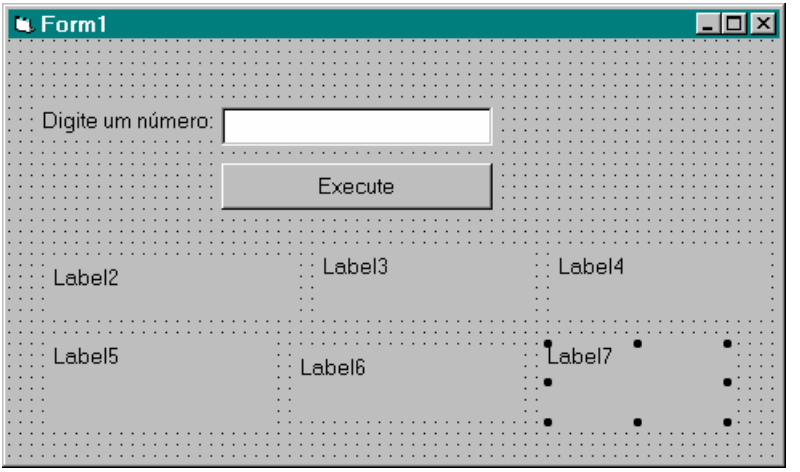
SGN: Retorna -1 se o argumento for um número negativo, e 1 se for um número positivo.
VariávelNumérica = Sgn(<expressãoNumérica>)

SIN: Calcula o seno de um ângulo.
VariávelNumérica = Sin(<expressãoNumérica>)

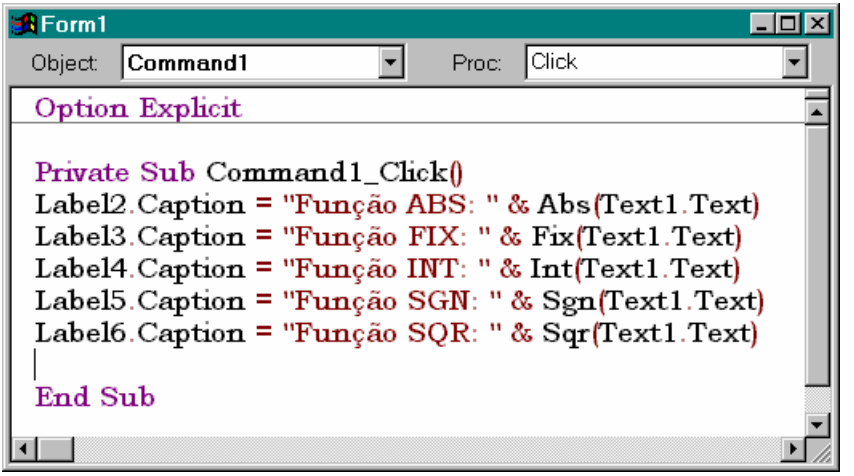
SQR: Calcula a raiz quadrada de um número.
VariávelNumérica = Sqr(<expressãoNumérica>)

TAN: Calcula a tangente de um ângulo.
VariávelNumérica = Tan(<expressãoNumérica>)

- Crie um novo formulário da seguinte forma:



- Para todos os labels deixe o Caption vazio e habilite o AutoSize.
- Faça a seguinte codificação para o botão:



10.1.2 Funções de Conversão

CBOOL: Converte uma expressão para um valor lógico (True ou false). Se o argumento for um zero, retornará False, caso contrário será True.

VariávelLógica = CBool(<expressão>)

Print Cbool(43)

Valor Impresso: True
Print Cbool(0)
Valor Impresso: False
Print Cbool(4 = 6)
Valor Impresso: False. O resultado da comparação não é verdadeira.

CBYTE: Converte uma expressão para um tipo Byte.
VariávelByte = cbyte(<expressão>)
Print Cbyte(155.56)
Valor Impresso: 156
Print Cbyte(355.56)
Erro. Overflow. Expressões do tipo byte não podem ser maior que 255.

CCUR: Converte uma expressão numérica para um tipo-moeda.
VariávelCurrency = Ccur(<expressão numérica>)
Print Ccur(120)
Valor Impresso: 120

CDATE: Converte uma expressão para um tipo Data. Entretanto, esta conversão se concretizará desde que a expressão usada como argumento seja mesmo no formato de uma data, ou seja, dia/mês/Ano. Se pedirmos para fazer conversão da palavra “teste” para data será retornado um erro.
VariávelData = Cdate(<expressão>)
Print Cdate(“12/04/95”)
Valor Impresso: 12/04/95
Print Cdate(“120495”)
Erro. O conteúdo informado como argumento não esta formatado como data.
Print Cdate(“30/02/95”)
Erro. O argumento não é uma data real
É importante salientar que essa conversão é necessário principalmente quando o usuário digita uma data e esta data vai ser objeto de cálculo. Então a conversão é necessária.

CDBL: Converte uma expressão numérica em um número de ponto flutuante de precisão dupla.
VariávelNumérica = Cdbl(<expressão numérica>)

CINT: converte uma expressão numérica em um número inteiro. Faz arredondamento.
VariávelNumérica = CInt(<expressão numérica>)

Print CInt(45.40)
Valor Impresso: 45
Print CInt(45.60)
Valor Impresso: 46

CLNG: Converte uma expressão numérica em um número inteiro longo.
VariávelNumérica = CLng(<expressão>)
VariávelNumérica = CLng(<expressão numérica>)
Print Clng(45.40)
Valor Impresso: 45
Print Clng(45.60)
Valor Impresso: 46
A diferença entre a função CINT e CLNG é a abrangência da própria variável.

CSNG: Converte uma expressão numérica em um número de ponto flutuante de precisão simples.
VariávelNumérica = CSng(<expressão >)

CSTR: Converte uma expressão numérica, data ou outra em uma string (texto).
VariávelString = CStr(<expressão>)
Print Cstr(452)
Valor Impresso: 452

CVAR: Converte uma expressão de qualquer tipo para o tipo variante.
VariávelVariant = Cvar(<expressão>)

STR: Converte um valor numérico para o tipo String (texto). Valido somente para argumentos numéricos.
VariávelString = Str(<expressãoNumérica>)
Print Str(452)
Valor Impresso: 452

STRCONV: Retorna uma string convertida de acordo com o tipo de conversão especificado.
VariávelString = Strconv(<ExpressãoString>, <TipoDeConversão>)
Tipos de conversão que podemos usar:
vbUpperCase Converte toda a expressão em letras maiúsculas.
vbLowerCase Converte toda a expressão em letras minúsculas.

vbProperCase Converte somente a primeira letra em maiúscula e o restante em minúsculo.

Print StrConv(“Lionardo”, vbUpperCase)

Valor Impresso: LIONARDO

Print StrConv(“LIONARDO”, vbProperCase)

Valor Impresso: Lionardo

ASC: Retorna o código ANSI do primeiro caractere de uma String.

VariávelNumérica = Asc(<string>)

Print Asc("B")

Valor Impresso: 66 (Numero correspondente na tabela ASCII da letra B.)

CHR: Retorna o caractere correspondente ao código na tabela ASCII

VariávelString = Chr(<códigoDoCaractere>)

Print Chr(66)

Valor Impresso: B

VAL: Converte uma String com caracteres numéricos em uma variável númerorica.

VariávelNumérica = Val(<stringNumérica>)

Print Var(“003”)

Valor Impresso: 3

Print Var(“123”)

Valor Impresso: 123

10.1.3 Funções de Data e Hora

DATE: Retorna a data corrente do sistema operacional.

VariávelData = Date

DATEADD: Incrementa uma data nos dias, meses ou anos especificados.

VariávelVariant = DateAdd(<Intervalo>, <Incremento>,<ExpressãoData>)

Retorna uma Variant que contém uma data à qual foi adicionado um determinado intervalo de tempo.

<Intervalo> Expressão de seqüência de caracteres que é o intervalo de tempo que você quer adicionar. Tipos de caracteres usados:

yyyy	Ano
q	Trimestre
m	Mês
y	Dia do ano
d	Dia
w	Dia da semana
ww	Semana
h	Hora
n	Minuto
s	Segundo

<Incremento> Expressão numérica que é o número de intervalos que você quer adicionar. Pode ser positivo (para obter datas no futuro) ou negativo (para obter datas no passado).

<ExpressãoData> Data que está sendo adicionada.

Print DateAdd("d",2,"31/07/96")
Valor Impresso: 02/08/96

A função DateAdd pode ser usada para adicionar a uma data ou subtrair dela um intervalo de tempo especificado. Por exemplo, você poderia usar DateAdd para calcular uma data 30 dias a partir de hoje ou uma hora que é 45 minutos a partir de agora.

Se quiser adicionar dias a data, você pode usar Dia do Ano ("y"), Dia ("d") ou Dia da Semana ("w").

A função DateAdd não retornará uma data inválida. O exemplo abaixo adiciona um mês a 31 de janeiro:

DateAdd("m", 1, "31-Jan-95")

Neste caso, DateAdd retorna 28-Fev-95 e não 31-Fev-95. Se date for 31-Jan-96, ele retornará 29-Fev-96 pois 1996 é um ano bissexto.

DATEDIFF: Calcula o intervalo entre duas datas.

VariávelData=DateDiff(<intervalo>, <expressãoData1>, <expressãoData2>)

<Intervalo> Expressão de seqüência de caracteres, que é o intervalo de tempo que você usa para calcular a diferença entre uma data e outra.

O argumento intervalo tem estas configurações:

yyyy	Ano
q	Trimestre

m	Mês
y	Dia do ano
d	Dia
w	Dia da semana
ww	Semana
h	Hora
n	Minuto
s	Segundo

<ExpressãoData1 e 2> Duas datas que você quer usar no cálculo.
Dim data1 As Date
Dim data2 As Date
Data1 = #31/7/96#
Data2 = #8/8/96#
Print DateDiff("d", data1, data2)
Valor Impresso: 8

A função DateDiff pode ser usada para determinar quantos intervalos de tempo especificados existem entre duas datas. Por exemplo, você poderia usar DateDiff para calcular o número de dias entre duas datas ou o número de semanas entre o dia de hoje e o final do ano.

Se quiser saber o número de dias entre date1 e date2, você pode usar tanto o Dia do Ano ("y") como o Dia ("d").

Quando interval é Dia da Semana ("w"), DateDiff retorna o número de semanas entre as duas datas. Se date1 cair numa segunda-feira, DateDiff contará o número de segundas-feiras até a date2. Ele conta date2, mas não date1. Se, contudo, interval for Semana ("ww"), a função DateDiff retornará o número de semanas do calendário entre as duas datas. Ele conta o número de domingos entre date1 e date2. DateDiff contará date2 se ela cair num domingo; mas não contará a date1, mesmo que caia num domingo.

Se date1 se referir a um ponto no tempo posterior à date2, a função DateDiff retornará um número negativo.

Se data for um literal de data (uma data entre sinais numéricos (#)), o ano, se especificado, se tornará uma parte permanente dessa data. Contudo, se data estiver entre aspas ("") e você omitir o ano, o ano atual será inserido no código sempre que a expressão data for avaliada. Isto torna possível escrever um código que pode ser usado em anos diferentes.

DATEPART: extrai de uma determinada data uma parte dela relativo a dia, mês, semana, quinzena, etc.

ParteDaData = DatePart(<intervalo>, <expressãoData>)
Intervalo pode ser:

yyyy	Ano
q	Trimestre
m	Mês
y	Dia do ano
d	Dia
w	Dia da semana
ww	Semana
h	Hora
n	Minuto
s	Segundo

Dim Data As Date
Data = #8/2/96#
Print DatePart("m", data)
Valor Impresso: 8

DATESERIAL: Retorna uma data para um dia, mês e ano especificados, ou seja, informe cada item separado que a função monta a data.

VariavelData = DateSerial(<ano>, <mês>, <dia>)
Print DateSerial("96","08","02")
Valor Impresso: 02/08/96

DATEVALUE: Retorna a data especificada numa string, ou seja, converte uma variável String para o tipo Data.

VariávelData = DateValue(<VariávelStringDeData>)
Print DateValue("02/08/96")
Valor Impresso: 02/08/96

DAY: Retorna o dia do mês referente a uma data.
VariávelNumérica = Day(<expressãoData>)
Print Day("02/08/96")
Valor Impresso: 2

HOUR: Retorna a hora de uma expressão de data e hora.
VariávelNumérica = Hour(<ExpressãoHora>)
Print Hour("12:34:12")
Valor Impresso: 12
Print Hour(now)
Valor Impresso: 15

MINUTE: Retorna o minuto de uma expressão de data e hora.
VariávelNumérica = Minute(<ExpressãoHora>)

MONTH: Retorna o mês de uma data.
VariávelNumérica = Month(<ExpressãoData>)

NOW: Retorna a data e a hora correntes do sistema operacional.
VariávelData = Now
Print Now
Valor Impresso: 02/08/96 15:08:07

SECOND: Retorna os segundos de uma expressão de data e hora.
VariávelNumérica = Second(<ExpressãoHora>)

TIME: Retorna a hora corrente do sistema operacional.
VariávelData = Time
Print Time
Valor Impresso: 15:08:07

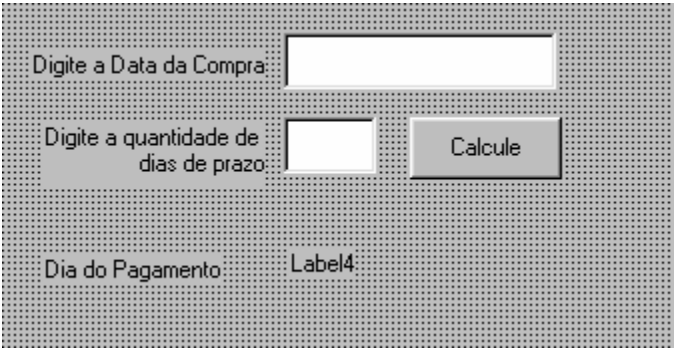
TIMER: Calcula a quantidade de segundos passados desde a meia noite.
VariávelNumérica = Timer

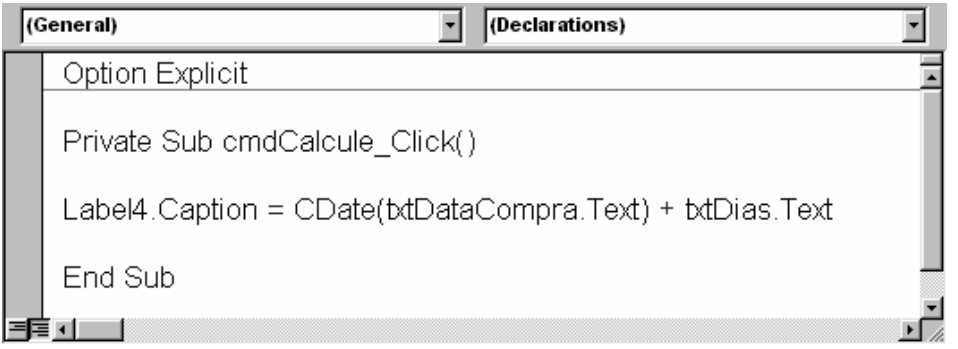
TIMEVALUE: Retorna a hora especificada numa string, ou seja, converte uma String cujo conteúdo esta no formato de hora para uma variável tipo Data e Hora.
VariávelData = TimeValue(<ExpressãoStringDeHora>)

WEEKDAY: Retorna o dia da semana de uma data, ou seja, seu numero correspondente: 1 para Domingo até 7 para Sábado.
VariávelNumérica = WeekDay(<ExpressãoData>)

YEAR: Retorna o ano de uma data.
VariávelNumérica = Year(<ExpressãoData>)

- Para criar um programa que faça calculos de data é bastante simples se usarmos as funções acima de forma correta.
- Abra um novo projeto e desenvolva um formulário como no modelo:

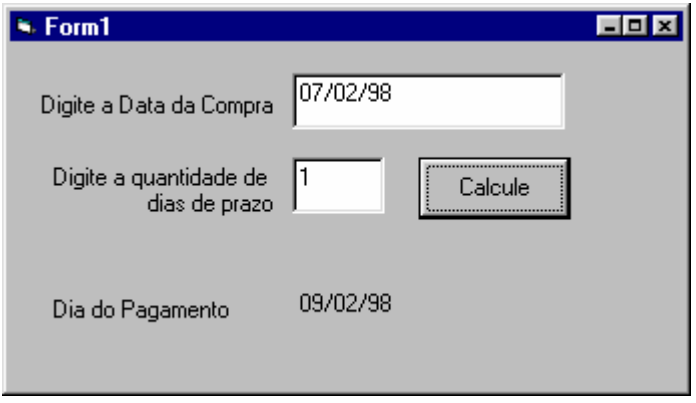




No botão "Calcule" crie a seguinte codificação:

- Na caixa de texto **txtDataCompra** é incluido dados no formato Texto, mesmo que seja uma data. Para o visual Basic entender que a expressão ali existente deve ser tratada como uma data e não como um texto, usamos a função de conversão **Cdate.**
- Vamos agora fazer a seguinte verificação: se o dia do pagamento cair num Domingo, fazer com que a data de pagamento passe para a Segunda-feira.

```
cmdCalcule                                   Click
Private Sub cmdCalcule_Click()
Dim DataPagamento As Date

DataPagamento = CDate(txtDataCompra.Text) + txtDias.Text

If WeekDay(DataPagamento) = 1 Then
   DataPagamento = DataPagamento + 1
End If

Label4.Caption = DataPagamento

End Sub
```



10.1.4 Funções de String

INSTR: Retorna a posição da primeira ocorrência de uma seqüência de caracteres dentro de outra

Variável = InStr ({<posiçãoInicial>,] <string>,
<SubStringAPesquisar>[, <MétodoDeComparação])

Posição Inicial: Expressão numérica que define a posição inicial de cada pesquisa. Se omitido, a pesquisa começa na posição do primeiro caractere.. Este argumento é necessário se o Método de Comparação for especificado.

String : Expressão de seqüência de caracteres que está sendo pesquisada.

SubStringAPesquisar : Expressão de seqüência de caracteres procurada.

MétodoDeComparação : Especifica o tipo comparação de seqüências de caracteres. Este argumento pode ser omitido, pode ser 0 ou 1. Especifique 0 (padrão) para realizar uma comparação binária. Especifique 1 para realizar uma comparação textual que desconsidere maiúsculas/minúsculas. Se este argumento for omitido, a configuração de **Option Compare** determinará o tipo de comparação.

Print InStr(8,"Lionardo Fonseca", "FO",1)

Valor Impresso: 10

A partir do 8º caractere procure na expressão “Lionardo Fonseca” em que posição começa a expressão “FO”, e ignore diferenciação maiúsculas/minúsculas.

Print InStr(8,"Lionardo Fonseca", "FO",0)

Valor Impresso: 0 (não encontrado).

LCASE: Converte uma expressão string para minúsculas.

VariávelString = Lcase (<stringAConverter>)

Print Lcase(“Editora Terra”)

Valor Impresso: editora terra

LEFT: Retorna uma quantidade de caracteres que se encontra da esquerda para a direita.

VariávelString = Left(<string>, <QuantidadeDeCaracteres>)

Print Left("Lionardo",4)

Valor Impresso: Lion

LEN: Retorna o número de caracteres de uma expressão String ou número de bytes requerido para armazenar uma variável.

VariávelNumérica = Len (ExpressãoCaractere>)

Print Len(“Lionardo”)

Valor Impresso: 8

LTRIM: Remove os espaços em branco à esquerda de uma String.

VariávelString = Ltrim (<ExpressãoString>)

MID: Retorna uma Substring de uma String, ou seja, retorna um número especificado de caracteres de uma seqüência de caracteres.

SubString=Mid(<string>,<posiçãoInicial>[, <quantidadeDeCaracteres>])

String : Expressão de seqüência de caracteres a partir da qual os caracteres são retornados.

PosiçãoInicial: Posição de caractere em String na qual a parte a ser considerada inicia. Se Posição Inicial for maior do que o número de caracteres em String, Mid retorna uma seqüência de caracteres de comprimento zero.

QuantidadeDeCaracteres: Número de caracteres a serem retornados. Se omitidos ou se o número de caracteres do texto for inferior ao QuantidadeDeCaracteres (inclusive o caractere em PosiçãoIncial), serão retornados todos os caracteres desde a PosiçãoInicial até o final da seqüência de caracteres.

Print Mid("Lionardo Fonseca Paiva",10,8)

Valor Impresso: Fonseca

Na expressão String “Lionardo Fonseca Paiva”, a partir do 10º caractere conte 8 para frente e retorne a String que esta neste intervalo.

RIGHT: Retorna uma substring com os caracteres que se encontram da direita para a esquerda dentro de uma expressão String.

VariávelNumérica=Right([<ExpressãoString>, <QuantidadeDeCaracteres>)

Print Right(“Lionardo Fonseca”,7)

Valor Impresso: Fonseca

RTRIM: Remove os espaços à direita em uma String.

VariávelString = Rtrim(<string>)

SPACE: Retorna uma String com uma determinada quantidade de espaços vázios.

VariávelString = Space(<quantidadeDeCaracteres>)

STR: Retorna a representação de um número como uma String.

VariávelString = Str (<ExpressãoNumérica>)

STRCOMP: Compara duas expressões Strings.

VariávelNumérica=StrComp(<string>,<string>[,Comparação>])

Comparação: Especifica o tipo de comparação de seqüências de caracteres. Este argumento pode ser omitido, pode ser 0 ou 1. Especifique 0 (padrão) para fazer uma comparação binária. Especifique 1 para fazer uma comparação de texto.

Valores de retorno da função: **-1** quando a primeira String for menor que a Segunda, **0** quando forem iguais e **1** quando a primeira String for maior que a segunda.

STRING: Repete um determinado caractere a quantidade de vezes estabelecido na função.

String = String (<QuantidadeDeCaracteres>, <caracteres>)
Print String(30,"-")
Valor Impresso: ----------

TRIM: Remove os espaços à esquerda e à direita de uma string.
VariávelString = Trim(<String>)

UCASE: Converte uma expressão String para maiúsculas.
VariávelString = Ucase (<string>)
Print Ucase(“Editora Terra”)
Valor Impresso: EDITORA TERRA

10.1.5 Funções de Manipulação de Matrizes

ARRAY: Retorna um Array do tipo Variant.
ArrayVariant = Array (<NúmeroDeElemento>)

NúmeroDeElemento: consiste de uma lista delimitada por vírgula de um número arbitrário de valores que são atribuídos aos elementos da matriz contidos em Variant. Se nenhum argumento for especificado, será criada uma matriz de comprimento zero.

Embora uma Variant que contém uma matriz seja conceitualmente diferente de uma matriz cujos elementos são do tipo Variant, o modo como são acessados os elementos da matriz é o mesmo. A notação usada para se referir a um elemento de uma matriz consiste no nome da variável seguido por parênteses que contêm um número do índice que indica o elemento desejado.

O limite inferior de uma matriz criada com a função Array é determinado pelo limite inferior especificado com a instrução Option Base.

Dim VariavelMatriz As Variant
VariavelMatriz = Array("Dom", "Seg", "Ter", "Qua", "Qui", "Sex", "Sáb")
Print VariavelMatriz(2)
Print VariavelMatriz(4)
Valor Impresso: Ter
Valor Impresso: Qui

LBOUND: Retorna o menor índice de uma matriz.
VariávelNumérica = Lbound(<NomeDaMatriz>)
Dim VariavelMatriz As Variant
VariavelMatriz = Array("Dom", "Seg", "Ter", "Qua", "Qui", "Sex", "Sáb")
Print LBound(VariavelMatriz)
Valor Impresso: 0

UBOUND: Retorna o maior índice de uma matriz.
VariávelMatriz = Ubound(<NomeDaMatriz>])

Dim VariavelMatriz As Variant
VariavelMatriz = Array("Dom", "Seg", "Ter", "Qua", "Qui", "Sex", "Sáb")
Print UBound(VariavelMatriz)
Valor Impresso: 6

10.1.6 Funções Lógicas

IIF: Analisa uma expressão lógica, e retorna valores para quando for falso ou verdadeiro.
ExpressãoDeRetorno = IIF (<ExpressãoLógica>,
<ExpressãoParaVerdadeiro>, <ExpressãoParaFalso>)

Print iif(10 > 2,"10 é maior que 2", "2 é maior que 10")
Valor Impresso: 10 é maior que 2

Print iif(10= 5,"Igual", "Diferente")
Valor Impresso: Diferente

SWITCH: Avalia uma lista de expressões e retorna o valor associado àquela primeiro avaliada como verdadeira.

Switch(Expr-1, Valor-1[, Expr-2, Valor-2 , Expr-etc, Valor-etc])

Expr : Expressão variant que você quer avaliar.
Valor : Valor ou expressão que é retornado se a expressão correspondente for True.

A lista de argumentos da função Switch consiste de pares de expressões e valores. As expressões são avaliadas à medida que aparecem

na lista da esquerda para a direita e é retornado o valor associado à primeira expressão para avaliar True. Se as partes não forem corretamente pareadas, ocorre um erro em tempo de execução. Por exemplo, se expr-1 for True, Switch retornará valor-1. Se expr-1 for False, mas expr-2 for True, Switch retornará valor-2, e assim por diante.

Switch retornará um valor Null se nenhuma das expressões for True.

Switch avalia todas as expressões, mesmo que retorne apenas uma delas. Por este motivo, você deve ficar atento para efeitos colaterais indesejáveis. Por exemplo, se a avaliação de qualquer expressão resultar num erro de divisão por zero, ocorrerá um erro.

A=2
B=2
C=2
Print Switch(A<2,"Letra A < 2", B=2,"Letra B = 2", C<3, "Letra C < 3")
Valor Impresso: Letra B = 2

A segunda expressão na lista foi impresso, pois foi a primeira a ter uma condição verdadeira. Note que o ultimo valor (C<3) também é verdadeiro, mas como a função Switch encerra sua execução quando encontra a primeira ocorrência verdadeira, ele foi ignorado.

A=2
B=3
C=2
Print Switch(A<2,"Letra A < 2", B=2,"Letra B = 2", C<3, "Letra C < 3")
Valor Impresso: Letra C < 3

10.1.7 Funções de Disco

CURDIR: Retorna o diretório corrente.
VariávelString = CurDir[(<drive>)]
Print CurDir("C:")
Valor Impresso: C:\WINDOWS\Desktop\

DIR: Retorna o nome de um arquivo, diretório ou pasta correspondente a um padrão especificado ou atributo de arquivo, ou a etiqueta de volume de uma unidade de disco.

VariávelString = Dir[(Nomedocaminho[, Atributos])]

Nomedocaminho : Expressão de seqüência de caracteres que especifica um nome de arquivo e pode incluir diretório ou pasta e unidade de disco. Null é retornado se nomedocaminho não for encontrado.

Atributos : Constante ou expressão numérica, cuja soma especifica atributos de arquivo. Se omitido, todos os arquivos normais que tiverem Nomedocaminho correspondente são retornados.

As configurações do argumento Atributos são:

vbNormal	0	Normal.
vbHidden	2	Oculto.
vbSystem	4	Arquivo de sistema,.
vbVolume	8	Etiqueta de volume; se especificada, todos os outros atributos são ignorados.
vbDirectory	16	Diretório ou pasta.

FILEDATETIME: Retorna a data e a hora da última atualização do arquivo.

VariávelData = FileDateTime(<NomeArquivo>)

Print FileDataTime("C:\WINDOWS\SYSTEM.INI")
Valor Impresso: 03/08/96 16:50:20

FILELEN: Retorna o tamanho do arquivo em bytes.
VariávelNumérica = FileLen(<NomeArquivo>)

Print FileLen("C:\WINDOWS\SYSTEM.INI")
Valor Impresso: 3481

GETATTR: Verifica os atributos de um arquivo ou diretório.
VariávelNumérica = GetAttr(<NomeArquivo>)
Veja os valores de retorno desta função:

0	vbNormal	Normal.
1	vbReadOnly	Somente Leitura.
2	vbHidden	Oculto.
4	vbSystem	Arquivo de sistema
16	vbDirectory	Diretório ou pasta.
32	vbArchive	O arquivo foi modificado desde o último backup.

Print GetAttr("C:\WINDOWS\SYSTEM.INI")
Valor Impresso: 32

10.1.8 Funções de Teste

ISARRAY: Testa se uma variável é uma matriz
VariávelBoolean = IsArray(<variável>)

ISDATE: Testa se o argumento pode ser convertido para uma data. Esta data deve estar dentro dos padrões de data.
VariávelBoolean = IsDate(<expressão>)

ISEMPTY: Verifica se uma variável foi inicializada.
IsEmpty retornará True se a variável estiver iniciada; caso contrário retornará False. Se a expressão contiver mais de uma variável, o retorno será sempre False.
VariávelBoolean = IsEmpty(<expressão>)

ISERROR: Testa se uma expressão é um valor de erro.
VariávelBoolean = IsError(<expressão>)

ISMISSING: Testa se um argumento opcional foi passado como parâmetro para uma procedure ou função.
VariávelBoolean = IsMissing(<NomedoArgumento>)

ISNULL: Testa se uma variável possui valor inválido.
VariávelBoolean = IsNull(<Expressão>)

ISNUMERIC: Testa se o argumento pode ser convertido para um número.
VariávelBoolean = IsNumeric(<Expressão>)

Print IsNumeric("AB")
Valor Impresso: False
Print IsNumeric("03")
Valor Impresso: True

ISOBJECT: Testa se uma expressão referencia a um objeto OLE válido.
VariávelBoolean = IsObject(<Expressão>)

VARTYPE: Retorna o tipo de variável especificada como argumento:
VariávelNumérica = VarType(<Variável>)
Retorna os valores abaixo:

0 Empty (Não iniciada).
1 Null Nenhum dado válido.
2 Inteiro (Integer).
3 Inteiro por extenso (Long)
4 Número de ponto flutuante de precisão simples (Single).
5 Número de ponto flutuante de precisão dupla (Double).
6 Moeda (Currency).
7 Data (Date).
8 Seqüência de caracteres textos (String).
9 objeto de Automação OLE (Object).
10 Erro (Error).
11 Booleano Boolean).
12 Variant (usada somente com matrizes de Variantes).
13 Um objeto que não seja de Automação OLE (DataObject).
17 Byte
8192 Matriz (Array).

TYPENAME: Retorna o nome descritivo do tipo de uma variável.
VariávelString = TypeName(<Variável>
Eis os nomes de retorno da função:

Byte	Um byte.
Integer	Um inteiro.
Long	Um inteiro por extenso.
Single	Um número de ponto flutuante de precisão simples.
Double	Um número de ponto flutuante de precisão dupla.
Currency	Um valor de moeda.
Date	Uma data.
String	Uma seqüência de caracteres.
Boolean	Um valor Booleano.
Error	Um valor de erro.
Empty	Não iniciado.
Null	Nenhum dado válido.
Object	Um objeto que suporta Automação OLE.

10.1.9 Funções de Escolha

CHOOSE: Seleciona um valor de uma lista de argumentos.
Variável = Choose (<índice>, <escolha>[, <escolha>]...)

Índice : Expressão numérica ou campo que resulta num valor entre 1 e o número de opções disponíveis.

Escolha : Expressão Variant que contém uma das possíveis opções.

Choose retorna um valor da lista de opções com base no valor de índice. Se índice for 1, Choose retorna a primeira opção da lista; se índice for 2, retorna a segunda opção e assim por diante.

Choose pode ser usado para pesquisar um valor numa lista de possibilidades. Por exemplo, se índice avalia para 3 e a opção-1 = "um", opção-2 = "dois" e opção-3 = "três", Choose retorna "três". Esta capacidade é particularmente útil se índice representar o valor num grupo de opções.

Se índice não for um número inteiro, ele será arredondado para o número inteiro mais próximo antes de ser avaliado.

Print Choose(2,"Apostilas","CD-ROM","Livros","Serviços Gráficos")
Valor Impresso: CD-ROM

INPUTBOX: Exibe um aviso numa caixa de diálogo, aguarda até que o usuário insira texto ou escolha um botão e retorna o conteúdo da caixa de texto.

Variável=Inputbox(<expressão>,<barraDeTítulo>,<escolhaDefault>)

EmpressãoPrompt : Expressão de seqüência de caracteres exibida como a mensagem numa caixa de diálogo. O tamanho máximo de prompt é de aproximadamente 1024 caracteres, dependendo da largura dos caracteres usados.

BarraDeTítulo : Expressão de seqüência de caracteres exibida na barra de títulos da caixa de diálogo. Se você omitir este argumento, o nome do aplicativo será incluído na barra de títulos.

EscolhaDefault : Expressão de seqüência de caracteres exibida na caixa de texto como a resposta padrão se nenhuma outra entrada for fornecida. Se você omitir EscolhaDefault, a caixa de texto será exibida vazia.

Se o usuário escolher "OK" ou pressionar ENTER, a função InputBox retorna o conteúdo da caixa de texto, qualquer que seja ele. Se o usuário escolher "Cancelar", a função retorna uma seqüência de caracteres de tamanho zero ("").

Observação : Se quiser especificar mais do que o primeiro argumento nomeado, você deverá usar InputBox em uma expressão. Se quiser omitir alguns argumentos posicionais, você deverá incluir o delimitador de vírgula correspondente.

Print InputBox("Digite Seu Nome","Cadastro de Cliente")

Valor Impresso: <Será o nome digitado na caixa de texto>

MSGBOX: Exibe uma caixa de diálogo para exibir informações e obter respostas simples através de botões de comando.

VariávelNumérica = MsgBox (<ExpressãoPrompt>,<BotõesDisponíveis>, <BarraDeTítulo>)

ExpressãoPrompt : Expressão de seqüência de caracteres exibida como a mensagem numa caixa de diálogo. O tamanho máximo de ExpressãoPrompt é de aproximadamente 1024 caracteres, dependendo da largura dos caracteres usados. Se ExpressãoPrompt for composto por mais de uma linha, você poderá separar as linhas usando um caractere de retorno de carro (Chr(13)), um caractere de alimentação de linha (Chr(10)) ou uma combinação de caracteres de retorno de carro e alimentação de linha (Chr(13) & Chr(10)) entre cada linha.

BotõesDisponível : Expressão numérica que é a soma de valores que especificam o número e tipo de botões a serem exibidos, o estilo de ícone a ser usado, a identidade do botão padrão e a modalidade da caixa de mensagem. Se omitido, o valor padrão será 0. Veja abaixo os tipos de botões disponível:

Constante	Valor	Descrição
vbOKOnly	0	Exibe apenas o botão "OK".
VbOKCancel	1	Exibe os botões "OK" e "Cancelar".
VbAbortRetryIgnore	2	Exibe os botões "Anular", "Repetir" e "Ignorar".
VbYesNoCancel	3	Exibe os botões "Sim", "Não" e "Cancelar".
VbYesNo	4	Exibe os botões "Sim" e "Não".
VbRetryCancel	5	Exibe os botões "Repetir" e "Cancelar".
VbCritical	16	Exibe o ícone "Mensagem crítica".

VbQuestion	32	Exibe o ícone "Consulta de advertência".
VbExclamation	48	Exibe o ícone "Mensagem de advertência".
VbInformation	64	Exibe o ícone "Mensagem de informação".
VbDefaultButton1	0	O botão "Primeiro" é o padrão.
VbDefaultButton2	256	O botão "Segundo" é o padrão.
VbDefaultButton3	512	O botão "Terceiro" é o padrão.
VbApplicationModal	0	Janela restrita do aplicativo; o usuário deve responder à caixa de mensagem antes de continuar seu trabalho no aplicativo atual.
VbSystemModal	4096	Janela restrita do sistema; todos os aplicativos são suspensos até que o usuário responda à caixa de mensagem.

O primeiro grupo de valores (05) descreve o número e tipo de botões exibidos na caixa de diálogo; o segundo grupo (16, 32, 48, 64) descreve o estilo de ícone; o terceiro grupo (0, 256, 512) determina qual botão é o padrão; e o quarto grupo (0, 4096) determina modalidade da caixa de mensagem. Quando adicionar números para criar um valor final para o argumento buttons, use somente um número de cada grupo.

Observação Estas constantes são especificadas pelo Visual Basic para aplicativos. Assim, os nomes podem ser usados em qualquer lugar do código em lugar dos valores reais.

BarraDeTítulo : Expressão de seqüência exibida na barra de títulos da caixa de diálogo. Se você omitir BarraDeTítulo, o nome do aplicativo será incluído na barra de títulos.

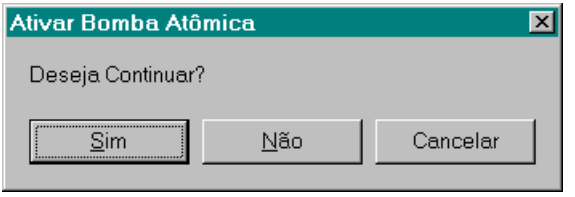
Estes são os valores que esta função retorna, para informar qual foi a ação do usuário:

Constante	Valor de Retorno	Botão escolhido
vbOK	**1**	"OK"
vbCancel	**2**	"Cancelar"
vbAbort	**3**	"Anular"
vbRetry	**4**	"Repetir"
vbIgnore	**5**	"Ignorar"
vbYes	**6**	"Sim"
vbNo	**7**	"Não"

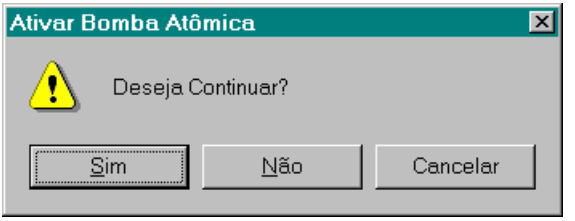
Se a caixa de diálogo exibir um botão "Cancelar", pressionar a tecla ESC terá o mesmo efeito que escolher "Cancelar".

Se quiser especificar mais do que o primeiro argumento nomeado, você deverá usar MsgBox em uma expressão. Se quiser omitir alguns argumentos de posição, você deverá incluir o delimitador de vírgula correspondente.

MsgBox ("Deseja Continuar?",VbYesNoCancel, "Ativar Bomba Atômica")

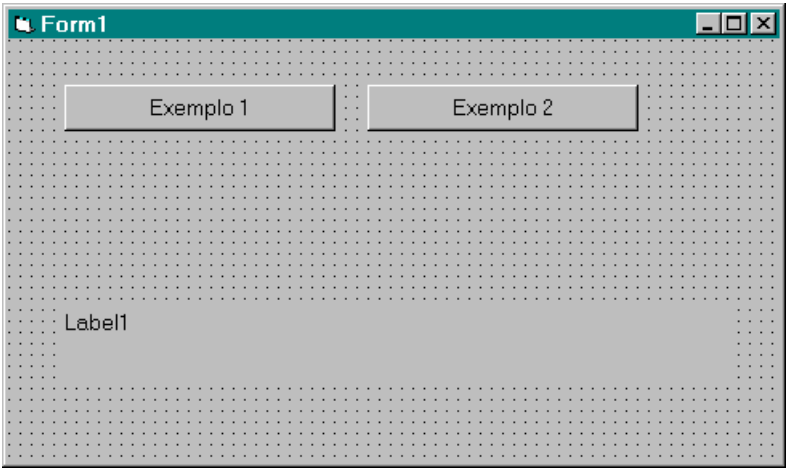


Print MsgBox ("Deseja Continuar?",VbYesNoCancel+VbExclamation, "Ativar Bomba Atômica")

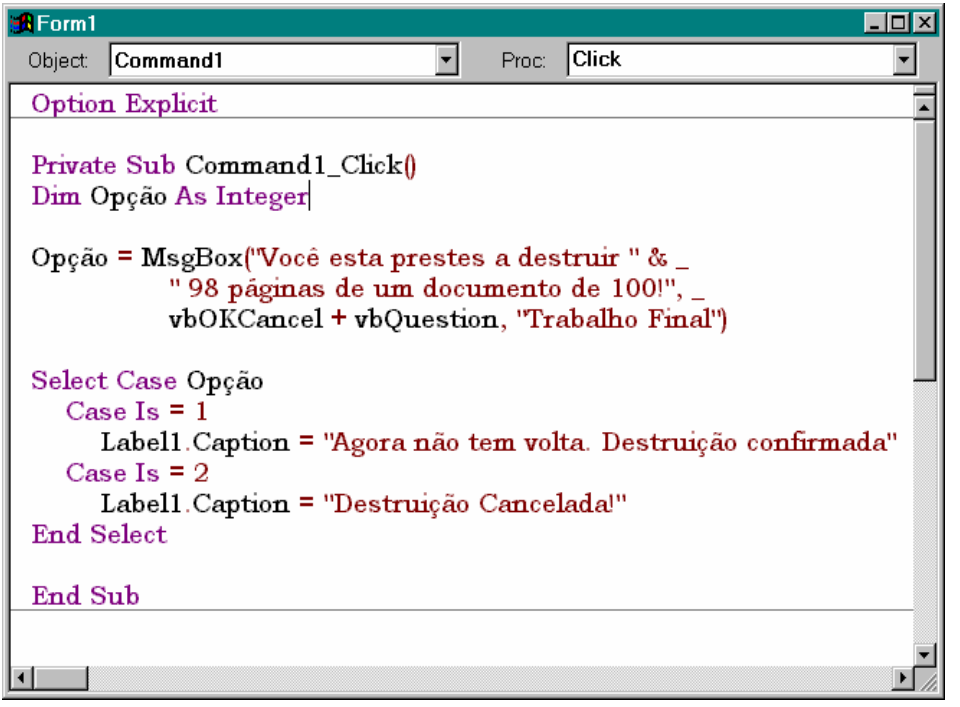


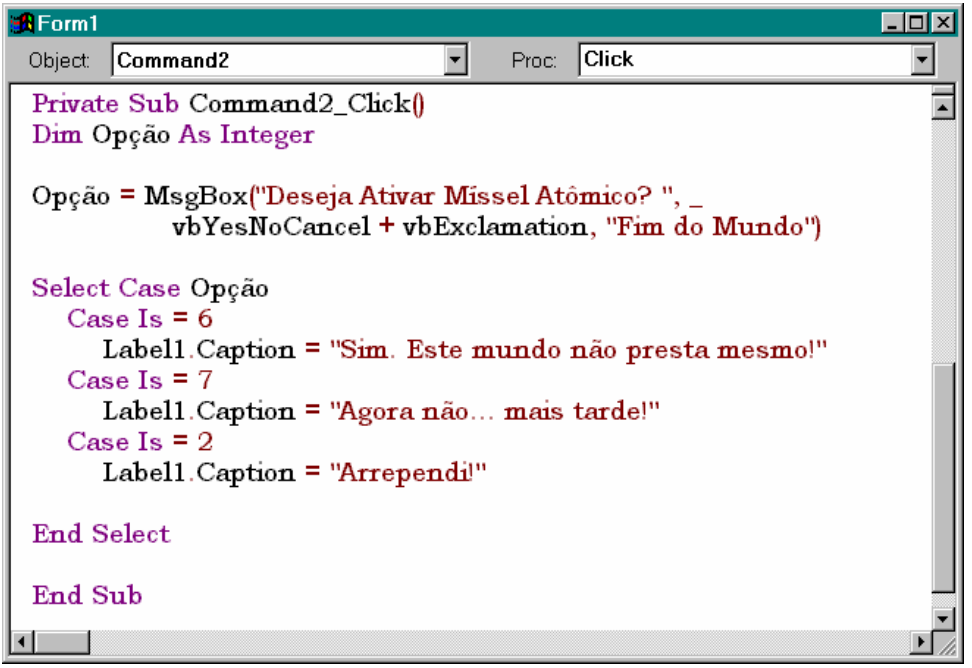
PRESTE ATENÇÃO

- Crie um formulário conforme modelo abaixo:



- Vamos fazer uma codificação para o botão 1 e o botão 2 como esta abaixo. Perceba que usamos o MSGBOX não só para enviar uma mensagem para o usuário, como para executar uma determinada ação dependendo do que o usuário escolher.





10.2 A FUNÇÃO FORMAT

Formata uma expressão qualquer de acordo com a máscara estabelecida na função.

ExpressãoFormatada = Format(<expressão>[, <Mascara de formatação>)

Formatação de Expressões Numéricas

Para formatar qualquer expressão numérica usamos "#" para representar um digito de 0 a 9, "," para representar os milhares e "." para representar as casas decimais. Como o padrão brasileiro para milhares e casas decimais é exatamente o contrário, o Visual Basic automaticamente irá colocar no nosso padrão, pois irá verificar qual o formato usado para o pais de acordo com o que foi estabelecido no painel de controle.

Print Format(12345.3,"##,###.##")

Valor Impresso: 12.345,3

Usamos o simbolo "#" para representar a disposição dos números. Não há necessidade de colocar a mesma quantidade de números e "#". Se tivessesmos colocado *Print Format(12345.3,"#,###,###.##"),* ainda assim seria impresso somente "12.345,3". Entretanto, se colocarmos:

Print Format(12345.3, "###.##")

Valor Impresso: 12345,3

Perceba que não podemos colocar uma formatação menor que os números que serão impressos, pois senão a formatação não irá alcançar toda a extensão dos números.

O símbolo "#" é substituído por números quando existir número para ser substituído. Note que o número decimal é ".3" e apesar de termos usado uma formatação para casas decimais com dois símbolos "##", não apareceu as duas casas decimais. Se quisermos forçar o aparecimento de zeros quando não tiver número para ser impresso, usados "0" no lugar de "#". Veja:

Print Format(12345.3,"##,###.00")

Valor Impresso: 12.345,30

Print Format(12345,"##,###.00")

Valor Impresso: 12.345,00

Isto vale também para formatação de números sem casas decimais:

Print Format(45,"0000")

Valor Impresso: 0045

Se quisermos uma formatação diferente para números negativos, basta colocar essa formatação após o ponto-e-virgula.

Print Format(12345,"##,###.00; (-)##,###.00")

Valor Impresso: 12.345,00

Print Format(-12345,"##,###.00; (-)##,###.00")

Valor Impresso: (-)12.345,00

Veja abaixo os caracteres que podem ser usados na formatação de valores numéricos:

0 Exibe um dígito ou um zero. Se a expressão tiver um dígito na posição em que o **0** aparece na seqüência de caracteres de formato, ele será exibido; caso contrário, é exibido um zero nessa posição.

Se o número possui um número de dígitos inferior ao de zeros (em qualquer lado da casa decimal) na expressão de formato, exibe zeros à esquerda ou à direita. Se o número tiver mais dígitos à direita do separador decimal do que zeros à direita do separador decimal na expressão de formato, arredonda o número para tantas casas decimais quantos forem os zeros existentes. Se o número tiver mais dígitos à esquerda do separador decimal do que zeros à esquerda do separador decimal na expressão de formato, exibe os dígitos a mais sem modificações.

Exibe um dígito ou nada. Se a expressão tiver um dígito na posição em que o símbolo # aparece na seqüência de caracteres de formato, ele será exibido; caso contrário, nada será exibido nessa posição.

Este símbolo funciona como o espaço reservado para o dígito **0**, mas os zeros à esquerda e à direita não são exibidos se o número tiver a mesma

quantidade ou menos dígitos do que existem # caracteres em qualquer um dos lados do separador decimal na expressão de formato.

. Espaço reservado para decimal

Para algumas localidades, é usada uma vírgula como separador decimal. O espaço reservado para decimal determina quantos dígitos são exibidos à esquerda e à direita do separador decimal. Se a expressão de formato contiver apenas sinais de números à esquerda deste símbolo, os números inferiores a 1 começam com um separador decimal. Se você quiser que um zero à esquerda seja sempre exibido com números fracionários, use 0 como o primeiro espaço reservado para dígito à esquerda do separador decimal. O caractere real utilizado como espaço reservado para decimal na saída formatada depende do Formato Numérico reconhecido pelo sistema.

% Espaço reservado para porcentagem

A expressão é multiplicada por 100. O caractere de porcentagem (%) é inserido na posição onde ele aparece na seqüência de caracteres de formato.

, Separador de milhar

Para algumas localidades, é utilizado um ponto como o separador de milhar. O separador de milhar separa milhar de centena dentro de um número que tenha quatro ou mais casas à esquerda do separador decimal. O uso padrão do separador de milhar é especificado no formato que contém um separador de milhar delimitado por espaços reservados de dígito (0 ou #). Dois separadores de milhar adjacentes ou um separador de milhar imediatamente à esquerda do separador decimal (sendo ou não especificado um decimal) significa "aplique uma escala ao número dividindo-o por 1000 e arredonde-o conforme necessário." Use essa técnica para aplicar escalas a números extensos. Por exemplo, a seqüência de caracteres de formato "##0,," pode ser usada para representar 100 milhões como 100. Números inferiores a 1 milhão são exibidos como 0. Dois separadores de milhar adjacentes em qualquer posição que não seja a imediatamente à esquerda do separador decimal serão considerados apenas como especificação do uso de um separador de milhar. O caractere real utilizado como o separador de milhar na saída formatada depende do Formato Numérico reconhecido pelo sistema.

E- E+ e- e+ Formato científico

Se a expressão de formato contiver pelo menos um espaço reservado para dígito (0 ou #) à direita de E-, E+, e- ou e+, o número é exibido em formato científico, sendo E ou e inserido entre o número e seu expoente. O número de espaços reservados para dígito à direita determina o número de dígitos do expoente. Use E- ou e- para incluir um sinal de subtração (-) ao lado

de expoentes negativos. Use E+ ou e+ para incluir um sinal de subtração ao lado de expoentes negativos e um sinal de adição (+) ao lado de expoentes positivos.

- + $ ( ) space Exibe um caractere literal

Para exibir uma caractere diferente dos listados, preceda-o com uma barra invertida (\) ou coloque-o entre aspas (" ").

**** Exibe o caractere seguinte da seqüência de caracteres de formato

Muitos caracteres da expressão de formato têm um significado especial e não podem ser exibidos como caracteres literais a menos que sejam precedidos por uma barra invertida. A barra propriamente não é exibida. Sua utilização equivale a colocar o caractere seguinte entre aspas. Para exibir uma barra invertida, use duas barras invertidas (\\).

Exemplos de caracteres que não podem ser exibidos como caracteres literais são caracteres de formatação de data e hora (a, c, d, h, m, n, p, q, s, t, w, y e /:), caracteres de formatação numérica (#, 0, %, E, e, vírgula e ponto) e os caracteres de formatação de seqüências de caracteres (@, &, <, >, e !).

"ABC" Exibe a seqüência de caracteres que está entre aspas.

Para incluir uma seqüência de caracteres em Format a partir do código, você deve usar Chr(34) para delimitar o texto (34 é código de caractere para aspas).

☞ Usamos também como argumento na formatação de expressões numéricas algumas palavras-chave que correspondem a algum tipo de formato padrão.

General Number : Exibe o número na forma em que se encontra, sem separadores de milhar.

Print Format(123456.7, "General Number")

Valor Impresso: 123456,7

Currency : Exibe o número com o separador de milhar, se apropriado; exibe dois dígitos à direita do separador de casa decimal. Note que a saída é baseada nas configurações do Painel de Controle.

Print Format(123456.7, "Currency")

Valor Impresso: R$123.456,70

Fixed : Exibe pelo menos um dígito à esquerda e dois dígitos à direita do separador de casa decimal.

Print Format(123456.7, "Fixed")

Valor Impresso: 123456,70

Print Format(1, "Fixed")

Valor Impresso: 1,00

Standard : Exibe o número com o separador de milhar, pelo menos um dígito à esquerda e dois dígitos à direita do separador de casa decimal.

Print Format(123456.7, "Standard")

Valor Impresso: 123.456,70

Percent : Exibe o número multiplicado por 100 com um sinal de porcentagem (%) anexado à direita; sempre mostra dois dígitos à direita do separador de casa decimal.

Print Format(123456.7, "Percent")

Valor Impresso: 12345670,00%

Print Format(1, "Percent")

Valor Impresso: 100,00%

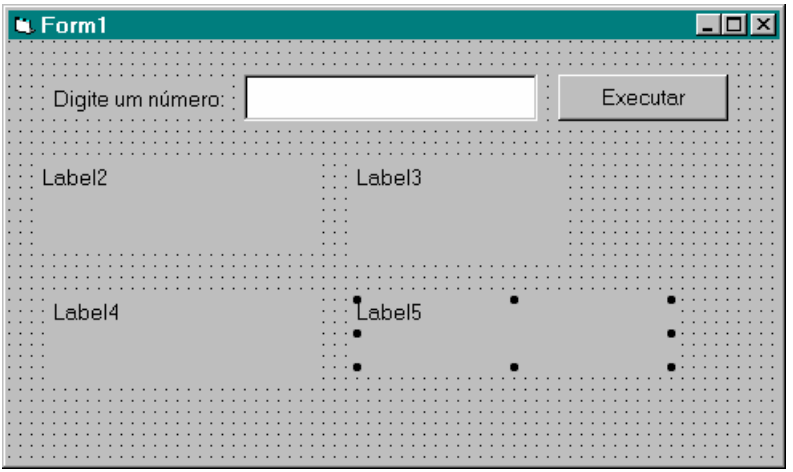
Scientific : Usa a notação científica padrão.

Print Format(123456.7, "Scientific")

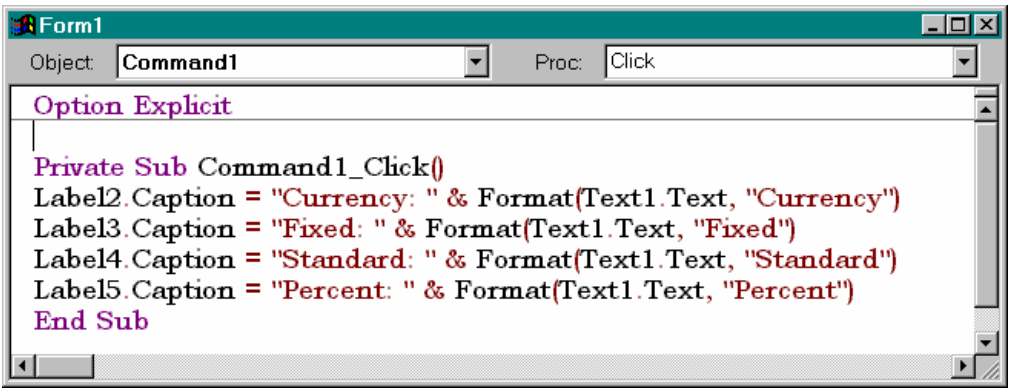
Valor Impresso: 1,23E+05

PRESTE ATENÇÃO

- Crie um formulário conforme o exemplo abaixo:



- Faça a seguinte codificação para o botão executar:



Formatação de Expressões Lógicas

Yes/No : Exibe “No” se o número for 0; caso contrário, exibe “Yes”.

True/False : Exibe “False” se o número for 0; caso contrário, exibe “True”.

On/Off : Exibe “Off” se o número for 0; caso contrário, exibe “On”.

Print Format(1,”True/False”)

Valor Impresso: True

Formatação de Expressões Data e Hora

Usamos a função Format também para formatar uma data ou hora, configurando assim o formato que será impresso. Veja os caracteres que podemos usar:

: Separador de hora. Em algumas localidades podem ser usados outros caracteres para representar o separador de hora. O separador de hora separa horas, minutos e segundos quando os valores de hora são formatados. O caractere real usado como o separador de hora na saída formatada é determinado pelas configurações de seu sistema.

/ Separador de data. Em algumas localidades podem ser usados outros caracteres para representar o separador de data. O separador de data separa o dia, mês e ano quando os valores de data são formatados. O caractere real usado como o separador de data na saída formatada é determinado pelas configurações de seu sistema.

c Exibe a data como **ddddd** e a hora como **ttttt**, nessa ordem. Exibe apenas informações de data se não houver parte fracionária para o número de série de data; exibe apenas informações de hora se não houver parte inteira.

Print Format("01/08/96","c")
Valor Impresso: 01/08/96
Print Format(now,"c")
Valor Impresso: 01/08/96 22:51:11

d Exibe o dia como um número sem zeros à esquerda.
Print Format("05/07/96","d")
Valor Impresso: 5

dd Exibe o dia como um número com zeros à esquerda.
Print Format("05/07/96","dd")
Valor Impresso: 05

ddd Exibe o dia da semana como uma abreviado em 3 letras.
Print Format("01/08/96","ddd")
Valor Impresso: Qui

dddd Exibe o dia da semana como um nome completo.
Print Format("01/08/96","dddd")
Valor Impresso: Quinta-feira

w Exibe o dia da semana como um número (1 para domingo até 7 para sábado).

ww Exibe a semana do ano como um número.
Print Format("01/08/96","ww")
Valor Impresso: 31

m Exibe o mês como um número sem zeros à esquerda. Se **m** vier imediatamente depois de **h** ou **hh**, é exibido o minuto em lugar do mês.
Print Format("01/08/96","m")
Valor Impresso: 8

mm Exibe o mês como um número com zeros à esquerda. Se **m** vier imediatamente depois de **h** ou **hh**, é exibido o minuto em lugar do mês.
Print Format("01/08/96","mm")
Valor Impresso: 08

mmm Exibe o mês como uma abreviado em três letras.
Print Format("01/08/96","mmm")
Valor Impresso: Ago

mmmm Exibe o mês como um nome completo.
Print Format("01/08/96","mmmm")
Valor Impresso: Agosto

q Exibe o trimestre do ano como um número.
Print Format("01/08/96","q")
Valor Impresso: 3

y Exibe o dia do ano como um número.
Print Format("01/08/96","y")
Valor Impresso: 214

yy Exibe o ano como um número de dois dígitos.
Print Format("01/08/96","yy")
Valor Impresso: 96

yyyy Exibe o ano como um número de quatro dígitos.
Print Format("01/08/96","yy")
Valor Impresso: 1996

h Exibe a hora como um número sem zeros à esquerda.

Print Format("09:13:55","h")
Valor Impresso: 9

hh Exibe a hora como um número com zeros à esquerda.
Print Format("09:13:55","h")
Valor Impresso: 09

n Exibe o minuto como um número sem zeros à esquerda.
nn Exibe o minuto como um número com zeros à esquerda.
s Exibe o segundo como um número sem zeros à esquerda.
ss Exibe o segundo como um número com zeros à esquerda.
t t t t t Exibe uma hora como uma hora completa (inclusive hora, minuto e segundo), formatada usando o separador de hora definido pelo formato de hora reconhecido pelo sistema.
Print Format(now,"ttttt")
Valor Impresso: 23:17:27

☞ Usando estes caracteres especiais podemos formatar uma data de várias maneiras, como por exemplo:
Print Format(“01/08/96”,”dd/mmmm/yyyy”)
Valor Impresso: 01/Agosto/1996
Print Format(“01/08/96”,”dd/mmm/yy”)
Valor Impresso: 01/Ago/96

☞ Veja abaixo a relação das palavras-chaves aceita pela função Format para expressões de data e hora:

General Date : Exibe a data e a hora nos formatos estabelecidos nas configurações do Windows. Caso a expressão seja somente uma data, será exibido
Print Format(now,"general date")
Valor Impresso: 01/08/96 23:21:25
Print Format("01/08/96","general date")
Valor Impresso: 01/08/96
Print Format("09:24:11","general date")
Valor Impresso: 09:24:11

Long Date : Exibe uma data de acordo com o formato por extenso de data de seu sistema.
Print Format("01/08/96","Long Date")

Valor Impresso: Quinta-feira, 1 de Agosto de 1996

Medium Date : Exibe uma data usando o formato médio de data apropriado para a versão de idioma do aplicativo host.

Print Format("01/08/96","Medium Date")

Valor Impresso: 01-Ago-96

Short Date : Exibe uma data usando o formato abreviado de data de seu sistema.

Print Format("01/08/96","Short Date")

Valor Impresso: 01/08/96

Long Time : Exibe uma hora usando o formato por extenso de hora de seu sistema: inclui horas, minutos, segundos.

Print Format("09:24:11","Long Time")

Valor Impresso: 09:24:11

Medium Time : Exibe uma hora no formato 12 horas usando horas e minutos e a designação AM/PM.

Print Format("09:24:11","Medium Time")

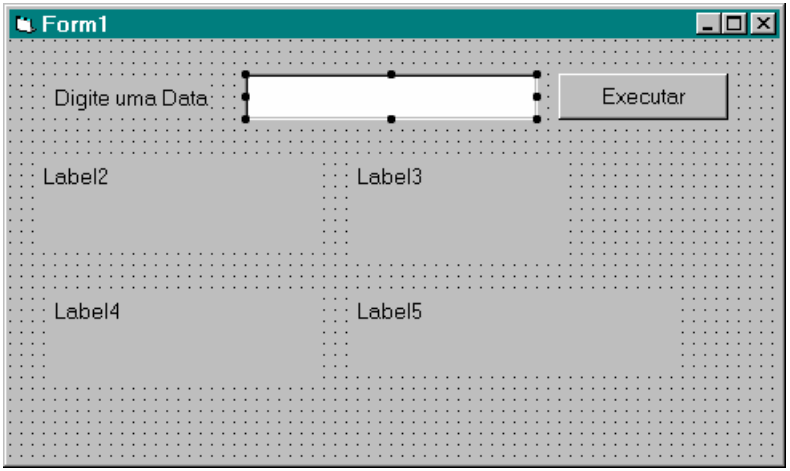
Valor Impresso: 09:24 AM

Short Time : Exibe uma hora usando o formato 24 horas.

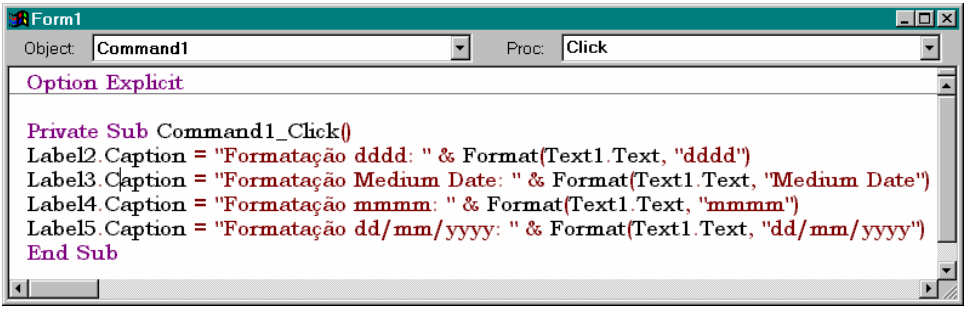
Print Format("09:24:11","Short Time")

Valor Impresso: 09:24

- Crie um formulário em branco da seguinte forma:



- Altere a propriedade AutoSize dos labels para True e digite a seguinte codificação:



Formatação de Expressões String

A manipulação de expressões String podem ser formatados usando os caracteres especiais abaixo:

@ Exibe um caractere ou um espaço. Se a seqüência de caracteres tem um caractere na posição em que @ aparece na seqüência de formato, ele será exibido; caso contrário, um espaço será apresentado nessa posição. Os espaços reservados são preenchidos da direita para a esquerda a menos que exista um caractere **!** na seqüência de caracteres de formato. Veja abaixo.

& Exibe um caractere ou nada. Se a seqüência de caracteres tem um caractere na posição em que **&** aparece, ele será exibido; caso contrário, nada será exibido. Os espaços reservados são preenchidos da direita para a esquerda a menos que exista um caractere **!** na seqüência de caracteres de formato. Veja abaixo.

! Força preenchimento da esquerda para a direita dos espaços reservados. O preenchimento padrão é feito da direita para a esquerda.

< Exibe todos os caracteres no formato de minúsculas.

> Exibe todos os caracteres no formato de maiúsculas.

EXERCÍCIOS PROPOSTOS

1 - Mostre a diferença entre as funções matemáticas FIX e INT:

2 - Para converter uma variável para o tipo Data usamos qual função?

3 - Para conversão de uma variável numérica para o formato String (texto) usamos qual função?

4 - Caso necessitamos de converter uma expressão string “INFORMATICA” para
“informatica” usamos qual função? Exemplifique:

5 - Como usariamos a função DATEADD para incrementar 2 meses na data “27/08/96”?

6 - Para saber o intervalo entre a data 27/08/96 e 31/12/96 em:
Quantidade de dias:
Quantidade de meses:
Quantidade de semanas:

7 - Para extrair o dia de uma data usamos qual função?

8 - Mostre a finalidade das funções:
NOW:
DATE:
TIME:

9 - Mostre a sintaxe completa das funções que:
a) Extrai da frase “Editora Terra” a expressão “Terra”:

b) Extrai da frase “Editora Terra” a expressão “Editora”:

c) Extrai da frase “Gráfica e Editora Terra” a expressão “Editora”:

10 - Mostre como ficaria a codificação de um programa que criasse uma matriz chamada MESES cujo conteúdo fosse os meses de Janeiro a Dezembro:

11 - Considerando que:
A = “Editora”
B = “Gráfica”
C = “Editora”
D = IIF(A=B, “Igual”, “Diferente”)
E = IIF(A=C, “Igual”, “Diferente”)
F = IIF(C=B, “Igual”, “Diferente”)
QuaL o conteúdo das variáveis D, E e F?

12 - Descreva a finalidade das Funções:

Isdate:

Isempty:

Isnumeric:

13 - Desenvolva um programa que tenha no formulário uma caixa de texto para o usuário digitar um numero de 0 a 5 e crie um evento Change cuja codificação é a seguinte:

Print Choose(Text1.text, “Zero”,”Um”, “Dois”,”Três”,”Quatro”,”Cinco”)

Descreva o que o programa fará:

14 - Em que situação usamos as funções:
INPUTBOX:

MSGBOX:

15 - Cite um exemplo da utilização da função Format para as expressões:
Numéricas:

Data e Hora:

String:

11 CRIAÇÃO DO BANCO DE DADOS

- Caracteristicas
- Visual Data Manager
- Índices

11.1 BANCO DE DADOS

Sempre que trabalhamos com qualquer tipo de dado devemos gravá-los em arquivos para não perdê-los quando desligamos o computador. Chamamos esses arquivos que recebem vários dados de Banco de Dados. Neste livro vamos abordar o padrão MDB (Microsoft Database) que representa a grande maioria de arquivos de Banco de Dados para Windows existente hoje. Esse padrão é usado pelo Access e o Visual Basic também o aceita.

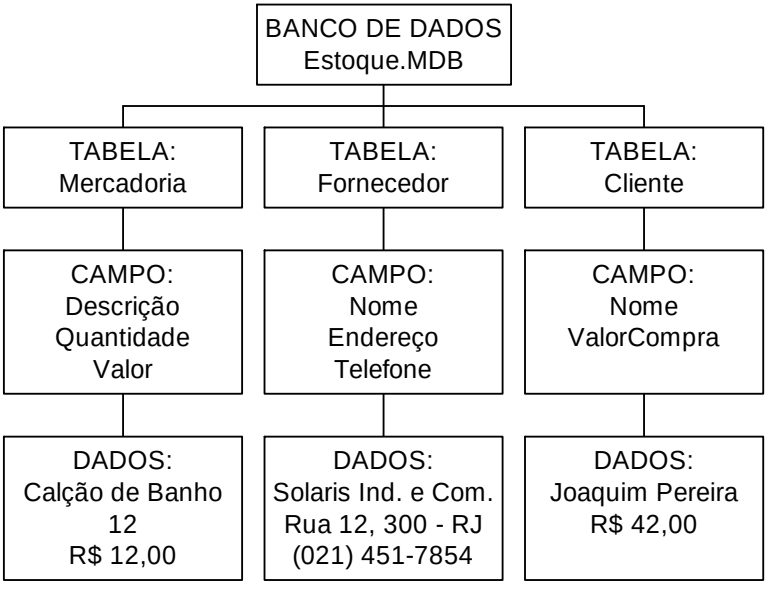
11.1.1 Características

Um arquivo que contém dados irá possuir a extensão MDB. Exemplo: Se temos um banco de dados de um controle de estoque possivelmente ele terá o nome de “Estoque.MDB”.

Dentro de um arquivo deste podemos ter várias tabelas. No exemplo de um controle de estoque, podemos ter uma tabela para Mercadoria, Fornecedor, Cliente, etc. Todas essas tabelas ficam armazenadas dentro do arquivo MDB.

Cada tabela possui Campos onde serão armazenado os dados. Por exemplo, na tabela Mercadoria temos o campo de Descrição da Mercadoria, Quantidade no Estoque, Valor de Venda, etc. Tudo isto representa os Campos que estão armazenados dentro da tabela.

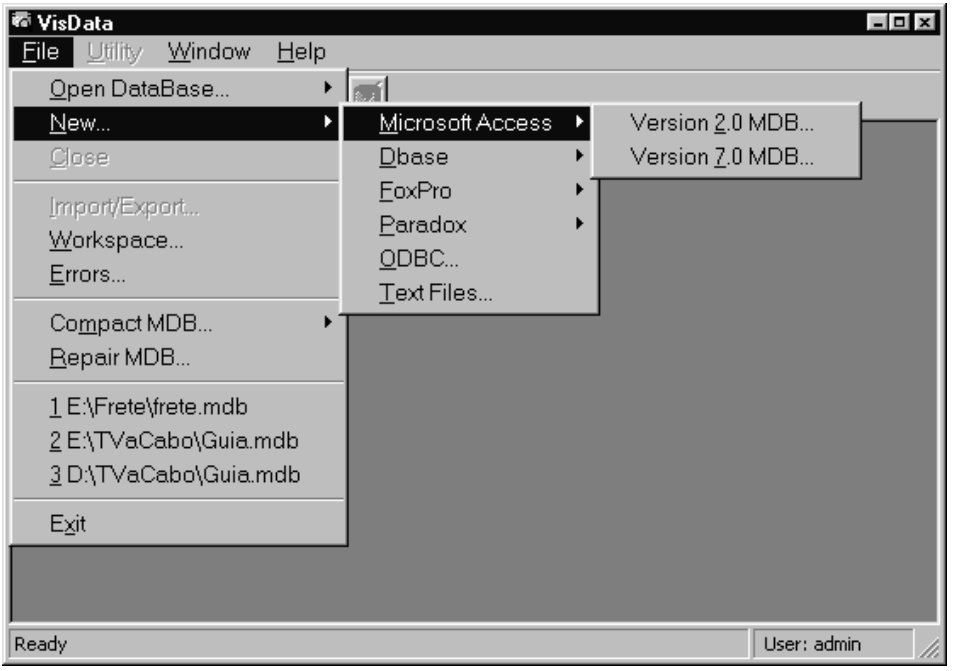
Dentro dos Campos que estão os dados que precisamos armazenar. Veja a disposição da hierarquia:



Quando estamos na fase de desenvolvimento de um programa que irá gerenciar um banco de dados, podemos deixar já criado o Banco de Dados com suas tabelas e campos. Somente os Dados é que serão inseridos pelo usuário na medida que for trabalhando com seu programa.

Para criar um banco de dados o Visual Basic traz uma ferramenta chamada Visual Data Manager. Ele se encontra no menu ADD-INS.

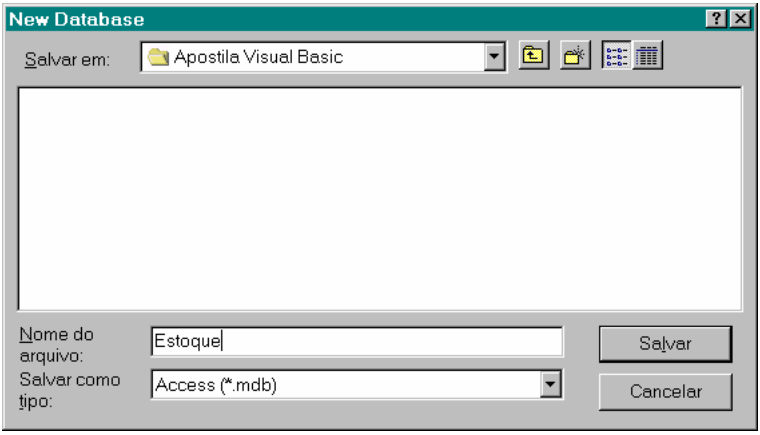
11.1.2 Visual Data Manager (VisData)



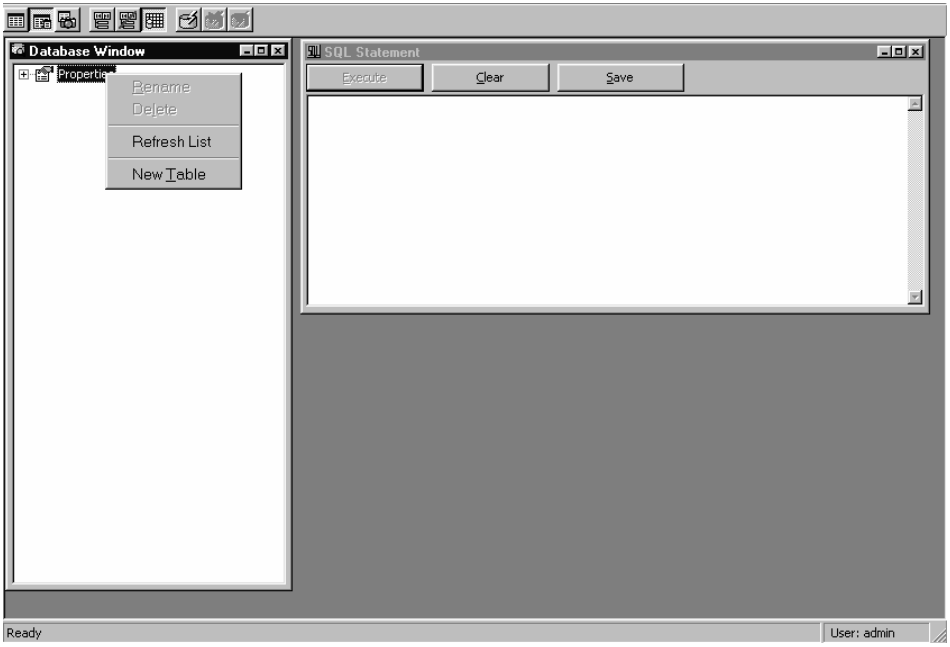
No Menu **File** usaremos duas opções de grande importância para nosso estudo: New Database para criação de novos Bancos de Dados e Open Database para abrir um banco de dados já criado.

Vamos criar nosso programa de Controle de Estoque rudimentar que apresentamos no organograma acima. Click na opção New. Pode-se criar Bancos de Dados de vários tipos, como Access, Dbase, FoxPro, etc. Mas o Visual Basic tem como padrão o Microsoft Access que cria arquivos com extensão MDB. Escolha então o Microsft Access/Version 7.0 MDB

Irá aparecer uma tela pedindo para digitarmos o nome do Banco de Dados que estamos criando: Digite “Estoque”. Não é necessário colocar a extensão pois o tipo de arquivo, por padrão, já é MDB.



A próxima janela que irá aparecer é a que gerência as tabelas. Para criarmos uma nova tabela é só apertar o botão da direita do mouse em "Propriedades" e acionar a opção “NEW TABLE”.



Irá aparecer outra janela onde nomeamos a tabela e criamos os campos que esta tabela terá:

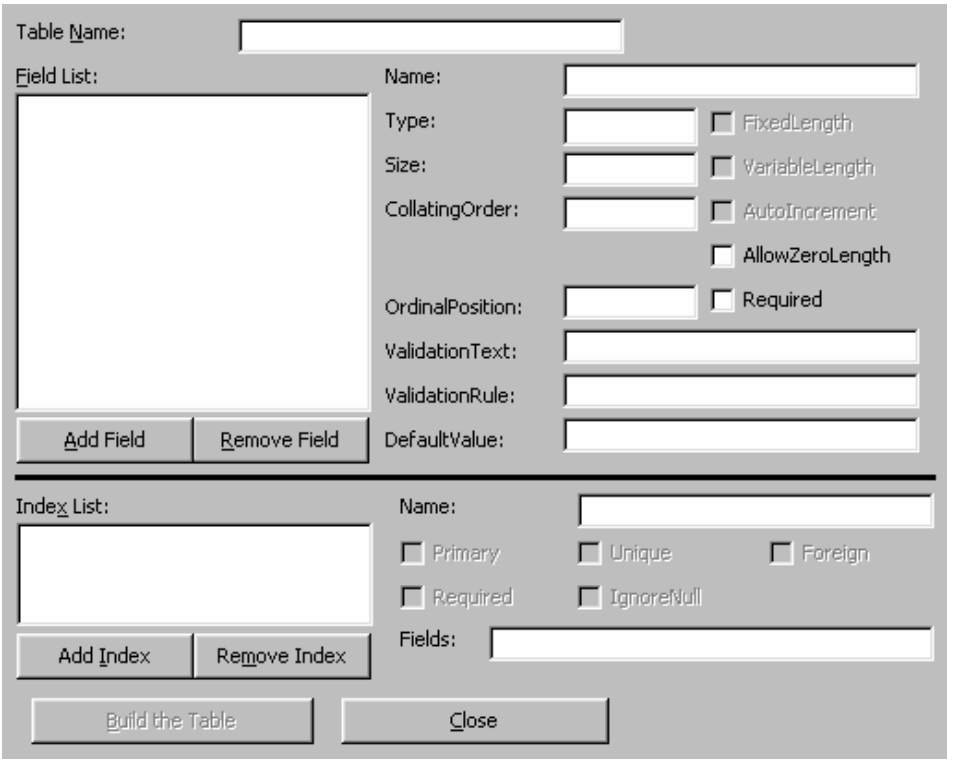
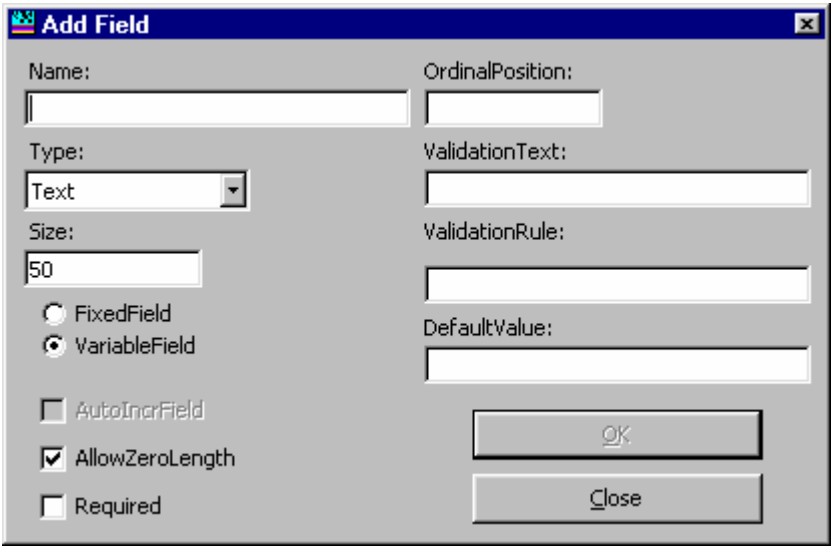


Table Name: Nome da Tabela, obedecendo as mesmas regras impostas para nomeação de variáveis, ou seja, não começar no números, não conter espaços, não usar símbolos como hifem (-), etc.

Digite: Mercadoria

Field List: é um Listbox onde aparece a lista dos campos que foram inseridos na tabela. Para inserir os campos aperte o botão "**Add Field**". Vai aparecer a seguinte janela:



Usamos essa janela para criar todos os campos da tabela.

Name: Nome do Campo.

Digite: Descrição

Type: Uma caixa de combinação onde escolhemos o tipo de dado que terá este campo. Usamos as mesmas regras para tipo de variável (veja o capítulo sobre variáveis).

Text: Textos e números, como nomes e endereços, números de telefone e códigos postais. Um campo de texto pode conter de 0 a 255 caracteres. Você deve definir a propriedade Size com a extensão máxima de seu texto, ao declarar um campo como sendo do tipo Text.

Memo: Textos e números extensos, como comentários ou explicações. O tamanho de um campo Memo é limitado pelo tamanho máximo do banco de dados.

Date/Time : Datas e horas.

Currency : Utilizado quando um campo contém valores monetários.Os números à direita da parte decimal são arredondados durante os cálculos. O tipo de dados Currency mantém um número fixo de dígitos à direita da parte decimal.

Boolean : Yes/No, True/False, On/Off ou campos que contêm apenas um valor, entre dois.

Binary : Objetos criados em outros programas usando o protocolo OLE. Pode ser Qualquer valor que possa ser expresso em binários com até 1.2 gigabytes de tamanho. Este tipo é utilizado para armazenar figuras, arquivos ou outros dados binários não processados.

Byte: Valores positivos de 0 a 255. Armazenado em variáveis tipo Byte.

Integer : Valores de –32.768 a +32.767. Armazenado em variáveis tipo Integer.

Long : –2.147.483.648 a 2.147.483.647 (cerca de 2.1 GB). Armazenado em variáveis tipo Inteiro Long.

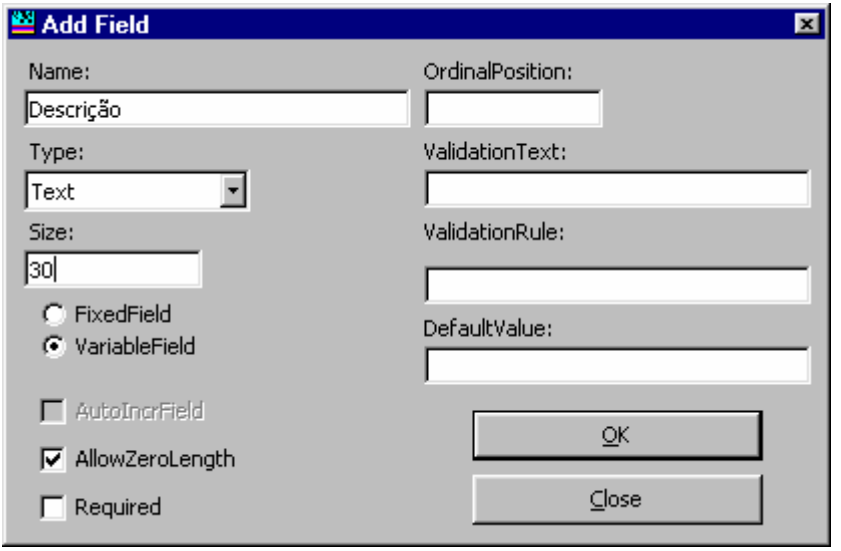
Single : –3,402823E38 a 1,401298E–45 para números negativos e 1,401298E–45 a –3,402823E38. Armazenado em variáveis tipo Single.

Double : 1,79769313486232E308 a 4,94065645841247E–324 para valores negativos e 4,94065645841247E–324 a 1,79769313486232E308 para valores positivos. Armazenado em variáveis tipo Double.

Escolha para nosso exemplo o tipo Text

Size : Tamanho definido para o campo. Se colocamos 2 para o campo, poderá ser digitado somente 2 caracteres. Esta opção é liberada somente para o tipo Text.

Digite 30



FixedField:Campo Fixo

VariableField: Campo Variável, que é o padrão.

AllowZeroLength: Permitir comprimento zero, ou seja, quando esta opção esta selecionada o campo aceita (""). Se esta opção não tiver habilitada e o usurio tentar gravar um campo que não tenha nada digitado (como um daado não-obrigatório no programa) ocorrerá um erro em tempo de execução.

Required: Quando esta opção esta marcada o campo se torna obrigatório, ou seja, é necessário informar valores para o campo, senão ocorre erro em tempo de execução.

DefaultValue: Quando temos um campo que não pode ficar vazio, sempre é interessante digitar algo aqui nesta opção, pois os valores que

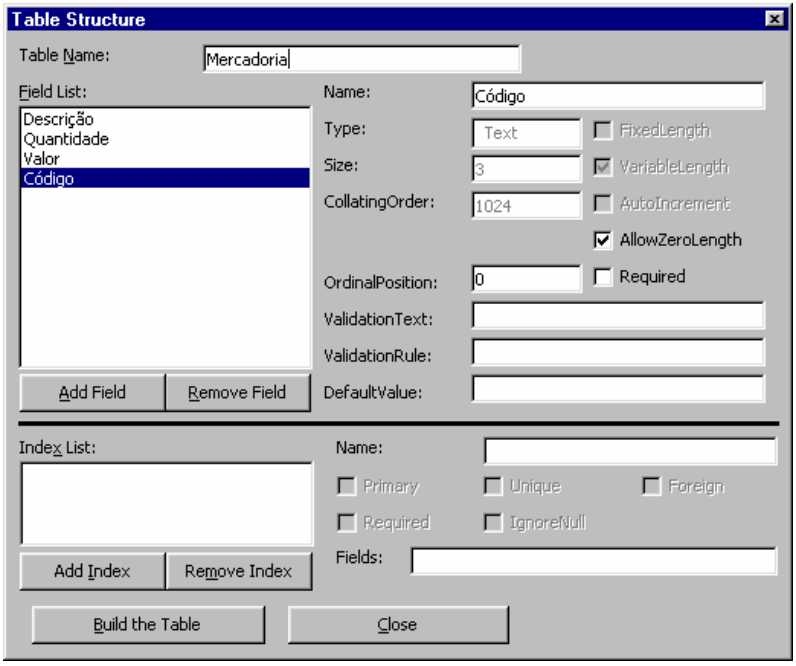
informarmos como *DefaultValue* será armazenado para o campo quando nada for digitado pelo usuário. Por Exemplo, existe um campo valor obrigatório, mas o usuário não digitou nada neste local, então no momento da gravação o programa grava automaticamente a expressão especificada no *DefaultValue*.

Aperto o botão "OK" para gravar o campo e continue inserindo outros campos como na tabela abaixo:

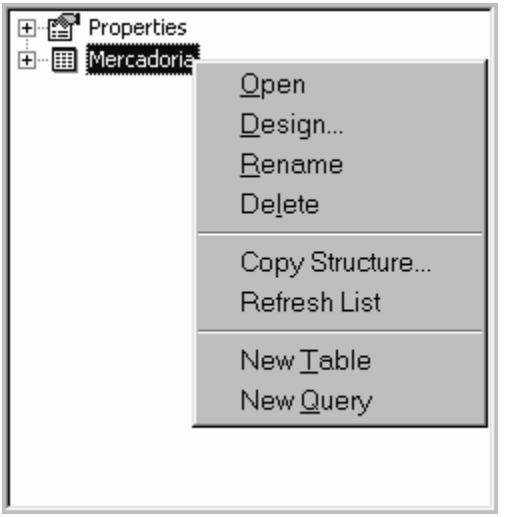
Field Name	Data Type	Size
Quantidade	Integer	N/A
Valor	Currency	N/A
Código	Text	3

Geralmente se usa colocar o código no inicio. Se quiser movê-lo para cima é só colocar "0" no ítem *OrdinalPosition*.

Veja como ficará depois de pronto:



Aperte o botão *Build The Table* para que a tabela seja gravada e inserida no banco de dados.



O nome da tabela "Mercadoria" agora já aparece dentro da janela "Database Windows", se clicarmos com o botão da direita em cima desta tabela ira aparecer um menu onde podemos criar uma outra tabela (*New Table*), digitar dados na tabela (*Open*) ou até modificar a estrutura da tabela (*Design*).

Crie agora as tabelas abaixo:

Tabela: Cliente

Field Name	Data Type	Size
CódCliente	Text	3
Nome	Text	30
LimiteCrédito	Currency	N/A

Tabela: Vendas

Field Name	Data Type	Size
Código	Text	3
CódCliente	Text	3
ValorVenda	Currency	N/A
QuantidadeVendida	Integer	N/A

Criamos uma tabela Vendas onde será lançado o código do cliente que efetuou a compra, o codigo da mercadoria que este cliente comprou, o valor total de sua compra e a quantidade de mercadoria que ele levou. Este exemplo é simples mas vai servir para nosso aprendizado.

11.1.3 Criando indices

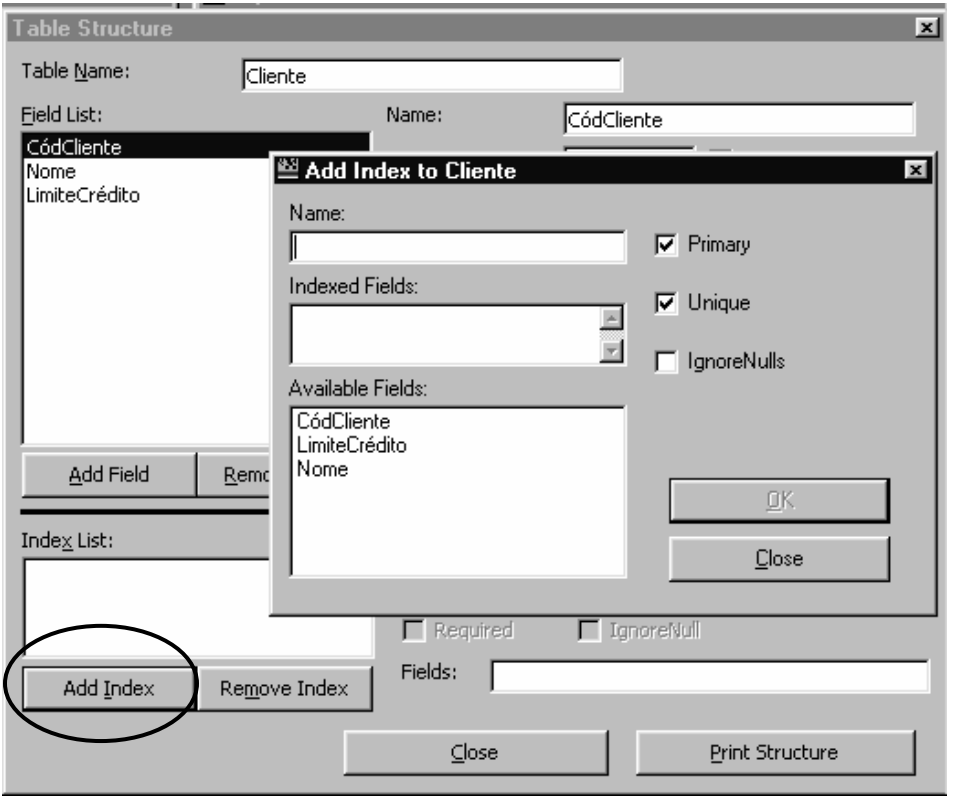
Usamos índices nas tabelas para melhorar a performance do acesso durante uma determinada procura. Quando criamos um índice o Visual Basic faz uma ordenação lógica da tabela através de uma chave composta por um ou mais campos.

Se precisamos ordenar uma tabela por código para que a procura de um determinado dado digitado seja feita pelo código de maneira mais rápida, então indexamos a tabela pelo campo que possui o código.

Se precisamos colocar uma relação de clientes em ordem alfabética pelo nome digitado, então indexamos a tabela pelo campo que possui o nome do cliente.

No nosso exemplo, durante a digitação de novos clientes e novas mercadorias, devemos ficar atentos para não cadastrar dois clientes com o mesmo código ou duas mercadorias com o mesmo código. Para verificar isto, sempre que o usuário digitar uma nova mercadoria o programa deve fazer uma procura nas mercadorias já cadastradas para saber se aquele novo código que o usuário digitou já existe ou não. Então, se temos que fazer uma procura rápida por código será necessário indexar a tabela Mercadoria pelo campo Código. O Visual Basic então ordenará logicamente a tabela pelo código de forma ascendente, e sempre que uma nova mercadoria for cadastrada ela será automaticamente inserida de forma ascendente dentro da tabela.

Para se criar um índice podemos usar também o Visual Data Manager através do Design, onde será apresentado novamente a estrutura que se criou da tabela. Use o botão "Add Index" para adicionar indices.



Name: O nome do indice. Através deste nome que ativaremos o indice em nosso programa.

Available Fields: Neste ListBox aparece a relação de campos que nossa tabela contém. Dê um click no campo que deseja indexar. Ao fazer isto o nome do campo ira aparecer no **Indexed Fields** sinalizando que aquele determinado campo irá ser indexado.

Crie então um indice com o nome de "IndCódigo" e selecione o campo CódCliente. Marque a caixa de verificação Primary Index, habilite também a propriedade Unique e click o botão OK.

Unique informa ao Visual Basic que a chave do indíce é exclusiva (única). Um índice chave ajuda a otimizar a procura por registros. Ele consiste de um ou mais campos que organizam unicamente todos os registros de uma tabela em uma ordem predefinida. Se o índice consiste de um campo, os valores desse campo precisam ser únicos. Se o índice consiste de mais de uma campo, valores duplicados podem ocorrer em cada campo, mas cada

combinação de valores de todos os campos indexados precisa ser única. Os campos que compõe a chave de índice não são necessariamente únicos.

Primary Index indica que um objeto Index representa um índice primário para uma tabela. Um índice primário consiste em um ou mais campos que identificam com exclusividade todos os registros de uma tabela em uma ordem predefinida. Como o índice do campo tem que ser exclusivo, a propriedade Unique do objeto Index é definida como True. Se o índice primário consistir de mais de um campo, cada campo pode conter valores duplicados mas as combinações de valores de todos os campos indexados devem ser únicas. Um índice primário consiste em uma chave para a tabela e geralmente contém os mesmos campos da chave primária.

Faça a mesma coisa para a tabela de mercadoria. O nome do indice também pode ser IndCódigo.

Para a tabela de Vendas nomeie o indice como indClienteMercadoria e defina para chave de índice os campos CódCliente e Código nesta ordem:

Colocamos estes dois campos como chave do índice pois para localizarmos os dados referente a venda, como quantidade e valor vendido, temos que saber (procurar por) qual cliente comprou e qual mercadoria ele comprou.

Agora que temos o arquivo "Estoque.Mdb" (que é o nosso banco de dado) já criado. Vamos fechar definitivamente o Visual Data Manager e voltar para o projeto do Visual Basic. A utilidade do Visual Data Manager é somente na criação do banco de dados, as tabelas, os índices, etc.; mas para manipular os dados, como incluir, alterar, consulta, exclusão e relatórios devemos desenvolver um programa para isto dentro do Visual Basic.

12 MANIPULAÇÃO DO BANCO DE DADOS

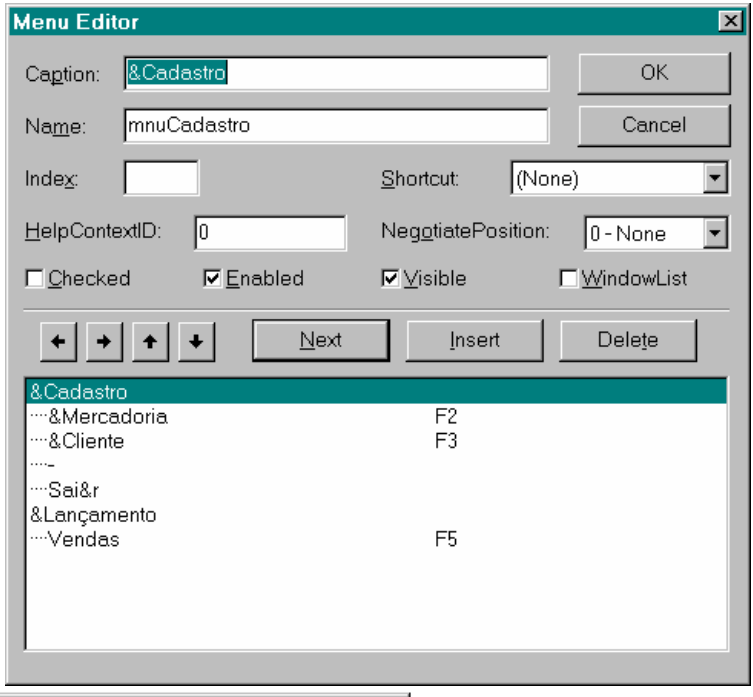
- Abertura do Banco de Dados e suas tabelas
- Adicionar
- Navegação
- Alteração
- Consulta
- Exclusão

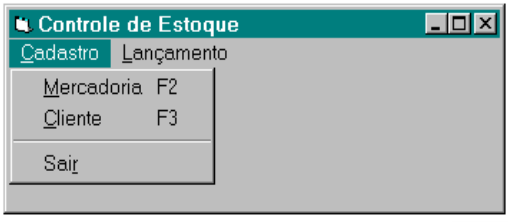
12.1 CRIANDO JANELAS

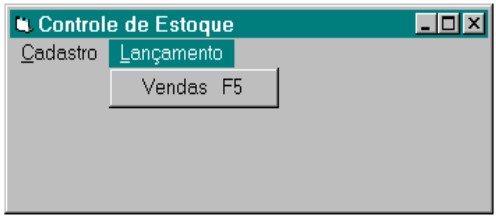
Precisamos criar inicialmente um formulário para cadastrar mercadoria e depois outro formulário para cadastrar o cliente. Vamos usar o que já aprendemos anteriormente:

Crie um novo projeto e para o formulário nomeio-o “frmEstoque”. O Caption será “Controle de Estoque”.

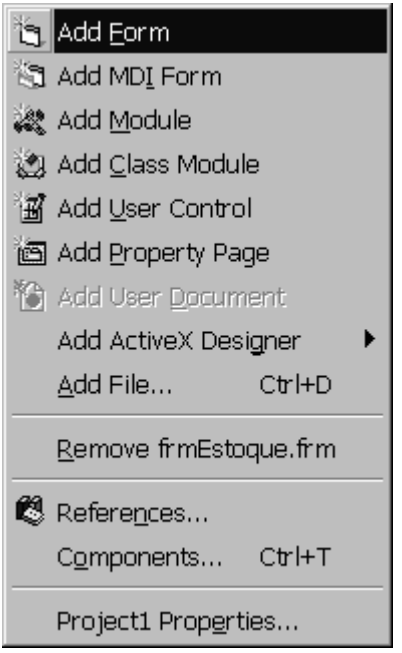
Neste formulário vamos criar os menus que irão acessar nossas bases de dados. Veja como ficará:

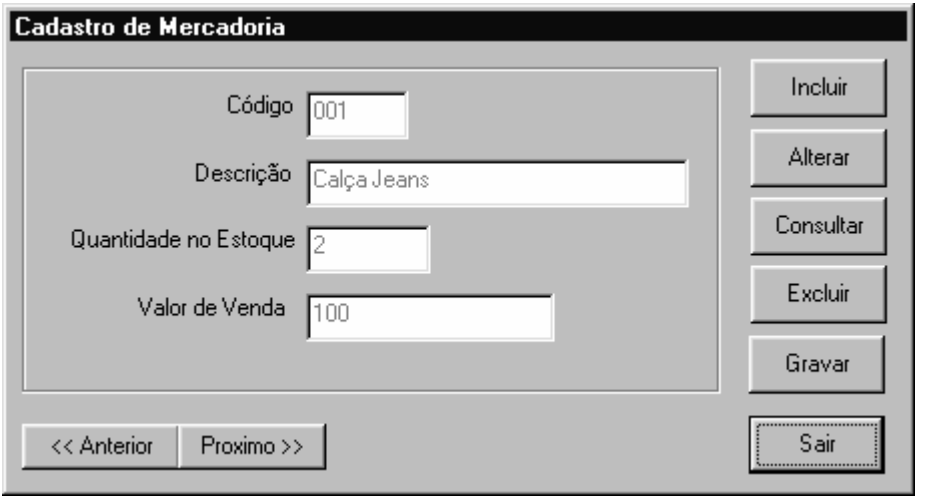






No menu "*Project*" do Visual Basic click na opção "*AddForm*" para inserirmos um novo formulário em nosso projeto. Será o formulário para cadastramento de mercadoria.





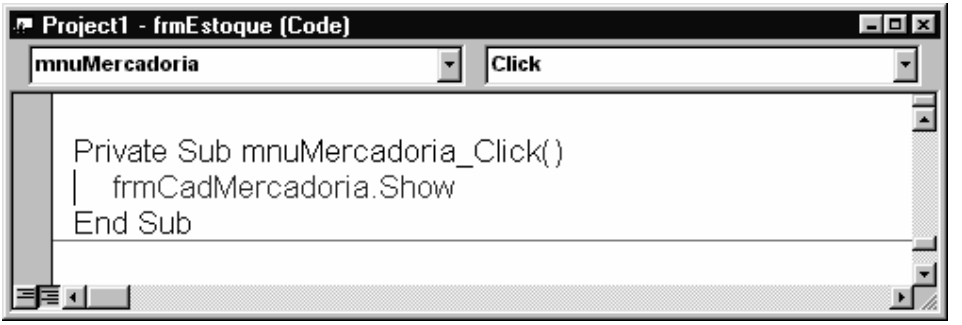
Neste novo formulário altere o **Caption** para “Cadastro de Mercadoria” e o **Name** para “frmCadMercadoria”. Vamos montar esta interface, depois abriremos outros **Form** para Cliente e Vendas.

Coloque um frame, 4 labels e 4 caixa de texto com os **Names**: txtCódigo, txtDescrição, txtQuantidade e txtValor.

Coloque também 7 botões de comando no formulário como na. Para nomea-los use os Captions de cada um prefixado com “cmd”:

Importante: Coloque a propriedade **TabIndex** do Botão "cmdIncluir " como "0" para que ele seja o primeiro a possuir o foco quando entrar na janela.

Para rodar este programa e ver o resultado será necessário ligar este formulário ao menu mnuMercadoria. Para fazer isto vamos até o menu localizado no frmEstoque, click no Cadastro para abrir as opções. Dê dois clickes na opção Mercadoria que uma janela de codificação será aberta. Digite:

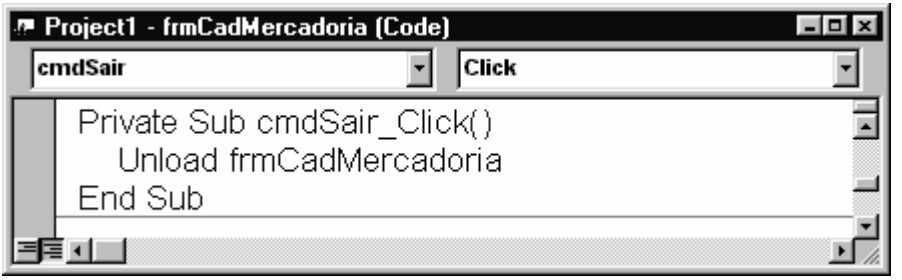


O Método **Show** abre um novo formulário a partir de outro, e usamos ele para abrir o formulário frmCadMercadoria.

Vamos aproveitar e codificar a opção Sair do menu também. Dê dois clicks nele e digite:

Este simples comando “END” encerra toda a aplicação fechando todas as janelas que podem estar abertas.

Para completar este raciocínio, vamos até o formulário frmCadMercadoria e para o botão cmdSair vamos fazer a seguinte codificação:

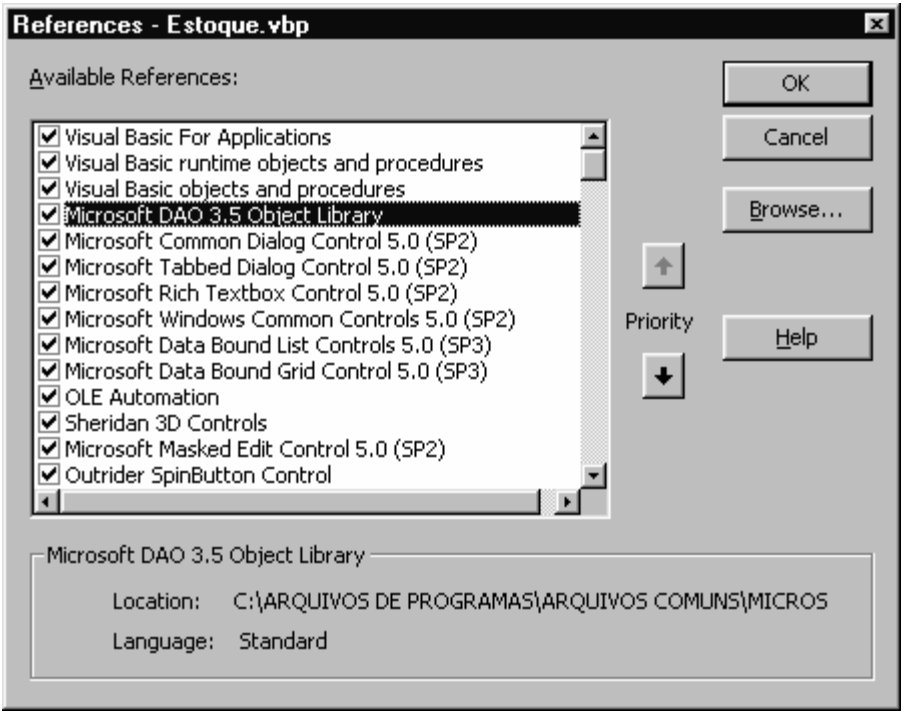


O Comando **Unload** retira a janela informada da memória e da tela, voltando a execução do programa para a janela anterior, que no nosso caso é o menu principal.

Agora estamos prontos para começar a codificar o nosso Banco de Dados.

12.1.1 Abrindo um banco de dados

Uma vez que vamos trabalhar com manipulação de dados, temos que verificar se o Visual Basic está apto para fazer esta manipulação. Os serviços necessários para executar essas funções estão em uma DLL do Microsoft DAO 3.5 Object Library. Para que nosso programa leia essa DLL temos que ir até o menu **Project/References** e selecionar esta DLL:

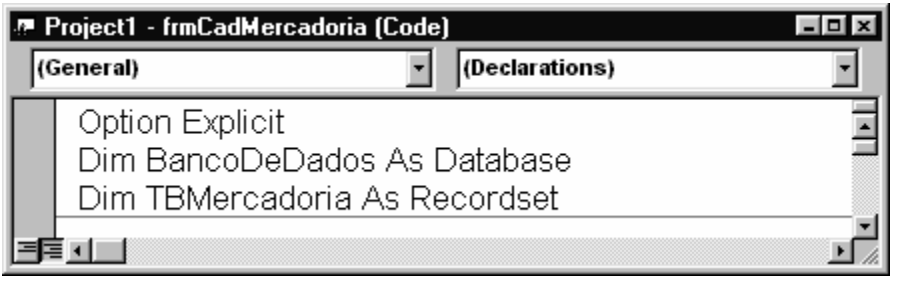


Se este ajuste não for feito e um objeto de banco de dados for declarado, o Visual Basic gerará um erro. A biblioteca DAO (Data Access Objects) fornece um conjunto de objetos de programação que precisaremos usar para gerência os Bancos de Dados.

Temos que criar agora duas variáveis do tipo Banco de Dados e outra do tipo Tabela. Uma servirá para fazer referência ao nome e caminho do Banco de Dados como um todo, e a outra irá fazer referência a uma tabela especifica.

Onde criar estas variáveis?

No nosso exemplo temos que criá-las na seção **General** da janela de codificação do formulário. Variáveis criadas ali possuem abrangência em todo o formulário. Se criarmos ela dentro de alguma outra rotina (ou vento) a abrangência destas variáveis serão somente dentro da própria rotina.



A variável BancoDeDados é do tipo Database pois ela sempre terá em seu conteúdo o nome do banco de dados que será aberto. Depois criamos uma variável para a tabela de Mercadoria de nome TBMercadoria.

O Objeto Recordset é a representação lógica de uma tabela. Usaremos este objeto para manipular todos os dados da tabela. Veja as principais propriedades de um Recordset (somente manipulados na codificação do programa):

Sintáxe: *NomeDoRecordSet.Propriedade*

AbsolutePosition : Indica o numero do registro corrente da tabela em uso.

BOF : Retorna True quando a tabela chegou ao inicio e False quando esta em outro ponto.

DateCreated : Retorna a data de criação da tabela manipulada pelo Recordset.

EOF : Retorna True quando a tabela chegou ao fim e False quando esta em outro ponto.

Index : Especificamos o nome do índice que será associado a tabela.

NoMatch : Retorna True se uma pesquisa efetuada dentro da tabela foi bem-sucedida.

PercentPosition : Retorna um percentual referente a posição que a tabela se encontra com comparação com o total de registros da tabela.

RecordCount : Retorna a quantidade de registro que uma tabela possui.

Os principais métodos usamos pelo Recordset:

AddNew : Prepara a tabela para receber um novo registro.

Close : Fecha a tabela, e conseqüentemente o objeto recordset.

Delete : Remove a registro atual da tabela.

Edit : Prepara a tabela para que o registro corrente seja alterado.

FindFirst <Condição> : Procura o primeiro registro dentro da tabela que satisfaça a condição estabelecida (Somente para objeto Recordset tipo dynaset ou snapshot).

FindLast <Condição> : Procura o ultimo registro dentro da tabela que satisfaça a condição estabelecida (Somente para objeto Recordset tipo dynaset ou snapshot).

FindNext <Condição > : Procura o próximo registro dentro da tabela que satisfaça a condição estabelecida (Somente para objeto Recordset tipo dynaset ou snapshot).

FindPrevious <Condição> : Procura o registro anterior dentro da tabela que satisfaça a condição estabelecida (Somente para objeto Recordset tipo dynaset ou snapshot).

MoveFirst : Move a tabela para o primeiro registro.

MoveLast : Move a tabela para o ultimo registro.

MoveNext : Move a tabela para o seguinte seguinte.

MovePrevious : Move a tabela para o registro anterior.

Seek <Operador de Comparação>,<Item a Procurar> : Localiza um registro dentro da tabela. A tabela tem que estar indexada. (Somente para objeto Recordset tipo table)

Update : Grava as alterações e inclusões na tabela.

O próximo passo será abrir o banco de dados e abrir a tabela de mercadoria. Para fazer isto selecione o formulário frmCadMercadoria e abra a janela de codificação dele criando um evento chamado FORM_LOAD. Este evento é chamado sempre que este formulário é chamado pelo Windows.

Associamos a variável BancoDeDados ao método **OpenDatabase**, que vai realmente abrir o arquivo na prática. Uma vez aberto ao arquivo, sempre que fizermos referência a ele usaremos a variável BancoDeDados.

Na abertura do arquivo usados o Objeto **App.Path** que retorna o nome do diretório corrente, onde esta localizado o arquivo pretendido. Depois concatenamos este diretório com o nome do arquivo a ser aberto.

Para abrir a tabela usamos o método **OpenRecordSet**, e como argumento informamos o nome da tabela a ser aberta e a constante ***dbOpenTable*** que informa que o RecordSet a ser aberto é do tipo tabela.

Note que é necessário antes do método **OpenRecordSet** usar o nome do objeto Database BancoDeDados, pois, dentro da hierarquia, a tabela esta dentro do Banco de Dados.

Fizemos tudo isto para deixar disponível para uso o banco de dados Estoque e sua tabela Mercadoria.

12.1.2 Abrindo um indice

Próximo passo é abrir o índice que esta tabela irá trabalhar. Este índice é necessário para ordenar os dados que forem digitados num determinada ordem (no nosso caso a ordem é por código) e fazer uma procura rápida pela chave de indexação.

O índice da tabela Mercadoria é o *IndCódigo* e possui como chave de indexação o campo *Código.*

Usamos a propriedade **Index** para TBMercadoria e determinamos que o índice corrente será IndCódigo.

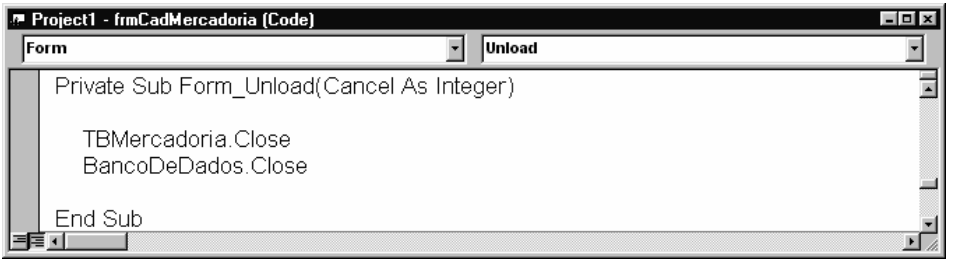
12.1.3 Fechando um banco de dados

É extremamente importante Fechar todas tabelas e banco de dados toda vez que sairmos de um programa ou formulário. Arquivos abertos é perigoso e pode trazer resultados indesejáveis.

Para fecharmos com segurança um banco de dados usamos o Evento do objeto Formulário chamado Unload. Este evento é chamado sempre que o formulário é retirado da tela (e da memória). Note que este evento é o contrário do evento Load, que é lido quando o formulário é colocado na tela.

Porque não podemos fechar os arquivos de Banco de Dados no evento click do botão Sair?

Por que o usuário pode querer inventar de sair deste formulário usando outros recursos do Windows que não seja o botão sair, como a opção Fechar do Control Box, Alt-F4, etc. Para não correr risco colocamos o fechamento no evento Unload.



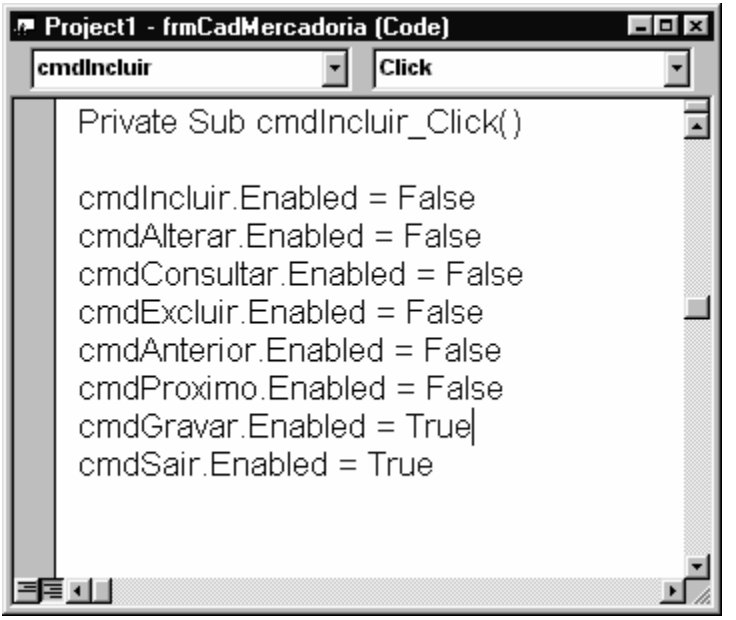
Fechamos primeiro a tabela Mercadoria representada pela variável TBMercadoria e depois o banco de dados Estoque representada pela variável BancoDeDados. Ambas usaram o método *Close*.

12.1.4 Cuidados especiais

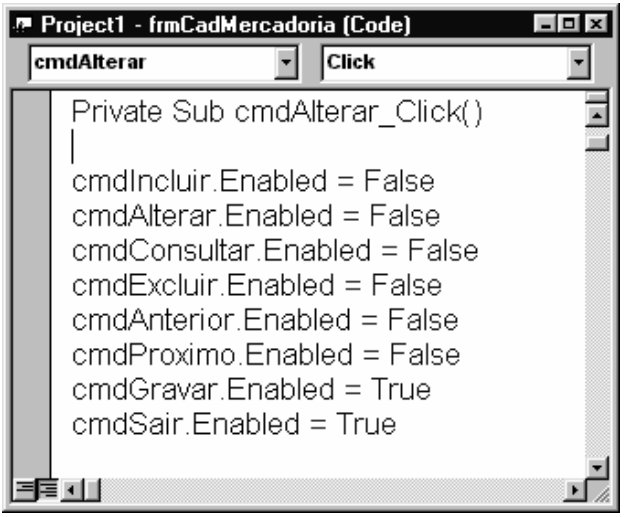
Algo que é extremamente importante no desenvolvimento de qualquer programa é tentar prever o máximo possivel de erros que o usuário pode cometer e tentar previni-los.

Imagine a situação: o usuário esta no meio de uma inclusão e resolver apertar a tecla excluir ou alterar, sendo que nem terminou a inclusão. Ou então esta numa alteração e resolve apertar o botão alterar novamente. Para podermos tentar cercar esses erros usamos a propriedade **Enabled** nos botões.

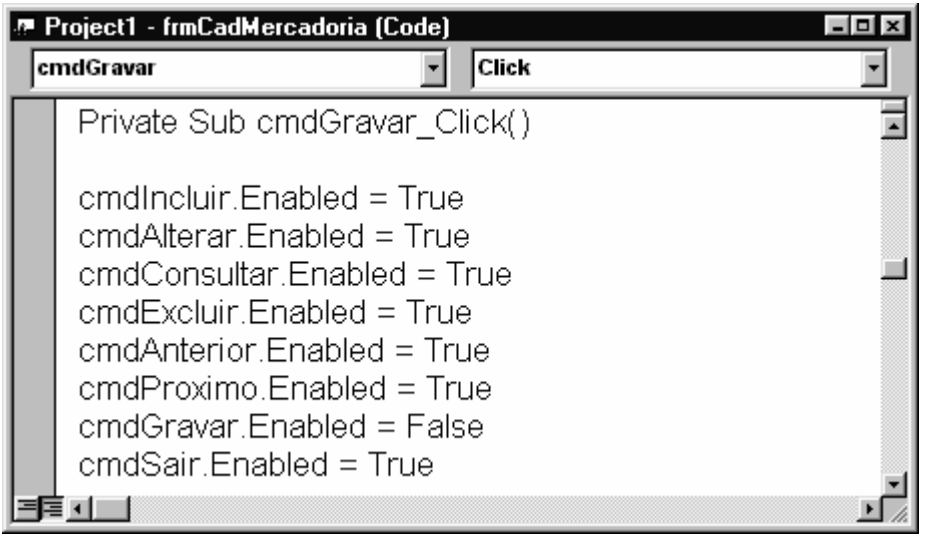
Quando o usuário estiver numa inclusão somente os botões Gravar e Cancelar deve estar habilitado. Ou seja, ou o usuário confirma o que esta fazendo ou cancela. Se esta numa alteração a mesma coisa ocorre. E assim por diante. Para fazermos isto temos que usar o **Enabled** em cada botão como veremos abaixo:



Fazendo isto quando o usuário apertar o botão incluir, os botões que ele não pode pressionar enquanto não concluir a inclusão ficarão desabilitados. Deve-se fazer isto para todos os botões.

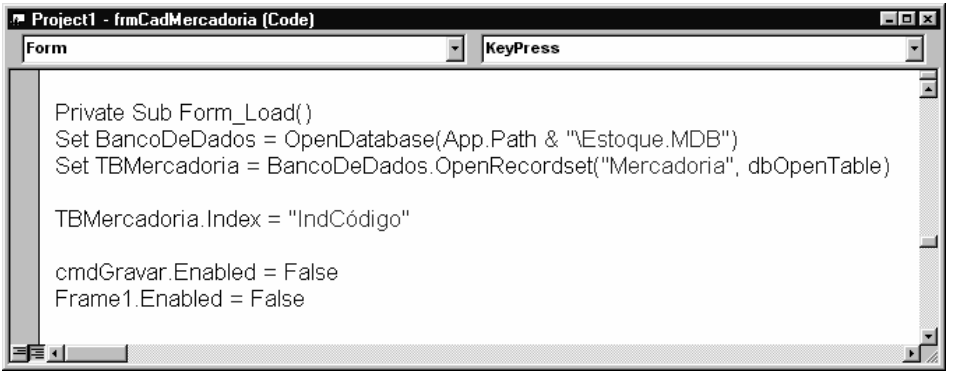


Na alteração o método é semelhando a inclusão. Durante a alteração o usuário só terá liberado para ele os botões "Gravar" e "Sair".

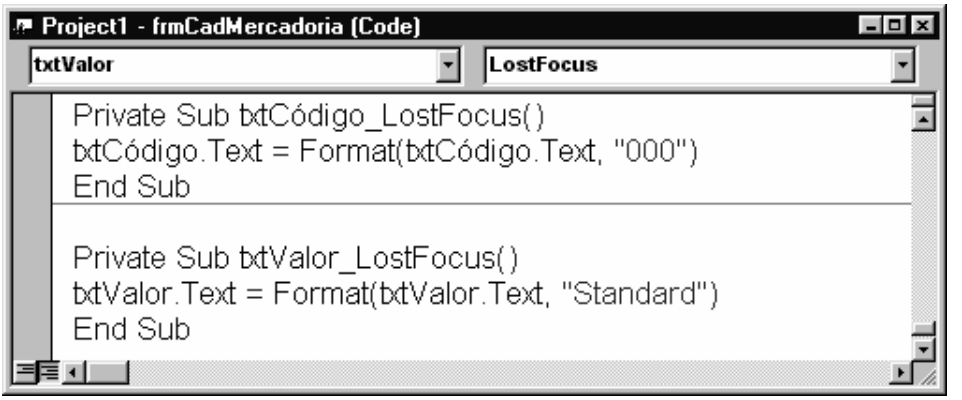


No evento click do botão gravar habilite todos novamente. Com exceção do próprio botão gravar que não pode ser habilitado ate que se incluia ou altere algo.

Outro detalhe que é bom lembrar é desabilitar o botão Gravar no evento **Form_Load** para que esteja devidamente desabilitado quando entrar na janela de cadastro de mercadoria. Desabilitamos também o **Frame**, pois assim todos os objetos contido dentro dele serão também desabilitados. Fazemos isto para que o usuário não fique "passeando" pelas caixas de texto sem definir antes (através dos botões) o que ele quer fazer.

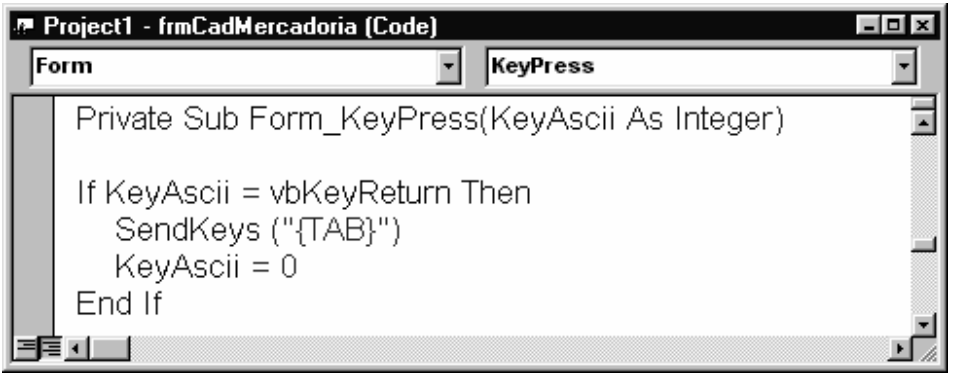


Uma formatação para as caixas de texto que receber numeros é bem vindo para dar um aspecto visual mais interessante.



Criamos uma formatação para as caixas de texto no evento **LostFocus** para quando o usuário terminar de digitar o valor se ajustar.

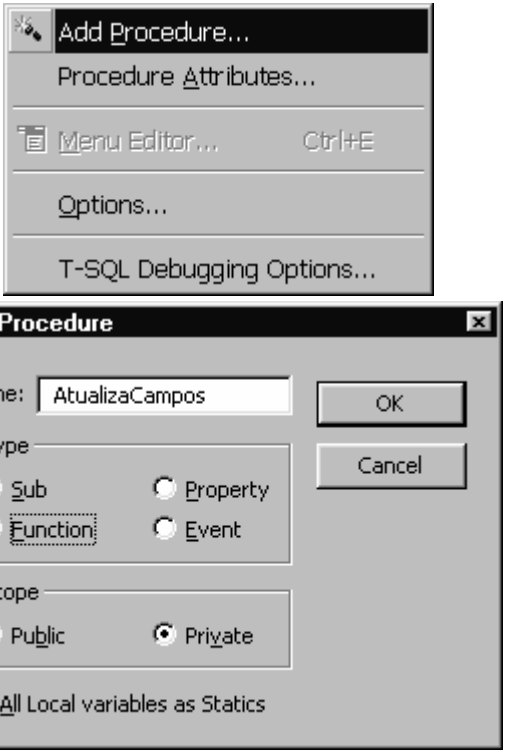
Habilitamos a tecla Enter criando um evento **KeyPress** no formulario. Não esquecer de habilitar a propriedade **KeyPreview**.



12.1.5 Funções de apoio

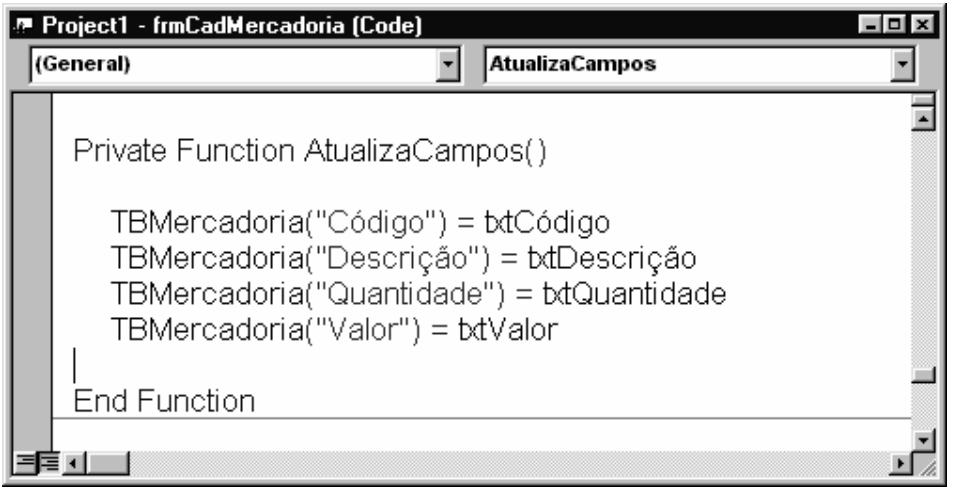
Para que nosso programa de cadastro tenha um código mais limpo e de fácil entendimento vamos criar algumas funções que serão úteis em nossa programação.

Para criar essas funções chame a janela de codificação, selecione TOOLS no menu principal do Visual Basic, e escolha a opção Add Procedure...



Nomeia a função que estamos criando de AtualizaCampos, coloque do tipo Function e o Scope (abrangência) Private. Com isto, estamos criando uma função que poderá ser usada somente no formulário corrente.

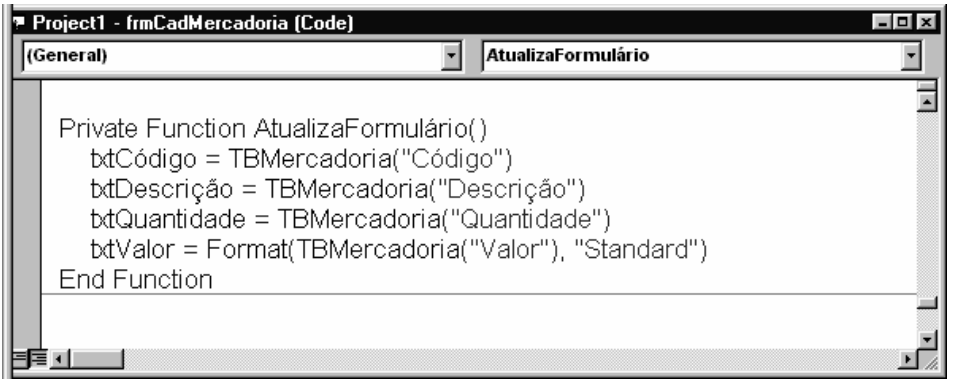
Ao clicar em Ok será aberta uma janela de codificação para podermos digitar o que a função ira conter:



A finalidade desta função é inserir o conteúdo existente nas caixas de textos do formulário para dentro dos campos da tabela de mercadoria.

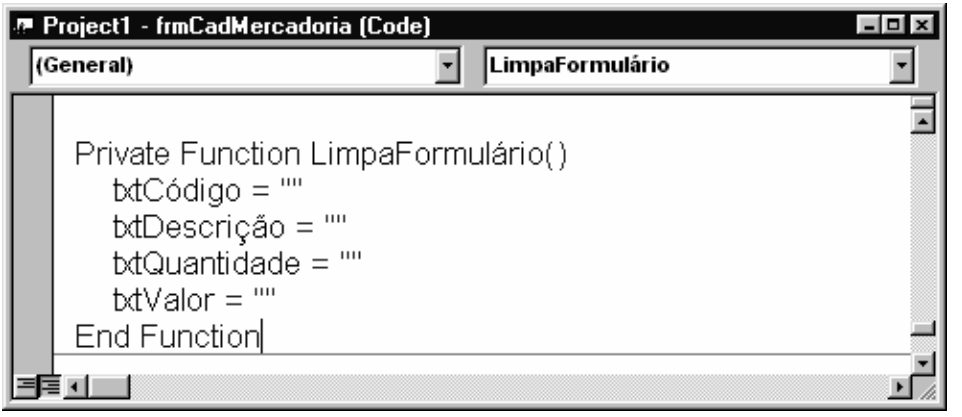
Com isto, sempre que precisarmos gravar os dados que o usuário digitou no formulário para dentro de seus respectivos campos na tabela usamos a função AtualizaCampos.

Repita o processo para criar agora a função AtualizaFormulário:



Note que o AtualizaFormulário faz o caminho inverso: Descarrega os dados existente na tabela de seus respectivos campos para dentro das caixas de texto existente no formulário.

Por fim, vamos criar uma função para limpar as caixas de texto. Dê o nome de LimpaFormulário:



Vamos usar esta função sempre que precisarmos que as caixas de texto fiquem vazias.

12.2 Adicionando Dados

Antes de começarmos a fazer a codificação para o botão Incluir, a fim de preparar nosso formulário para manipular a tabela de Mercadoria, vamos fazer uma modificação nas propriedades MaxLength das caixas de texto: todas devem possuir o mesmo valor da propriedade **Size** usada para os campos na tabela. Usamos isto para não deixar que um usuário coloque um nome com 40 caracteres num campo que foi definido para receber 30 caracteres.

Outra modificação será efetuada no evento Form_Load do formulário frmCadMercadoria:

Quando uma tabela é aberta o Visual Basic se posiciona no primeiro registro existente dentro dela. Entretanto, quando ainda não existe nada digitado dentro da tabela, conseqüentemente não existirá nenhum registro para se posicionar. Neste caso o fim de arquivo é encontrado logo na abertura da tabela.

A propriedade EOF é quem informa se o fim de arquivo esta ativo ou não. Se EOF for True (verdadeiro) significa que a tabela chegou ao fim de arquivo, se for False, significa que a tabela esta posicionada em algum registro em algum lugar.

Recapitulando: A propriedade EOF retorna True se a posição do registro atual for posterior ao último registro da tabela, ou seja, se o fim do arquivo for localizado, e False se a posição do registro atual for no último registro ou anterior a ele.

O Visual Basic possui também a propriedade BOF para informar se o inicio do arquivo foi encontrado.

A propriedade BOF retorna True se a posição do registro atual for anterior ao primeiro registro e False se a posição do registro atual estiver no primeiro registro ou for posterior a ele.

Os valores de retorno das propriedades BOF e EOF são determinados pela localização do indicador do registro atual.

Podemos usar as propriedades BOF e EOF para determinar se um objeto Recordset contém registros ou se você foi além dos limites de um objeto Recordset ao percorrer seus registros.

Perceba que somente se EOF for False que atualizamos o formulário, pois caso contrário, não poderíamos atualizar o formulário pois não haveria dados a serem atualizados.

O primeiro botão que vamos codificar é o incluir. Ele funcionará da seguinte forma: Quando clicarmos nele as caixas de textos serão limpas e o método **AddNew** será acionado. Este método é usado para avisar à tabela que alguns dados serão incluídos.

Depois de usar o **AddNew** temos que atualizar os campos, ou seja, passar tudo que foi digitado nas caixas de texto para seus respectivos campos dentro da tabela TBMercadoria. Fazendo isto usamos o método **Updated** para efetuar a gravação propriamente dita e atualizar a tabela com os novos campos.

Veja um exemplo:

```
TBMercadoria.AddNew
TBMercadoria("Código") = “001”
TBMercadoria("Descrição") = “Calça Jeans”
TBMercadoria("Quantidade") = “12”
TBMercadoria("Valor") = “100”
TBMercadoria.Updated
```

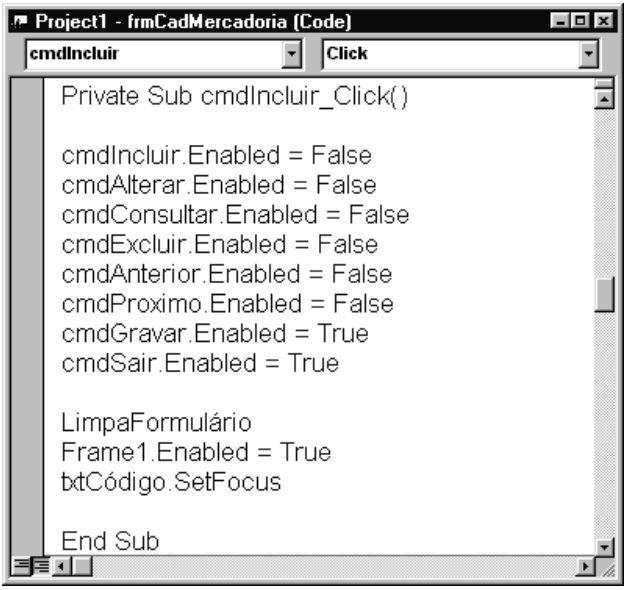
Esta rotina é o padrão para inclusão de dados.Repare que sempre inicia com **AddNew** e termina com **Updated** Neste exemplo colocamos expressões string para serem inseridas nos campos, mas poderiamos ter colocado variáveis.

No nosso exemplo vamos colocar as caixas de texto. Ficaria assim:

```
TBMercadoria.AddNew
TBMercadoria("Código") = txtCódigo
TBMercadoria("Descrição") = txtDescrição
TBMercadoria("Quantidade") = txtQuantidade
TBMercadoria("Valor") = txtValor
TBMercadoria.Updated
```

Desta forma o que o usuário digitar nas caixas de texto serão incluídos dentro dos campos da tabela.

Isto é uma forma simplificada de fazer inclusão de dados. Mas nós vamos usar um forma mais complexa:

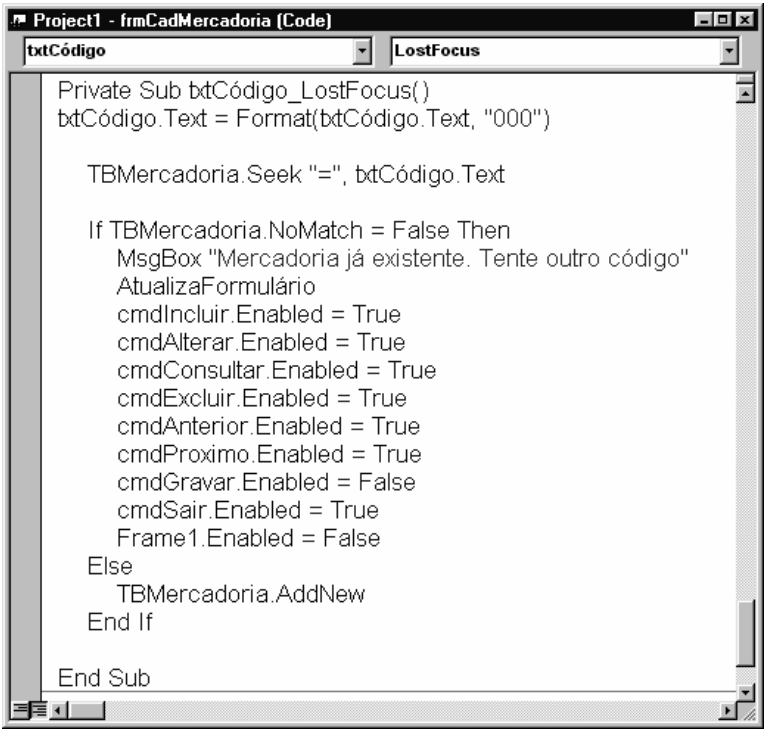


Veja o que foi feito: Sempre que o usuário clicar neste botão o programa limpa o formulário para deixar as caixas de texto vazias para serem preenchidas com novos dados. O **Frame** é habilitado para que as caixas de texto possam ser manipuladas. Em seguida o foco é passado para o caixa de texto **txtCódigo** pois é a primeira que deve ser digitada pelo usuário

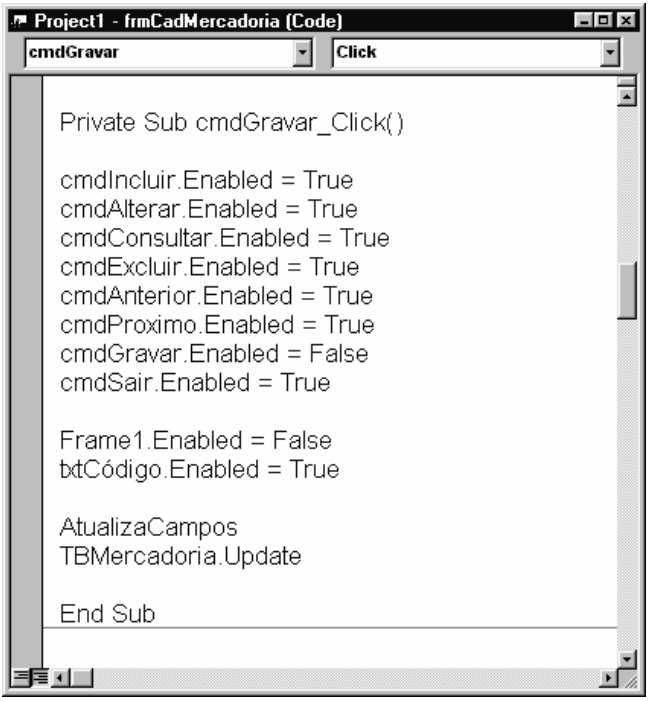
Ainda não aplicamos o método **AddNew** pois ainda não sabemos se o usuário vai incluir um código válido. Primeito o programa tem que analizar o código que ele digitou, verificar se ele realmente não existe para depois usar o método **AddNew.** Fazemos isto no evento **LostFocus** da Caixa de Texto txtCódigo.

Neste evento verificamos primeiramente se o botão CmdGravar esta habilitado. Se estiver é porque a inclusão foi acionada.

O programa procura o código digitado. Se ele existir sera dado um aviso ao usuario e o processo de inclusão será cancelado, caso contrário (ou seja, se o código não existir dentro da tabela) o método **AddNew** é chamado a fim de preparar a tabela para receber um novo registro.



Pronto, a janela já esta preparada para receber os dados. Mas agora falta fazer a codificação para o botão Gravar.

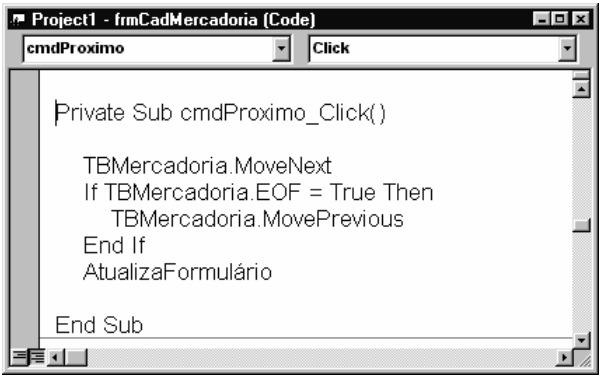


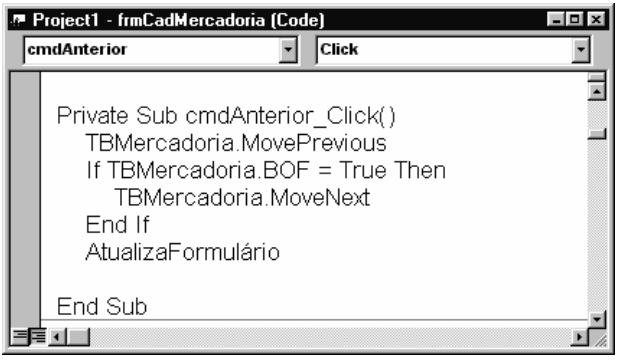
Atualizamos os campos da tabela com os dados digitado nas caixas de textos e depois chamamos o método **Update** que fará a gravação fisica dentro do banco de dados.

- Tente incluir vários dados na tabela para podermos usá-los posteriormente nos próximos exercícios.
- Um recurso interessante que pode ser acrescentado é desabilitar os botões de Alteração, Consulta , Exclusão, Anterior e Próximo quando a tabela estiver vazia, pois se não houver nenhum registro, não vai haver nada para alterar, consultar, excluir, etc. Somente o botão de inclusão que poderá ficar habilitado.
- A maneira mas usual de saber se a tabela esta vazia ou não é através da propriedade **RecordCount** que infoma quantos registros existem gravados dentro da tabela.

12.3 PRÓXIMO E ANTERIOR

Estes botões serão usados o usuário “passear” pela tabela, verificando os dados de todos os registros cadastrados na ordem especificada pelo Index.



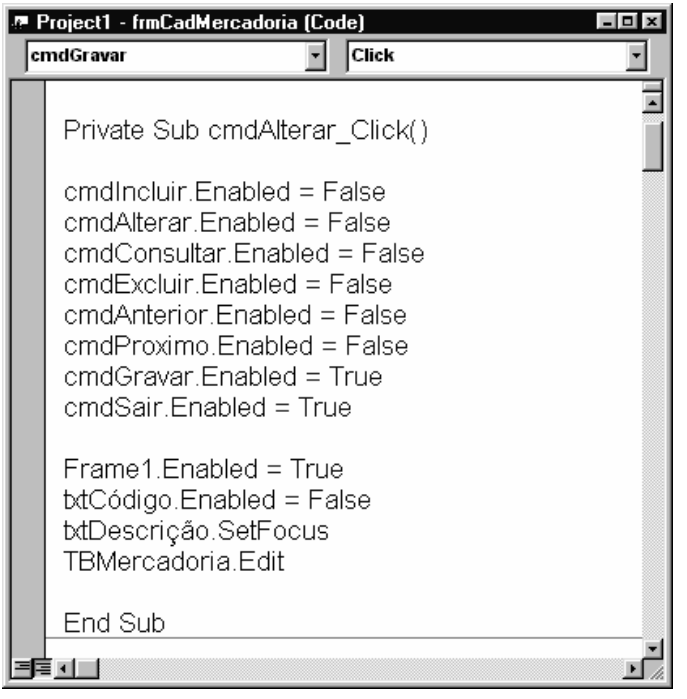


O botão Próximo irá mostrar na tela o próximo registro existente. Note que para fazer este movimento usamos a propriedade MoveNext (mova para o próximo), depois fazermos um teste com a propriedade EOF para verificar se foi movido para o ultimo registro na tabela. Se estiver no último, dali ele não pode passar pois não encontrará nada. Encontrando então o ultimo registro, se tentar passar dali a propriedade MovePrevious (move para o anterior) é acionado. Depois disto atualizamos o formulário.

O botão Anterior possui a mesma lógica, mas invertemos as outras propriedades para fazer agora o teste de inicio de arquivo. Ou seja, se o usuário estiver no primeiro registro ele não poderá voltar um registro, pois antes do primeiro não existira nada.

Se você usar MoveNext quando o último registro for o atual, a propriedade EOF será configurada para True e não haverá registro atual. Se você usar MoveNext novamente, um erro ocorrerá; EOF permanece True. Se recordset referir-se a um Recordset tipo table (que é o nosso caso), a movimentação segue o índice atual. Você pode definir o índice atual usando a propriedade Index. Se você não configurar o índice atual, a ordem de registros retornados será indefinida.

12.4 ALTERAÇÃO



Para alteração vamos fazer algo semelhante ao que fizemos na inclusão. A diferença ficará numa propriedade nova para nós que é EDIT. Esta

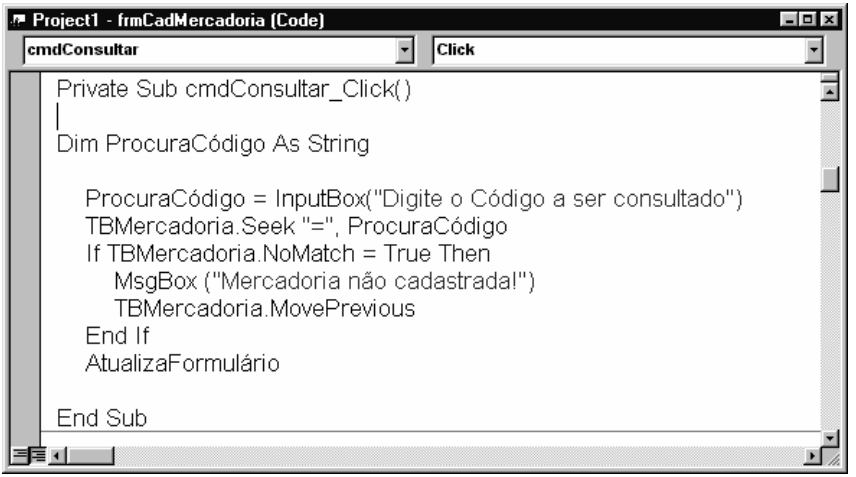
propriedade abre a tabela para que o dado que esta sendo alterado seja editado.

Note que não liberamos a caixa de texto txtCódigo para alteração, pois sendo ela a chave de índice, não pode ser alterada. Quando necessitar de alteração no código, ele deve ser excluído e incluído novamente.

12.5 CONSULTA

Para consulta vamos usar a função INPUTBOX. Aprendemos a usá-la nos capítulos anteriores. Ela solicita ao usuário a digitação de algum dado e armazena numa determinada variável.

Então criamos uma variável de nome ProcuraCódigo e usamos ela para receber o que o usuário digitar.



Feito isto, usamos o método Seek para efetuar uma procura dentro da tabela.

Seek: Localiza o registro de um objeto Recordset tipo table indexado que satisfaça os critérios especificados para o índice atual e torna esse registro o registro atual.

Após o Seek colocamos um sinal de comparação para determinar o tipo de procura que este método fara. Podemos usar “=” ,”<”, “>”, “<=”, “>=”, etc

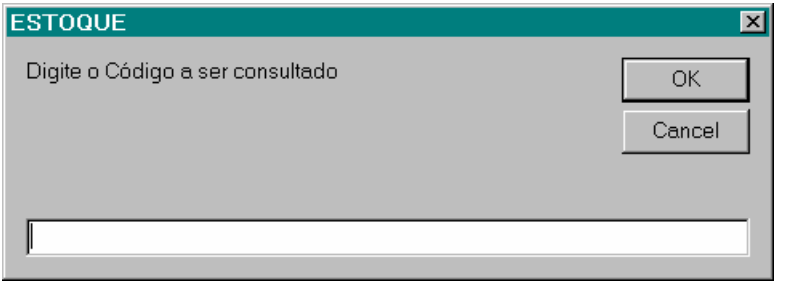
O método Seek pesquisa os campos-chave especificados e localiza o primeiro registro que satisfaça os critérios especificados na comparação dentro da chave de índice. Uma vez encontrado, esse registro torna-se o registro atual e a propriedade **NoMatch** é definida como False. Se o método **Seek** falhar em localizar um registro correspondente, a propriedade **NoMatch** é configurada como True e o registro atual é indefinido.

Se comparação for igual (=), maior ou igual a (>=) ou maior do que (>), o **Seek** começa a pesquisa no início do índice. Se comparação for maior do que (<) ou maior ou igual a (<=), o **Seek** começa a pesquisa no final do índice e continua em direção ao início a menos que existam entradas de índice duplicadas no final. Neste caso, **Seek** começa por uma entrada arbitrária entre as entradas de índice duplicadas no final do índice.

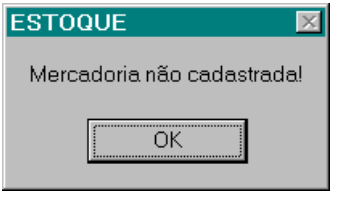
Quando informamos uma chave para o método **Seek** procurar, esta chave deve ser no mesmo formato que o estabelecido no índice do arquivo (index). Por exemplo: no TBMercadoria a chave do índice é o *Código*, então o *Seek* somente faz a procura pelo código. Não podemos pedir para procurar pela Descrição da mercadoria, pois para isto deveria existir um índice para descrição.

Em nosso exemplo se **NoMatch** for True (ou seja, se não encontrou nenhum registro que seja igual ao conteúdo da variável *ProcuraCódigo*) movemos para o registro anterior e depois o formulário é atualizado. Fazemos isto pois quando uma procura não é bem sucedida a tabela é posicionada no fim de arquivo, e como não existe nenhum registro nesta posição, movimentamos para um registro antes, onde o ultimo registro incluído se encontra.

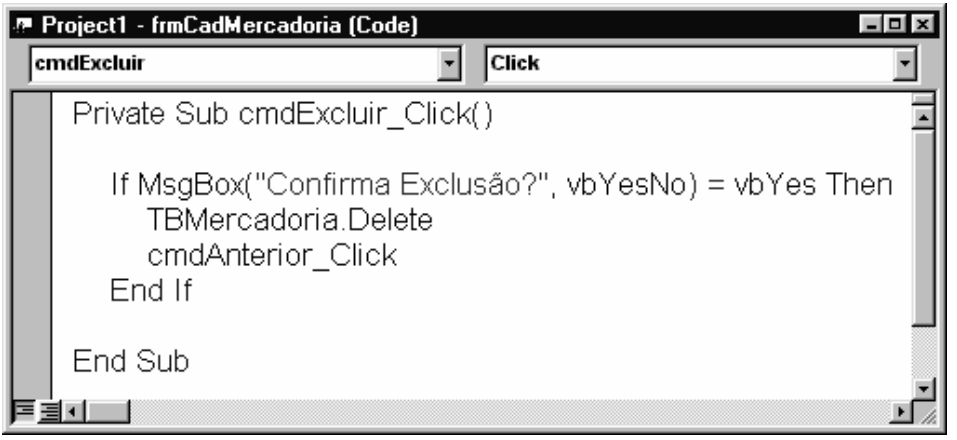
Quando clicarmos no botão Consultar uma janela como a que esta abaixo aparecerá no video:



Digitamos o código que queremos procurar e ao clicar Ok a busca é iniciada. Se não encontrar uma mensagem será apresentada:



12.6 EXCLUSÃO



Quando se ai deletar algo é bom solicitar ao usuário uma confirmação, pois a deleção acidental pode causar resultados catastróficos. Então é bom que o usuário tenha certeza do que esta fazendo.

Quando usamos a propriedade *Delete* o registro atual (aquele que esta aparecendo no vídeo) é excluído da tabela.

Mas para a tela não ficar vazia (uma vez que aquele registro que estava lá foi eliminado) usamos a procedure *cmdAnterior_click* que é a rotina que usamos quando o botão anterior é acionado.

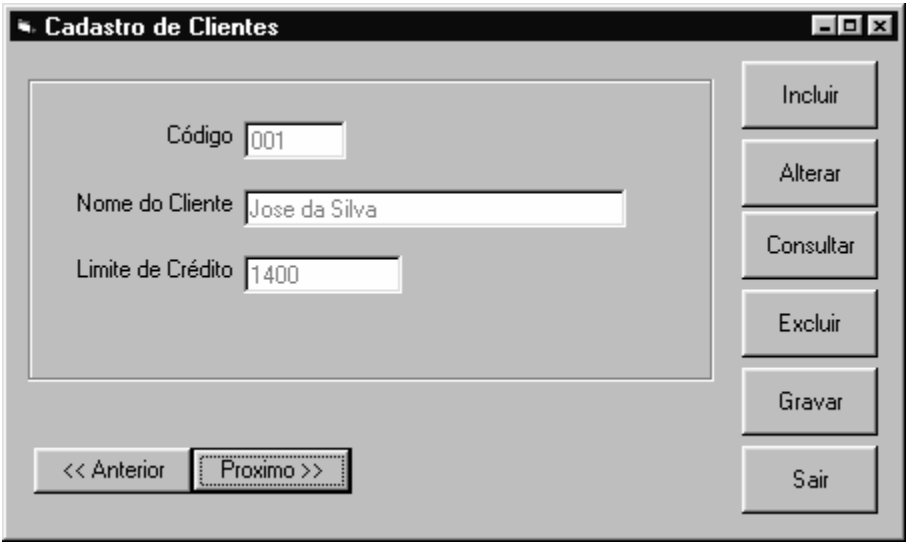
Fazemos isto para posicionar a tabela no registro anterior ao que foi deletado e mostrá-lo na tela. Desta forma a tela não fica vazia.

Em nosso programa podemos localizar um registro com a consulta ou através dos botões Anterior e Próximo. Localizando podemos clicar no botão Excluir que o registro desaparece.

12.7 CONSIDERAÇÕES FINAIS

Pronto. Esta finalizado este formulário frmCadMercadoria. Agora usando estes mesmos processos usados aqui programe o Cadastro de Cliente. As vendas é um pouco diferente pois esta tabela depende da tabela de Mercadoria e da tabela de Clientes, ou seja, é necessário, nas vendas, informar quem é o cliente que esta comprando e qual mercadoria se esta comprando. Estas informações são coletadas em suas respectivas tabelas. Mas vamos aprender a codificá-la após terminarmos com os Clientes.

12.8 CADASTRO DE CLIENTES




```
Option Explicit
Dim BancoDeDados As Database
Dim TBCliente As Recordset

Private Sub cmdAlterar_Click()

cmdIncluir.Enabled = False
cmdAlterar.Enabled = False
cmdConsultar.Enabled = False
cmdExcluir.Enabled = False
cmdAnterior.Enabled = False
cmdProximo.Enabled = False
cmdGravar.Enabled = True
cmdSair.Enabled = True

Frame1.Enabled = True
txtCódCliente.Enabled = False
txtNome.SetFocus
TBCliente.Edit

End Sub

Private Sub cmdAnterior_Click()
    TBCliente.MovePrevious
```

```
    If TBCliente.BOF = True Then
        TBCliente.MoveNext
    End If
    AtualizaFormulário

End Sub

Private Sub cmdConsultar_Click()

Dim ProcuraCódCliente As String

    ProcuraCódCliente = InputBox("Digite o Código do
Cliente a ser consultado")
    TBCliente.Seek "=", ProcuraCódCliente
    If TBCliente.NoMatch = True Then
        MsgBox ("Cliente não cadastrado!")
        TBCliente.MovePrevious
    End If
    AtualizaFormulário

End Sub

Private Sub cmdExcluir_Click()

    If MsgBox("Confirma Exclusão?", vbYesNo) = vbYes Then
        TBCliente.Delete
        cmdAnterior_Click
    End If

End Sub

Private Sub cmdGravar_Click()

cmdIncluir.Enabled = True
cmdAlterar.Enabled = True
cmdConsultar.Enabled = True
cmdExcluir.Enabled = True
cmdAnterior.Enabled = True
cmdProximo.Enabled = True
cmdGravar.Enabled = False
cmdSair.Enabled = True
```

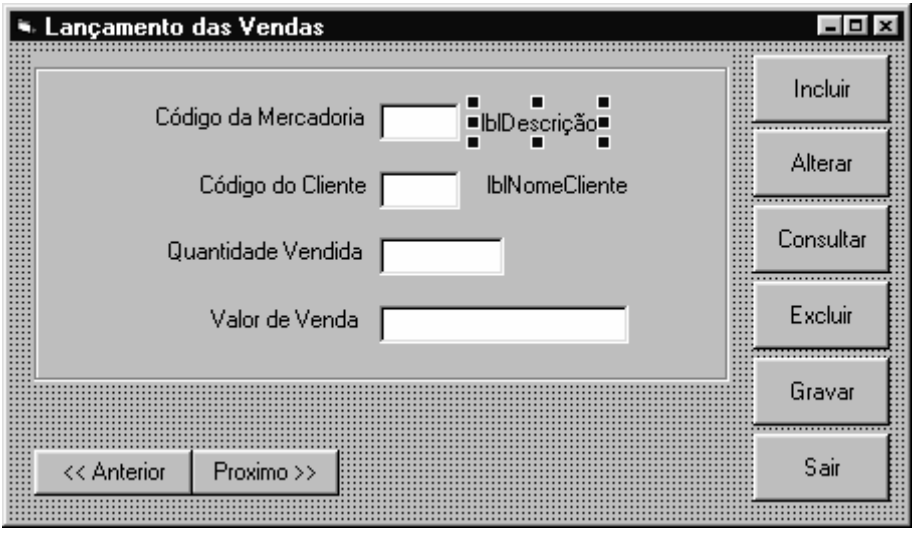
```
Frame1.Enabled = False
txtCódCliente.Enabled = True

AtualizaCampos
TBCliente.Update

End Sub

Private Sub cmdIncluir_Click()

cmdIncluir.Enabled = False
cmdAlterar.Enabled = False
cmdConsultar.Enabled = False
cmdExcluir.Enabled = False
cmdAnterior.Enabled = False
cmdProximo.Enabled = False
cmdGravar.Enabled = True
cmdSair.Enabled = True

LimpaFormulário
Frame1.Enabled = True
txtCódCliente.SetFocus

End Sub

Private Sub cmdProximo_Click()

    TBCliente.MoveNext
    If TBCliente.EOF = True Then
        TBCliente.MovePrevious
    End If
    AtualizaFormulário

End Sub

Private Sub cmdSair_Click()
    Unload frmCadClientes
End Sub

Private Sub Form_KeyPress(KeyAscii As Integer)
```

```
If KeyAscii = vbKeyReturn Then
    SendKeys ("{TAB}")
    KeyAscii = 0
End If

End Sub

Private Sub Form_Load()
Set BancoDeDados = OpenDatabase(App.Path & "\Estoque.MDB")
Set   TBCliente   =   BancoDeDados.OpenRecordset("Cliente",
dbOpenTable)

TBCliente.Index = "IndCódigo"

cmdGravar.Enabled = False
Frame1.Enabled = False

If TBCliente.EOF = False Then
    AtualizaFormulário
End If

End Sub

Private Sub Form_Unload(Cancel As Integer)

    TBCliente.Close
    BancoDeDados.Close

End Sub


Private Sub txtCódCliente_LostFocus()
txtCódCliente.Text = Format(txtCódCliente.Text, "000")

    TBCliente.Seek "=", txtCódCliente.Text

    If TBCliente.NoMatch = False Then
        MsgBox "Cliente já existente. Tente outro Código"
        AtualizaFormulário
        cmdIncluir.Enabled = True
```

```
            cmdAlterar.Enabled = True
            cmdConsultar.Enabled = True
            cmdExcluir.Enabled = True
            cmdAnterior.Enabled = True
            cmdProximo.Enabled = True
            cmdGravar.Enabled = False
            cmdSair.Enabled = True
            Frame1.Enabled = False
        Else
            TBCliente.AddNew
        End If

    End Sub

    Private Sub txtValor_LostFocus()
    txtLimiteCrédito.Text    =    Format(txtLimiteCrédito.Text,
    "Standard")
    End Sub


    Private Function AtualizaCampos()

        TBCliente("CódCliente") = txtCódCliente
        TBCliente("Nome") = txtNome
        TBCliente("LimiteCrédito") = txtLimiteCrédito

    End Function

    Private Function AtualizaFormulário()

        txtCódCliente = TBCliente("CódCliente")
        txtNome = TBCliente("Nome")
        txtLimiteCrédito.Text                                    =
    Format(TBCliente("LimiteCrédito"), "Standard")

    End Function

    Private Function LimpaFormulário()

        txtCódCliente = ""
        txtNome = ""
```

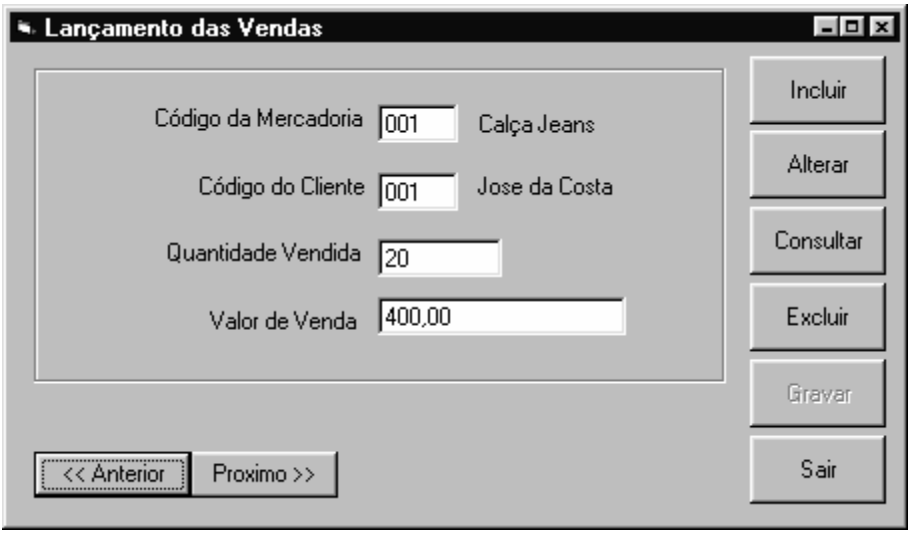
```
    txtLimiteCrédito = ""

End Function
```

12.9 LANÇAMENTO DAS VENDAS



A criação do formulário será como no exemplo acima; ou seja, comparando com os anteriores, a diferença é somente o acréscimo de dois labels que ficarão ao lado do código da mercadoria e do cliente. O primeiro label conterá a descrição da mercadoria, e vamos nomeá-lo para *lblDescrição,* e o segundo terá o nome do cliente, e vamos nomeá-lo para *lblNomeCliente.*

Para esta tabela é necessário criar um índice em que a chave de procura seja o Código da Mercadoria e o Código do Cliente. Usamos para isto o Visual Data Manager:



Note que esses dados não existem na tabela de Vendas. O que existe nesta tabela é somente o código da mercadoria. A descrição esta em outra tabela (Mercadoria), assim como o nome do cliente também esta em outra tabela (Cliente). Para podermos acessar estas tabelas e capturar somente a descrição da mercadoria e nome do cliente, temos somente um ponto de referência, que é o código deles que existe na tabela de Vendas.

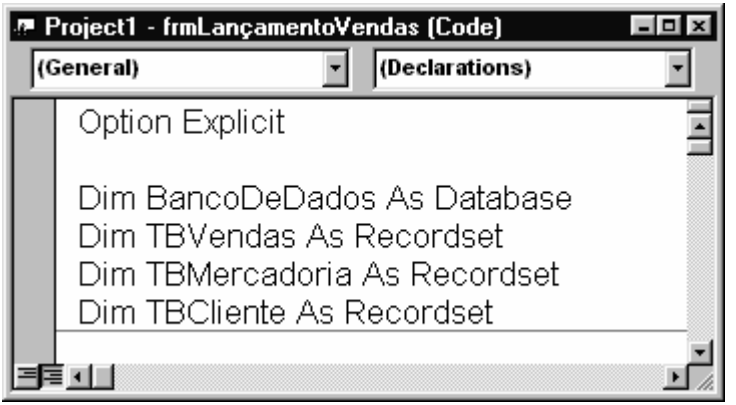
A lógica usada neste programa consiste em pegarmos o código existente na tabela de vendas e buscar a descrição daquele código em outra tabela. Para fazer isto todas as tabelas envolvidas devem ser abertas, assim como seus índices, pois será através do índice que a procura será feita.

Exemplificando através da tabela de mercadoria:

Como na tabela de vendas existe o código da mercadoria que foi vendida, e na tabela de mercadoria existe a descrição da mercadoria para

aquele código, usamos o índice para procurar dentro da tabela de mercadoria, o código existente dentro da tabela de vendas. Isto é possível desde que a tabela de mercadoria esteja indexada pelo código, pois a procura será feita por ele.

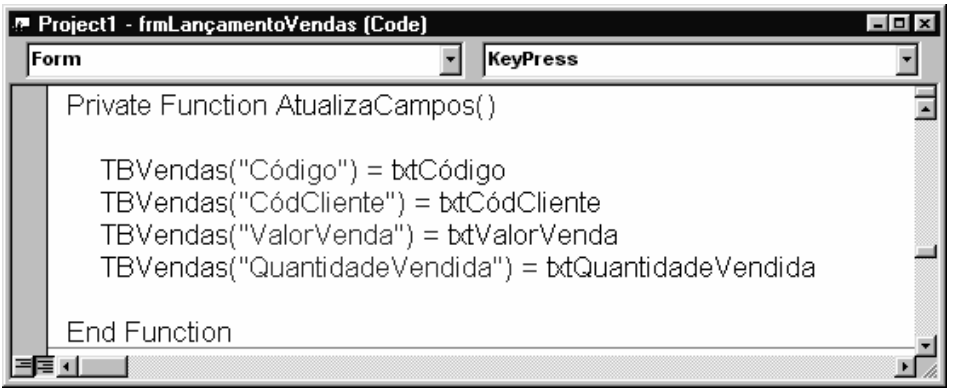
Para podermos então codificarmos o formulário que terá acesso aos dados da tabela de vendas, vamos criar as variáveis que serão usadas para lidar com o banco de dados e as tabelas:

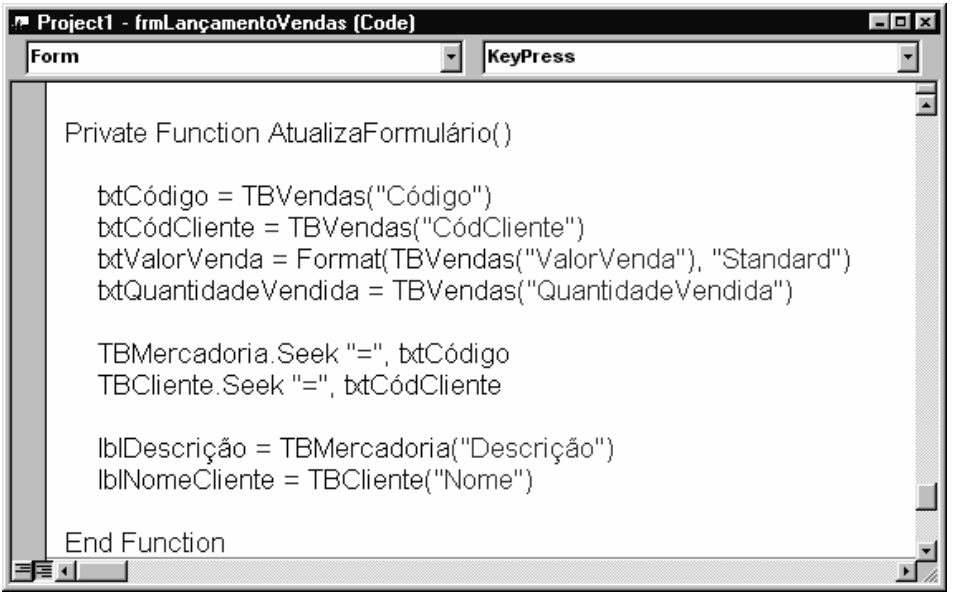


Criamos uma variável para cada tabela que será aberta. Precisamos agora criar um evento Form_Load para abrir todas essas tabelas.

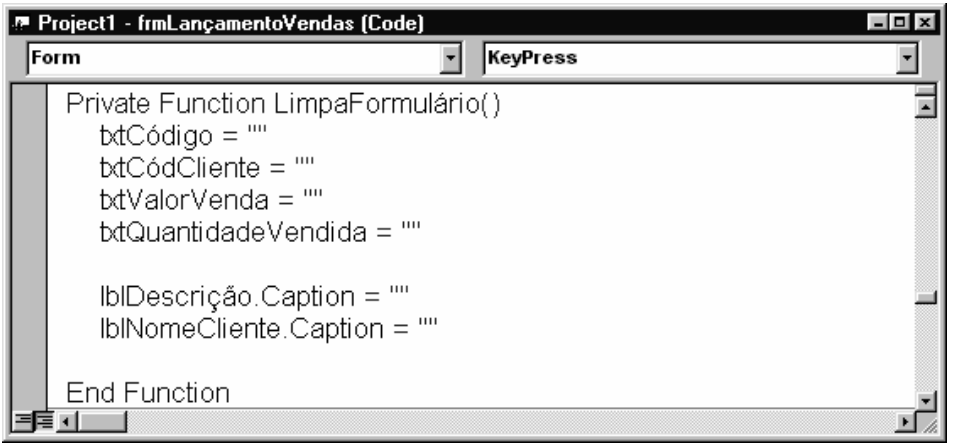
Veja que abrimos todas as três tabelas que serão usadas neste formulário, e abrimos também todos os índices relativo a cada uma.

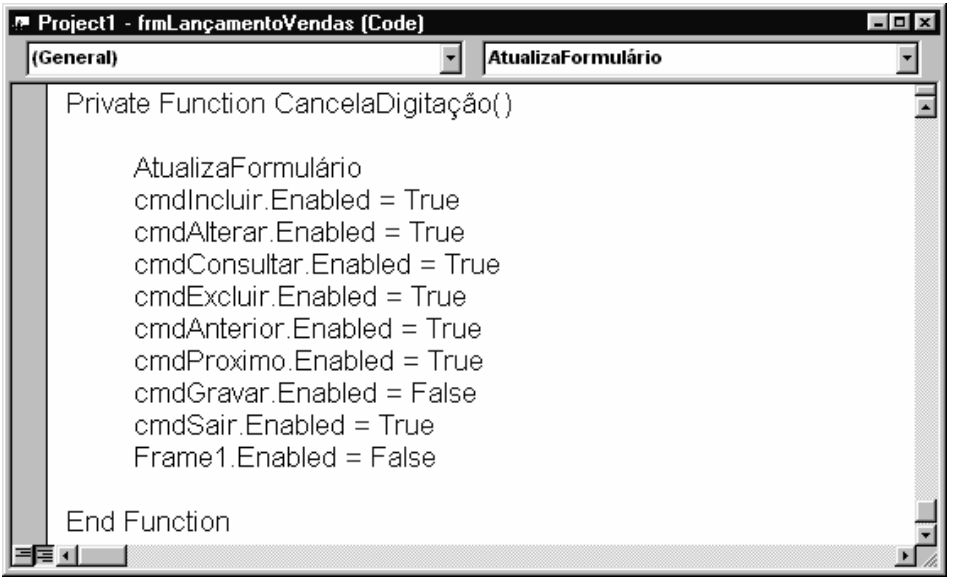
As nossas funções auxiliares sofrerão alguma mudança pois acrescentamos em nosso formulário dois **labels**, e um conterá a descrição da mercadoria e outro o nome do cliente.





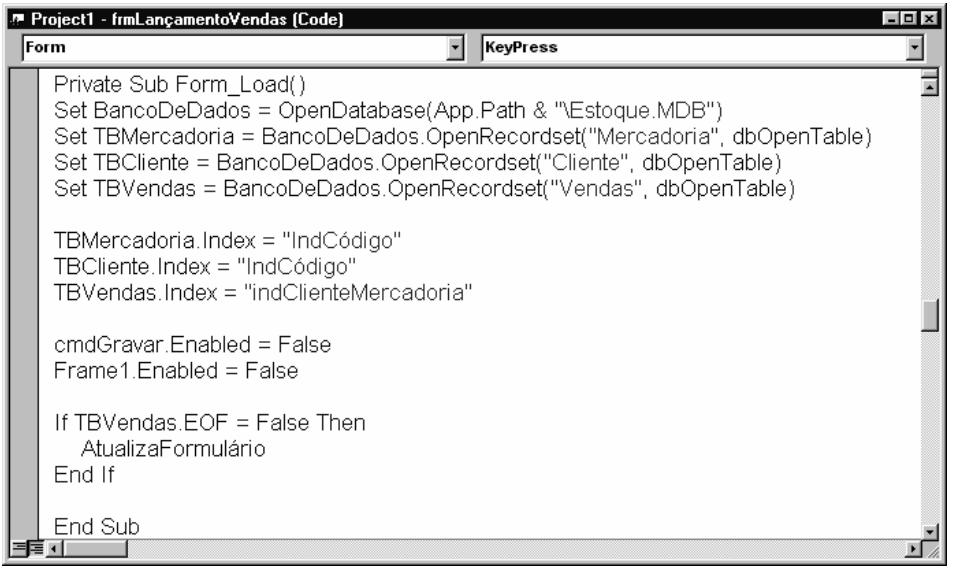
Note que quando formos atualizar o formulário será necessário apresentar nos dois **labels** os dados contido em outras tabelas, por isto é necessário colocar um **Seek** para cada item a ser procurado. Quando o usuário, por exemplo, aperta o botão "Avançar" o programa avança um registro na tabela de Vendas e usa os códigos contidos nela para tentar localizar suas respectivas descrições e nome do cliente.





Para facilitar nosso trabalho, criamos aqui uma função **CancelaDigitação** que tem por finalidade anular o que o usuário estiver fazendo, voltar os botões e **frame** para seu estado natural. Usaremos esta função quando o usuário digitar algum código inválido.

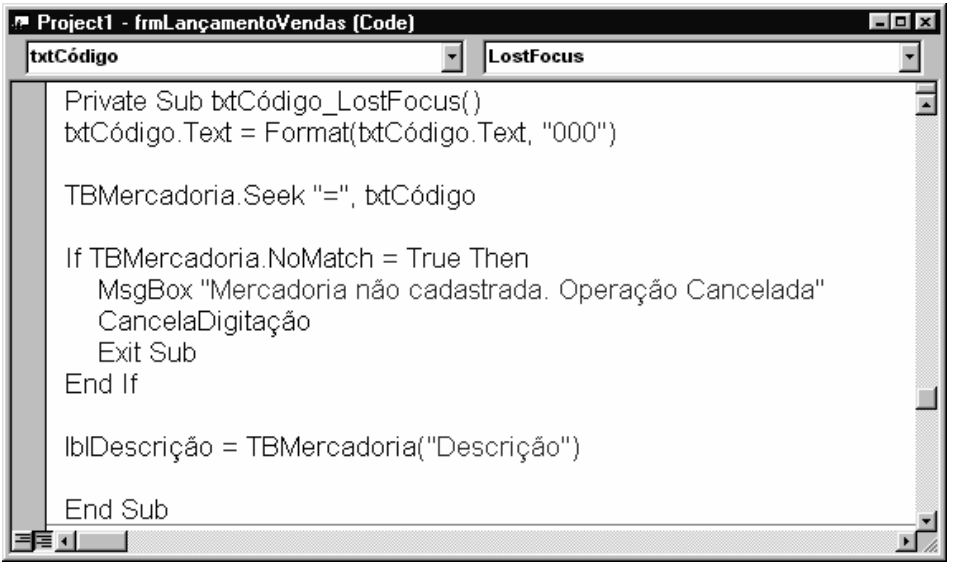
Com as funções prontas, vamos completar a codificação do **Form_Load**, pois apenas abrimos as tabelas, precisamos agora verificar se a tabela de vendas está vazia, e desabilita o botão "Gravar" e o **frame**:



A inclusão para esta tabela segue o estilo que usamos para os outros formulários. Existe somente uma diferença fundamental que é o aparecimento do nome do cliente e a descrição da mercadoria quando o usuário digitar o código correspondente. Ou seja, numa inclusão, quando o usuário digitar o código da mercadoria que foi vendida, o programa terá que acessar a tabela de Mercadoria, procurar o código que o usuário acabou de digitar, e trazer de lá a descrição daquele código.

Este trabalho todo podemos fazer no evento **LostFocus** da caixa de texto. Por que este evento? Porque esta procura será feita DEPOIS que o usuário digitar o código e passar para a próxima caixa de texto (ou seja, quando o objeto perder o foco).

Então, diante disto, precisamos criar dois evento **LostFocus**. Uma para a caixa de texto txtCódigo e outro para txtCódCliente:



Usamos o **SEEK** para efetuar uma procura dentro da tabela TBMercadoria.

O que irá procurar?

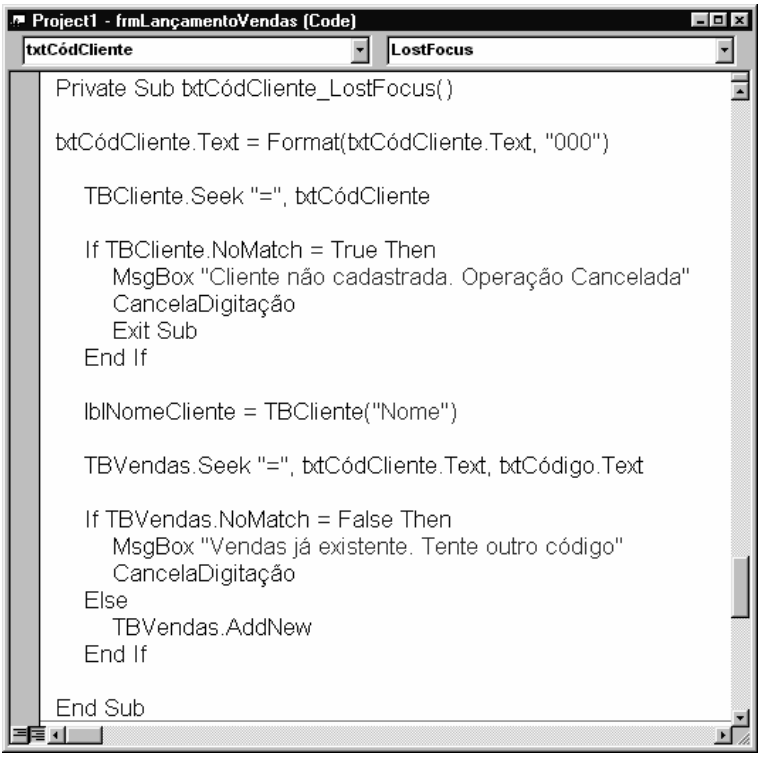
Irá procurar o código digitado na caixa de texto txtCódigo.

Se **NoMatch** for igual a True é sinal que não existe nenhuma mercadoria cadastrada com o código digitado. Então uma mensagem será mostrada na tela: “Mercadoria não cadastrada”.

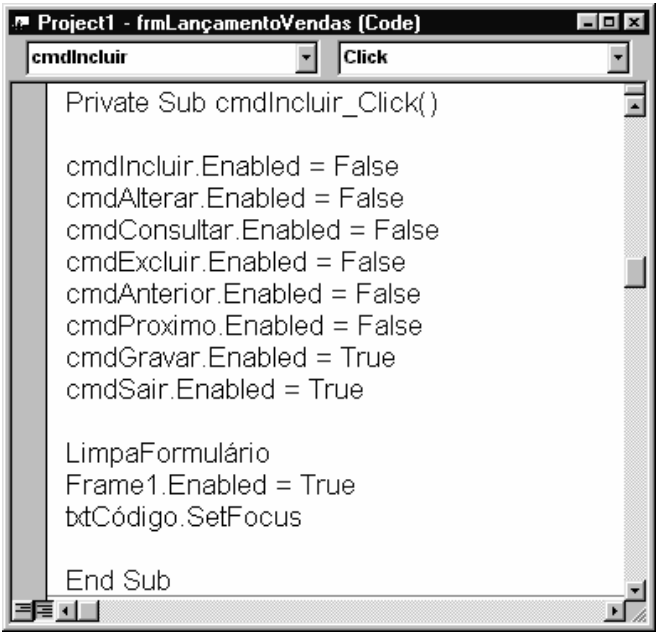
Depois chamamos aquela função que criamos chamada Cancela Digitação. Logicamente que se quisermos incrementar ainda mais nosso programa poderíamos fazer com que o programa abrisse uma tela mostrando quais as mercadorias que existem cadastradas para o usuário escolher uma, ou algo assim. Mas para resumir nosso exemplo deixaremos assim.

Se o código digitado na caixa de texto for encontrado então a descrição é inserida dentro do label "lblDescrição" para que seja mostrada no formulário.

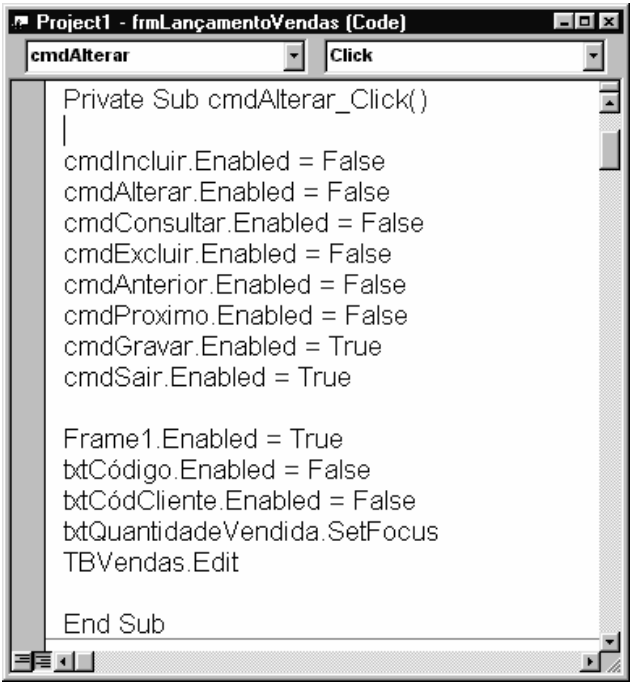
Fazemos a mesma coisa com a evento **LostFocus** de txtCódCliente:

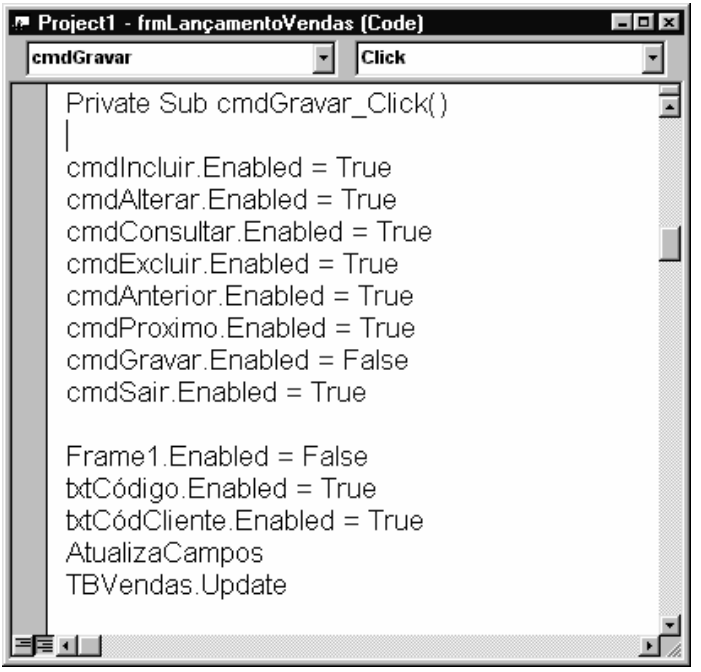


A inclusão de registros dentro da tabela de vendas é igual ao apresentado nos outros programas:

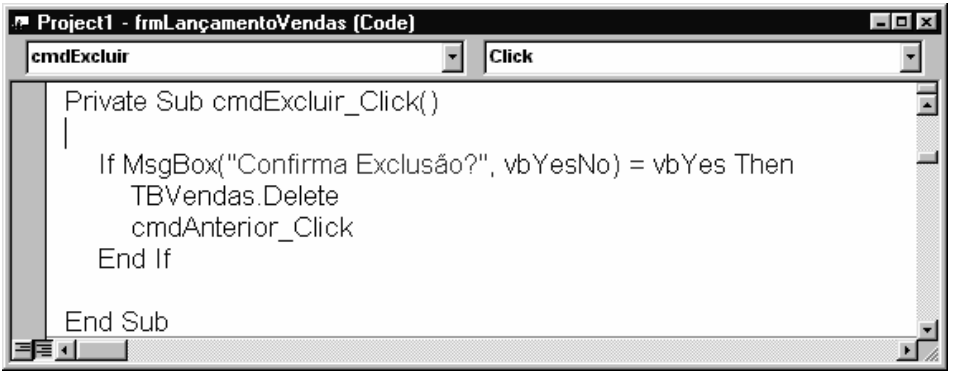


A alteração segue os padrões anteriores, lembrando que nesta tabela de vendas não podemos alterar o Código da Mercadoria e o Código do Cliente, pois ambos fazem parte da chave do índice da tabela. Se necessitar alterar o código, exclua e inclua novamente:

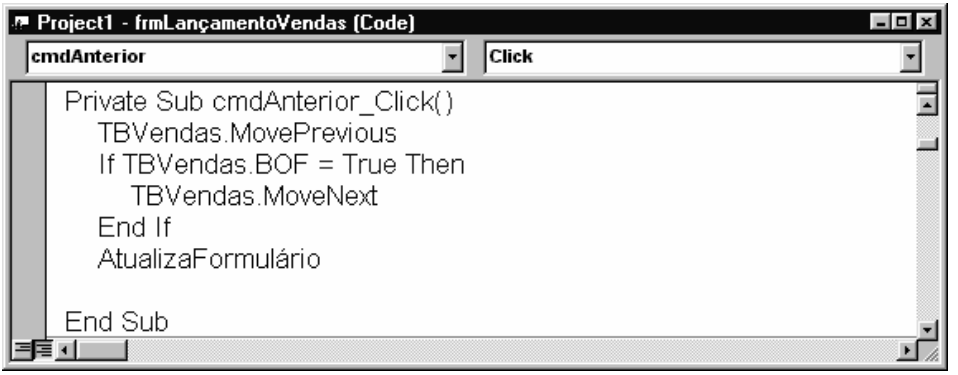




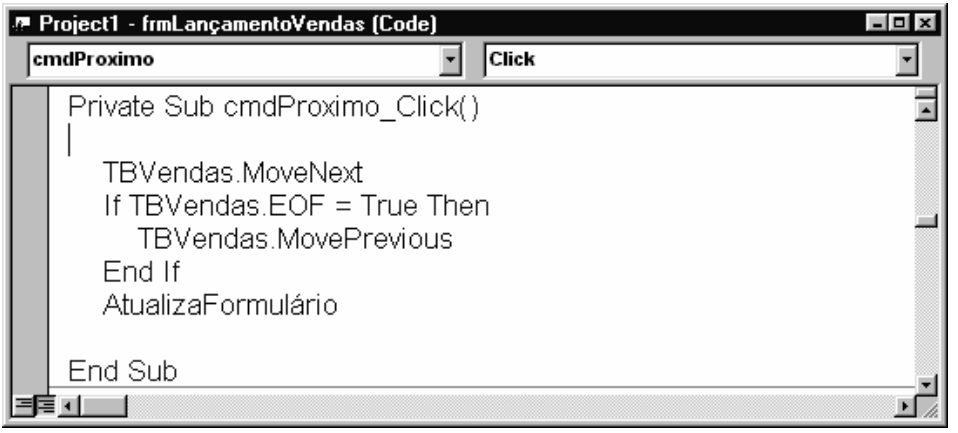
A exclusão também é semelhante aos anteriores:



Os botões Anterior e Próximo devem fazer uma pesquisa na tabela de Mercadoria e Clientes para atualizar os **labels**, a fim de que apareça a descrição da mercadoria e o nome do cliente:



Perceba que esta rotina move a tabela de vendas para o registro anterior, e depois ao chamar a função **AtualizaFormulário** faz uma procura dentro da tabela de mercadoria e depois dentro da tabela de clientes usando como chave de busca o conteúdo existente dentro da tabela de vendas.



Dê uma conferida em seu código fonte agora:

```
Option Explicit

Dim BancoDeDados As Database
Dim TBVendas As Recordset
Dim TBMercadoria As Recordset
Dim TBCliente As Recordset

Private Sub cmdAlterar_Click()

cmdIncluir.Enabled = False
cmdAlterar.Enabled = False
cmdConsultar.Enabled = False
cmdExcluir.Enabled = False
cmdAnterior.Enabled = False
cmdProximo.Enabled = False
cmdGravar.Enabled = True
cmdSair.Enabled = True

Frame1.Enabled = True
txtCódigo.Enabled = False
txtCódCliente.Enabled = False
txtQuantidadeVendida.SetFocus
TBVendas.Edit

End Sub

Private Sub cmdAnterior_Click()
    TBVendas.MovePrevious
    If TBVendas.BOF = True Then
        TBVendas.MoveNext
    End If
    AtualizaFormulário
End Sub

Private Sub cmdExcluir_Click()

    If MsgBox("Confirma Exclusão?", vbYesNo) = vbYes Then
        TBVendas.Delete
        cmdAnterior_Click
    End If
```

```
End Sub

Private Sub cmdGravar_Click()

cmdIncluir.Enabled = True
cmdAlterar.Enabled = True
cmdConsultar.Enabled = True
cmdExcluir.Enabled = True
cmdAnterior.Enabled = True
cmdProximo.Enabled = True
cmdGravar.Enabled = False
cmdSair.Enabled = True

Frame1.Enabled = False
txtCódigo.Enabled = True
txtCódCliente.Enabled = True
AtualizaCampos
TBVendas.Update

End Sub

Private Sub cmdIncluir_Click()

cmdIncluir.Enabled = False
cmdAlterar.Enabled = False
cmdConsultar.Enabled = False
cmdExcluir.Enabled = False
cmdAnterior.Enabled = False
cmdProximo.Enabled = False
cmdGravar.Enabled = True
cmdSair.Enabled = True

LimpaFormulário
Frame1.Enabled = True
txtCódigo.SetFocus

End Sub

Private Sub cmdProximo_Click()
```

```
    TBVendas.MoveNext
    If TBVendas.EOF = True Then
        TBVendas.MovePrevious
    End If
    AtualizaFormulário

End Sub

Private Sub cmdSair_Click()
    Unload frmLançamentoVendas
End Sub

Private Sub Form_KeyPress(KeyAscii As Integer)

If KeyAscii = vbKeyReturn Then
    SendKeys ("{TAB}")
    KeyAscii = 0
End If

End Sub

Private Sub Form_Load()
Set BancoDeDados=OpenDatabase(App.Path & "\Estoque.MDB")
SetTBMercadoria=BancoDeDados.OpenRecordset"Mercadoria",
dbOpenTable)
Set TBCliente=BancoDeDados.OpenRecordset("Cliente",
dbOpenTable)
Set TBVendas=BancoDeDados.OpenRecordset("Vendas",
dbOpenTable)

TBMercadoria.Index = "IndCódigo"
TBCliente.Index = "IndCódigo"
TBVendas.Index = "indClienteMercadoria"

cmdGravar.Enabled = False
Frame1.Enabled = False

If TBVendas.EOF = False Then
    AtualizaFormulário
End If
```

```
End Sub

Private Sub Form_Unload(Cancel As Integer)

    TBVendas.Close
    TBMercadoria.Clone
    TBCliente.Close
    BancoDeDados.Close

End Sub

Private Function AtualizaCampos()

    TBVendas("Código") = txtCódigo
    TBVendas("CódCliente") = txtCódCliente
    TBVendas("ValorVenda") = txtValorVenda
    TBVendas("QuantidadeVendida") = txtQuantidadeVendida

End Function


Private Function LimpaFormulário()
    txtCódigo = ""
    txtCódCliente = ""
    txtValorVenda = ""
    txtQuantidadeVendida = ""

    lblDescrição.Caption = ""
    lblNomeCliente.Caption = ""

End Function

Private Sub txtCódigo_LostFocus()
txtCódigo.Text = Format(txtCódigo.Text, "000")

TBMercadoria.Seek "=", txtCódigo

If TBMercadoria.NoMatch = True Then
    MsgBox "Mercadoria não cadastrada. Operação Cancelada"
    CancelaDigitação
    Exit Sub
```

```
End If

lblDescrição = TBMercadoria("Descrição")

End Sub

Private Sub txtCódCliente_LostFocus()

txtCódCliente.Text = Format(txtCódCliente.Text, "000")

    TBCliente.Seek "=", txtCódCliente

    If TBCliente.NoMatch = True Then
        MsgBox "Cliente não cadastrada. Operação
Cancelada"
        CancelaDigitação
        Exit Sub
    End If

    lblNomeCliente = TBCliente("Nome")

    TBVendas.Seek "=", txtCódCliente.Text, txtCódigo.Text

    If TBVendas.NoMatch = False Then
        MsgBox "Vendas já existente. Tente outro código"
        CancelaDigitação
    Else
        TBVendas.AddNew
    End If

End Sub

Private Sub txtValorVenda_LostFocus()
txtValorVenda.Text = Format(txtValorVenda.Text,
"Standard")
End Sub

Private Function AtualizaFormulário()

    txtCódigo = TBVendas("Código")
    txtCódCliente = TBVendas("CódCliente")
```

```
    txtValorVenda = Format(TBVendas("ValorVenda"),
"Standard")
    txtQuantidadeVendida = TBVendas("QuantidadeVendida")
    TBMercadoria.Seek "=", txtCódigo
    TBCliente.Seek "=", txtCódCliente
    lblDescrição = TBMercadoria("Descrição")
    lblNomeCliente = TBCliente("Nome")

End Function

Private Function CancelaDigitação()
        AtualizaFormulário
        cmdIncluir.Enabled = True
        cmdAlterar.Enabled = True
        cmdConsultar.Enabled = True
        cmdExcluir.Enabled = True
        cmdAnterior.Enabled = True
        cmdProximo.Enabled = True
        cmdGravar.Enabled = False
        cmdSair.Enabled = True
        Frame1.Enabled = False
End Function
```

Deixamos o botão Consulta de fora propositadamente. Vamos voltar nele nos próximos capítulos.

13 USANDO O CONTROLE DATA

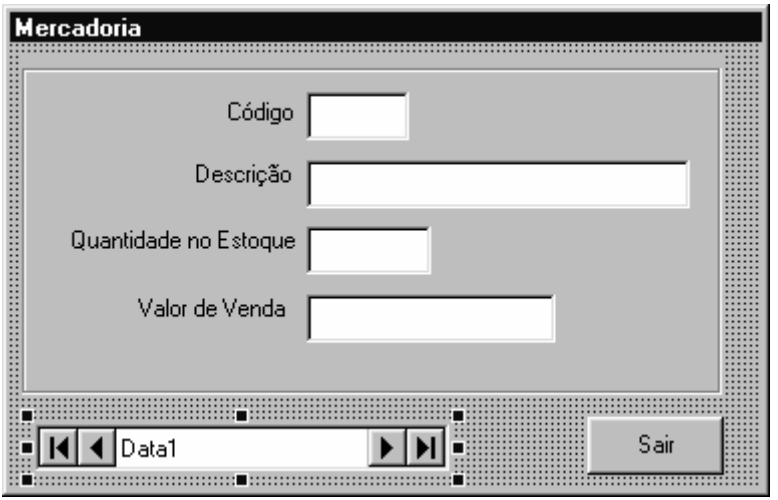
- Data
- DBGrid

13.1 CONTROLE DATA

Data

Este controle economiza uma séria de linhas de programação, fazendo o intercâmbio entre os controles de um formulário com a tabela do Banco de Dados.

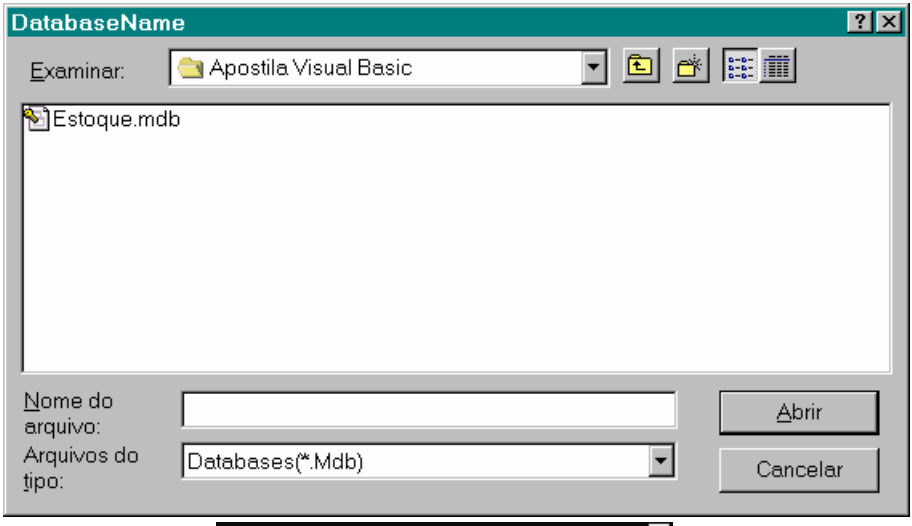
Vamos usar a mesma tabela de Mercadoria para criar um formulário e manipular esta tabela através do controle **data.**

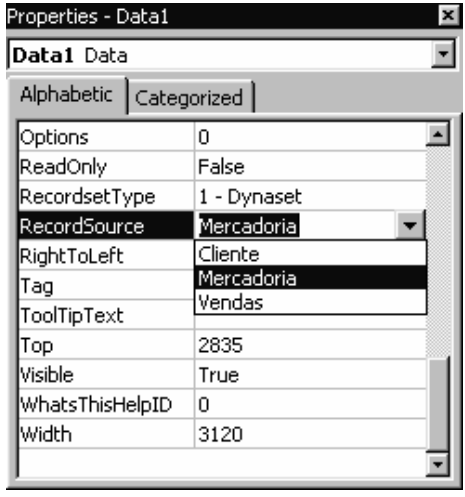


Crie um projeto novo e coloque os seguintes controles nele. Repare que é semelhante ao que fizemos no capítulo anterior, mas na verdade tem grandes mudanças na codificação. Mude o **Caption** deste controle para "Mercadoria".

As duas propriedades principais deste controle são: DatabaseName e RecordSource.

DatabaseName: Nesta propriedade selecionamos o nome do banco de dados que será manipulado. Uma caixa igual a que aparece abaixo será apresentada. Em nosso exemplo, selecione Estoque.mdb que é o banco de dados que estamos trabalhando.

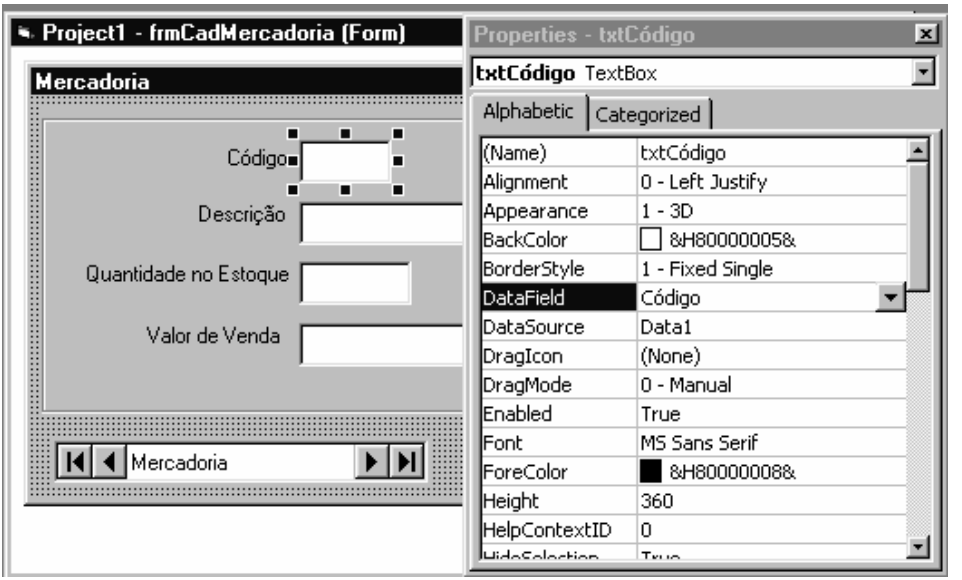




RecordSource: Nesta propriedade deve ser selecionado o nome da tabela que será manipulada pelo objeto Data. Dentro do Banco de Dados escolhido (Estoque.mdb) existem 3 tabelas: Vendas, Mercadoria e Cliente. As 3 irão aparecer. Escolha "Mercadoria".

Cada caixa de texto deve ser agora anexada ao controle **data** e, por conseguinte, aos seus respectivos campos. Por exemplo: A caixa de Texto *txtCódigo* deve ser anexada ao campo *Código*.

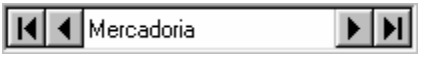
Esta anexação é feita usando as propriedades **DataField** e **DataSource** das caixas de texto.



DataSource: Origem dos dados. Aqui é informado o nome da controle **data** que possui a tabela que vamos manipular. Selecione *Data1*

DataField: Nome do Campo. Na caixa de combinação que aparece, esta o nome de todos os campos da tabela existente no controle **data**. Escolha o campo que corresponda com a caixa de texto selecionada.

Pronto. Não precisa digitar nada na codificação do programa. Com somente este passos já temos pronto um programa que altera os dados da tabela, pois tudo que for digitado nas caixa de texto será inserido automaticamente dentro dos registros da tabela. Use o controle **data** para avançar ou voltar, ir para o primeiro registro ou o ultimo.



Este controle data anexa dados não somente em caixas de texto, mas tambem em outros controles, como por exempo o **DBGrid**, que é muito útil para o programador.

13.2 DBGRID

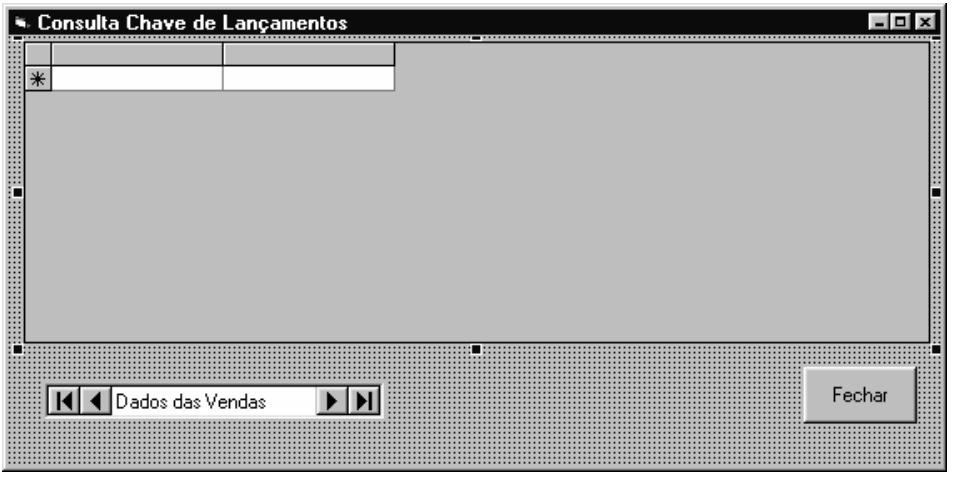
No projeto desenvolvido no capítulo anterior, deixamos de codificar o botão de Consulta. Vamos fazer isto agora usando um DBGrid para auxiliar na consulta. Adicione um novo formulário e mude a propriedade **Name** para

frmConsulta. Neste formulário vamos criar todas as rotinas de consulta que irão aparecer para o usuário.

Insira um **CommandButton** no formulário, um objeto do tipo **Data** e um objeto do tipo **DBGrid**:

 DBGrid  Data

Estes Controles serão inseridos na seguinte disposição:



No objeto **Data** mude a propriedade **name** para datVendas e o **Caption** para “Dados das Vendas”.

No objeto **DBGrid** mude a propriedade **name** para gridVendas e o **Caption** para “Lançamento das Vendas”.

O Objeto **Data** possui algumas propriedades que devem ser alteradas, como o **DatabaseName** e **RecordSource.** Escolha "Estoque.MDB" para uma e "Vendas" para outra.

Trabalhando somente com essas duas propriedade anexamos a tabela Vendas do Banco de Dados Estoque.mdb no objeto **Data**, que nomeamos para datVendas.

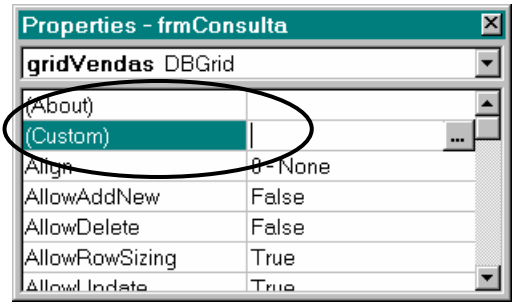
Com isto, sempre que fizermos agora qualquer evento no datVendas será refletido diretamente na tabela Vendas.

O objeto que manipulará os dados foi então criado, que é o **Data**, mas onde os registros do irão aparecer?

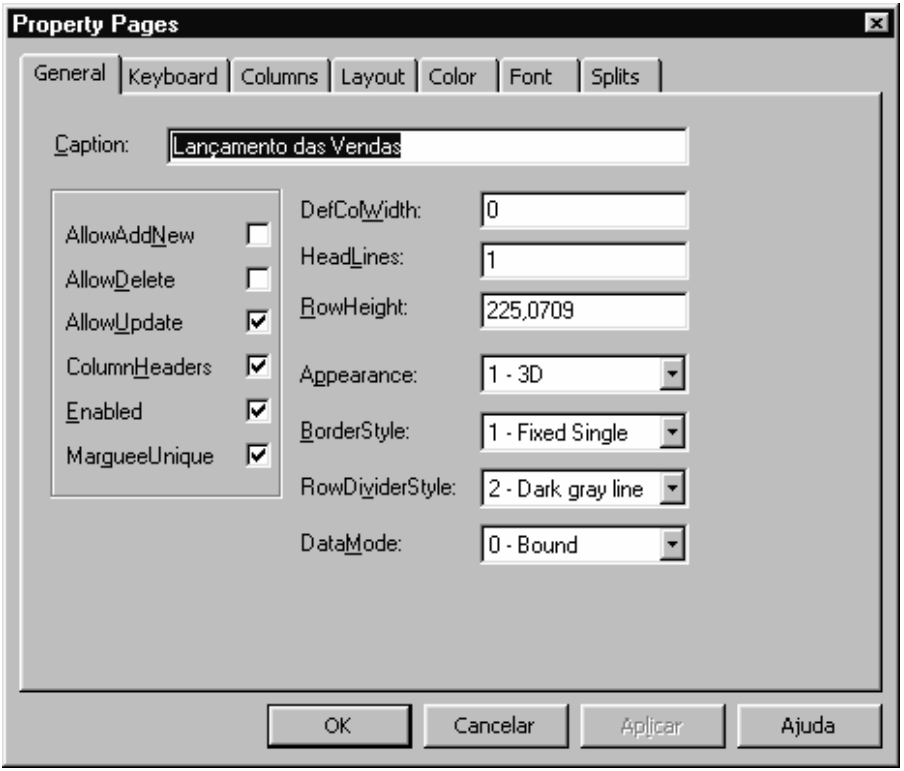
Poderíamos colocar cada Campo anexado a uma caixa de texto, mas em lugar disto vamos colocar todos os registros existentes na tabela Vendas dentro de um objeto **DBGrid**.

O **DBGrid** possui uma propriedade chamada **DataSource.** Nesta propriedade informamos o nome do objeto **Data** que inserimos no formulário.

Fazendo isto estaremos fazendo uma ligação entre o banco de dados e a tabela, já definida no objeto Data, para o objeto DBGrid. Com isto o DBGrid já saberá qual tabela que ele irá manipular.



A propriedade Custom deste objeto abre uma janela onde estão as principais propriedades deste objetos. Ao clicar será mostrado uma janela para configurar estas propriedades que irão dar forma ao DBGrid.



No **Caption** digitamos o título que será exibido no Grid.

AllowAddNew: Habilita a possibilidade de se poder acrescentar registros na tabela.

AllowDelete : Habilita a possibilidade de apagar registros

AllowUpdate : Habilita a possibilidade de alterar os registros existentes

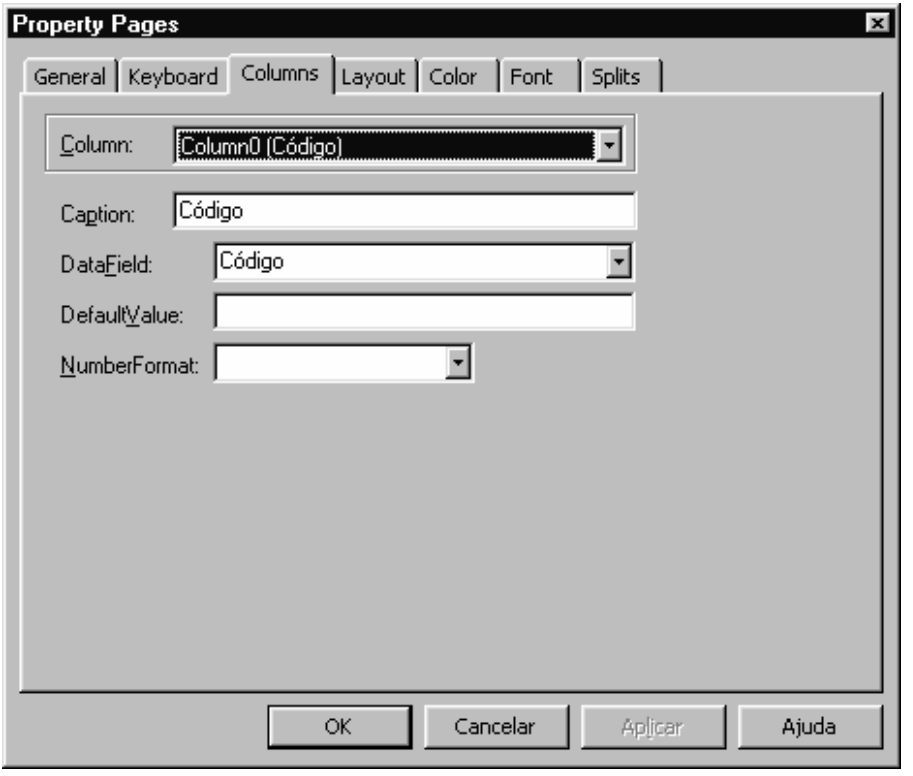
ColumnHeaders : Determina se o cabeçalho de cada coluna será exibido.

RecordSelectors : Determina se no DBGrid irá aparecer um seletor de registros

AllowRowSizing : Habilita a possibilidade do usuário mudar o tamanho das colunas.

Enabled : Habilita ou não a interação do DBGrid com o usuário.

Selecionando no topo desta janela a seção **Columns** abriremos novas propriedades para serem configuradas.



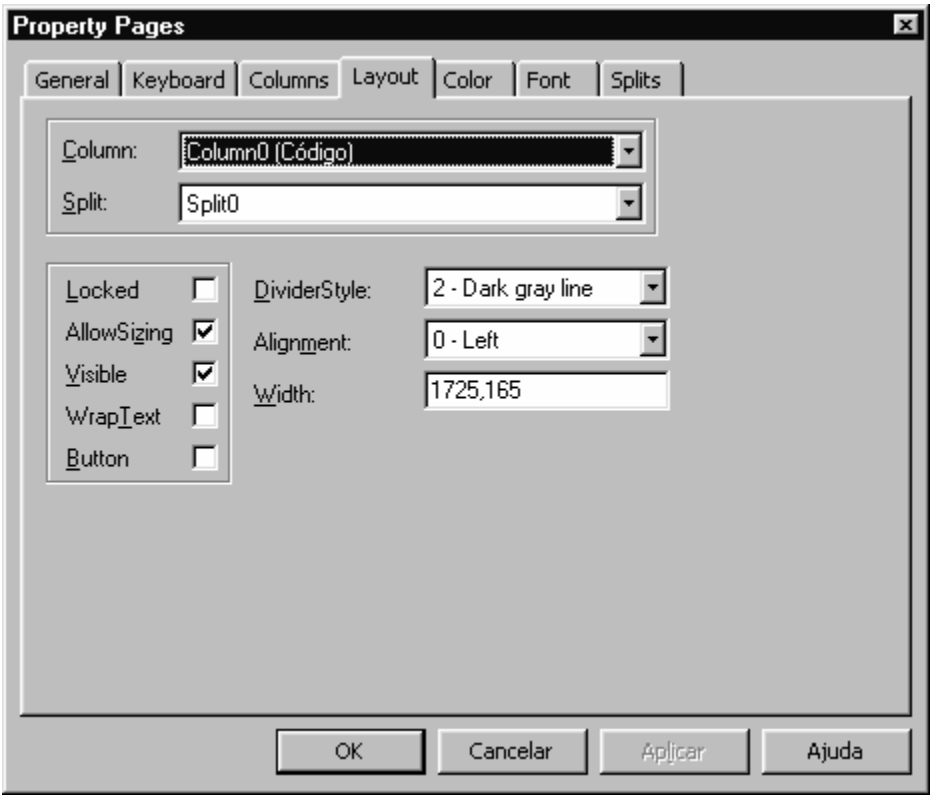
Columns : Seleciona em qual coluna de dados do DBGrid que terá as propriedades abaixo disponível para serem configuradas.

Caption : Título da Coluna selecionada.

DataField : Campo da tabela que terá seus dados exibidos na coluna selecionada.

DefaultValue : Valor padrão para um determinado campo.

NumberFormat : Determina o formato com que os dados da coluna serão mostrados no vídeo. Usa as mesmas regras da função FORMAT.



Locked : Trava os dados contido nas colunas.

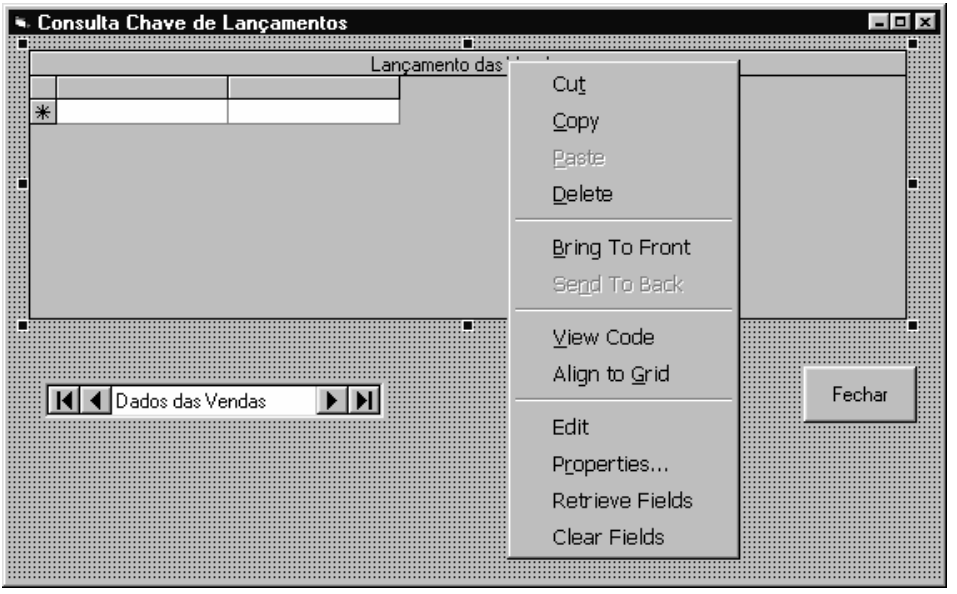
AllowSizing : Habilitada a possibilidade de mudar o tamanho da coluna

Visible : Indica se a coluna será visível ou não.

Alignment : Alinha dos dados na coluna na forma determinada nesta propriedade.

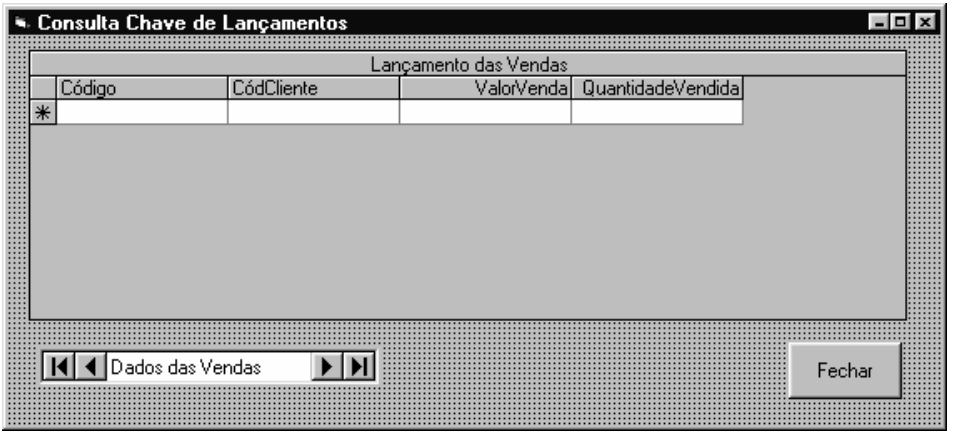
Uma forma rápida e recomendada para transferir os dados da tabela para suas respectivas propriedades é a seguinte:

- Selecione o DBGrid no formulário.
- posicione o ponteiro do mouse sobre ele e pressione o botão da direita.
- Irá aparecer um menu como o que esta abaixo:

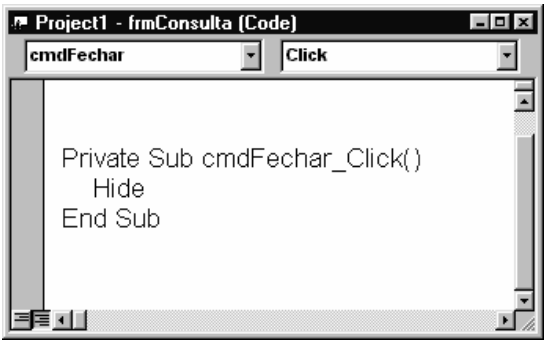


- Selecione a opção **Retrieve Fields**.

Isto fará com que todos os dados relativo a tabela que foi informada na propriedade **DataSource** sejam automaticamente ajustadas no **DBGRid**. Use a propriedade (Custom) somente para modificar os **Caption** das Colunas, se houver necessidade. Desta forma o **DBGrid** ficará assim:



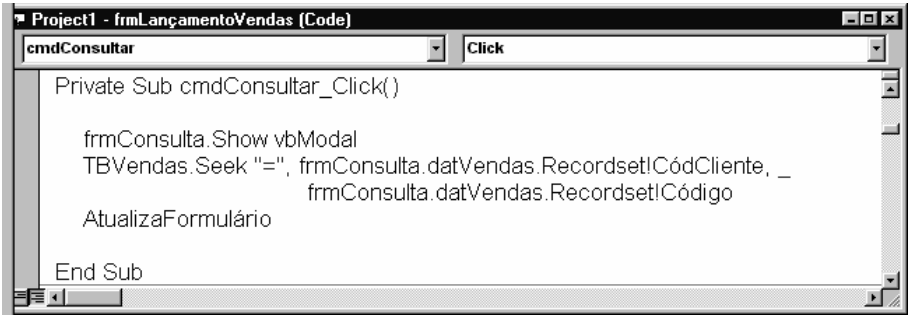
O botão fechar será configurado para esconder este formulário e retornar para o formulário anterior.



Note que usamos o **Hide** no lugar de **Unload,** pois o **Unload** retirna o formulário da tela e da memória também, enquanto que o **hide** mantém o formulário na memória. A vantagem disto é que o acesso a esta janela se torna mais rápido no segundo acesso que for feito a ela e os valores das variáveis do formulário ainda continuará em uso. Neste caso vamos precisar dos valores dos campos que o usuário selecionou para, na tela de Vendas, usá-las.

Este tipo de consulta que foi feito é bastante simples. Logicamente que pode-se usar outros métodos mais sofisticados, como o SQL por exemplo. Mas isto já faz parte dos recursos avançados do Visual Basic.

Volte agora na tela de Vendas para podermos codificar o botão "Consultar"



No método **Show** de frmConsulta, usamos como argumento a constante "vbModal" para indicar que essa janela deve ser do tipo modal, ou seja, exclusiva. Não se tem acesso a outra janela enquanto o usuario não concluir o que deseja fazer na janela aberta.

Usamos o **Seek** para procurar os códigos que foi selecionado na janela frmConsulta.

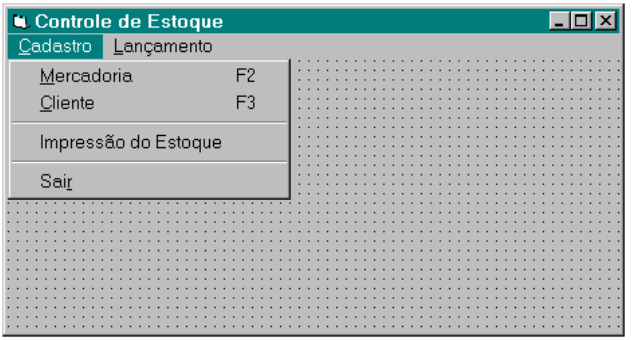
Note que usamos o "_" para fazer uma quebra de linha.

14 IMPRESSÃO

- Printer
- Crystal Reports

14.1 PRINTER

Em nosso programa vamos acrescentar um item no menu principal chamado “impressão do Estoque” e com isto vamos imprimir o conteúdo da tabela mercadoria.



Usaremos um novo objeto para nos chamado **Printer.** E será este objeto que manipulará tudo que enviaremos para a impressora, criando, assim, um canal de comunicação entre nosso programa e a porta em que a impressora esta ligada.

Muitas peculiaridades de uma impressão, como falta de papel, impressora em linha, fila de impressão, etc. Tudo isto será organizado pelo próprio Windows, deixando o programador livre para executar somente uma tarefa: Despachar seu relatório para a impressora.

As propriedades principais do **Printer** são:

ColorMode: Determina ou mostra a capacidade de impressão colorida do dispositivo.

Copies: Especifica a quantidade de cópias de uma página que deve ser impressa.

CurrentX: Determina a coordenada horizontal que a impressora irá imprimir.

CurrentY: Determina a coordenada vertical que a impressora irá imprimir.

DeviceName: Mostra o nome da impressora padrão suportada pelo dispositivo

FontName: Determina qual fonte de letra a impressora usará para impressão.

FontBold: Determina se a fonte será em negrito.

FontItalic: Determina se a fonte será em itálico.

Fonts: Fornece uma lista de todas as fontes disponíveis para impressão.

FontSize: Determina o tamanho que a fonte de letra escolhida usará.

FontUnderline: Determina se a fonte será sublinhada.

Orientation: Determina a orientação do papel: Retrato ou Paisagem. Usamos as constantes para definir o tipo escolhido:

vbPRORPortrait1 Retrato

vbPRORLandscape 2 Paisagem

Page : Retorna o número da página que esta sendo impressa.

PaperSize: Determinamos o tamanho do papel. Podemos usamos as seguintes constantes:

vbPRPSLetter	1	Letter, 8 1/2 x 11 in.
VbPRPSLetterSmall	2	Letter Small, 8 1/2 x 11 in.
VbPRPSTabloid	3	Tabloid, 11 x 17 in.
VbPRPSLedger	4	Ledger, 17 x 11 in.
VbPRPSLegal	5	Legal, 8 1/2 x 14 in.
VbPRPSStatement	6	Statement, 5 1/2 x 8 1/2 in.
VbPRPSExecutive	7	Executive, 7 1/2 x 10 1/2 in.
vbPRPSA3	8	A3, 297 x 420 mm
vbPRPSA4	9	A4, 210 x 297 mm
vbPRPSA4Small	10	A4 Small, 210 x 297 mm
vbPRPSA5	11	A5, 148 x 210 mm
vbPRPSB4	12	B4, 250 x 354 mm
vbPRPSB5	13	B5, 182 x 257 mm
vbPRPSFolio	14	Folio, 8 1/2 x 13 in.
VbPRPSQuarto	15	Quarto, 215 x 275 mm
vbPRPS10x14	16	10 x 14 in.
vbPRPS11x17	17	11 x 17 in.
VbPRPSNote	18	Note, 8 1/2 x 11 in.
vbPRPSEnv9	19	Envelope #9, 3 7/8 x 8 7/8 in.
vbPRPSEnv10	20	Envelope #10, 4 1/8 x 9 1/2 in.
vbPRPSEnv11	21	Envelope #11, 4 1/2 x 10 3/8 in.
vbPRPSEnv12	22	Envelope #12, 4 1/2 x 11 in.
vbPRPSEnv14	23	Envelope #14, 5 x 11 1/2 in.
VbPRPSCSheet	24	C size sheet
vbPRPSDSheet	25	D size sheet
vbPRPSESheet	26	E size sheet
vbPRPSEnvDL	27	Envelope DL, 110 x 220 mm
vbPRPSEnvC3	29	Envelope C3, 324 x 458 mm
vbPRPSEnvC4	30	Envelope C4, 229 x 324 mm
vbPRPSEnvC5	28	Envelope C5, 162 x 229 mm
vbPRPSEnvC6	31	Envelope C6, 114 x 162 mm

vbPRPSEnvC65	32	Envelope C65, 114 x 229 mm
vbPRPSEnvB4	33	Envelope B4, 250 x 353 mm
vbPRPSEnvB5	34	Envelope B5, 176 x 250 mm
vbPRPSEnvB6	35	Envelope B6, 176 x 125 mm
vbPRPSEnvItaly	36	Envelope, 110 x 230 mm
vbPRPSEnvMonarch	37	Envelope Monarch, 3 7/8 x 7 1/2 in.
vbPRPSEnvPersonal	38	Envelope, 3 5/8 x 6 1/2 in.
vbPRPSFanfoldUS	39	U.S. Standard Fanfold, 14 7/8 x 11 in.
vbPRPSFanfoldStdGerman	40	German Standard Fanfold, 8 1/2 x 12 in.
vbPRPSFanfoldLglGerman	41	German Legal Fanfold, 8 1/2 x 13 in.
vbPRPSUser	256	User-defined

Port: Retorna o nome da porta de impressão que será usada pela impressora padrão.

PrintQuality: Determina ou seta a resolução que a impressora irá usar.

VbPRPQDraft	-1	Resolução Draft
vbPRPQLow	-2	Baixa Resolução
vbPRPQMedium	-3	Média Resolução
vbPRPQHigh	-4	Alta Resolução

Os métodos:

EndDoc: Finaliza a impressão de um relatório

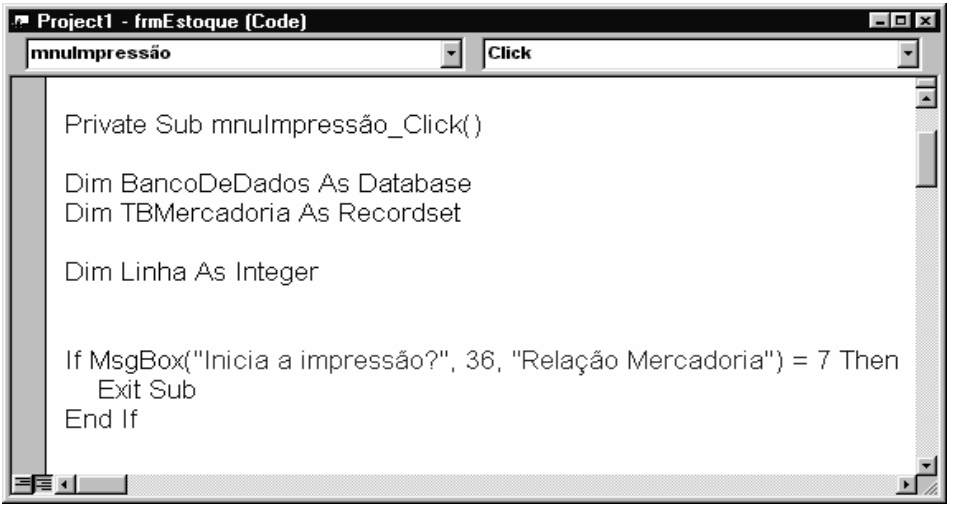
KillDoc: Termina imediatamente a impressão

NewPage: Força a impressão passar para a próxima página.

Print <expressão> : Imprime uma expressão ou variável especificada.

O Objeto Printer não aparece na caixa de ferramentas, nem possui um ícone como outros objetos. Ele é interno que não possui formato visual.

Para desenvolver nossa rotina de impressão, dê um click na opção “Impressão Estoque” no menu e vamos codificar o programa:



Criamos primeiro as variáveis que irão manipular o Banco de Dados e a tabela. Como vamos imprimir o conteúdo da tabela de mercadoria isto se torna necessário.

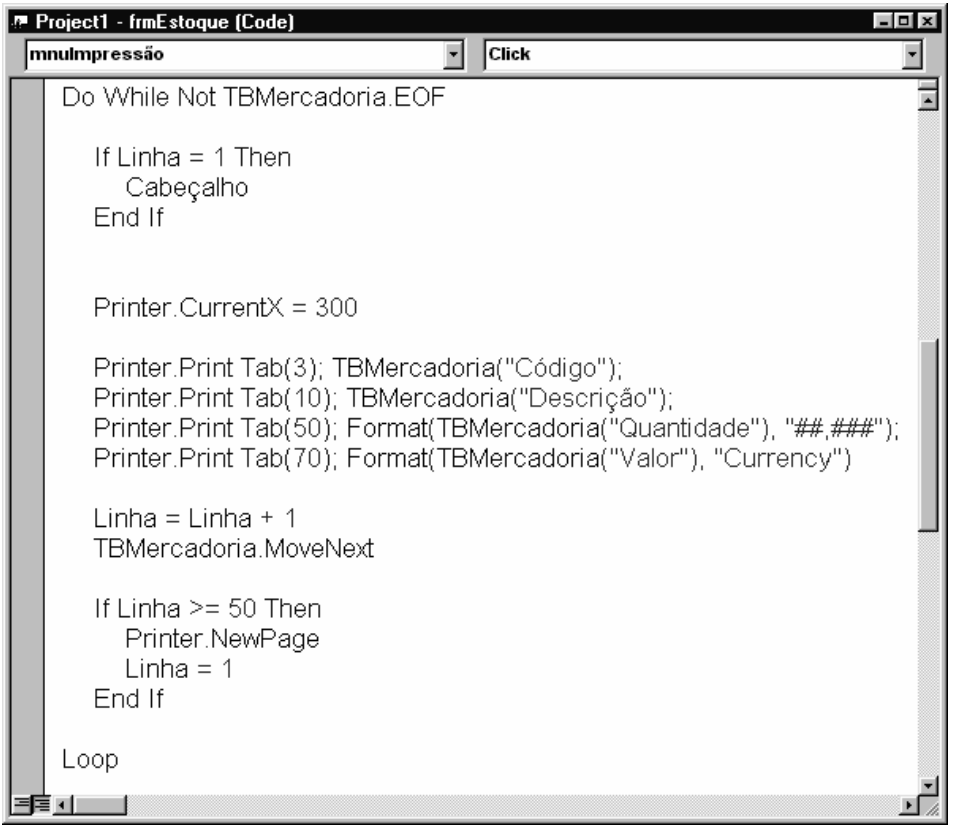
A variável Linha irá fazer um contador de linhas, para podermos controlar quantas linhas esta sendo impressa numa página.

Depois colocamos uma mensagem para o usuário ter certeza que realmente quer imprimir o relatório. Se o MsgBox retornar o número 7 é sinal que o usuário digitou escolheu **não.**

Na seqüência definimos o contador de linha para 1, e abrimos o banco de dados e a tabela de mercadoria. Não é necessário abrir o índice. Se não tivéssemos aberto a ordem em que os registros iriam aparecer no relatório seria a ordem em que foram digitados. Como abrimos um índice que ordena todos os dados na ordem de código, o relatório será impresso então nesta ordem. Caso houvesse necessidade de criar um relatório na ordem alfabética da descrição das mercadorias, então seria necessário criar um índice pelo

campo “Descrição”, ou uma SQL ordenado pela "Descrição", e abri-lo na rotina de impressão.

Usamos o objeto Printer para definir com qual fonte de letra o relatório será impresso e qual o tamanho usado



Abrimos um DO WHILE para sempre verificar se o fim do arquivo foi atingido. Enquanto não for (*Not TBMercadoria.EOF*) executa as rotinas definidas entre o DO WHILE e o LOOP.

Em seguida é verificado se o contador de linha é igual a 1. Se for chama uma função chamada Cabeçalho. Esta função irá imprimir o cabeçalho de nosso relatório.

O conteúdo dos campos da tabela de mercadoria é inserido dentro das variáveis, para depois podermos imprimir no papel estas variáveis. Usamos a função Format para melhorar a formatação numérica.

Usamos a propriedade CurrentX para definir qual será a coordenada horizontal a ser utilizado durante a impressão.

A propriedade Print é quem efetivamente faz a impressão. Em conjunto com o Print usamos a função TAB que efetua uma tabulação para cada variável que será impressa. Isto ajuda a alinhar as expressões impressas.

Note que colocamos um ";" após o nome da variável que será impressa. Usamos isto para o Objeto Printer continuar imprimindo na mesma linha. No ultimo campo a ser impresso ***TBMercadoria("Valor")*** já não é necessário, pois a próxima impressão já será na linha seguinte.

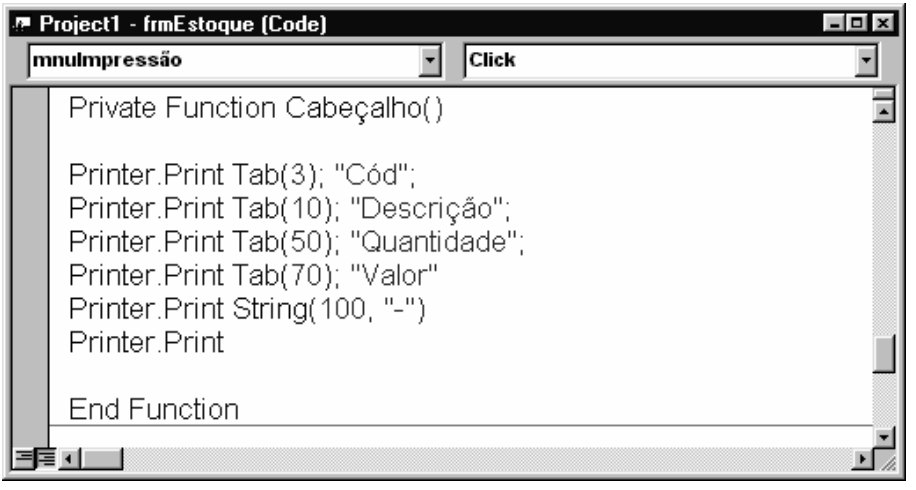
Fazemos depois um incremento de + 1 na variável linha. Isto é para atualizar o contador de linha.

MoveNext posiciona a tabela no próximo registro, pois o atual já foi impresso e agora é necessário ler o registro seguinte. Se o registro seguinte for o fim do arquivo, quando a execução do programa encontrar o LOOP esta rotina será encerrada.

No nosso exemplo definimos que numa página pode conter no máximo 50 linhas. Então quando o contador de linha chegar neste ponto, o Objeto Printer chamará uma nova página (NewPage) e o contador de linha volta a ser igual a 1.

Encerramos nossa rotina com um EndDoc que finaliza a impressão e um Close fechando a tabela e o banco de dados.

Neste programa esta faltando, para encerrar, a função que cria o cabeçalho:



Quando colocamos um print sozinho, sem nenhuma expressão acompanhado, indica que queremos forçar a impressão de uma linha em branco.

```
Private Sub mnuImpressão_Click()

Dim BancoDeDados As Database
Dim TBMercadoria As Recordset

Dim Linha As Integer

If MsgBox("Inicia a impressão?",36,"Relação Mercadoria")=7
Then
    Exit Sub
End If

Linha = 1

Set BancoDeDados = OpenDatabase(App.Path & "\Estoque.MDB")
Set TBMercadoria =
BancoDeDados.OpenRecordset("Mercadoria", dbOpenTable)
TBMercadoria.Index = "IndCódigo"

Printer.FontName = "Arial"
Printer.FontSize = 10

Do While Not TBMercadoria.EOF

    If Linha = 1 Then
        Cabeçalho
    End If


    Printer.CurrentX = 300

    Printer.Print Tab(3); TBMercadoria("Código");
    Printer.Print Tab(10); TBMercadoria("Descrição");
    Printer.Print Tab(50); Format(TBMercadoria
("Quantidade"), "##,###");
    Printer.Print Tab(70); Format(TBMercadoria("Valor"),
"Currency")
```

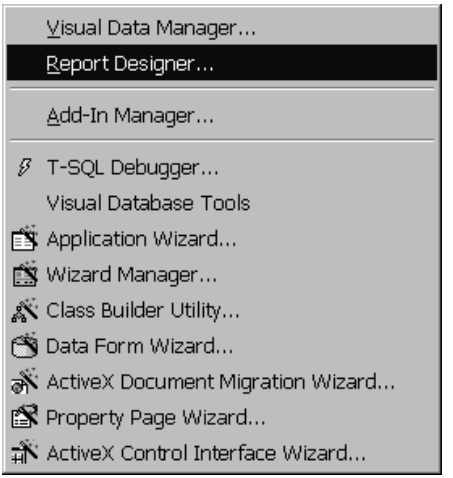
```
    Linha = Linha + 1
    TBMercadoria.MoveNext

    If Linha >= 50 Then
        Printer.NewPage
        Linha = 1
    End If

Loop
Printer.EndDoc
TBMercadoria.Close
BancoDeDados.Close
End Sub

Private Function Cabeçalho()

Printer.Print Tab(3); "Cód";
Printer.Print Tab(10); "Descrição";
Printer.Print Tab(50); "Quantidade";
Printer.Print Tab(70); "Valor"
Printer.Print String(100, "-")
Printer.Print

End Function
```

14.2 CRYSTAL REPORTS

O Crystal Reports consegue dar uma aparecia mais profissional para os relatórios, alem do usuário poder dar um "preview" de seus relatórios antes deles serem impressos.

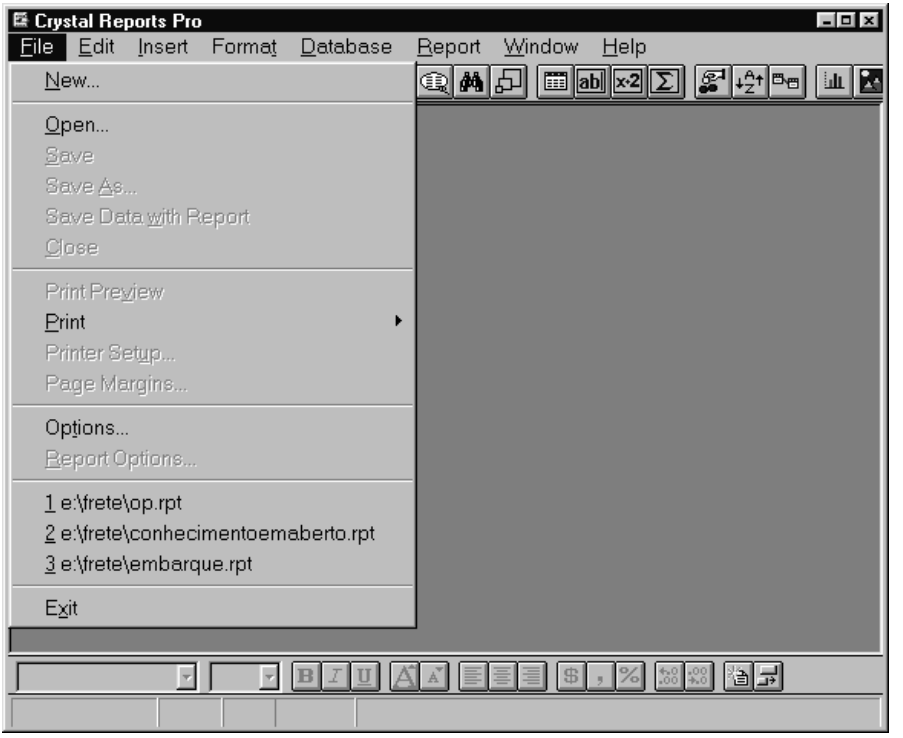
Usar o Crystal é muito simples e fácil. Muito mais fácil que usar o **Printer** para relatórios.

Entre no programa através do menu ***Add-Ins*** e entre na opção ***Report Designer.***

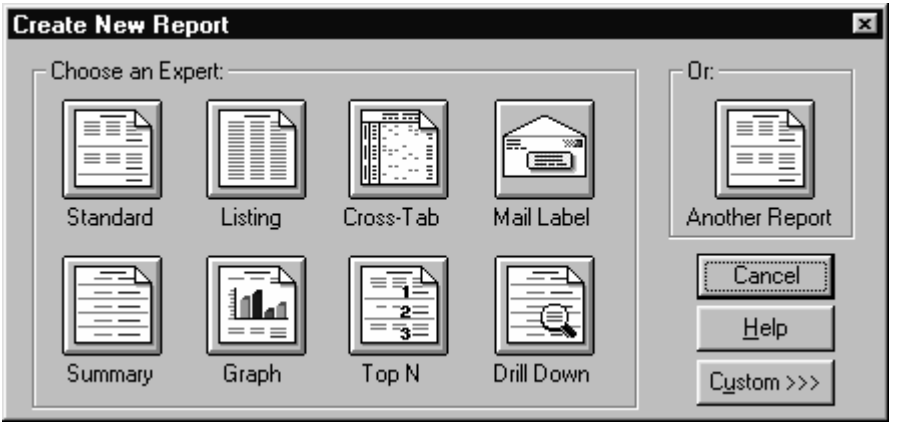


Vamos usar este gerador de relatório para criar um relatório de Mercadoria. Não será necessário codificações complexas, pois o programa automatiza muitas tarefas.

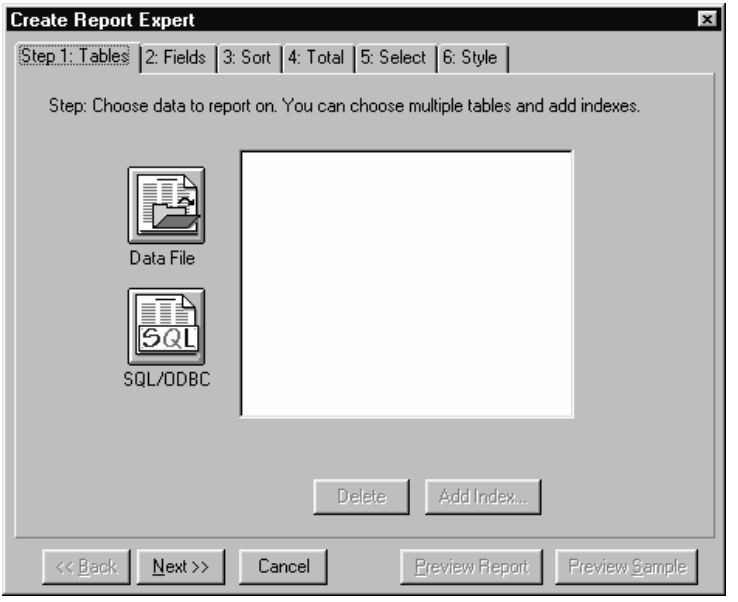
Ao aparecer o Programa Crystal Reports entre na opção NEW.



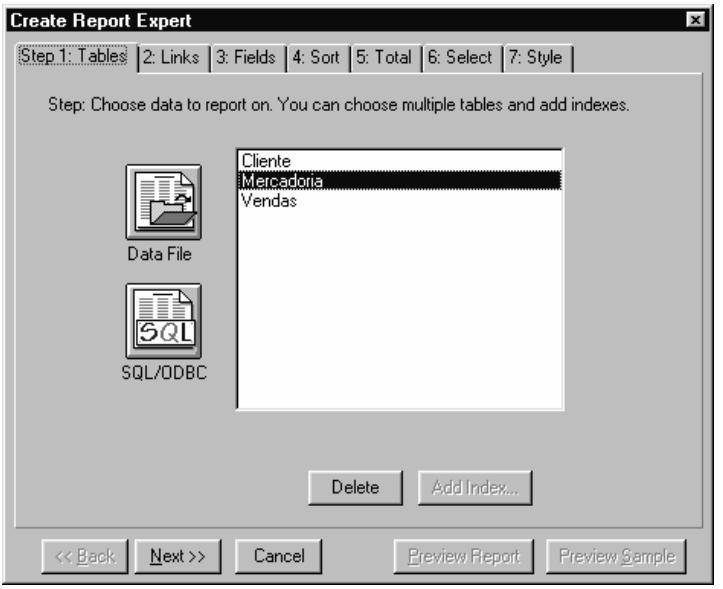
Irá aparecer uma janela onde devemos escolher qual estilo de relatório será criado. Vamos usar o "Standard"



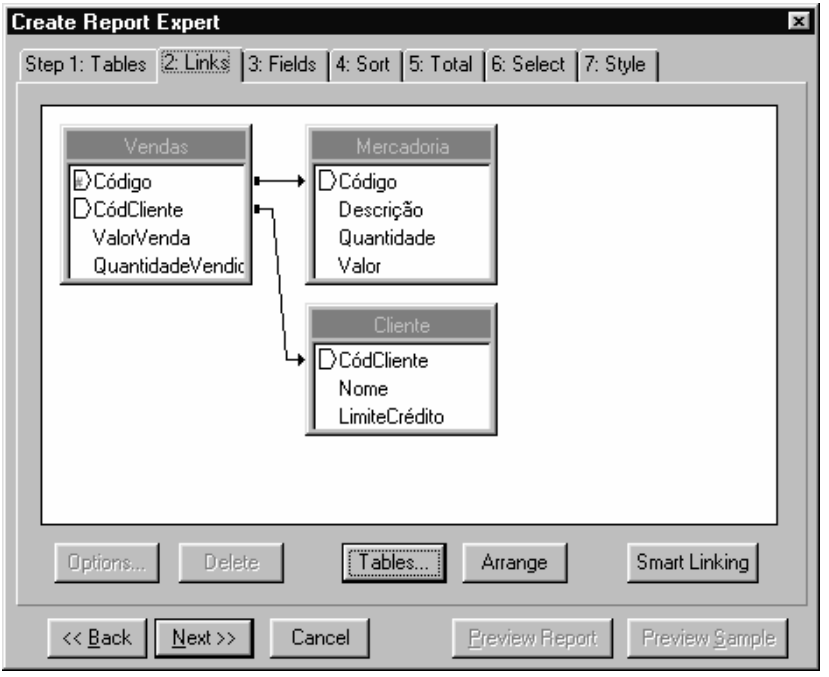
Para que o programa crie nosso relatório devemos passar algumas informações, e essas informações serão coletadas através dessas opções que vem a seguir:



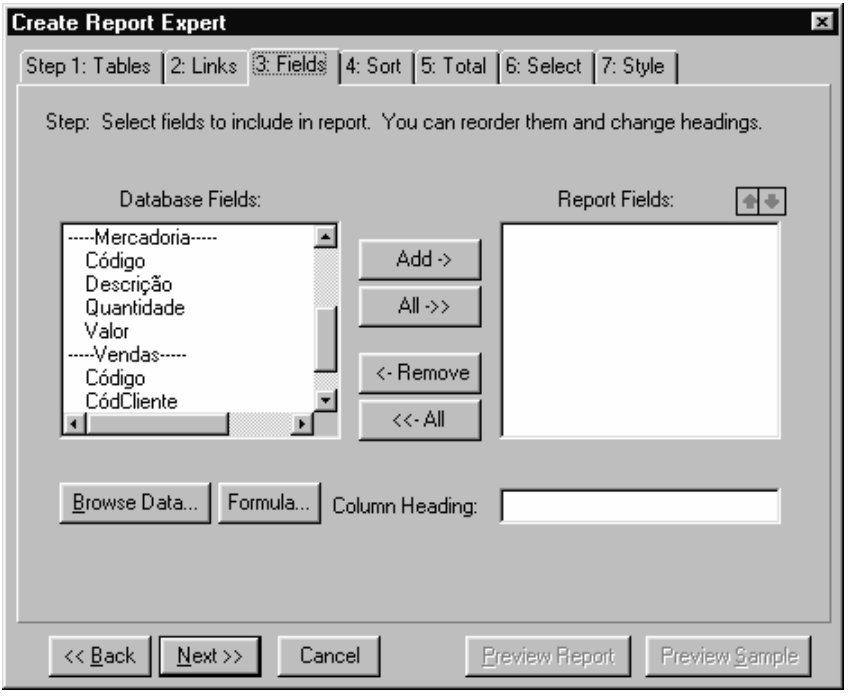
Aperte o botão **Data File** para escolher o banco de dados que sera usado no relatório. Depois de escolhido irá aparecer a relação de todas as tabelas existentes no Banco de Dados. Escolha o Arquivo "Estoque.MDB".



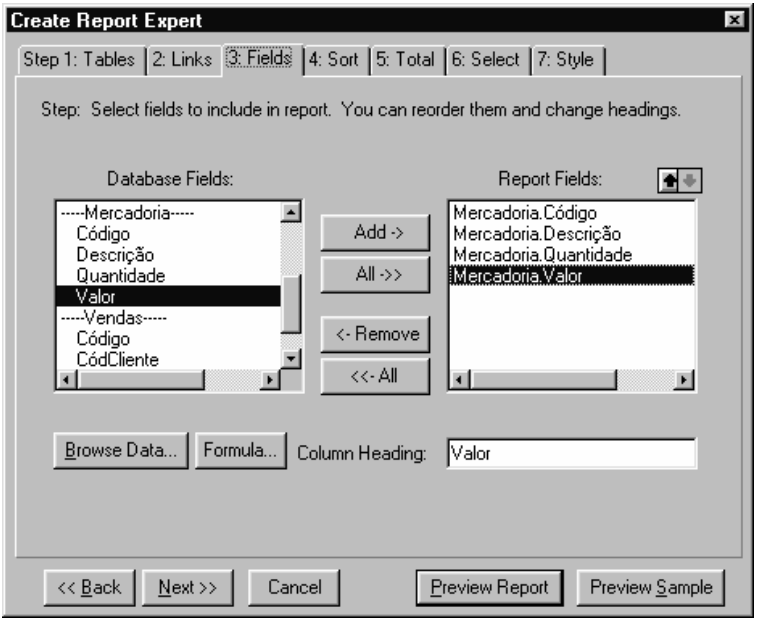
Aperte o botão "Next >>"



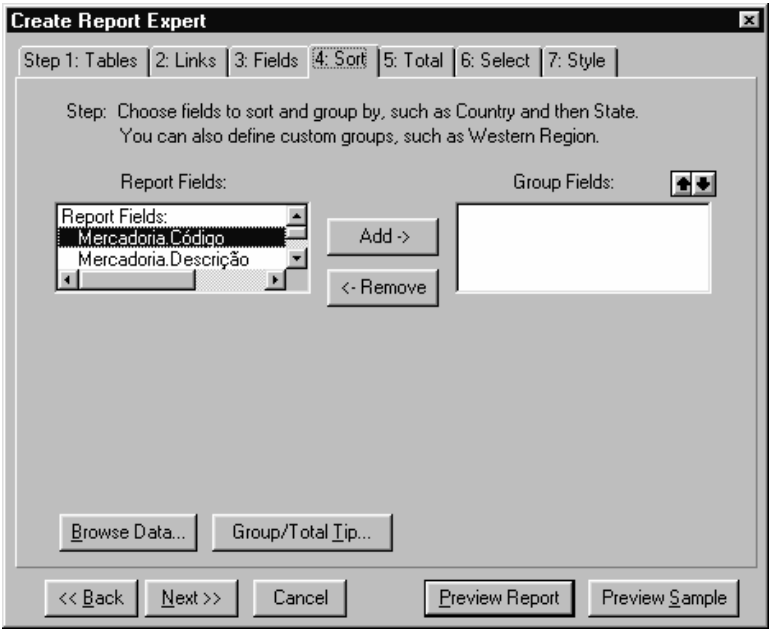
Na próxima tela aparece as tabelas e um gráfico mostrando o relacionamento entre as tabelas.



Nesta tela selecionamos os campos que vão compor o relatório. Escolha um campo na lista que aparece no "**Database Fields**", aperte o botão "**Add**" e dê um título para o campo (que irá aparecer no cabeçalho) no item **Column Heading**.

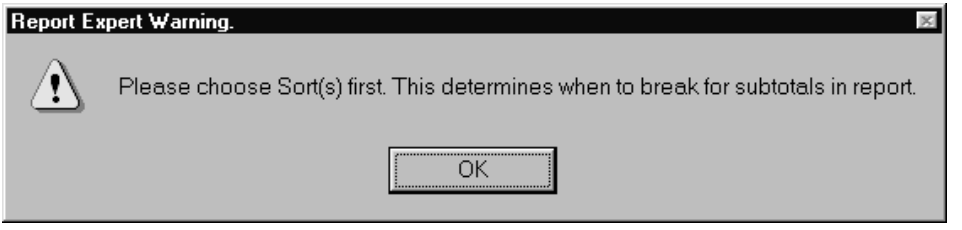


Escolha somente os campos da tabela de Mercadoria.

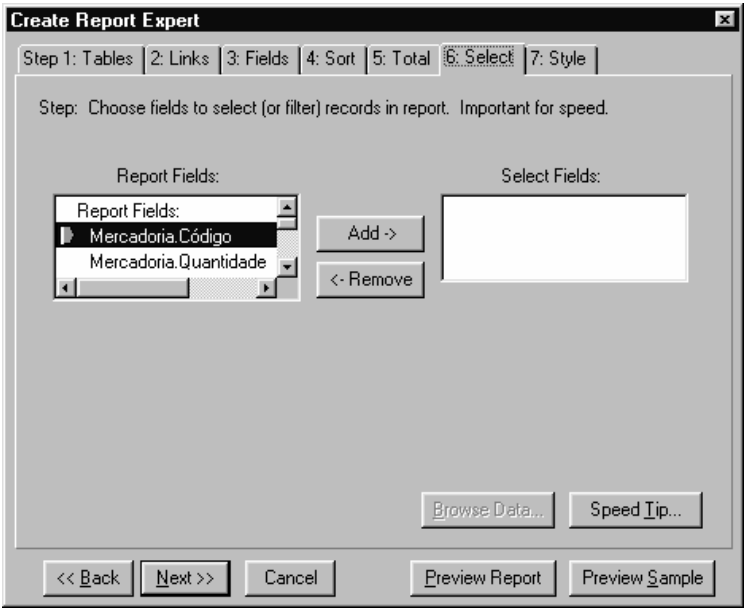


Se quisermos criar grupos ou subgrupos, usamos esta tela. Como por exemplo, um relatório que aparece primeiro o grupo de Material Esportivo e depois o Grupo de Roupa Social. Quando escolhemos um campo que será a chave do grupo, o relatório ordena o relatório primeiro pelo grupo.

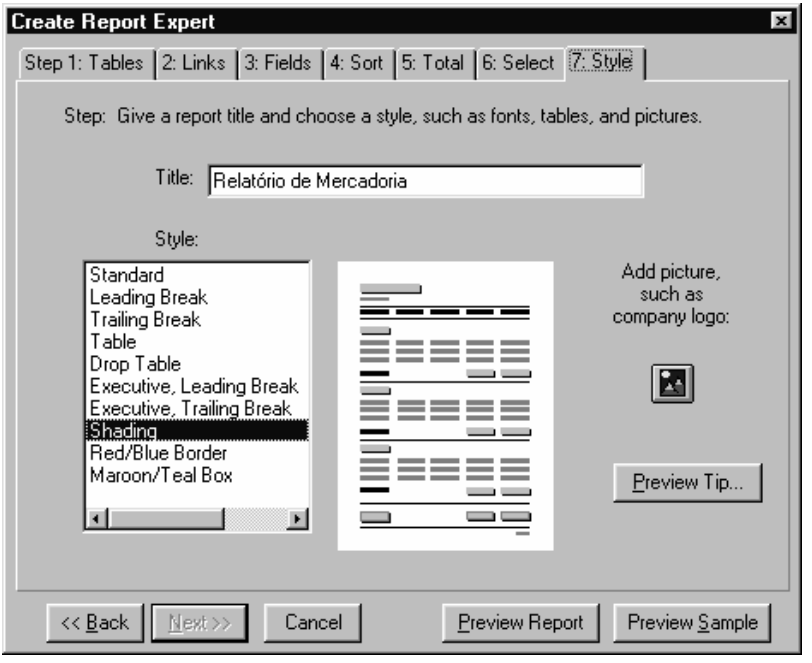
No nosso caso, não existe grupo. Aperte o "Next"



Irá aparecer esta mensagem, alertando que como não foi escolhido nenhum grupo não será possível entrar nesta opção. Aperte o botão OK e vamos direto para a opção "6 - Select"



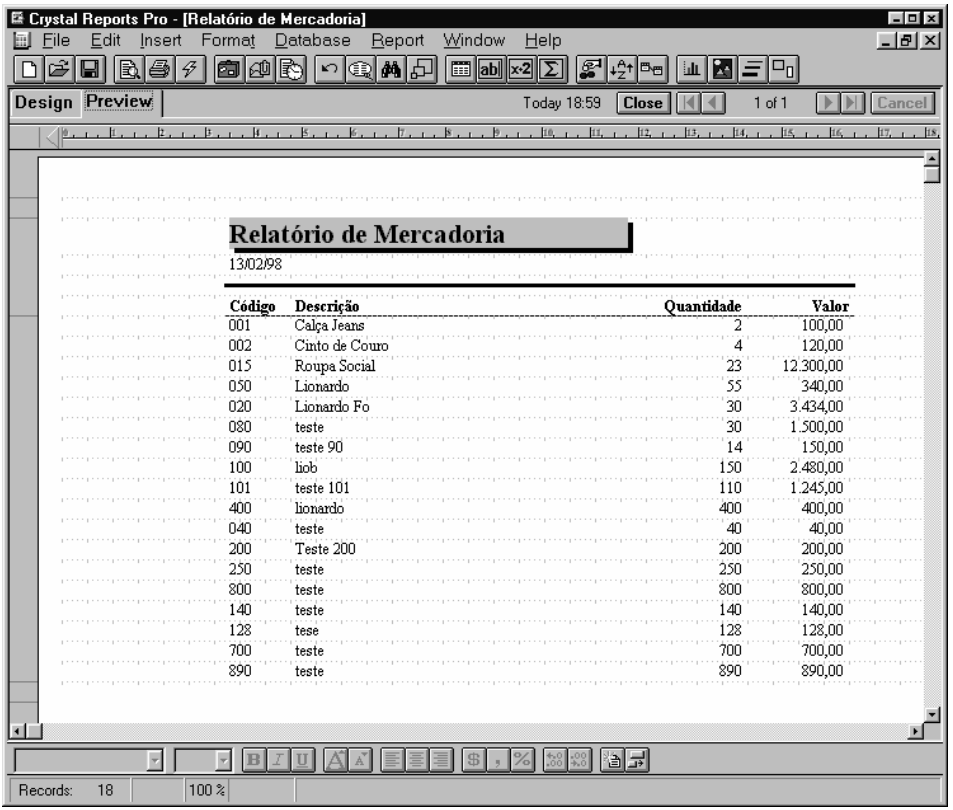
Nesta tela pode-se fazer um filtro dos registros que irão compôr o relatório. Não faremos nenhum tipo de filtro.



Aqui colocamos um título para o relatório e escolhemos um estilo na lista que aparece em "Style". Escolha ***Shading .*** Cada estilo monta o relatório

num formato diferente. Deve-se escolher o que mais se adapta ao tipo que relatório que se esta criando.

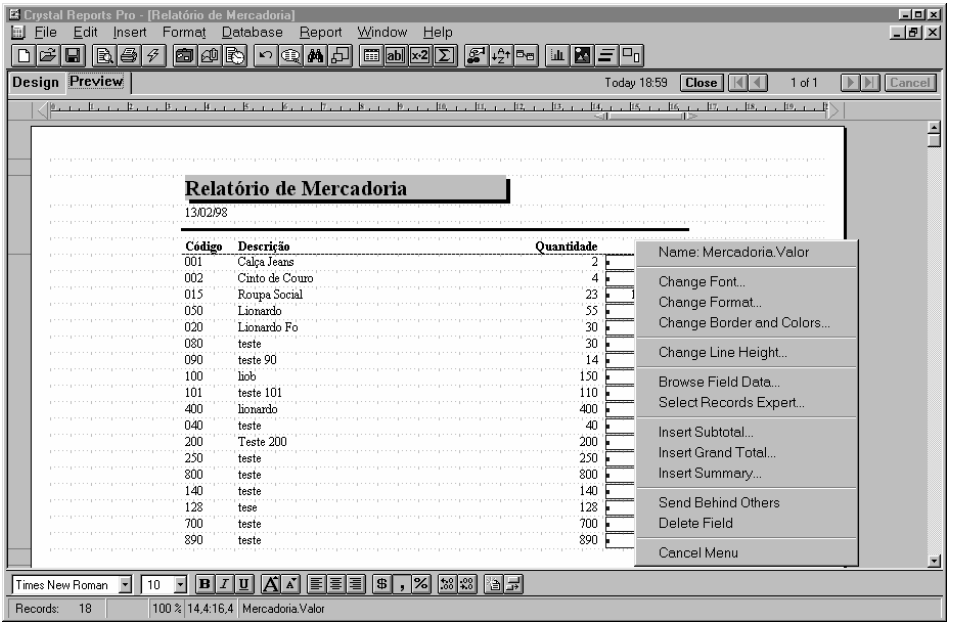
Aperte agora o botão "Preview Report" para ver como ficou.

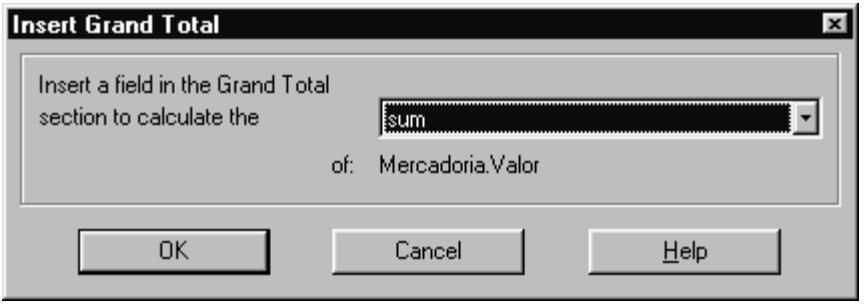


Código	Descrição	Quantidade	Valor
001	Calça Jeans	2	100,00
002	Cinto de Couro	4	120,00
015	Roupa Social	23	12.300,00
050	Lionardo	55	340,00
020	Lionardo Fo	30	3.434,00
080	teste	30	1.500,00
090	teste 90	14	150,00
100	liob	150	2.480,00
101	teste 101	110	1.245,00
400	lionardo	400	400,00
040	teste	40	40,00
200	Teste 200	200	200,00
250	teste	250	250,00
800	teste	800	800,00
140	teste	140	140,00
128	tese	128	128,00
700	teste	700	700,00
890	teste	890	890,00

Vamos agora colocar um total para o "Valor". Dê um click com o mou se em cima do registro e aperte o botão da direita.

Dentre as várias opções que irá aparecer, escolha "Inset Grand Total". Será então inserido um totalizador para o "Valor".



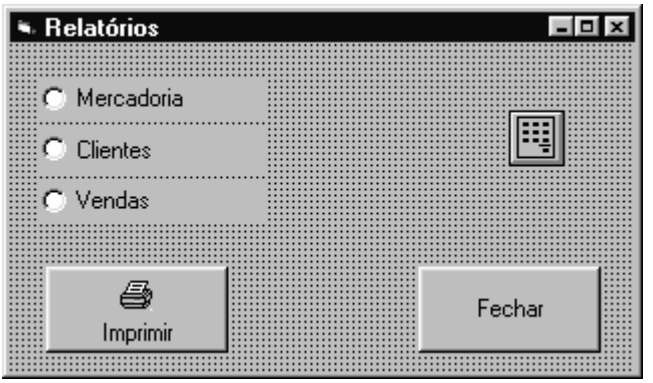


Uma vez feito tudo isto, nosso relatório esta pronto. Vamos agora salvar o arquivo e mandar executar em nosso programa.

Va no menu "**File**", opção "**Save**" e grave este relatório com o nome de "Mercadoria".

Feche o Cristal Reports e volte para o nosso programa.

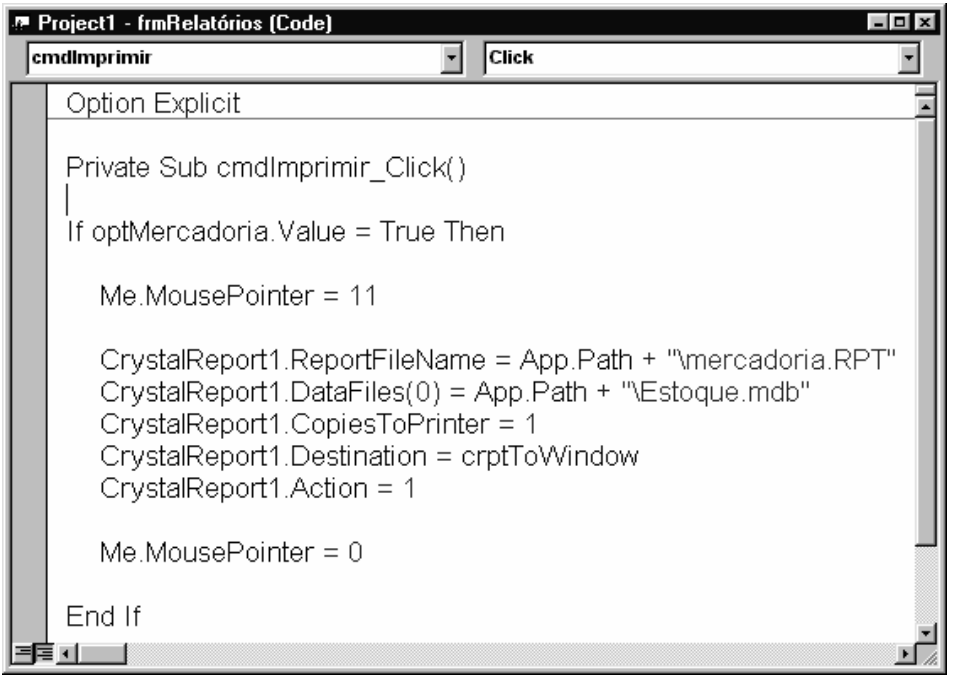
Crie mais um formulário para o projeto, e dê o nome de "frmRelatórios". Abra uma opção no menu principal do programa e acrescente-o nele.



Nesta janela o usuário irá escolher o tipo de relatório que ele deseja e pressiona o botão "Imprimir".

Note que inserimos no formulário um novo controle chamado CrystalReports. Ele quem irá na verdade fazer todo o trabalho "pesado" de gerar o relatório.

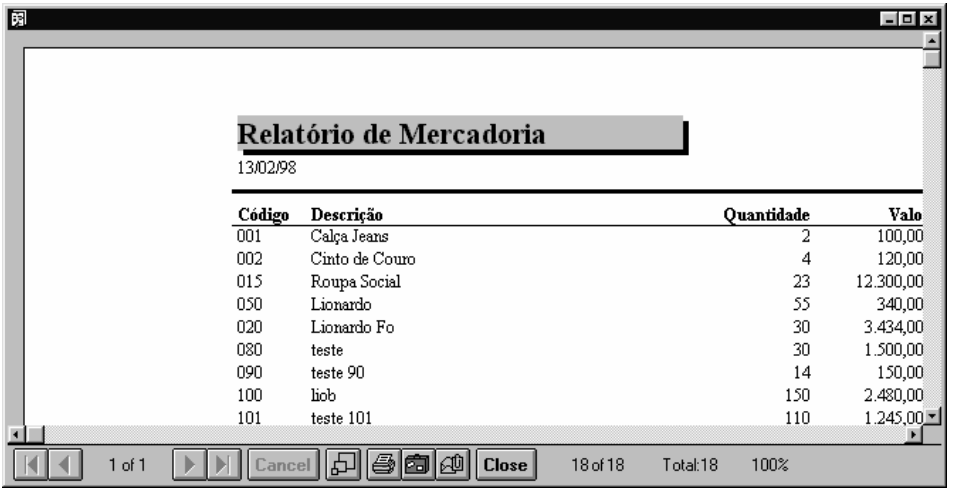
Vamos então codificar o botão imprimir:



Primeiro verificamos se o usuário escolheu o relatório de mercadoria. Caso seja positivo mudamos o ponteiro do mouse para uma ampulheta (para não deixar o usuário fazer nada ate que o relatório seja concluido.

Na propriedade **ReportFileName** coloque o arquivo que criamos. Lá existe a matriz de nosso relatório. Em **DataFiles** especificamos o nome do banco de dados que será lido pelo relatório.

CopiesToPrinter determina quantas copias terá o relatório. Na propriedade **Destination** informamos que o relatório deve ser gerado na tela primeiro antes de ir para a impressora. Por fim, usamos o **Action** que envia todas as informações captadas para o Crystal e cria o relatório.



Quando o relatório é criado e mostrado na tela, usamos os botões que fica na parte inferior para manipular o relatório.

Os botões em forma de seta são utilizados para navegar nas páginas do relatório. O botão em forma de impressora envia o relatório para a porta de impressão.

O relatório de Vendas tem uma particulariadade. Nele deve conter o nome do cliente e o nome da mercadoria comprada, entretanto na tabela de Vendas não existe esses dados, existe somente os códigos relacionados. Como o Crystal Reports faz o relacionamento entre tabelas de forma quase que automaticas, temos que somente inserir os respectivos campos.