Introdução

Quando se pensa em Windows, algumas palavras nos vêm à cabeça, tais como: interface gráfica, objetos, ícones, janelas, mouse, botões, etc., e, como o próprio nome diz, janelas. Aliás, uma verdadeira confusão de janelas: maximizadas, restauradas ou minimizadas; pop-up's, modais ou não-modais; caixas de mensagem ou de diálogo; janelas mães e janelas filhas. Logicamente, então, o ponto de partida para a programação em ambiente Windows só poderia ser uma janela, pois é através dela que o usuário vai interagir com um sistema.

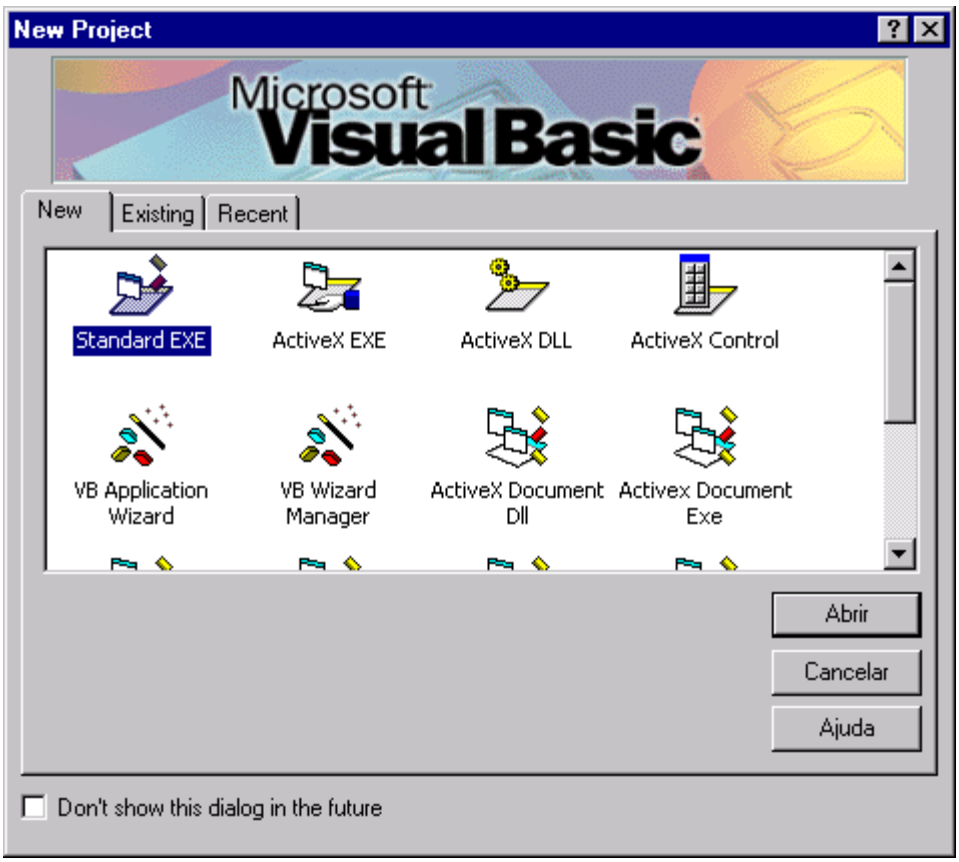
O Microsoft Visual Basic é um pacote para desenvolvimento de aplicações visuais para ambiente Windows baseado na linguagem de programação Basic. É orientado a eventos, o que quer dizer que trata ocorrências que dão início a alguma rotina de trabalho: o programa fica parado até que algo aconteça. Quer dizer também que ele permite o uso de objetos, mas não a sua criação, pois não é uma linguagem orientada a objetos.

Objetos são estruturas que combinam *propriedades* e *métodos*. As propriedades são características dos objetos, que podem ser acessadas e/ou alteradas pelo programador tanto em *tempo de projeto* (quando o projeto está sendo desenvolvido) quanto em *tempo de execução* (quando o aplicativo está sendo executado). Já os métodos são rotinas internas ao objeto que servem para executar determinadas ações. Para exemplificar, pense em uma bicicleta azul. A cor azul é uma característica da bicicleta, ou uma propriedade dela. Já um método seria o ato de pedalar, que é a rotina necessária para fazer a bicicleta andar.

Para programação em VB, usamos uma versão da linguagem Basic estruturada para a construção de procedimentos e funções que podem estar associados aos eventos dos objetos de sua aplicação. O VB também faz uma verificação automática de sintaxe dos comandos, e possui recursos avançados de compilação e rastreamento de erros.

Iniciando o VB

Clique no botão Iniciar do Windows, depois em Programas, e procure uma opção chamada **Microsoft Visual Basic 6.0**. Clique nela, e será exibida durante alguns segundos uma tela de apresentação do VB, enquanto o software está sendo carregado. Depois disso, aparecerá uma janela onde você deve indicar o tipo de projeto que deverá ser iniciado:



- Para iniciar um novo projeto, escolha a opção *Standard EXE* na aba *New*. As demais opções dessa aba iniciam outros tipos de projetos, que não serão abordados nesse curso por tratarem-se de programação avançada, como a criação de controles *ActiveX*;
- A aba *Existing* serve para abrir um projeto já existente;
- Finalmente, a aba *Recent* contém os projetos mais recentemente utilizados, e pode ser usada para abrir um deles de maneira mais rápida que na aba *Existing*.

A tela do VB

Ao se utilizar o VB, tem-se a impressão que estamos "esculpindo" nossa aplicação diretamente no Windows. Isto se deve ao fato de que ele não possui uma janela que ao ser maximizada ocupe toda a área de trabalho. Na verdade, o VB é constituído apenas por uma barra de títulos, uma barra de menus e uma barra de ferramentas (veja a figura na próxima página). Dessa forma, quando construímos os nossos formulários estamos vendo exatamente como eles aparecerão no Windows. As janelas auxiliares, que podem ser a caixa de ferramentas, a janela de propriedades, a janela imediata, etc., aparecem "flutuando" na área de trabalho do Windows. Os principais componentes da interface do VB são os seguintes:

A janela Projeto (*Project*)

No VB, sempre que pensamos em uma aplicação, pensamos em um projeto, que é o conjunto dos arquivos que compõem a aplicação. Esses arquivos podem ser formulários ou módulos. Existem outros arquivos que compõem um projeto, tais como controles customizados (*Custom Controls*), bibliotecas de funções, etc. Entretanto, estes não são parte integrante do arquivo que o VB gera. No VB, cada componente de um projeto está em um arquivo separado, que pode ser visualizado e acessado através da janela Projeto.

Formulários (*Forms*)

No VB, as janelas são chamadas de formulários. Sempre que se começa a desenvolver uma nova aplicação, o VB nos fornece um formulário vazio como ponto de partida, e atribui a ele o nome provisório de Form1. Podemos inserir diversos tipos de controles neste formulário, controles estes que estão representados por botões na caixa de ferramentas.

A caixa de ferramentas (*Toolbox*)

Procure na figura a caixa de ferramentas do VB. Ela geralmente está à esquerda do formulário, e contém os botões correspondentes aos controles que podem ser adicionados a ele.

Controles

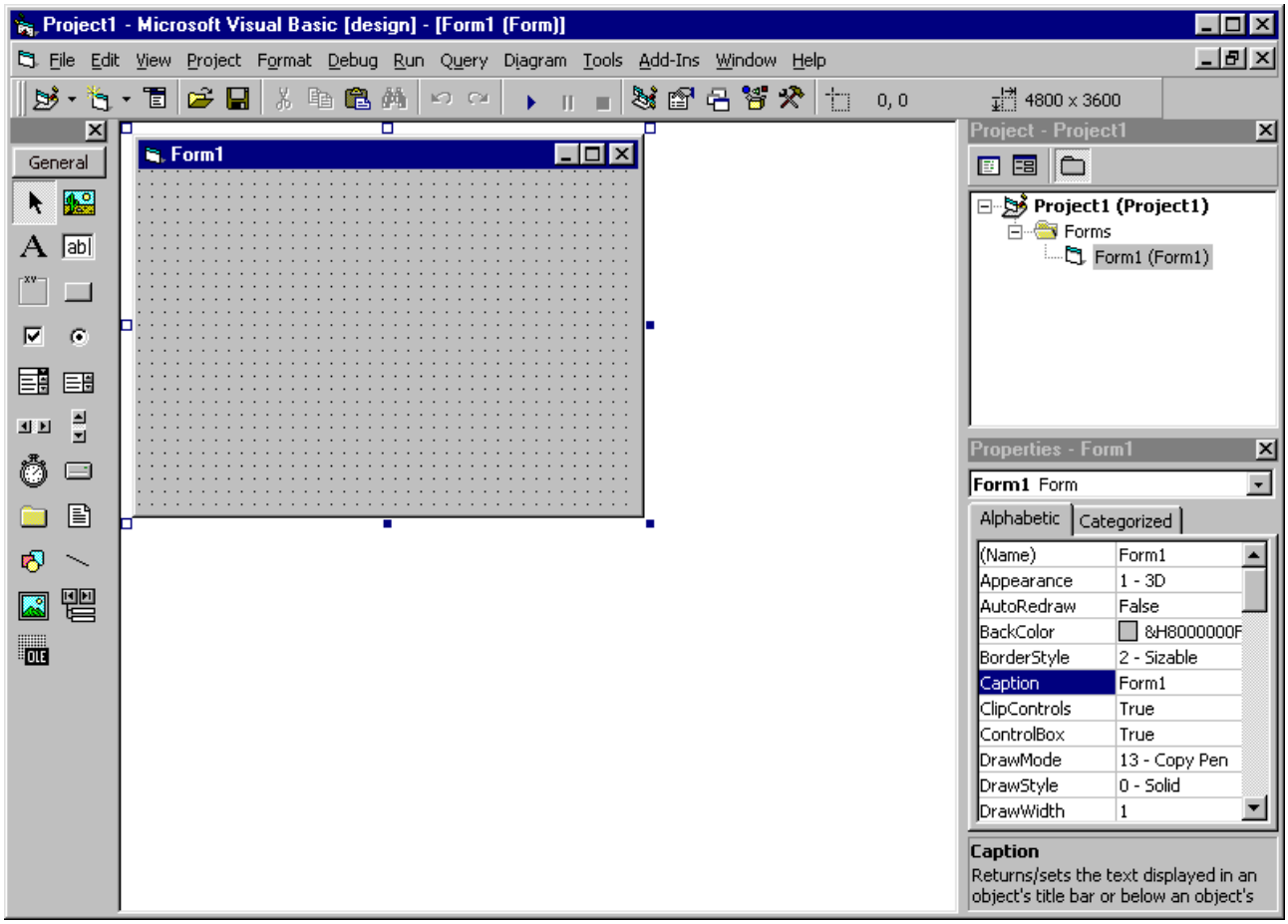
Os controles são arquivos com extensão OCX e constituem a base da programação visual do VB, além de poderem ser utilizados em qualquer outro aplicativo ou linguagem que aceite objetos com esse padrão. Existem dois tipos básicos de controles: internos ou *padronizados* e personalizados ou *customizados*. Basicamente, os controles padronizados fazem parte do "corpo" do VB e são disponibilizados na caixa de ferramentas quando se inicia um novo projeto. Já os customizados podem ou não ser fornecidos junto com o VB, dependendo da versão utilizada, e precisam ser incluídos no projeto pelo programador - quando se inclui um controle customizado em um projeto, aparece um novo botão na caixa de ferramentas. Aliás, qualquer pessoa pode criar um controle OCX (o próprio VB permite isso), existindo inclusive vários deles que são comerciais, desenvolvidos por software-houses especializadas.

A utilização dos dois tipos de controles é rigorosamente a mesma: ambos possuem propriedades e métodos, respondem a eventos, aparecem na caixa de ferramentas, e são manipulados da mesma forma. A diferença é que os customizados não estão disponíveis enquanto não forem explicitamente incorporados à caixa de ferramentas do VB pelo programador. Mas, depois que isso acontece, tudo fica transparente.

A propriedade Name

A propriedade **Name** determina o nome de um objeto. Todos os objetos dentro de um projeto, incluindo formulários e controles, precisam ter um nome, pois é através desse nome que nos referimos a eles quando estamos escrevendo o código. Quando você inicia o VB, o formulário apresentado recebe o nome genérico de Form1; da mesma maneira, os controles recebem nomes genéricos: Command1, Command2, CommandN

para os botões de comando; Text1, Text2, TextN para as caixas de texto, e assim por diante. É recomendável, porém, que os objetos recebam nomes mais descritivos e de fácil memorização e identificação, especialmente aqueles usados com muita freqüência, como caixas de texto e labels. Em casos especiais – barras de botões, timers, barras de status e outros, usados mais raramente – esse cuidado pode ser dispensado.



As janelas do VB

A propriedade Name de um objeto deve sempre iniciar com uma letra, possuir no máximo 40 caracteres, e pode incluir números e o caracter sublinhado "_", mas não pode incluir espaços nem sinais de pontuação.

Existe uma convenção usada para a nomeação dos objetos, normalmente adotada como regra pelos programadores VB, que determina a adoção de um prefixo de duas ou três letras minúsculas identificando o tipo do objeto, seguido do nome propriamente dito. Por exemplo: se você quiser chamar um formulário de Aviso e usar a convenção, deverá determinar para a propriedade Name o valor frmAviso, pois "frm" é o prefixo que identifica um formulário. O uso regular dessa convenção ajuda na documentação de um projeto, facilitando muito sua manutenção. Veja na tabela a seguir os prefixos convencionais utilizados para a propriedade Name dos objetos mais comuns do VB:

Objeto	***Prefixo***	***Exemplo***
Formulário	frm	frmMenu
Caixa de Figura (*PictureBox*)	pic	picFoto
Legenda (*Label*)	lbl	lblResultado
Caixa de Texto (*TextBox*)	txt	txtAviso
Moldura (*Frame*)	fra	fraConceito
Botão de Comando (*CommandButton*)	cmd	cmdApagar
Caixa de Verificação (*CheckBox*)	chk	chkNegrito
Botão de Opção (*OptionButton*)	opt	optSexo

Objeto	***Prefixo***	***Exemplo***
Caixa de Combinação (*ComboBox*)	cbo	cboCidades
Caixa de Listagem (*ListBox*)	lst	lstClientes
Barra de Rolagem Horizontal (*HScrollBar*)	hsb	hsbIdade
Barra de Rolagem Vertical (*VScrollBar*)	vsb	vsbTaxa
Temporizador (*Timer*)	tmr	tmrAlarme
Caixa de Listagem de Unidades (*DriveListBox*)	drv	drvUnidades
Caixa de Listagem de Diretórios (*DirListBox*)	dir	dirDestino
Caixa de Listagem de Arquivos (*FileListBox*)	fil	filArquivos
Forma (*Shape*)	shp	shpQuadrado
Linha (*Line*)	lin	linDivisão
Caixa de Imagem (*Image*)	img	imgCarro
Controle de Dados (*DataControl*)	dat	datBancos
Grid	grd	grdConsulta
Menu	mnu	mnuEditar
Banco de Dados (*DataBase*)	db	dbContabilidade
Conjunto de Registros (*RecordSet*)	rs	rsDespesas
Conexão com provedor de dados (*Connection*)	cnn	cnnDados

Lição 1: Primeiros Passos

Nessa lição vamos criar nossa primeira aplicação no VB. Precisaremos seguir os seguintes passos:

1. **Criação da interface com o usuário**: para isso, você vai usar os controles da caixa de ferramentas, que serão inseridos nos formulários de modo à "desenhar" a interface;
2. **Definição das propriedades dos objetos**: após a inserção dos controles, você deve definir as propriedades com as quais eles devem ser iniciados quando o programa for executado;
3. **Codificação dos eventos**: escrever a programação Basic necessária para que o programa responda corretamente aos eventos que vão ocorrer durante sua execução.

Criação da interface com o usuário

Vamos então para a primeira etapa do processo de criação do nosso programa: criar a interface com o usuário. Para isso você vai precisar dos seguintes controles:

Apontador (*Pointer*): esse não é bem um controle, e sim uma ferramenta usada para inserir, movimentar e/ou dimensionar os diversos objetos de uma aplicação.

Botão de comando (*CommandButton*): sua função principal é executar comandos quando clicado. É, provavelmente, o controle mais usado em programação visual.

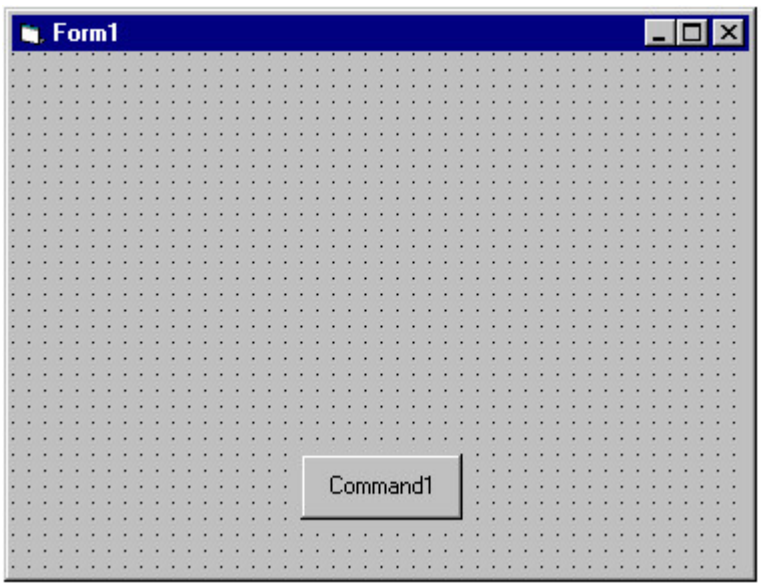
Botão de opção (*OptionButton*): usado para determinar se uma informação é verdadeira ou falsa, somente um botão pode estar ligado, num dado momento, para cada conjunto desses controles.

Legenda (*Label*): serve para exibir um texto que não pode ser editado pelo usuário, como uma mensagem ou o rótulo de um campo a ser digitado.

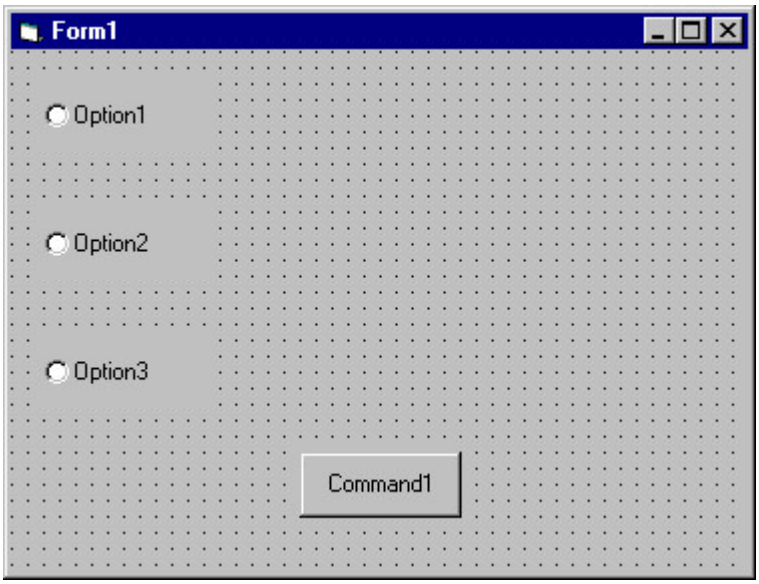
Caixa de figura (*PictureBox*): permite a exibição de figuras e o agrupamento controles, tais como botões de comando, em formulários do tipo MDI (que serão estudados mais adiante).

Bem, mãos à obra:

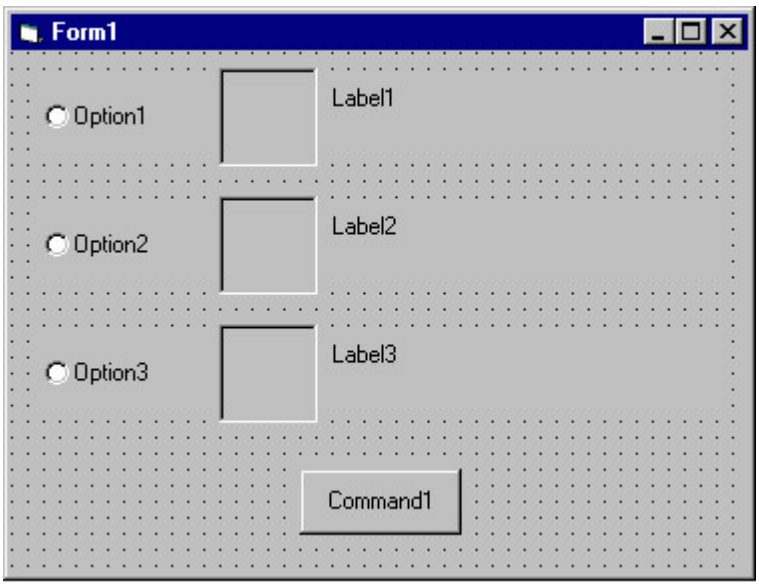
1. Inicie o Windows e o VB, se ainda não o fez. Na janela *New Project*, escolha a opção *Standard EXE* da aba *New*: será exibida a tela do VB com um formulário em branco;
2. Insira um botão de comando nesse formulário. Para isso, você pode agir de duas maneiras: dar um click no ícone *CommandButtom* da caixa de ferramentas e arrastar o mouse sobre o formulário de modo a definir o tamanho desejado para o botão, ou então dar um duplo click no ícone (nesse caso o botão será inserido no centro do formulário com um tamanho padronizado):



3. Usando o mesmo procedimento, insira três botões de opção (*OptionButtom*) e posicione-os um abaixo do outro do lado esquerdo do formulário, de modo a ficar parecido com o exemplo a seguir:



4. Ao lado dos botões, insira três caixas de figura (*PictureBox*). Desenhe-as como quadrados de aproximadamente 1 centímetro de lado;
5. Insira agora três *Labels* à direita das caixas de figura, indo até perto da borda direita do formulário. Seu formulário deve estar parecido com esse:



6. No menu *File*, selecione *Save Project*. Como é a primeira vez que o projeto será gravado, aparece a caixa de diálogo *Save File As*, onde devemos informar o nome que será dado ao **formulário**;
7. Digite "Hello" e selecione *Save*. Seu formulário foi gravado em um arquivo chamado **Hello.FRM**;
8. Aparece então a caixa de diálogo *Save Project As*, onde devemos informar o nome que será dado ao **projeto**. Digite "Hello" novamente e selecione *Save*. Seu projeto foi gravado no arquivo **Hello.VBP**;

Meus parabéns! Você terminou a criação de sua primeira interface com o usuário no VB. Vamos agora passar à segunda fase do nosso programa, que é a definição de propriedades para os objetos do formulário.

Definição das propriedades dos objetos

Antes de começar, verifique se a janela Propriedades (*Properties*) está aberta. Caso não esteja, você pode abrí-la pressionando F4 ou selecionando a opção *Properties* do menu *View*. A definição de propriedades é um processo muito simples, basta seguir os seguintes passos:

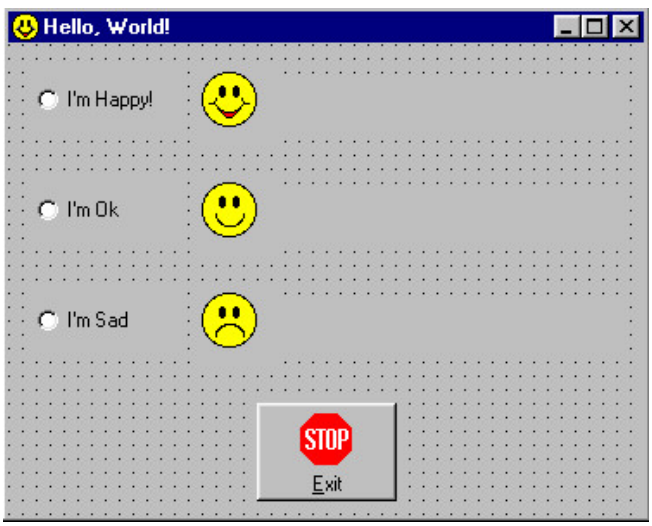
1. Dê um click no objeto cujas propriedades você quer mudar (você também pode clicar com o botão direito sobre ele, e escolher a opção *Properties* do menu de contexto);
2. Clique na propriedade que deseja alterar;

3. Digite o novo valor para a propriedade. Algumas propriedades possuem uma caixa de combinação em que você pode indicar sua opção. Outras abrem uma caixa de diálogo, como a propriedade Picture de uma caixa de figura – quando isso acontecer, será indicado por um botão com reticências "..." ao lado do valor da propriedade.

Seguindo os passos acima, altere as propriedades dos objetos do nosso formulário, de acordo com a tabela:

Objeto	***Nome (propriedade Name)***	***Outras Propriedades = Valor***
Formulário	frmHello	Caption = Hello, World! Icon = FACE05.ICO (obs.: os ícones do VB normalmente são encontrados na pasta C:\Arquivos de Programas\Microsoft Visual Studio\Commom\Graphics\Icons, e o arquivo citado está na subpasta Misc)
Botão de opção	optHappy	Caption = I'm Happy!
Botão de opção	optOk	Caption = I'm Ok
Botão de opção	optSad	Caption = I'm Sad
Caixa de figura	picHappy	Picture = FACE03.ICO Visible = False Appearance = 0 – Flat BorderStyle = 0 – None
Caixa de figura	picOk	Picture = FACE02.ICO Visible = False Appearance = 0 – Flat BorderStyle = 0 – None
Caixa de figura	picSad	Picture = FACE04.ICO Visible = False Appearance = 0 – Flat BorderStyle = 0 – None
Label	lblHappy	Caption = "" (em branco) Font = Times New Roman, Negrito, tamanho 12
Label	lblOk	Caption = "" Font = Times New Roman, Negrito, tamanho 12
Label	lblSad	Caption = "" Font = Times New Roman, Negrito, tamanho 12
Botão de comando	cmdExit	Caption = &Exit Style = 1 – Graphical Picture = TRFFC14.ICO (procure na subpasta Traffic do diretório de ícones do VB)

Note que usamos o caracter "&" na definição da propriedade Caption do botão cmdExit. Fazemos isso quando queremos habilitar o acionamento de um controle pela tecla ALT. No formulário, o rótulo do botão é "Exit" – com a letra E sublinhada, indicando que podemos acioná-lo através da combinação de teclas ALT+E. Verifique se você fez tudo corretamente, comparando seu formulário com o exemplo abaixo:



Bem, vamos conhecer melhor as propriedades que usamos nesses controles, e algumas outras interessantes:

Propriedade Caption

Uma boa tradução para Caption é rótulo: para um formulário, Caption determina o texto que será exibido na barra de título, e também abaixo do ícone do aplicativo, se ele for minimizado. Para um controle, determina o texto exibido dentro ou próximo a ele. Quando um objeto é criado o conteúdo de Caption é igual ao nome padrão atribuído pelo VB ao controle.

Propriedade Picture

A propriedade Picture determina o nome de um arquivo gráfico para ser exibido pelo controle. Como você já viu, podemos escolher o arquivo gráfico desejado em uma janela de diálogo que é aberta quando clicamos no botão com "..." (reticências) exibido por Picture. Os formatos gráficos suportados pelo VB 4 são: ícones (arquivos de extensão ICO), Windows Metafile (extensão WMF – o formato dos cliparts do Word), Bitmap (extensão BMP – o formato dos arquivos criados pelo Paint do Windows), e arquivos JPG e GIF (muito usados em páginas da Internet). No caso do botão de comando, Picture só tem efeito se a propriedade Style for igual a 1 – Graphical.

Propriedade Style

Essa propriedade indica o estilo da aparência do controle. No caso dos botões de comando, os valores possíveis são **0 – Standard** e **1 – Graphical**. Na prática, a diferença é que, alterando o estilo do *CommandButton* para Graphical, será permitida a exibição de ícones, enquanto que em Standard somente texto poderá ser exibido por ele. Vários controles possuem a propriedade Style, sendo que os valores possíveis variam de um tipo para outro.

Propriedade Visible

Como o próprio nome indica, essa propriedade define se um controle deve ou não estar visível em um determinado momento. Ela só pode assumir os valores True (verdadeiro) ou False (falso);

Propriedade Enabled

Não usamos essa propriedade em nosso projeto, mas vamos citá-la agora para evitar confusões com Visible. Você já reparou quando abre uma janela qualquer do Windows e aparece um botão ou uma opção de menu cujo rótulo está com uma cor mais "apagada", indicando que aquele controle não está acessível no momento? A propriedade Enabled é quem faz isso: ela habilita (quando True) ou não (quando False) o acesso ao controle em um determinado momento. Um exemplo: você cria um botão "Ok" para ser utilizado quando forem fornecidos alguns valores necessários para se realizar um cálculo. Você pode usar Enabled para desabilitar o botão Ok até que o usuário tenha fornecido esses valores, pois só depois disso ele poderá usá-lo.

Propriedade Appearance

A propriedade Appearance define a aparência do controle no formulário. Ela só pode ser modificada em tempo de projeto, e normalmente só pode assumir dois valores: 0 – Flat (o controle terá a mesma aparência do formulário, não haverá destaque) ou 1 – 3D (o controle será exibido em perspectiva 3D).

Propriedade BorderStyle

Essa propriedade normalmente é aplicada ao formulário, e, como o próprio nome indica, define o estilo de sua borda. Existem seis tipos de bordas, sendo que alguns controles também possuem a propriedade BorderStyle, mas não todos os tipos. Os tipos de bordas são os seguintes:

- **0 – None (Nenhuma)**;
- **1 - Fixed Single (Fixa Simples)**: indica que o formulário não será dimensionável, ou seja, o usuário não poderá alterar o tamanho no formulário em tempo de execução.
- **2 – Sizable (Dimensionável)**;
- **3 - Fixed Dialog (Diálogo Fixo)**: o formulário não possuirá os botões Maximizar e Minimizar, e, além disso, não será dimensionável. É o estilo da borda das caixas de diálogo do Windows;

- **4 - Fixed ToolWindow (Ferramenta Fixa)**: esse tipo não possui o menu de controle e os botões Maximizar e Minimizar. Não é dimensionável, e mostra somente o botão Fechar e a barra de título com a fonte reduzida. Além disso, o formulário não aparece na barra de tarefa do Windows.
- **5 - Sizable ToolWindow (Ferramenta Dimensionável)**: semelhante a Fixed ToolWindow, só que dimensionável.

Propriedade StartUpPosition

Aplicada ao formulário, essa propriedade determina sua posição na tela do computador quando for acionado. Os valores possíveis são:

- **0 – Manual**: a posição inicial será a mesma que o formulário ocupa em tempo de projeto;
- **1 – Center Owner**: a posição inicial será o centro do formulário principal da aplicação, quando temos mais de um formulário no projeto;
- **2 – Center Screen**: a posição inicial será o centro da tela do computador;
- **3 – Windows Default**: o Windows determinará a posição inicial do formulário.

Propriedade Font

A propriedade Font permite a formatação da fonte usada nas propriedades Text e Caption. Quando você clica em Font, é aberta a janela de formatação de fontes do Windows, de modo que você poderá aplicar fonte, tamanho e efeitos ao texto das propriedades citadas. Essa propriedade não pode ser alterada em tempo de execução, mas para isso o VB fornece algumas propriedades relacionadas a Font, que são as seguintes:

- **FontBold (Negrito)**, **FontItalic (Itálico), FontStrikethru (Riscado) e FontUnderline (Sublinhado)**: podem assumir os valores True (liga o efeito) ou False (desliga o efeito);
- **FontName (Nome da Fonte)**: indica o nome da fonte a ser aplicada ao controle;
- **FontSize (Tamanho da Fonte)**: recebe um valor numérico referente ao tamanho a ser aplicado em uma fonte True Type.

Obs.: com exceção de Font, essas propriedades só podem ser alteradas em tempo de execução.

Codificação dos eventos

Essa é a etapa na qual vamos agora fazer nosso programa funcionar. Como já foi dito anteriormente, o Visual Basic é uma linguagem orientada a eventos, o que quer dizer que os objetos respondem a eventos que contém a codificação necessária. Cada evento contém uma subrotina (ou *procedure*) associada a ele. Ou seja, a codificação será feita em procedures de acordo com a ação que o programa deverá executar quando um determinado evento ocorrer.

Antes de escrever uma procedure, você precisa dizer ao VB para qual objeto deseja escrevê-la. A maneira mais simples de se fazer isso é dando um duplo click sobre o objeto desejado, e a janela Código (*Code*) será apresentada. Na janela Código temos três destaques:

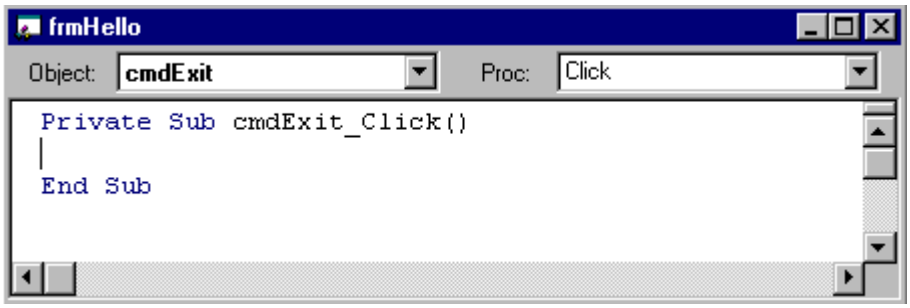
- Quando aberta, a janela Código apresenta a procedure correspondente ao evento padrão do objeto. Por exemplo: o evento padrão de um botão de comando é o Click, portanto quando abrirmos a janela Código de um botão de comando o evento que aparecerá será o Click. Porém, no caso do evento padrão não conter nenhuma codificação, mas outro qualquer sim, então a janela será aberta nesse outro evento;
- Na parte superior da janela temos duas caixas de combinação. A da esquerda, chamada **Object**, contém os nomes de todos os objetos do nosso formulário, e da direita, chamada **Proc**, uma lista com todas as procedures associados ao objeto atualmente selecionado;
- Existe uma opção na caixa Object chamada **General,** que a princípio só contém uma procedure, chamada **Declarations**. Esse não é um objeto, e sim uma **Seção**: uma parte do programa onde são escritas instruções especiais, como declaração de variáveis e funções de uso geral.

Quando estamos programando em VB, é freqüente a necessidade de se modificar o valor das propriedades dos objetos em tempo de execução. Para fazer isso, o VB refere-se às propriedades de um objeto da seguinte maneira:

```
Objeto.Propriedade = <valor>
```

Mas vamos fazer a codificação do nosso projeto. Comecemos pelo botão cmdExit: para escrever o código necessário ao seu funcionamento, siga os seguintes passos:

1. Dê um duplo click sobre o botão cmdExit: a janela Código se abre no evento Click:



2. Escreva o seguinte código:

```
Private Sub cmdExit_Click()
      End
End Sub
```

A primeira linha da procedure indica seu nome – cmdExit_Click – e sua classe – Private (privada). A última linha indica o fim da procedure, através do comando End Sub. Entre o nome da procedure e o comando End Sub é que escrevemos a codificação necessária, que nesse caso resumiu-se ao comando End (que nome sugestivo, não?), cuja função é encerrar a execução do programa.

Vamos agora testar nosso programa. Para isso, basta clicar no botão *Start* da barra de ferramentas ou então pressionar a tecla F5. Teste o funcionamento de seu aplicativo: se tudo foi feito corretamente, quando você clicar no botão "Exit" o programa será encerrado e você retornará ao VB.

Uma observação: quando estamos criando um formulário ou escrevendo código dizemos que estamos **em tempo de projeto**; quando estamos executando um programa dizemos que estamos **em tempo de execução**. Essa diferenciação é importante, pois existem algumas propriedades que não podem ter seu valor alterado em tempo de execução.

Planejando a codificação

Planejamento é a palavra chave para nós, programadores, quando chegamos na fase de escrever a programação do aplicativo. Antes de começar a codificação de um programa você precisa saber o que ele deve fazer e quando: identificar quais os eventos que podem ser acionados pelo usuário e como eles vão influenciar a execução do aplicativo, para poder assim programá-los de maneira a chegar ao resultado desejado. Vamos então planejar nosso primeiro aplicativo:

- Temos em nosso formulário três botões de opção, que servirão para indicar nosso "humor matinal";
- Temos também três caixas de figura, inicialmente invisíveis, cada uma delas contendo uma "carinha" diferente, e três *Labels* que não contém texto nenhum;
- Quando clicarmos em um botão qualquer, a figura e o texto correspondente a ele deverão aparecer, e a figura e o texto dos outros botões não deverão mais ser exibidos;

Você já deve ter percebido que o evento que vamos trabalhar é o Click dos botões de opção, pois é ele que vai gerar as ações necessárias para cumprirmos nosso planejamento. Agora que já sabemos o que fazer, vamos escrever a codificação para isso:

1. Dê um duplo-click no botão de opção optHappy. A janela Código será aberta e a procedure optHappy_Click exibida;
2. Digite os comandos indicado:

```
Private Sub optHappy_Click()
    picHappy.Visible = True
    picOk.Visible = False
    picSad.Visible = False
    lblHappy.Caption = "I'm going to Disneyworld!"
    lblOk.Caption = Empty
    lblSad.Caption = Empty
End Sub
```

Obs.: **Empty** é uma função do VB que retorna uma string vazia (nula).

3. Dê um click na lista Object e escolha o objeto optOk. Se a procedure exibida pela janela Código não for a optOk_Click, selecione-a na lista Proc;

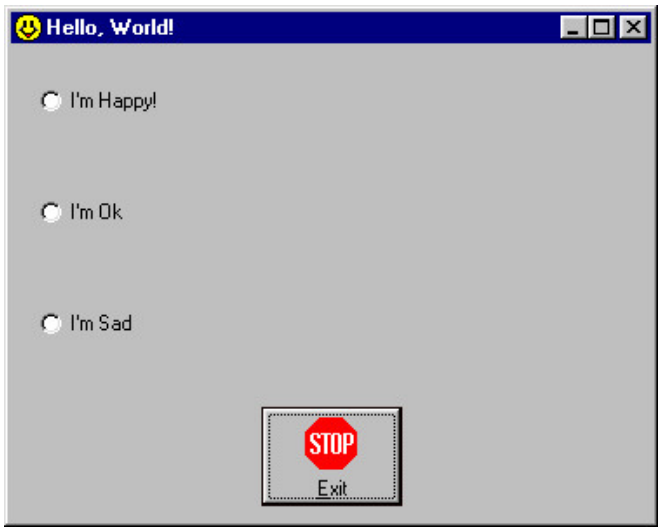
4. Digite a procedure:

```
Private Sub optOk_Click()
    picHappy.Visible = False
    picOk.Visible = True
    picSad.Visible = False
    lblHappy.Caption = Empty
    lblOk.Caption = "I'm Ok today."
    lblSad.Caption = Empty
End Sub
```

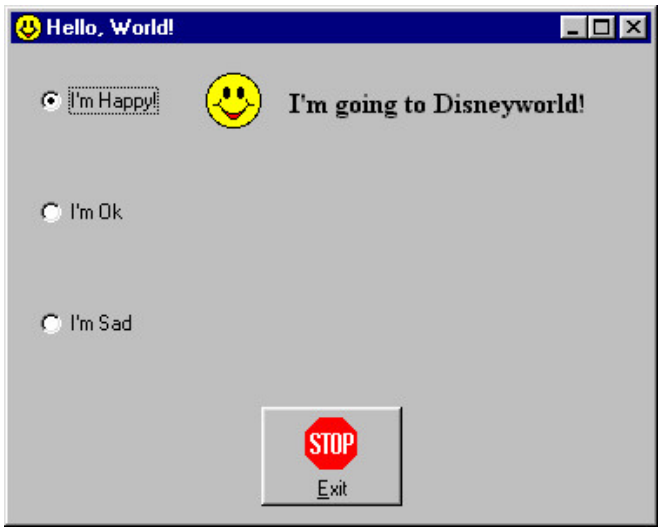
5. Selecione agora o objeto optSad e a procedure optSad_Click, e escreva o código para ela (você pode usar os recursos de copiar e colar do Windows):

```
Private Sub optSad_Click()
    picHappy.Visible = False
    picOk.Visible = False
    picSad.Visible = True
    lblHappy.Caption = Empty
    lblOk.Caption = Empty
    lblSad.Caption = "Goodbye, cruel world..."
End Sub
```

6. Pressione F5 para executar o programa. Se você digitou tudo corretamente, o VB exibirá seu formulário na tela, como no exemplo a seguir:



7. Dê um click no botão com o rótulo “I’m Happy” e veja se o resultado é o esperado:



8. Verifique o funcionamento dos outros botões de opção. Se algo estiver errado, confira a codificação e execute o aplicativo novamente.

Criando um arquivo executável

O último passo de qualquer projeto é a criação do arquivo executável. Ou você vai entregar para o cliente os arquivos que o compõem? Além das razões monetárias óbvias, você não vai correr o risco de um curioso qualquer alterar ou copiar o código que você tanto suou para criar, vai? Além disso, se você não criar um executável, o cliente deverá possuir um cópia do VB instalada em sua máquina para poder rodar o aplicativo, o que não é lá muito prático. A criação de arquivos executáveis é feita pela opção *Make <nome do projeto>.exe* do menu *File*:

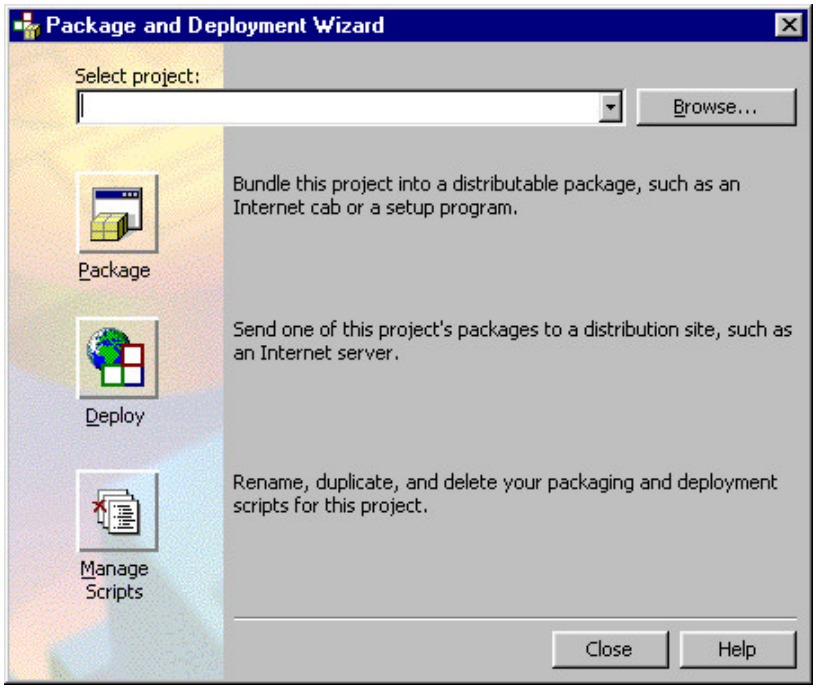
1. Escolha a opção *Make hello.exe* do menu *File* do VB. Uma caixa de diálogo será exibida, onde você deverá informar o nome do arquivo executável, caso queira mudá-lo. Após indicar o nome, clique em Ok;
2. Crie um atalho para o arquivo executável na área de trabalho do Windows. Veja que, como ícone do atalho, será exibida a "carinha feliz".

Package and Deployment Wizard

A criação do executável, apesar de importante, ainda não é suficiente para a distribuição de uma aplicação desenvolvida em VB. Como você sabe, o Windows é um sistema composto de inúmeros arquivos e programas, cada qual com sua função específica, sendo que alguns são necessários para a execução de aplicativos. Além disso, o programa escrito em VB não tem o poder de "rodar" sozinho, mesmo após a criação do módulo executável: para funcionar, necessita de algumas bibliotecas chamadas de *run-time*, e dos arquivos OCX que estão gravados no diretório do Windows. Chamamos a esses arquivos de *dependências*. Mas, como saber quais são as dependências de um aplicativo?

Um dos utilitários que acompanham o VB em suas versões comerciais é o *Package and Deployment Wizard*, cuja função é justamente resolver esse problema. Ele cria um programa de instalação para sua aplicação, que vai copiar para a máquina onde o programa for instalado todos os arquivos de *run-time*, DLL's e OCX's que ele precisar, da mesma maneira que em um aplicativo comercial.

O uso desse utilitário é muito simples:



1. Na caixa *Select Project*, selecione o projeto para o qual deve ser criado o programa de instalação;
2. Clique no botão *Package* para iniciar o assistente;
3. Você deverá responder algumas perguntas (por exemplo, o nome da opção que será criada no menu Iniciar do Windows) sendo que para a maioria delas não é necessário fazer qualquer alteração. Devemos porém destacar uma delas na qual deve-se indicar, se for o caso, o nome dos arquivos externos ao projeto que também precisam ser copiados. O caso mais comum é o de arquivos de banco de dados, quando da primeira instalação do aplicativo;
4. Depois que você responder essas perguntas, deverá indicar em que unidade e (se for o caso) em que pasta onde devem ser gravados os arquivos da instalação. Se você indicar uma unidade de disquetes, o Wizard criará os disquetes de instalação automaticamente;
5. Pronto, o Wizard resolve todas as dependências e cria o programa de instalação para você.

Lição 2: Entrada e Saída de Dados

Entrada e saída são dois conceitos importantíssimos em Processamento de Dados, pois envolvem a interação do usuário com a máquina: por entrada, entende-se toda solicitação ou fornecimento de dados ao computador pelo operador, e por saída, o retorno pelo computador do resultado obtido. Nessa lição você vai aprender a implementar esses dois conceitos em seus programas.

Caixas de Texto

Em programação visual, existem várias maneiras de se permitir a entrada de dados, mas a principal é o uso das caixas de texto (*TextBox*). Seu evento padrão é o *Change*, que ocorre sempre que acontece uma digitação, e a propriedade principal é a *Text*. Vejamos então algumas propriedades importantes:

Propriedade Text

A propriedade Text de uma caixa de texto armazena o texto contido na área de edição. Ela pode ser alterada pelo usuário em tempo de execução, o que permite então a digitação de dados no programa. Como a propriedade Text armazena textos, quando houver a necessidade do dado a ser digitado ser numérico teremos que converter seu conteúdo usando funções específicas para esse fim.

Propriedade MaxLength

Essa propriedade serve para determinar o tamanho máximo em caracteres do dado digitado em uma caixa de texto. Por exemplo: sabendo que a digitação em uma caixa de texto chamada txtNome não pode ter mais de 30 caracteres, determinamos para a MaxLength de txtNome o valor 30. Se MaxLength não for definida, o valor padrão é zero, e não haverá limite para a digitação no campo.

Propriedade Multiline

A função dessa propriedade é permitir ou não a digitação de mais de uma linha em uma caixa de texto. Os valores possíveis são True (Multiline habilitada) ou False. Com MuliLine = True, quando Enter for pressionada, a digitação passa para a próximo linha. Além disso, a propriedade Alignment de uma caixa de texto só funciona de Multiline estiver habilitada.

Propriedade PasswordChar

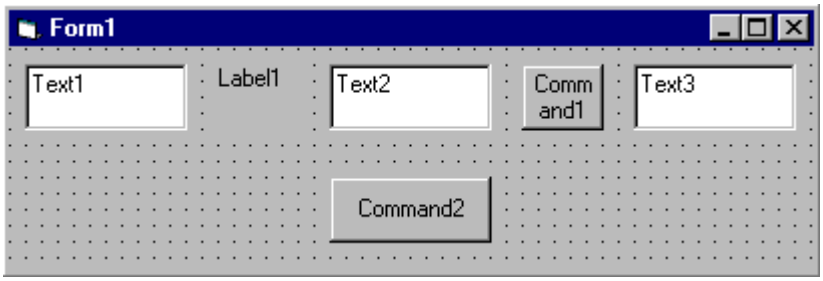
Eis uma propriedade muito interessante: você já digitou uma senha em um programa qualquer em que aparece na caixa um caractere especial (normalmente é um “*”) ao invés daquele digitado? Quem faz isso é PasswordChar: basta informar em seu valor o caracter desejado, e o efeito será exatamente esse.

Propriedades ForeColor e BackColor

A propriedade ForeColor define a cor da fonte usada no texto exibido pelo controle, e BackColor indica a cor do fundo de um objeto. Essas propriedades recebem como valor um código numérico ou hexadecimal que representa a cor desejada. Em tempo de projeto, para alterá-las a maneira mais simples é clicar sobre a propriedade que deve ser modificada e depois no botão com as reticências que aparece ao lado do código da cor atual. Uma janela com as opções de cores disponíveis se abre e basta escolher a que mais lhe agrade. Para alterar seus valores em tempo de execução, são usadas as funções RGB e QBColor, que estudaremos mais adiante.

Para exemplificar o uso das *TextBoxes*, vamos elaborar o projeto de uma calculadora simples:

1. Inicie um novo projeto;
2. Dimensione o formulário e insira os controles necessários de modo a ficar parecido com o modelo da próxima página;



3. Altere as propriedades dos objetos:

Objeto	***Nome***	***Outras Propriedades = Valor***
Formulário	frmCalculadora	Caption = Calculadora BorderStyle = 1 – Fixed Single StartUpPosition = 2 – Center Screen
Caixa de Texto	txtNum1	Text = “” (em branco) MaxLenght = 10
Caixa de Texto	txtNum2	Text = “” MaxLenght = 10
Caixa de Texto	txtResultado	Text = “”
Label	lblMultiplicação	Caption = X (maiúsculo) Font = Arial, Negrito, tamanho 16
Botão de Comando	cmdIgual	Style = 1 – Graphical Picture = Misc\MISC22.ICO
Botão de Comando	cmdLimpar	Caption = &Limpar

Nesse projeto, digitaremos um número em txtNum1 e outro em txtNum2. Quando clicarmos em cmdIgual, o resultado da multiplicação aparecerá em txtResultado. Para limpar as caixas de texto, clicaremos em cmd-Limpar. O projeto irá trabalhar, portanto, basicamente com dois eventos:

- Click em cmdIgual (=)
- Click em cmdLimpar (Limpar)
- Não se preocupe com o cálculo agora, pois voltaremos a esse assunto na próxima lição. Vamos então escrever a codificação do nosso programa:

1. Dê um duplo click no botão cmdIgual e entre com o seguinte código:

```
Private Sub cmdIgual_Click()
      vValor1 = Val(txtNum1.Text)
      vValor2 = Val(txtNum2.Text)
      txtResultado.Text = vValor1 * vValor2
End Sub
```

2. Alterne para a procedure cmdLimpar_Click e digite os comandos a seguir:

```
Private Sub cmdLimpar_Click()
      txtResultado.Text = Empty
      txtNum1.Text = Empty
      txtNum2.Text = Empty
End Sub
```

3. Execute o projeto. Para testar, entre com um número qualquer em txtNum1, outro em txtNum2, dê um click no botão “=”, e o resultado da multiplicação aparecerá em txtResultado;

4. Grave o formulário como **frmCalc** e o projeto como **Calc**.

Obs.: você deve ter percebido que na codificação foram usados alguns recursos novos: por enquanto, o que nos interessa é que “vValor1” e “vValor2” são variáveis, e “Val” é uma função de conversão.

Saída de Dados

Você já usou na lição anterior um dos principais meios de exibição de dados do VB: o controle *Label*. Ele tem, nesse caso, uma vantagem muito grande sobre as caixas de texto, fácil de perceber: execute novamente

o programa Calc, faça um cálculo qualquer e clique com o mouse sobre a caixa de texto txtResultado. Agora escreva alguma coisa: a digitação é aceita, o que, convenhamos, não é lá muito recomendável para a exibição de dados. Já um label não permite digitação, resolvendo esse problema. Vamos então alterar nosso programa:

1. Apague do formulário a caixa txtResultado;
2. Inclua um label em seu lugar, e altere suas propriedades para:
   - *Name* = lblResultado
   - *Caption* = "" (em branco)
   - *Alignment* = 2 – Center
   - *BorderStyle* = 0 – None
   - *BackStyle = 1 – Opaque*
   - *BackColor* = (clique nas reticências e escolha a cor branca)
3. Altere a última linha da procedure cmdIgual_Click para:

```
lblResultado.Caption = vValor1 * vValor2
```

4. Altere a primeira linha da procedure cmdLimpar_Click para:

```
lblResultado.Caption = Empty
```

5. Execute o programa, faça um cálculo qualquer e tente alterar o resultado: como o label não permite digitação, isso não é mais possível.

Saída Formatada

É muito comum a necessidade de se aplicar máscaras para a exibição dos dados. Isso fica claro quando, em nossa calculadora, um cálculo resulta em um valor alto, pois não existem separadores de milhar ou decimal. O VB possui algumas funções específicas para a formatação de dados de saída:

Função Format

A função Format é usada na aplicação de máscaras em dados numéricos ou datas. Sua sintaxe é a seguinte:

```
Format(<expressão>,<formato>)
```

Onde **Expressão** é o dado a ser formatado e **Formato** é a máscara que será aplicada a ele, e o resultado será uma string correspondente à expressão formatada com a máscara. Veja alguns exemplos:

Formato	5 positivo	5 negativo	5 decimal
0	5	-5	1
0,00	5,00	-5,00	0,50
#,##0	5	-5	1
#,##0.0	5,0	-5,0	0,5
$#,##0;($#,##0)	$5	($5)	$1
$#,##0.00;($#,##0.00)	$5,00	($5,00)	$0,50
0%	500%	-500%	50%
0.00E+00	5,00E+00	-5,00E+00	5,00E-1

Em "formato" o número 0 será mostrado ou trocado pelo caractere em sua posição, já o sustenido (#) não será mostrado. Podemos usar símbolos na máscara, como no exemplo: $, % e E (que representa exponenciação). Note que a separação de milhares e decimais na máscara deve ser feita no padrão americano, ou seja, "," para separação de milhares e "." para decimais, mas o resultado vai seguir as configurações feitas na opção Internacional do Painel de Controle do Windows. Format também pode ser usada para formatação de data e hora:

Formato	Exibição
d/m/yy	10/7/96
dd-mm-yyyy	01-Jun-1996
dd-ddd	02-Dom
hh:mm AM/PM	08:50 AM
h:mm:ss a/p	8:50:20 a
d/m/yy h:mm	03/12/95 9:30

Formato	Exibição
General Date	06/09/96 9:40:18
Long Date	Sexta, 9 de setembro de 1996
Medium Date	09-set-96
Short Date	09/09/96
Long Time	9:40:19
Medium Time (12 horas)	09:40 AM
Short Time (24 horas)	09:40

Funções UCase e LCase

Essas funções são usadas na conversão de textos em letras maiúsculas (UCase) ou minúsculas (LCase).

Função StrConv

A função StrConv também é usada na formatação de strings, mas é mais flexível que UCase e LCase, pois recebe um parâmetro que indica como deve ser feita a conversão de maiúsculas para minúsculas e vice-versa. Esse parâmetro pode ser:

- **1** – **vbUpperCase**: converte a string em caracteres maiúsculos. O mesmo que UCase;
- **2 – vbLowerCase**: converte a string em caracteres minúsculos. O mesmo que LCase;
- **3 – vbProperCase**: converte em letra maiúscula a primeira letra de cada palavra na seqüência de caracteres, e as demais em minúsculas.

Por exemplo:

Função	Exibição
UCase(“MICROSOFT Visual basic”) ou StrConv(“MICROSOFT Visual basic”, vbUpperCase)	MICROSOFT VISUAL BASIC
LCase(“MICROSOFT Visual basic”) ou StrConv(“MICROSOFT Visual basic”, vbLowerCase)	microsoft visual basic
StrConv(“MICROSOFT Visual basic”, vbProperCase)	Microsoft Visual Basic

Vamos alterar a exibição do resultado na calculadora:

1. Abra a janela Código no evento Click do botão cmdIgual;
2. Altere a última linha da procedure como segue:

```
lblResultado.Caption = Format(vValor1 * vValor2, “###,##0.00”)
```

3. Execute o programa e teste a alteração;
4. Grave seu trabalho. Guarde-o com cuidado, pois esse projeto será usado novamente na próxima lição.

Lição 3: Fundamentos da Linguagem

Como já foi dito anteriormente, a programação em VB é feita através de uma versão da linguagem Basic estruturada. É essa linguagem que começaremos a estudar a partir de agora. Nessa lição, veremos a criação e o uso de variáveis e constantes, e a razão da existência dos computadores: cálculos. Estudaremos também um conceito muito importante em programação visual: o foco.

Variáveis

Variável é uma área da memória que identificamos com um nome e onde podem ser guardados dados com possibilidade de alteração em tempo de execução. O nome de uma variável pode ter até 255 caracteres, deve começar com uma letra e tem que ser único. O nome pode conter números e sublinhados, e não pode ser uma palavra reservada. Nesse texto adotaremos como padrão iniciar todos os nomes de variáveis com a letra "v". Por exemplo: para uma variável que identifica uma quantidade poderemos adotar o nome vQuantidade.

Existem vários tipos de variáveis, dependendo do dado que queremos que ela armazene:

Tipo	*Tamanho (Bytes)*	*Sufixo*	*Faixa de Dados*
Byte	1	nenhum	0 a 255
Boolean	2	nenhum	True (-1) ou False (0)
Date	8	nenhum	1/Jan/100 a 31/Dez/9999
Integer	2	%	-32.768 a 32.767
Long	4	&	-2.147.483.647 a 2.147.483.647
Single	4	!	-3,402823E38 a -1,401298E-45; 1,401298E-45 a 3,402823E38
Double	8	#	-1,79769313486232E308 a -4,94065645841247E-324; 4,94065645841247E-324 a 1,79769313486232E308
Currency	8	@	-922.337.203.685.477,5808 a 922.337.203.685.477,5807
String	variável	$	Qualquer dado alfanumérico
Variant	variável	nenhum	Qualquer tipo de dado: será assumido o tipo e tamanho mais apropriado ao dado a ela atribuído

A coluna Sufixo contém caracteres que servem para indicar o tipo da variável desejada, quando a usamos pela primeira vez. Por exemplo: para iniciar uma variável de nome vTroco e declará-la como sendo do tipo Currency, poderemos usar a linha de comando:

```
vTroco@ = 0
```

Uma observação: internamente, o tipo Date é um número do tipo Long que representa uma data, sendo que a parte inteira desse número indica a data propriamente dita, e a parte decimal a hora.

Declaração de Variáveis

Declarar uma variável é indicar ao VB que deve ser reservada uma área na memória para o armazenamento de um dado, e de que tipo será esse dado. Existem três maneiras de declarar variáveis:

1. Inicializar a variável onde ela for necessária. Como o VB não obriga a declaração de variáveis, podemos simplesmente atribuir um valor a ela e a variável assumirá o tipo mais adequado para o dado que está sendo armazenado: é o que chamamos de *declaração implícita*;
2. Usar o mesmo método acima, porém indicando o tipo da variável através dos sufixos, na primeira vez em que a utilizarmos;
3. Usar os comandos Dim, Static, Private ou Public para alocar o espaço na memória para a variável e indicar seu tipo. A sintaxe é a mesma para todos os comandos:

   ```
   <comando> Variável1, Variável2, Variável3, ..., VariávelN As <tipo>
   ```

Também podemos obrigar a declaração de todas as variáveis de uma aplicação, o que é muito interessante, pois declarar variáveis é uma prática de programação altamente recomendável. Para isso usamos o comando **Option Explicit** na seção Declarations do formulário ou módulo de programação.

Escopo das Variáveis

Escopo são os pontos da aplicação de onde podemos acessar a variável. O escopo de uma variável é determinado pelo comando usado em sua declaração (Dim, Static, Private ou Public):

- **Variáveis Locais**: são reconhecidas apenas pela procedure na qual foram declaradas. Diferentes procedures podem ter variáveis locais com o mesmo nome sem que isso cause conflitos, pois quando uma procedure é encerrada o conteúdo de suas variáveis locais será perdido. A declaração de variáveis locais é feita com os comandos Dim ou Private. Importante: variáveis com declaração implícita serão sempre locais;
- **Variáveis Estáticas:** são variáveis locais cujo conteúdo será retido ao término da procedure. Nesse caso, quando a procedure que contém uma variável estática for executada novamente, a variável não será reinicializada e seu conteúdo será o mesmo que ela continha quando a procedure foi encerrada da última vez. Para declarar uma variável como estática usa-se o comando Static;
- **Variáveis Privadas**: compartilham informações entre todas as procedures em um módulo de programação ou formulário, mas não para outros módulos ou formulários. A declaração de variáveis privadas é feita com os comandos Dim ou Private na seção Declarations de um formulário ou módulo de programação;
- **Variáveis Públicas**: estão disponíveis para todos os módulos de programação e formulários do aplicativo. Para a declaração de variáveis públicas usamos o comando Public.

Obs.: por uma questão de compatibilidade com versões anteriores do VB, também pode ser usado o comando Global para declaração de variáveis públicas.

Inicialização de Variáveis

De modo geral, para inicializar uma variável basta atribuir um conteúdo a ela. Obviamente, esse conteúdo deve ser compatível com o tipo usado na declaração da variável.

- Para dados numéricos basta indicar o valor desejado:

```
vSalárioBase = 1200
```

- Valores reais devem usar o ponto para separação da parte decimal, **jamais a vírgula**, mesmo que a configuração do Painel de Controle do Windows indique o contrário:

```
vPi = 3.1416
```

- Dados tipo String devem vir entre aspas:

```
vNomeEscola = “CEMEP”
```

- Dados tipo Date são identificados pelo caracter “#” (sustenido) no seu início e fim. Por exemplo: para armazenar a data 15/07/1998 em uma variável, podemos usar o comando:

```
vDataCurso = #15/07/1998#
```

Constantes

Constantes também são posições de memória que têm as mesmas características das variáveis e podem ser dos mesmos tipos, mas, como o próprio nome indica, não podem ter seu valor alterado durante a execução do programa. São úteis para guardar parâmetros e valores que serão usados em várias procedures do sistema, pois com elas podemos centralizar a atribuição desses valores em alguns poucos locais, o que facilita muito a programação. Por exemplo: quando um cliente compra um sistema, obviamente pretende que o nome de sua empresa apareça nos relatórios emitidos por ele. Se você incluir o nome da empresa em cada programa de relatório, e depois vender esse sistema para outro cliente, terá que rastreá-lo todo a procura dessas ocorrências e fazer as alterações necessárias. Mas, se você definir uma constante pública no início do sistema contendo o nome da empresa, só terá que fazer uma alteração. Constantes só podem ser declaradas com os comandos Public (constantes públicas) ou Private (constantes privadas ou locais). A sintaxe usada na sua criação é:

```
<escopo> Const NomeDaConstante As <tipo> = <valor>
```

Por exemplo:

```
Public Const Pi As Single = 3.1415926535
```

Constantes de Sistema

O VB possui diversas constantes pré-definidas, chamadas de Constantes de Sistema, que podem ser usadas a qualquer momento pelo programador. A principal vantagem de seu uso é a clareza da codificação e a conseqüente facilidade de manutenção. Um bom exemplo do uso desse tipo de constante é o de uma aplicação que trate com dias da semana. Se a comparação do número do dia da semana de uma data com 1 for verdadeira, saberemos que esse dia é um domingo, pois o VB identifica um domingo pelo valor 1. Existe uma constante de sistema, de nome vbSunday, cujo valor é 1, que foi criada justamente para ser usada em casos como esse: ao invés do programador comparar o dia da semana com 1 e ter que se lembrar a toda hora que 1 é domingo, pode comparar com vbSunday, e seu código ficará muito mais legível. A lista completa das constantes de dias da semana é a seguinte:

Constante	***Valor***	***Dia da semana***
vbSunday	1	Domingo
vbMonday	2	Segunda
vbTuesday	3	Terça
vbWednesday	4	Quarta
vbThursday	5	Quinta
vbFriday	6	Sexta
vbSaturday	7	Sábado

Um observação importante é que uma constante de sistema é facilmente identificável, pois seu nome geralmente começa com "vb" – repare na tabela acima. Você também já viu algumas delas na lição anterior, quando estudamos a função StrConv: os valores vbUpperCase, vbLowerCase e vbProperCase são constantes de sistema cujos valores são, respectivamente, 1, 2 e 3. Estudaremos mais algumas delas durante o curso.

Funções de Conversão

Como já vimos anteriormente, a entrada de dados é feita pelas caixas de texto, obrigando o uso de conversores quando houver necessidade do dado ser numérico ou data. Os conversores são funções que recebem dados de um tipo e os convertem em outro. Cada tipo de conversão tem uma função correspondente. Por exemplo: não podemos usar um conversor de string para Byte se sabemos que o texto representa um valor maior que 255, pois esse é o valor limite do tipo Byte: o tamanho do dado não será suportado na conversão.

A tabela a seguir mostra as funções para conversão de dados tipo String em numéricos. Considere que a variá-vel vPi contém um texto correspondente ao valor de π ("3.1415926535"):

Função/Sintaxe	***Exemplo***	***Tipo Resultante***	***Resultado da conversão***
Val(<string>)	Val(vPi)	Long	3
CInt(<string>)	Cint(vPi)	Integer	3
CLng(<string>)	CLng(vPi)	Long	3
CSng(<string>)	CSng(vPi)	Single	3.14159
CDbl(<string>)	CDbl(vPi)	Double	3.1415926536
CCur(<string>)	Ccur(vPi)	Currency	3.1416

Observação: as funções Val e CLng diferem-se na conversão de um dado não numérico: Val é a única função que consegue tratar um dado String que não represente um valor. Nesse caso, ela retornará zero. As outras funções, inclusive CLng, geram um erro de execução se forem usadas para esse tipo de conversão.

A conversão de um dado tipo String em uma data é feita pela função CDate. Essa função também gera um erro de execução se o texto a ser convertido não representar uma data válida. Veja sua sintaxe e um exemplo de utilização:

```
CDate(<data>)

' Define a data:
vDia = "12/02/1969"
' Converte vDia para vDataNasc, do tipo Date.
vDataNasc = CDate(vDia)
```

Obs.: as linhas iniciadas com o apóstrofo (') são comentários.

O VB também possui uma função para conversão de expressões numérica ou data em dados tipo String: é a função Str, cuja sintaxe é:

```
Str(<valor>)
```

Por exemplo: o resultado da linha de comando Str(3.1416) será o texto "3.1416".

Operadores

Depois que o programa recebe dados do usuário, o próximo passo é fazer algo com esses dados, normalmente cálculos. Qualquer tipo de cálculo envolve uma operação; portanto, os comandos especiais para trabalhar com dados são chamados *operadores*. O VB fornece cinco tipos de operadores:

Operadores Aritméticos

São os operadores matemáticos clássicos, que permitem somar, subtrair, multiplicar e dividir números ou variáveis que contenham números:

Operador	***Função***	***Exemplo***
=	Nesse caso, atribuição de valor	Variável = 0
+	Soma de dois números	vSoma = X + Y
–	Subtração de dois números Inversão de positivo/negativo	vSalLiq = vSalBruto – vImposto vNegativo = –vPositivo
*	Multiplicação de dois números	vTotal = vPreço * vQuantidade
/	Divisão de dois números, sendo que o resultado será um número com ponto flutuante (decimal), como 5.8547	vTeste = 1910 / 25 (vTeste será igual a 76.4)
\	Divisão de dois números, sendo que o resultado será um número inteiro	vTeste = 1910 \ 25 (vTeste será igual a 76)
Mod	Resto da divisão inteira de um número por outro	vTeste = 1910 Mod 25 (vTeste será igual a 10)
^	Exponenciação	vQuadrado = vNúmero ^ 2

Operador de String

O mais simples de todos. É o operador "&", que realiza a **concatenação** (união) de dois ou mais dados tipo String. Veja um exemplo:

```
vLinguagem = "Visual " & "Basic"
```

O comando acima resulta em "Visual Basic", pois o dado após o operador é agrupado por este ao final do primeiro, resultando na união de ambos: foram concatenados. Também pode ser usado para isso o operador "+", mas recomenda-se usar "&" porque este converte dados de outros tipos antes da concatenação, evitando erros de execução.

Operadores de Datas

São aqueles que permitem efetuar cálculos com dados tipo Date. Como as datas são representadas internamente pelo VB como números, podemos obter outras datas somando valores a elas, ou então determinar o número de dias entre uma data e outra através de subtração. Veja a tabela:

Operador	***Função***	***Exemplo***
+	Obter a partir da soma de uma data inicial e um número uma outra data. O número será entendido como quantidade de dias.	vVencimento = Date + 30 (Date retorna a data de hoje, à qual serão somados trinta dias)
–	Obter a diferença em dias entre duas datas	vAtraso = vPagto – vVenc (vAtraso receberá o número de dias entre o vencimento e o pagamento)

Um lembrete: como o tipo Date também serve para indicar horas, para cálculos de horas usamos os mesmos operadores, mas trabalhamos com a parte decimal das variáveis ou expressões.

Operadores Lógicos

São aqueles que manipulam os valores lógicos *True* (verdadeiro) e *False* (falso). O VB também representa esses valores como –1 e 0, respectivamente. Não entraremos em muitos detalhes com esses operadores agora, pois eles serão mais bem estudados quando abordarmos as estruturas de controle.

Operador	***Função***	***Exemplo***
And	*E* lógico: retorna verdadeiro se todos os valores da expressão forem verdadeiros e falso em qualquer outro caso	Expressão1 And Expressão2
Or	*Ou* lógico: retorna falso somente se todos os valores da expressão forem falsos e verdadeiros em qualquer outro caso	Expressão1 Or Expressão2
Xor	*Ou* lógico exclusivo: a diferença desse operador para Or é que, se todos os valores da expressão forem verdadeiros, ele retorna **falso** (não há erro, é isso mesmo!)	Expressão1 Xor Expressão2
Not	*Não* lógico: "nega", ou melhor, inverte o valor lógico da expressão	Expressão1 = Not Expressão2

Operadores de Comparação

Servem para a comparação de dados. Assim como os operadores lógicos, estudaremos melhor esse assunto juntamente com as estruturas de controle.

Operador	***Função***	***Exemplo***
<	Menor que	Expressão1 < Expressão2
<=	Menor ou igual a	Expressão1 <= Expressão2
>	Maior que	Expressão1 > Expressão2
>=	Maior ou igual a	Expressão1 >= Expressão2
=	Nesse caso, igual a	Expressão1 = Expressão2
<>	Diferente de	Expressão1 <> Expressão2

Muito cuidado ao comparar strings com os operadores "=" e "<>", pois **duas strings são iguais apenas se forem absolutamente idênticas**. Isso quer dizer que, se você fizer a comparação "a" = "A", o resultado será falso, pois "a" (minúsculo) possui um código ASCII diferente de "A" (maiúsculo), e portanto não são iguais.

O Foco

No Windows, apenas uma janela (formulário) ou controle pode, num dado instante, receber ações (cliques) do mouse ou entradas via teclado. Dizemos que esse objeto **possui o foco** (*focus*, em inglês). O foco é geralmente indicado por uma legenda ou moldura destacada, ou então pelo cursor estar posicionado no controle, e pode ser alterado pela ação do usuário ou através de código. Quando um objeto recebe o foco, ocorre o evento Got-Focus, e quando ele o perde ocorre o evento LostFocus.

Outra característica importante a ser observada é que, quando um programa VB é executado, o cursor se posiciona no primeiro controle que você inseriu no formulário. Por exemplo: se o primeiro controle a ser inserido foi o txtNum1, quando você iniciar o aplicativo é nele que o cursor estará posicionado. Usando a tecla Tab, o cursor vai se alternando entre os campos, sempre seguindo a ordem em que os controles foram colocados no formulário. Isso chama-se **ordem de tabulação** e é determinada pela propriedade TabIndex dos controles.

Propriedade TabIndex

Como já foi dito, conforme você insere controles em um formulário eles recebem uma ordem para recebimento do foco, que é o valor da propriedade TabIndex. O primeiro controle receberá o valor 0 para TabIndex, o segundo 1, e assim por diante. Essa ordem pode ser mudada pelo programador, bastando alterar o valor de TabIndex de modo a ajustá-la. No caso de ser digitado para TabIndex um valor já pertencente a outro controle, o VB vai reorganizá-los automaticamente. TabIndex não pode ser alterada em tempo de execução.

Propriedade TabStop

Essa propriedade impede o acesso ao controle pela tecla Tab se estiver desligada. Isso acontece porque TabStop determina se o controle deve ou não atender à ordem de tabulação. Mas é importante ressaltar que TabStop não evita que o usuário acesse o controle usando o mouse. Os valores possíveis para TabStop são True (TabStop ligada) ou False (TabStop desligada).

Propriedade Default

Existem, nas aplicações para Windows, botões de comando que podem ser acionados pela tecla Enter, independente da ordem de tabulação, como os botões Ok das caixas de diálogo do Windows. Quem determina o acionamento do botão por Enter é a propriedade Default, que pode receber os valores True ou False. O objeto cuja propriedade Default for igual a True aparecerá com um contorno mais espesso, dando a indicação que, se Enter for pressionada, ele será acionado.

Método SetFocus

Para posicionar o foco em um determinado objeto em tempo de execução usamos o método SetFocus, sendo que somente formulários ou controles visíveis podem recebê-lo. A sintaxe é a seguinte:

```
objeto.SetFocus
```

Modificando a Calculadora

Vamos agora aplicar o que aprendemos:

1. Se ainda não o fez, abra a VB e o projeto da calculadora que iniciamos na lição anterior;
2. Execute o aplicativo e verifique se a ordem de tabulação corresponde à seguinte: txtNum1 = 0, txtNum2 = 1, cmdIgual = 2 e cmdLimpar = 3. Se necessário, altere as propriedades TabIndex dos controles para que fiquem de acordo com a ordem indicada;
3. No formulário da calculadora, selecione o botão cmdIgual e pressione a tecla Delete. Lembra-se que havia programação associada a ele? E agora que esse objeto apagado, para onde ela foi?
4. Abra a janela Código e procure a seção General do formulário. Note que a lista Proc agora contém, além da seção Declarations, a procedure cmdIgual_Click, que está lá como uma procedure geral. Uma procedure geral é aquela que pode ser chamada por qualquer outra procedure do formulário, ou seja, funciona como uma subrotina ou função. Ela não é executada por estar associada a um objeto e a partir de um evento, e sim quando chamada. Estudaremos isso melhor na próxima lição;
5. Agora altere o formulário da calculadora como no exemplo:

6. Altere as propriedades dos novos objetos seguindo a tabela:

Objeto	***Nome***	***Outras Propriedades = Valor***
Label	Label1	Caption = Primeiro Número:
Label	Label2	Caption = Segundo Número:
Botão de Comando	cmdMultiplicar	Style = 1 – Graphical Picture = Misc\MISC20.ICO
Botão de Comando	cmdDividir	Style = 1 – Graphical Picture = Misc\MISC21.ICO
Botão de Comando	cmdSomar	Style = 1 – Graphical Picture = Misc\MISC18.ICO
Botão de Comando	cmdSubtrair	Style = 1 – Graphical Picture = Misc\MISC19.ICO

7. Chame a janela Código e procure a procedure cmdIgual_Click (está na seção Declarations, lembra-se?) e altere seu nome para cmdMultiplicar_Click. Ao término da alteração, o código do antigo botão cmdIgual será associado ao botão cmdMultiplicar (veja nas caixas de combinação da janela Código);

8. Altere a procedure cmdMultiplicar para que fique de acordo com a codificação a seguir. Note que foi incluída na programação a declaração das variáveis como do tipo Single, e a função de conversão usada foi substituída pela CSng, que é apropriada a esse tipo:

```
Private Sub cmdMultiplicar_Click()
      Dim vValor1 As Single
      Dim vValor2 As Single
      vValor1 = CSng(txtNum1.Text)
      vValor2 = CSng(txtNum2.Text)
      lblResultado.Caption = Format(vValor1 * vValor2, "###,##0.00")
End Sub
```

9. Selecione todo o texto da procedure, menos o cabeçalho e End Sub, e copie para a área de transferência usando a opção *Copy* do menu *Edit* ou teclando Ctrl+C;

10. Abra a janela código no evento Click de outro botão e cole o trecho de programação usando a opção *Paste* do menu *Edit* ou então Ctrl+V;

11. Faça as alterações necessárias para que o botão funcione corretamente;

12. Repita esse procedimento para os outros operadores;

13. Grave seu trabalho e teste o aplicativo.

Lição 4: Estruturas de Controle

Assim como em outras linguagens, como C ou Pascal, o VB também possui estruturas de tomada de decisão e para controle de repetição (loops). Nessa lição aprenderemos a lidar com essas estruturas e também mais alguns recursos da linguagem.

Estruturas de Decisão

Estruturas de decisão são aquelas que permitem o desvio do fluxo do programa de acordo com uma condição. As estruturas de decisão do VB são similares às encontradas em outras linguagens:

Estrutura If...Then

Executa um comando ou uma série de comandos de acordo com uma condição. Se a condição não for verdadeira, os comandos serão ignorados e a execução do programa passa para o próximo comando após a estrutura. Essa estrutura pode ser escrita de duas maneiras:

```
If <condição> Then <comando>
```

ou:

```
If <condição> Then
      <comando1>
      <comando2>
      ...
      <comandoN>
End If
```

O que diferencia as duas estruturas é o número de comandos executados se <condição> for verdadeira. Na primeira somente um comando é executado, por isso esse comando pode vir logo após a palavra reservada Then, e o comando End If não é necessário. Exemplo:

```
If txtNome.Text = Empty Then MsgBox “Um nome deve ser digitado!”
```

Na segunda vários comandos devem ser executados se a condição for verdadeira, então cada comando deverá ser escrito em uma linha e o comando End If deve ser usado para fechar a estrutura. Veja dois exemplo:

```
If txtNome.Text = Empty Then
      MsgBox “Um nome deve ser digitado!”
End If

If vSaldoMédio > 1000 Then
      lblStatus.Caption = “Cliente Especial”
      vTaxaJuros = 0.015
End If
```

Estrutura If...Then...Else

Semelhante ao If...Then, porém contém um ou mais comandos para serem executados no caso da condição ser falsa, indicados pelo Else da sintaxe:

```
If <condição> Then
      <comandos para condição verdadeira>
Else
      <comandos para condição falsa>
End If
```

A condição é testada, e, se for verdadeira, o primeiro conjunto de comandos será executado. Se for falsa, o segundo conjunto será executado. Exemplo:

```
If vSaldoMédio > 1000 Then
      lblStatus.Caption = “Cliente Especial”
      vTaxaJuros = 0.015
Else
      lblStatus.Caption = “Cliente Comum”
      vTaxaJuros = 0.018
End If
```

Estrutura Select Case

Quando houver a necessidade de se testar um número maior de situações o VB nos oferece a estrutura Select Case, que possibilita a construção de código mais eficiente e legível do que um encadeamento de If's. Sua sintaxe é a seguinte:

```
Select Case <expressão>
      Case <valor1>
            <comandos para valor1>
      Case <valor2>
            <comandos para valor2>
      ...
      Case <valorN>
            <comandos para valorN>
      [Case Else]
            [<comandos>]
End Select
```

Valor1, valor2, valorN são os valores que a expressão pode assumir: se a expressão assumir o valor1 os comandos para o valor1 serão executados, se assumir valor2 os comandos para valor2 serão executados, e assim por diante. Se o valor assumido não for nenhum dos valores indicados, os comandos após o Case Else serão executados. É interessante destacar que Case Else é opcional. Exemplo:

```
Select Case vCodigoCliente
      Case 1
            lblStatus.Caption = "Cliente Preferencial"
            vTaxaJuros = 0.012
      Case 2
            lblStatus.Caption = "Cliente Especial"
            vTaxaJuros = 0.015
      Case 3
            lblStatus.Caption = "Cliente Comum"
            vTaxaJuros = 0.018
      Case Else
            lblStatus.Caption = "Código Inválido"
End Select
```

Estrutura If...ElseIf

Essa é uma variação do comando If que possui uma sintaxe muito parecida com o Select Case:

```
If <expressão1> Then
      <comandos para expressão1>
ElseIf <expressão2> Then
      <comandos para expressão2>
ElseIf <expressão3> Then
      <comandos para expressão3>
      ...
ElseIf <expressãoN> Then
      <comandos para expressãoN>
[Else]
      [<comandos>]
End If
```

Por exemplo: o comando que codificamos com Select Case, se escrito com If...ElseIf, ficaria assim:

```
If vCodigoCliente = 1 Then
      lblStatus.Caption = "Cliente Preferencial"
      vTaxaJuros = 0.012
ElseIf vCodigoCliente = 2 Then
      lblStatus.Caption = "Cliente Especial"
      vTaxaJuros = 0.015
ElseIf vCodigoCliente = 3 Then
      lblStatus.Caption = "Cliente Comum"
      vTaxaJuros = 0.018
Else
      lblStatus.Caption = "Código Inválido"
End Select
```

Comando With

Esse é um comando muito útil quando temos várias instruções que serão aplicadas a um mesmo objeto. Ele informa ao VB qual objeto deve ser considerado sempre que em um comando qualquer não houver nenhuma indicação da aplicação de uma propriedade, método ou campo. Sua sintaxe é:

```
With <objeto>
      <comandos a serem aplicado em objeto>
End With
```

É mais fácil entender vendo um exemplo prático: suponhamos que você precisa aplicar uma série de comandos a uma caixa de texto:

```
txtTeste.Visible = True
txtOutra.Visible = False
txtTeste.FontName = “Arial”
txtTeste.FontSize = 20
txtTeste.FontBold = True
txtTeste.Text = “Isso é um teste.”
```

Com o comando With, você poderia escrever o mesmo trecho de programação da seguinte maneira:

```
With txtTeste
      .Visible = True
      txtOutra.Visible = False
      .FontName = “Arial”
      .FontSize = 20
      .FontBold = True
      .Text = “Isso é um teste.”
End With
```

Todas as propriedades onde não foi indicada a caixa de texto txtTeste estão sendo aplicadas à ela da mesma maneira, devido ao comando With. Note que a instrução aplicada à txtOutra não mudou, mesmo estando dentro do conjunto de comandos contidos entre o With e o End With.

É importante destacar que podem existir um bloco With dentro de outro, sendo que o mais interno terá efeito sobre todos os objetos a ele subordinados enquanto não for encontrado um End With fechando-o. Antes e depois do bloco mais interno, quem tem efeito é o With mais externo.

Evento Load

Load é o evento que ocorre quando um formulário é lido. Usamos Load normalmente para a inicialização de variáveis privadas ou então para escrever comandos e rotinas que devem ser executadas logo que o formulário é aberto. Ou melhor: no evento Load de um formulário usamos comandos e rotinas cuja execução é necessária ao seu funcionamento e de seus controles de uma maneira geral.

O Aplicativo TesteCor

Vamos testar o que aprendemos em um novo aplicativo. Ele funcionará assim: quando um botão de comando for clicado, a cor do texto de um label mudará para a próxima cor numa escala pré-definida, e o rótulo desse label deverá informar qual é a cor atual. Por exemplo: se a cor atual for “2 – Verde”, passará a ser “3 – Ciano”. Um outro botão fará o processo inverso, ou melhor: a cor passará a ser a anterior na escala.

Mas qual é a melhor maneira de codificar esse processo todo? Vejamos:

- Algumas cores serão modificadas a cada vez que um dos botões for clicado. Então, precisamos guardar o código da cor atual em uma variável que possa ser acessada por várias procedures, já que cada evento é ligado a uma procedure. Resumindo: precisamos de variáveis privadas.

- Existe também uma seqüência de comandos repetida para alguns botões: a mudança de uma das cores do *label* e a exibição do nome dessa cor. Então, podemos criar uma subrotina que contenha essa seqüência de comandos e possa ser chamada pela procedure Click de um dos botões, de modo a não haver necessidade de repetição de código.

Funções QBColor e RGB

Essas funções geram cores de acordo com valores que representam códigos associados a elas. A diferença entre ambas é que QBColor só aceita os valores de 0 a 15, cada um representando uma cor diferente, enquanto RGB permite a composição de uma cor pela mistura das cores básicas Vermelho (Red), Verde (Green) e Azul (Blue), daí seu nome. A quantidade de vermelho, verde e azul que deve ser usada na composição da cor é representada por valores inteiros que variam de 0 a 255. Veja as sintaxes:

```
QBColor(<código>)
```

```
RGB(<vermelho>,<verde>,<azul>)
```

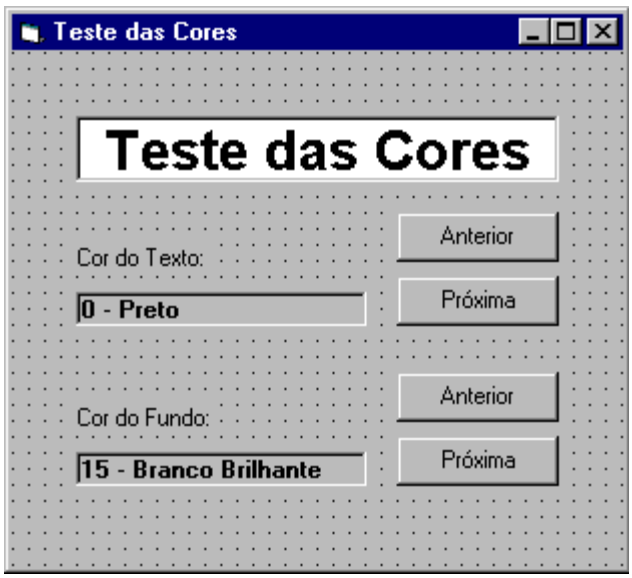
Os códigos aceitos por QBColor são os seguintes:

Código	*Cor*	*Código*	*Cor*	*Código*	*Cor*	*Código*	*Cor*
0	Preto	4	Vermelho	8	Cinza	12	Vermelho Claro
1	Azul	5	Magenta	9	Azul Claro	13	Magenta Claro
2	Verde	6	Amarelo	10	Verde Claro	14	Amarelo Claro
3	Ciano	7	Branco	11	Ciano Claro	15	Branco Brilhante

Na tabela abaixo estão os valores de vermelho, verde e azul necessários para se conseguir algumas cores com a função RGB:

Cor Desejada	*Vermelho*	*Verde*	*Azul*
Preto	0	0	0
Azul	0	0	255
Verde	0	255	0
Azul Claro	0	255	255
Vermelho	255	0	0
Magenta	255	0	255
Amarelo	255	255	0
Branco	255	255	255

Como você com certeza já percebeu, em nosso caso a função QBColor é a que melhor se encaixa. Mas vamos começar a trabalhar: crie o formulário de acordo com o exemplo:



Altere as propriedades dos objetos:

Objeto	*Nome*	*Outras Propriedades = Valor*
Formulário	FrmTeste	Caption = Teste das Cores StartUpPosition = 2 – Center Screen
Label	Label1	Caption = Cor do Texto:
Label	Label2	Caption = Cor do Fundo:

Objeto	***Nome***	***Outras Propriedades = Valor***
Label	LblTeste	Caption = Teste das Cores Alignment = 2 – Center BackColor = Branco BorderStyle = 1 – Fixed Single Font = Arial, Negrito, tamanho 20
Label	LblTexto	Caption = 0 – Preto BorderStyle = 1 – Fixed Single Font = Negrito
Label	LblFundo	Caption = 15 – Branco Brilhante BorderStyle = 1 – Fixed Single Font = Negrito
Botão de Comando	cmdTextoAnt	Caption = Anterior
Botão de Comando	cmdTextoProx	Caption = Próxima
Botão de Comando	cmdFundoAnt	Caption = Anterior
Botão de Comando	cmdFundoProx	Caption = Próxima

Sendo assim, vamos criar nossa codificação:

1. Abra a janela Código na seção Declarations. Vamos declarar as variáveis privadas:

```
Dim vCorTexto As Integer
Dim vCorFundo As Integer
```

2. Dê agora um duplo click sobre uma área do formulário que não contenha nenhum controle. A janela Código é aberta no evento Load do objeto Form, aonde vamos inicializar as variáveis que declaramos no item anterior:

```
Private Sub Form_Load()
    vCorTexto = 0
    vCorFundo = 15
End Sub
```

3. Acesse a procedure Click do botão cmdTextoAnt e escreva os comandos abaixo:

```
Private Sub cmdTextoAnt_Click()
    vCorTexto = vCorTexto - 1
    If vCorTexto < 0 Then vCorTexto = 15
    MudaCorTexto
End Sub
```

Note como o comando If...Then foi usado: já que só existe um comando após Then, End If não foi necessário. No entanto, poderíamos escrever essa estrutura da seguinte maneira, com o mesmo resultado:

```
If vCorTexto < 0 Then
      vCorTexto = 15
End If
```

Você deve ter percebido também que temos algo novo nessa procedure: o comando MudaCorTexto. Na verdade, isso é a chamada a uma subrotina, aquela que citamos quando estávamos planejando nosso aplicativo. Podemos criar uma subrotina através da opção *Add Procedure* do menu *Tools*: o VB abre uma caixa de diálogo onde devemos informar, entre outras coisas, o nome da subrotina que será criada (veja a figura na próxima página), mas existe uma maneira mais simples:

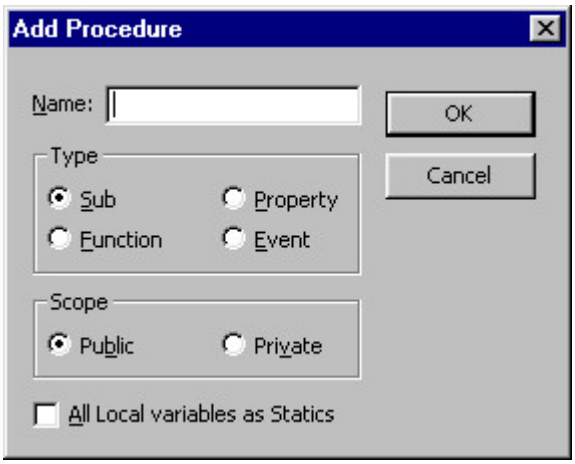
Criando uma Subrotina

A criação de subrotinas pode ser feita diretamente na janela Código, bastando para isso fazer a declaração da procedure (ou funcão – veremos mais adiante). Subrotinas podem ser declaradas como privadas ou públicas. Usamos a palavra reservada Sub para inicar uma procedure, e o comando “End Sub” para fechá-la.

1. Com a janela Código aberta, escolha a opção *General* na caixa de combinação *Object*. *Proc* deve estar na seção *Declarations*;

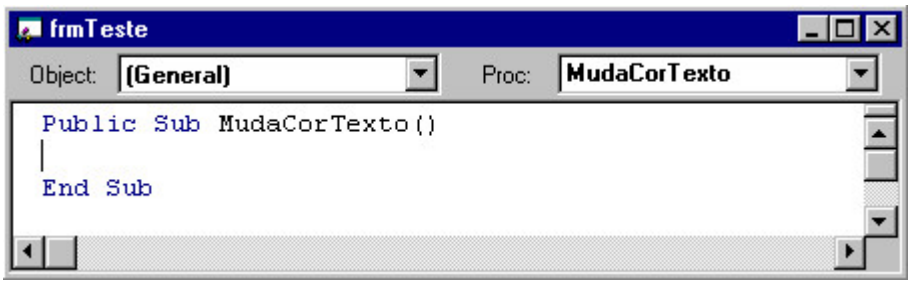
2. Declare a procedure MudaCorTexto escrevendo o cabeçalho a seguir:

```
Public Sub MudaCorTexto()
```



A janela Add Procedure

3. Note que, quando terminar a declaração, o próprio VB insere o comando End Sub fechando a procedure, que deve estar parecida com a da figura abaixo:



4. Vamos então escrever os comandos da subrotina:

```
Private Sub MudaCorTexto()
    lblTeste.ForeColor = QBColor(vCorTexto)
      With lblTexto
            Select Case vCorTexto
                  Case 0
                        .Caption = "0 - Preto"
                  Case 1
                        .Caption = "1 - Azul"
                  Case 2
                        .Caption = "2 - Verde"
                  Case 3
                        .Caption = "3 - Ciano"
                  Case 4
                        .Caption = "4 - Vermelho"
                  Case 5
                        .Caption = "5 - Magenta"
                  Case 6
                        .Caption = "6 - Amarelo"
                  Case 7
                        .Caption = "7 - Branco"
                  Case 8
                        .Caption = "8 - Cinza"
                  Case 9
                        .Caption = "9 - Azul Claro"
                  Case 10
                        .Caption = "10 - Verde Claro"
                  Case 11
                        .Caption = "11 - Ciano Claro"
                  Case 12
                        .Caption = "12 - Vermelho Claro"
                  Case 13
                        .Caption = "13 - Magenta Claro"
                  Case 14
                        .Caption = "14 - Amarelo Claro"
                  Case Else
                        .Caption = "15 - Branco Brilhante"
```

```
            End Select
      End With
End Sub
```

5. Execute o aplicativo, teste o funcionamento do botão e verifique se está de acordo com o planejado.
6. Escreva a procedure cmdTextoProx_Click: note que ela é quase igual à do botão cmdTextoAnt, só que muda para a próxima cor na escala:

```
Private Sub cmdTextoProx_Click()
    vCorTexto = vCorTexto + 1
    If vCorTexto > 15 Then vCorTexto = 0
    'Chamada da subrotina:
    MudaCorTexto
End Sub
```

7. Escreva agora as procedures para os eventos click dos botões cmdFundoAnt e cmdFundoProx, e a subrotina a ser chamada por elas, que deve ter o nome de MudaCorFundo. A propriedade a ser alterada para exibir a nova cor é a BackColor;
8. Teste o funcionamento de todos os botões e verifique se está de acordo com nosso planejamento.
9. Salve o formulário com o nome de **frmTeste** e o projeto com o nome de **TesteCor**.

Estruturas de Repetição

As Estruturas de Repetição permitem que uma ou mais linhas de código sejam executadas um determinado número de vezes ou até que uma condição seja verdadeira. Resumindo: criam *loops*.

Estruturas Do While

A estrutura Do While (faça enquanto) permite que um determinado bloco de comandos seja executado enquanto uma condição for verdadeira. Existem dois tipos de estruturas Do While:

Loop com teste *a priori*: como você sabe, esse tipo de loop testa sua condição de término *antes* da execução dos comandos que estão nele contidos, o que implica na possibilidade desses comandos não serem executados, caso a condição de término seja satisfeita logo de início. A sintaxe para a criação de loops com teste a priori em VB usando Do While é a seguinte:

```
Do While <condição>
      <comandos>
Loop
```

Loop com teste *a posteriori*: já esse tipo de loop testa sua condição de término após a execução dos comandos nele contidos, ou seja, os comandos serão executados ao menos uma vez, mesmo que a condição de término seja satisfeita logo de início. A sintaxe de Do While com teste a posteriori é a seguinte:

```
Do
      <comandos>
Loop While <condição>
```

Veja exemplos da estrutura Do While usando os dois tipos de loop para a mesma operação:

```
Do While vTotal < 500
    vTotal = vTotal + Quantidade
Loop
```

```
Do
    vTotal = vTotal + Quantidade
Loop While vTotal < 500
```

Estruturas Do Until

Do Until quer dizer “faça até que”. Essa estrutura difere da anterior por executar os comandos do loop enquanto a condição for falsa. Também existem duas sintaxes para Do Until:

Loop com teste *a priori*:

```
Do Until <condição>
    <comandos>
Loop
```

Loops com teste *a posteriori*:

```
Do
    <comandos>
Loop Until <condição>
```

Para que a diferença entre as duas estruturas fique mais clara, vamos alterar os exemplos usados em Do While para estruturas Do Until. Note que foi necessária uma mudança na formulação da condição de término do loop para obtermos o mesmo efeito:

```
Do Until vTotal >= 500
    vTotal = vTotal + Quantidade
Loop
```

```
Do
    vTotal = vTotal + Quantidade
Loop Until vTotal >= 500
```

Estrutura For Next

Essa estrutura deve ser usada quando se conhece de antemão o número de vezes que o loop será executado. Sua sintaxe é a seguinte:

```
For <contador> = <valor inicial> To <valor final> [Step <valor do incremento>]
    <comandos>
Next <contador>
```

Onde:

- **Contador** é uma variável que será usada para controlar o número de vezes que o loop será executado;
- **Valor Inicial** é o primeiro valor que a variável de controle assumirá;
- **Valor Final** é o último valor que o contador assumirá na execução do loop;
- **Valor do Incremento** é o valor que será somado ou subtraído (pois podemos usar valores negativos) do contador a cada vez que Next for executado.

Ao entrar em um loop For Next o VB faz com que o contador seja igual ao valor de início. A cada vez que os comandos são executados e a estrutura atinge o Next, o contador é incrementado pelo valor indicado em Step (que é opcional – se não for declarado será assumido o valor 1) e comparado com o valor final indicado. **Se o contador ultrapassar o valor final, o loop será terminado**, caso contrário a seqüência de comandos será executada novamente.

Exemplo: a estrutura abaixo calcula a soma dos números de 1 a 100:

```
vSoma = 0
For i = 1 To 100
    vSoma = vSoma + i
Next i
```

Com uma pequena mudança na estrutura, somaremos apenas os números ímpares entre 1 e 100:

```
vSoma = 0
For i = 1 To 100 Step 2
    vSoma = vSoma + i
Next i
```

Comando Exit

O comando Exit permite que um loop ou uma subrotina sejam abandonados (terminem antes do final normal de sua execução). A sintaxe do comando Exit exige que indiquemos que estrutura queremos abandonar, assim:

- **Para abandonar um loop Do** : `Exit Do`
- **Para abandonar um loop For** : `Exit For`
- **Para abandonar uma subrotina Sub** : `Exit Sub`
- **Para abandonar uma subrotina Function** : `Exit Function`

And, Or, Xor e Not

Os operadores lógicos And, Or, Xor e Not (E, Ou, Ou exclusivo e Não) são normalmente usados em estruturas de controle para estabelecer uma relação entre duas ou mais condições em uma mesma expressão. Como já vimos na lição 2, a função desses operadores é a seguinte:

- **And** faz uma combinação das condições, retornando verdadeiro somente se todas as condições da expressão forem verdadeiras.
- **Or** funciona de maneira contrária: a expressão só será falsa se todas as condições forem falsas.

- **Xor** é semelhante a Or, mas retorna falso se todas as condições forem verdadeiras.
- **Not** inverte a condição de verdadeiro para falso e vice-versa.

Veja a tabela de resultados:

Operador	***Condição 1***	***Condição 2***	***Resultado***
And	Verdadeiro	Verdadeiro	Verdadeiro
	Verdadeiro	Falso	Falso
	Falso	Verdadeiro	Falso
	Falso	Falso	Falso
Or	Verdadeiro	Verdadeiro	Verdadeiro
	Verdadeiro	Falso	Verdadeiro
	Falso	Verdadeiro	Verdadeiro
	Falso	Falso	Falso
Xor	Verdadeiro	Verdadeiro	Falso
	Verdadeiro	Falso	Verdadeiro
	Falso	Verdadeiro	Verdadeiro
	Falso	Falso	Falso
Not	Verdadeiro		Falso
	Falso		Verdadeiro

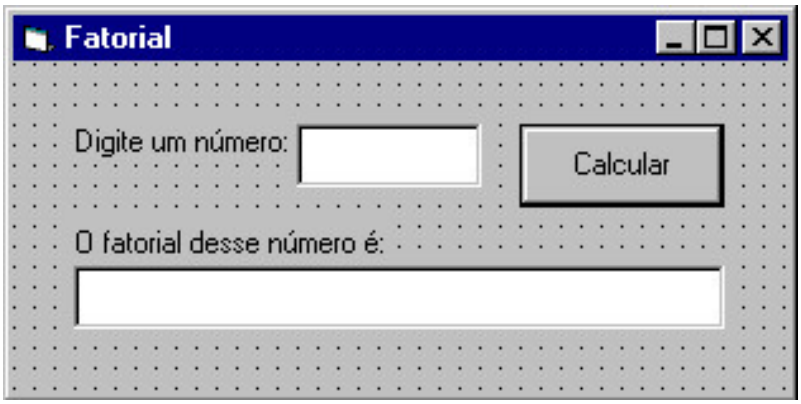
O Aplicativo Fatorial

Para testar o uso das estruturas de repetição, vamos escrever um pequeno aplicativo para calcular o fatorial de um número inteiro (n!), que é o resultado da multiplicação dos números inteiros de 1 até n. Exemplo:

$5! = 1 \times 2 \times 3 \times 4 \times 5 = 120$

Veja o modelo e a tabela de propriedades no início da próxima página. A codificação do aplicativo envolve apenas um evento: quando o usuário clicar em cmdCalcular ou teclar Enter o fatorial do número digitado em txtNúmero será calculado e exibido em lblFatorial.

1. Inicie um novo projeto;
2. Formate o formulário e altere as propriedades dos objetos como a seguir:



Objeto	***Nome***	***Outras Propriedades = Valor***
Formulário	frmFatorial	Caption = Fatorial StartUpPosition = 2 – Center Screen
Label	Label1	Caption = Digite um número:
Label	Label2	Caption = O fatorial desse número é:
Caixa de Texto	txtNúmero	Text = “”
Label	lblFatorial	Caption = “” Alignment = 2 – Center BackColor = Branco BorderStyle = 1 – Fixed Single
Botão de Comando	cmdCalcular	Caption = Calcular Default = True

3. Abra a janela Código no evento Click do botão Calcular e escreva os comandos para o cálculo:

```
Private Sub cmdCalcular_Click()
Dim vNúmero As Double, vFatorial As Double
    vNúmero = Val(txtNúmero.Text)
    If vNúmero < 0 Then
        Beep
        lblFatorial.Caption = Empty
    Else
        If vNúmero = 0 Or vNúmero = 1 Then
            lblFatorial.Caption = 1
        Else
            vFatorial = 1
            Do
                vFatorial = vFatorial * vNúmero
                vNúmero = vNúmero - 1
            Loop Until vNúmero <= 1
            lblFatorial.Caption = Str(vFatorial)
        End If
    End If
    txtNúmero.SetFocus
End Sub
```

Observações:

- Por definição, 0! = 1;
- O comando Beep faz com que o computador emita um sinal sonoro (um “apito”).

Note o uso do operador Or no segundo If: poderíamos escrever a condição vNúmero <= 1 ao invés da expressão usada, já que nesse ponto temos a certeza que vNúmero não é negativo (veja o If anterior), mas fizemos isso para exemplificar o uso de Or;

Como exercício de fixação, reescreva o loop do cálculo do fatorial usando as estruturas Do While e For Next. Tente também modificar as estruturas de teste a posteriori para teste a priori.

4. Grave o formulário com o nome de **frmFator** e o projeto com o nome de **Fatorial**.

Lição 5: Tratamento de Erros de Execução

Eis um assunto importantíssimo: tratamento de erros irrecuperáveis de execução. Quando um erro irrecuperável acontece, o VB emite uma mensagem informando o erro e encerra a execução do aplicativo. Esses erros acontecem sob várias condições: um arquivo não encontrado, incompatibilidade entre tipos de variáveis, uma chamada de função inválida, falta de memória, etc.

É fácil conferirmos essa característica: execute a calculadora que fizemos na lição 2 e, ao invés de um número, digite uma letra qualquer nas caixa de textos. Quando você executar um cálculo, uma caixa de mensagem com os dizeres "Run-time Error" aparecerá informando o tipo do erro que ocorreu, e o aplicativo será encerrado.

Nessa lição aprenderemos a trabalhar com esses erros de modo que, mesmo que um deles aconteça, o aplicativo não seja encerrado. Adicionalmente, aprenderemos a usar as caixas de mensagem e de entrada da dados do VB.

Comando On Error

Esse comando é a chave para o tratamento de erros no VB: sua função é a de desviar o fluxo da execução de uma procedure para um parágrafo em caso de erro irrecuperável. Sua sintaxe é a seguinte:

```
On Error GoTo <nome_de_parágrafo>
```

O parágrafo para o qual o fluxo será desviado deve obrigatoriamente fazer parte da mesma procedure, e deve estar posicionado no final da mesma, antes do comando End Sub. O nome do parágrafo é identificado por ser terminado com dois pontos – ":".

Comando Resume

O comando Resume indica ao VB que o aplicativo deve continuar a ser executado mesmo em caso de erro irrecuperável. Veja a sintaxe:

```
Resume <destino>
```

Sendo que o <destino> da execução não é obrigatório. Se não for informado, a execução passa para a próxima instrução na seqüência normal do programa. As opções para <destino> são:

- **Next**: produz um desvio para a próxima instrução na seqüência normal da execução do aplicativo;
- **0**: faz com que o execução retorne ao início da procedure atual (parâmetro recursivo);
- **Nome_De_Parágrafo**: desvia a execução para um parágrafo da procedure atual.

Objeto Err

O objeto Err indica o código do erro irrecuperável ocorrido, através da propriedade **Number**, de tipo Integer. Pelo código informado por Err.Number podemos identificar o erro e escrever os comandos necessários para que o aplicativo continue funcionando. Outra propriedade interessante do objeto Err é a **Description**, que contém a descrição do erro ocorrido.

Para obter a lista completa com os códigos dos erros tratáveis do VB, acesse o menu *Help* e escolha a seqüência *Contents – Trappable Errors – Miscellaneous Messages*.

Vejamos então um exemplo da utilização desses recursos:

```
Public Sub Exemplo_Erro()
Dim vQualquer As Integer
    On Error GoTo Erro_Na_Conversão
    vQualquer = CInt(txtNúmero.Text)

Erro_Na_Conversão:
    If Err.Number = 13 Then
            '13 é o código de erro para Tipo Incompatível de Dados:
        vQualquer = 0
        Resume Next
    End If
End Sub
```

Como já citamos anteriormente, a função CInt gera um erro irrecuperável em caso de tipo incompatível de dados (*Type Mismatch*) na tentativa de conversão de uma string que não represente um valor. Na procedure Exemplo_Erro estamos dizendo para o VB que, se isso acontecer, a execução deve ser desviada para o parágrafo Erro_Na_Conversão, onde testamos o valor de Err.Number e escrevemos a codificação necessária para que o programa continue normalmente. O comando Resume Next indica então ao VB que a execução deve ser desviada para a próxima instrução na seqüência normal da programação.

Caixas de Mensagem

Até agora falamos sobre o tratamento de erros mas não informamos ao usuário o que exatamente aconteceu, o que é muito importante em qualquer aplicativo que se preze. Uma das formas que o VB nos oferece para fornecer mensagens ao usuário são as caixas de mensagens padronizadas. Por exemplo: supondo que o campo Nome de um aplicativo qualquer não possa ser deixado em branco pelo operador, se isso acontecer, poderíamos exibir na tela uma caixa de mensagem como a da figura:

Caixas de mensagem padronizadas como essa podem ser obtidas com o comando MsgBox e a função MsgBox. Veja a sintaxe de ambos:

```
MsgBox <mensagem>,<tipo>,<título>
<variável> = MsgBox(<mensagem>,<tipo>,<título>)
```

Onde:

- **Mensagem**: é uma string com a mensagem a ser exibida pela caixa. Esse é o único parâmetro obrigatório da sintaxe. No nosso exemplo: “O campo Nome deve ser preenchido!”;
- **Tipo**: é um valor obtido a partir da soma dos códigos que especificam o número e o tipo dos botões exibidos pela caixa, o ícone a ser utilizado, o botão default e a modalidade de exibição da caixa. No nosso exemplo esse valor é 4144. Esse valor também pode ser obtido a partir de algumas constantes de sistema. Veja a seguir uma lista com os valores admitidos e seu significado;
- **Título**: é uma string a ser exibida como título da caixa. No nosso exemplo: “Aviso”;
- **Variável** (somente para a função): nome da variável que receberá o número do botão selecionado pelo usuário.

Veja a seguir as constantes de sistema usadas para se obter o tipo da caixa com seus respectivos valores:

Constante	***Valor***	***Descrição***
vbOKOnly	0	Exibe somente o botão OK.
vbOKCancel	1	Exibe os botões OK e Cancelar.
vbAbortRetryIgnore	2	Exibe os botões Abortar, 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 Aviso: 
vbExclamation	48	Exibe o ícone Mensagem de Aviso: 
vbInformation	64	Exibe o ícone Mensagem de Informação: 
vbDefaultButton1	0	O primeiro botão é o default.
vbDefaultButton2	256	O segundo botão é o default.
vbDefaultButton3	512	O terceiro botão é o default.

Constante	*Valor*	*Descrição*
vbDefaultButton4	768	O quarto botão é o default.
vbApplicationModal	0	Janela restrita do aplicativo: o usuário deve responder à caixa de mensagem antes de continuar o trabalho no aplicativo atual.
vbSystemModal	4096	Janela restrita do sistema: todos os aplicativos são suspensos até que o usuário responda à caixa de mensagem.

No nosso exemplo, o valor é 4144 pois usamos apenas o botão Ok (0), o ícone Exclamação (48), o botão Ok é o default por ser o único (0), e a modalidade é sistema (4096). Então, 0 + 48 + 0 + 4096 = 4144. O comando completo para se obter a caixa de mensagem do exemplo é a seguinte:

```
MsgBox “O campo Nome deve ser preenchido!”, 4144, “Aviso”
```

- Podemos também usar as constantes correspondentes à configuração desejada:

```
MsgBox “O campo Nome deve ser preenchido!”, _
      vbOk + vbExclamation + vbDefaultButton1 + vbSystemModal, “Aviso”
```

- Ou então apenas aquelas que terão algum efeito na configuração (são diferentes de zero):

```
MsgBox “O campo Nome deve ser preenchido!”, _
      vbExclamation + vbSystemModal, “Aviso”
```

- Ou ainda uma variável:

```
vConf = vbExclamation + vbSystemModal
MsgBox “Nome deve ser preenchido!”, vConf, “Aviso”
```

Obs.: o símbolo “_” (subscrito) indica continuação da codificação na próxima linha.

Identificando o botão selecionado

A diferença entre o comando MsgBox e a função MsgBox é que a função retorna um número inteiro representando o botão selecionado pelo usuário na caixa de mensagem. Assim, podemos identificar esse botão e executar a ação correspondente a ele. Veja os valores retornados por MsgBox:

Constante	*Valor*	*Descrição*
vbOK	1	Selecionado o botão Ok.
vbCancel	2	Selecionado o botão Cancelar.
vbAbort	3	Selecionado o botão Abortar.
vbRetry	4	Selecionado o botão Repetir.
vbIgnore	5	Selecionado o botão Ignorar.
vbYes	6	Selecionado o botão Sim.
vbNo	7	Selecionado o botão Não.

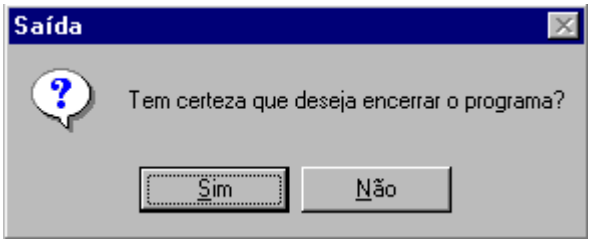
Por exemplo: suponha um aplicativo qualquer que possua um botão de comando para encerrar sua execução, mas que antes disso solicite ao usuário uma confirmação. A programação para isso poderia ser a seguinte:

```
Private Sub cmdFechar_Click()
Dim vOk As Integer
      vOk = MsgBox("Tem certeza que deseja encerrar o programa?", 36, "Saída")
      If vOk = 6 Then
            End
      End If
End Sub
```

Ou então, se usarmos as constantes de sistema:

```
Private Sub cmdFechar_Click()
Dim vOk As Integer
      vOk = MsgBox("Tem certeza que deseja encerrar o programa?", _
            vbYesNo + vbQuestion, "Saída")
      If vOk = vbYes Then
            End
      End If
End Sub
```

Em qualquer dos casos, a caixa de mensagem resultante da função MsgBox acima é a seguinte:



Mas qual é a vantagem de se usar constantes de sistema se a codificação fica maior? A resposta é: documentação! Leia novamente os dois trechos de programação e seja honesto: qual deles ficou mais fácil de entender?

Caixas de Entrada de Dados

Além das caixas de mensagem o VB dispõe de outro tipo de caixa de diálogo padronizada: as caixas de entrada de dados, que são obtidas com a função InputBox. A sintaxe dessa função é a seguinte:

```
<variável> = InputBox(<mensagem>,<título>,<valor padrão>)
```

Onde:

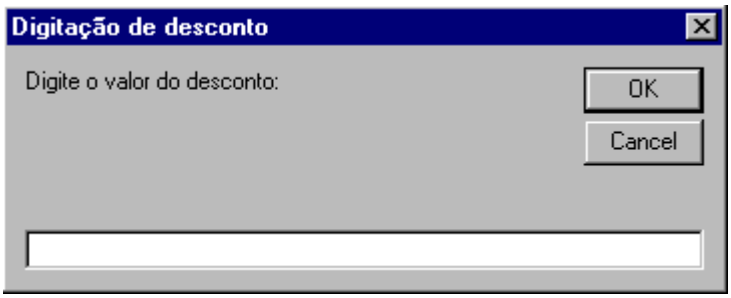
- **Mensagem**: único parâmetro obrigatório, é um texto a ser exibida dentro da caixa como legenda;
- **Título**: é uma string a ser exibida como título da caixa;
- **Valor padrão**: é o valor que a variável deve receber de InputBox caso o usuário não digite nada.

Importante: a função InputBox sempre retorna um dado tipo string, portanto se você precisar de um dado de qualquer outro tipo deverá convertê-lo.

Por exemplo: suponha um botão de comando que permita a digitação de um desconto para a emissão de uma nota fiscal de compra. A programação desse botão poderia ser a seguinte:

```
Private Sub cmdDesconto_Click()
      Dim vDesconto As String
      vDesconto = InputBox("Digite o valor do desconto:", _
            "Digitação de desconto")
End Sub
```

A InputBox resultante dessa programação é a da figura abaixo:



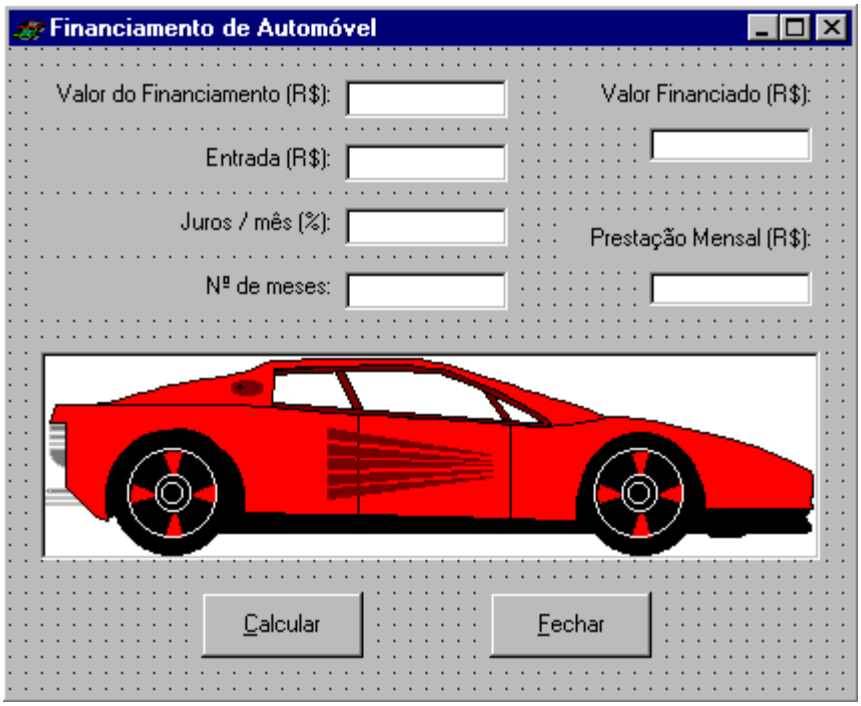
O Aplicativo Carro

Para usar os conceitos vistos nessa lição, vamos criar um aplicativo que calcule o valor da prestação mensal para financiamento de automóvel. Antes de começar, vamos estudar um novo controle que será usado em nosso projeto:

Caixa de Imagem (*ImageBox*)

Esse controle é muito semelhante a uma *PictureBox*, pois também exibe figuras ou ícones nos formulários. A principal diferença é que responde ao evento Click, funcionando como um botão de comando quando clicado. Outra diferença importante é a propriedade ***Stretch***, que pode aceitar os valores True e False: quando *Stretch* = True, o tamanho da imagem será ajustado ao tamanho da caixa de imagem, permitindo a exibição de figuras que devido ao seu tamanho não caberiam em um formulário. Se *Stretch* = False, então o tamanho da *ImageBox* é que será ajustado ao da figura.

Bem, vamos começar: inicie um novo projeto e formate o formulário de acordo com o modelo:



As propriedades dos objetos são as seguintes:

Objeto	***Nome***	***Outras Propriedades = Valor***
Formulário	frmCarro	Caption = Financiamento de Automóvel StartUpPosition = 2 – Center Screen
Caixa de Imagem	imgCarro	Stretch = True BorderStyle = 1 – Fixed Single Picture = CARRO.WMF (Procure na pasta ClipArt\Popular do diretório do MS-Office)
Label	Label1	Caption = Valor do Financiamento (R$):
Label	Label2	Caption = Entrada (R$):
Label	Label3	Caption = Juros / mês (%):
Label	Label4	Caption = Nº de meses:
Label	Label5	Caption = Valor Financiado (R$):
Label	Label6	Caption = Prestação Mensal (R$):
Label	lblValFinanciado	Caption = “” Alignment = 2 – Center BackColor = Branco BorderStyle = 1 – Fixed Single
Label	lblPrestMensal	Caption = “” Alignment = 2 – Center BackColor = Branco BorderStyle = 1 – Fixed Single
Caixa de Texto	txtValor	Text = “”
Caixa de Texto	txtEntrada	Text = “”
Caixa de Texto	txtJuros	Text = “”
Caixa de Texto	txtMeses	Text = “”
Botão de Comando	cmdCalcular	Caption = &Calcular Enabled = False
Botão de Comando	cmdFechar	Caption = &Fechar Enabled = False TabStop = False

Inicialmente o botão Calcular estará desabilitado pois só após a digitação de todos os dados necessários o cálculo da prestação mensal poderá ser feito. A fórmula para esse cálculo é a seguinte:

$$\text{Prestação Mensal} = \text{Valor Financiado} * \frac{\text{Juros} * (1 + \text{Juros})^{\text{Meses}}}{(1 + \text{Juros})^{\text{Meses}} - 1}$$

Antes de começar a programar, vamos conhecer um evento que será necessário para nossa codificação:

Evento KeyPress

Sempre que é pressionada uma tecla, estando o foco sobre um controle, ocorre o evento KeyPress. Associada a ele está a variável KeyAscii, que é carregada com o código ASCII da tecla pressionada. Dessa maneira podemos identificar quando o usuário usa uma tecla qualquer, e, se for o caso, escrever um código a ser executado quando essa tecla for utilizada. Por exemplo: podemos fazer com que o foco passe para outro controle quando a tecla Enter for pressionada numa caixa de texto. Sabendo-se que o código ASCII da tecla Enter é 13, correspondente à constante vbKeyReturn, o código para isso poderia ser o seguinte:

```
Private Sub Text1_KeyPress(KeyAscii As Integer)
      If KeyAscii = vbKeyReturn Then Text2.SetFocus
End Sub
```

Propriedade KeyPreview

Essa propriedade dos formulários indica se, quando pressionada uma tecla qualquer, ela deve ser ou não identificada pelo formulário. Os valores possíveis são True (habilita a verificação das teclas pelo formulário) e False (desabilita). Usando esse recurso e a função SendKeys, podemos fazer o Enter funcionar em qualquer campo, através do evento KeyPress do formulário (as linhas precedidas pelo apóstrofo – ‘ – são comentários):

```
Private Sub Form_KeyPress(KeyAscii As Integer)
    If KeyAscii = vbKeyReturn Then       'Identifica o Enter
        SendKeys "{Tab}"                 'Grava no buffer do teclado um Tab
        KeyAscii = 0                     'Zera KeyAscii para eliminar o Enter
    End If
End Sub
```

Na procedure acima, a cada vez que o usuário teclar Enter em um controle para o qual não surta efeito, ele será substituído por uma tabulação, e o foco passará para o próximo controle na seqüência de TabIndex.

Mas vamos codificar o aplicativo:

1. Abra a janela Código e acesse a seção Declarations. Vamos declarar as variáveis que receberão os dados digitados como privadas, pois elas serão acessadas por várias procedures:

```
Dim vValor As Currency
Dim vEntrada As Currency
Dim vJuros As Single
Dim vMeses As Integer
```

2. Vá agora ao evento Form_Load. Vamos inicializar as variáveis:

```
Private Sub Form_Load()
    vValor = 0
    vEntrada = 0
    vJuros = 0
    vMeses = 0
End Sub
```

3. Em tempo de execução, faremos com que o foco passe de uma caixa de texto para outra quando teclarmos Enter. Para isso vamos alterar o evento KeyPress do formulário. Dê um duplo clique sobre uma parte vazia do formulário e procure o evento KeyPress na lista de procedures;

4. Escreva a procedure como segue:

```
Private Sub Form_KeyPress(KeyAscii As Integer)
    If KeyAscii = vbKeyReturn Then
        SendKeys "{Tab}"
        KeyAscii = 0
    End If
End Sub
```

5. Vamos testar o evento KeyPress: execute o aplicativo e dê um Enter na caixa de texto txtValor. Nada aconteceu? É porque a propriedade KeyPreview do formulário não está habilitada. Saia do aplicativo para fazer a correção;

6. Altere o valor da propriedade KeyPreview do formulário para True;

7. Teste o aplicativo novamente: se tudo correu bem, o foco passou para a caixa txtEntrada. Desta, o foco vai para txtJuros, e assim por diante. Após testar em todos os controles, saia do aplicativo para podermos continuar;

8. Agora vamos fazer a codificação necessária para carregar a variável vValor. Como você pode observar, como o tipo de vValor é Currency, usaremos a função CCur para converter o texto contido em txtValor para a variável, o que pode gerar um erro irrecuperável de execução se txtValor contiver qualquer texto que não represente um valor. Devemos então usar uma rotina de tratamento de erro para evitar problemas. Mas, onde escrever essa rotina? Podemos fazer isso no evento LostFocus do controle, pois assim uma mensagem de erro será exibida assim que txtValor perder o foco, se esta contiver um dado inválido. Abra a janela código no evento LostFocus do controle txtValor e escreva a codificação a seguir:

```
Private Sub txtValor_LostFocus()
    On Error GoTo Valor_Errado
    vValor = CCur(txtValor.Text)

Valor_Errado:
    If Err = 13 Then
        MsgBox "Dado inválido na digitação do valor do financiamento", _
                vbExclamation + vbSystemModal, "Aviso"
        txtValor.Text = InputBox("Informe o valor correto do financiamento:", _
                "Valor do Financiamento")
        Resume 0
    End If
End Sub
```

- Note que usamos a função InputBox() para solicitar ao usuário que digite um dado válido em caso de erro de execução;
- Note também que o comando Resume é seguido do valor 0, o que vai fazer com que o VB retorne ao início da procedure em caso de erro e refaça a conversão do conteúdo de txtValor. Isso é importante pois, caso o usuário insista em escrever um dado inválido na caixa de entrada, o processo todo será repetido.

9. Vamos testar nossa procedure. Execute o aplicativo e escreva um texto qualquer na caixa txtValor. Digamos, “aaa”. O resultado deve ser o da figura a seguir:

10. Clicando no botão Ok, deverá aparecer a caixa de entrada:

11. Agora digite um valor correto, como 12000. O texto aparecerá na caixa txtValor e o foco passará para txtEntrada. Saia do aplicativo para continuarmos a programação;

12. Usando a procedure txtValor_LostFocus como base, escreva as rotinas para o evento LostFocus das outras caixas de texto. Lembre-se que a variável vJuros é do tipo Single, e portanto você deve usar a função CSng para a conversão. Além disso, o valor obtido deve ser dividido por 100, pois é um percentual. Já para a variável vMeses, que é Integer, você deverá usar a função CInt;

13. Após isso, ainda falta escrevermos a rotina de cálculo que será chamada pelo botão cmdCalcular. Vá até a seção Declarations e adicione uma subrotina de nome Prestação;

14. Na subrotina Prestação escreva a seguinte codificação (note o uso do operador And):

```
Public Sub Prestação()
Dim vValFinanciado As Currency
Dim vPrestMensal As Currency
    If vValor <> 0 And vJuros <> 0 And vMeses <> 0 Then
        If vEntrada >= vValor Then
            MsgBox "O valor da Entrada deve ser menor que o do Financiamento", _
                  vbExclamation + vbSystemModal, "Aviso"
            lblValFinanciado.Caption = Empty
            lblPrestMensal.Caption = Empty
            txtEntrada.SetFocus
            Exit Sub
        End If
        vValFinanciado = vValor - vEntrada
        vPrestMensal = vValFinanciado * vJuros * (1 + vJuros) ^ vMeses / _
            ((1 +  vJuros) ^ vMeses - 1)
        lblValFinanciado.Caption = Format(vValFinanciado, "###,##0.00")
        lblPrestMensal.Caption = Format(vPrestMensal, "###,##0.00")
        cmdCalcular.Enabled = True
        cmdFechar.Enabled = True
    Else
        lblValFinanciado.Caption = Empty
        lblPrestMensal.Caption = Empty
    End If
End Sub
```

Observações:

- O primeiro If verifica se todos os dados necessários para o cálculo estão disponíveis;
- O segundo If verifica se o valor da entrada não é maior ou igual ao do financiamento, pois senão o resultado do cálculo seria uma “prestação negativa” (se é que isso existe...);

- A propriedade Enabled dos botões de comando só foi habilitada dentro dessa rotina pois assumimos que o usuário fará ao menos um cálculo quando acionar o programa.

15. Agora só falta fazer com que o botão cmdCalcular execute a subrotina:

```
Private Sub cmdCalcular_Click()
    Prestação
End Sub
```

16. Você também pode fazer que, após o primeiro cálculo, toda vez que um valor for alterado os resultados sejam imediatamente atualizados. Para isso, altere a procedure txtValor_LostFocus como a seguir:

```
Private Sub txtValor_LostFocus()
    On Error GoTo Valor_Errado
    vValor = CCur(txtValor.Text)

    'É só incluir a linha abaixo:
    Prestação

Valor_Errado:
    If Err = vbKeyReturn Then
        MsgBox "Dado inválido na digitação do valor do financiamento", _
           vbExclamation + vbSystemModal, "Aviso"
        txtValor.Text = InputBox("Informe o valor correto do financiamento:", _
           "Valor do Financiamento")
        Resume 0
    End If
End Sub
```

17. Faça o mesmo para as outras caixas de texto;

18. Nosso aplicativo está praticamente terminado. Só falta a programação do botão cmdFechar. Nesse botão, vamos solicitar ao usuário uma confirmação de que ele realmente deseja sair do programa, usando a função MsgBox:

```
Private Sub cmdFechar_Click()
Dim vOk As Integer
    vOk = MsgBox("Tem certeza que deseja encerrar o programa?", _
        vbYesNo + vbQuestion, "Saída")
    If vOk = vbYes Then End
End Sub
```

19. Grave o formulário com o nome de **frmCarro** e o projeto com o nome de **Carro**;

20. Por fim, execute o aplicativo e teste-o bem. Procure usar vários valores diferentes nas caixas de texto para observar os resultados. Tente também entrar com dados inválidos e verifique se o comportamento do projeto é o esperado. Se necessário, faça os ajustes na programação para que seu funcionamento seja perfeito.

Lição 6: Iniciando a Construção de um Sistema

Um projeto em VB é um conjunto de objetos. Até agora trabalhamos com apenas um objeto em cada projeto: o formulário. Sim, o formulário também é um objeto: você não reparou que ele também possui propriedades e responde a métodos? Obviamente, em um aplicativo mais complexo, como será o caso do sistema de Controle de Bibliotecas que começaremos a desenvolver nessa lição, além do formulário principal existem outros destinados a entrada de dados, exibição de mensagens ou de resultados de consultas, etc.

Existem outros tipos de objetos que podem ser adicionados ao projeto, como módulos de programação, relatórios, controles ActiveX e módulos de classe. O que os caracteriza é o fato de serem gravados em arquivos separados. Assim, o projeto pode ser considerado como um conjunto de arquivos, cada um deles correspondente a um objeto.

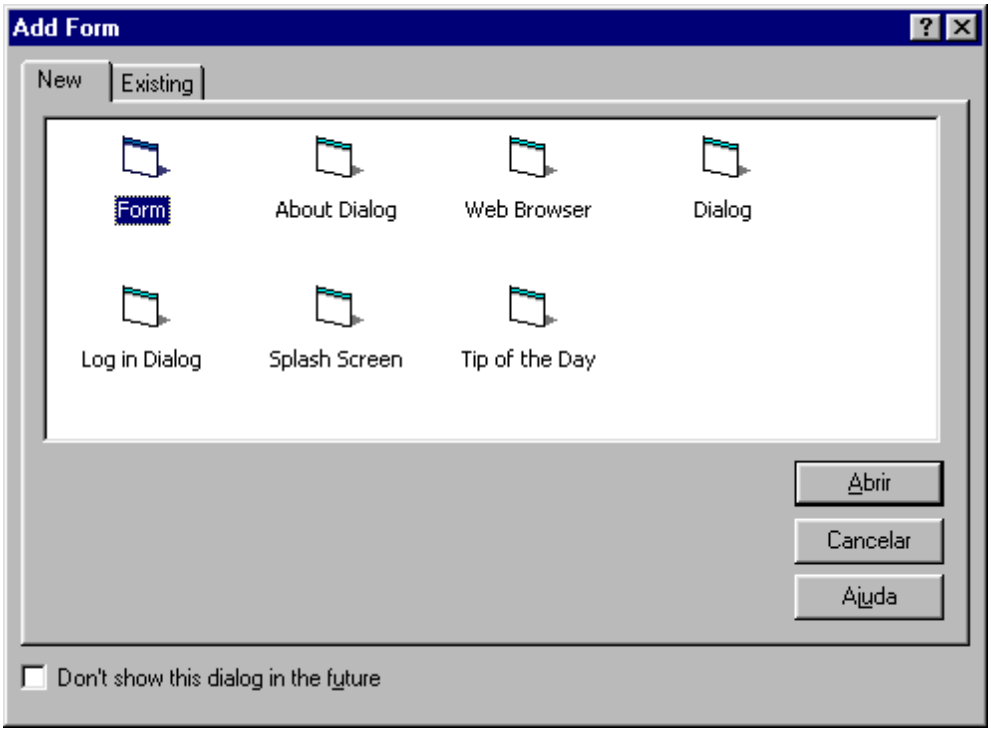
Trabalhando com Vários Formulários

Quando estamos desenvolvendo um projeto do qual vários formulários farão parte, obviamente vamos precisar adicionar a ele formulários novos e, muitas vezes, prontos. Além disso, uma aplicação Windows freqüentemente trabalha com outros tipos de formulários, diferentes daquele que usamos até agora.

Adicionando formulários ao projeto

Para incluir um novo formulário ou um formulário existente em um projeto, siga os seguintes passos:

- Abra o menu *Project* do VB;
- Escolha a opção *Add Form*: a janela Add Form será aberta, com várias opções de tipos de formulários:



- Se desejar acrescentar um formulário já existente, clique na aba *Existing* e indique sua localização;
- Se o formulário ainda não existe (formulário novo), você deve escolher um dos tipos da aba *New*. Para um formulário padrão, escolha a opção *Form*.

Em ambos os casos, o formulário será incluído na lista *Forms* da janela Projeto. A partir desse momento, ele poderá ser manipulado como outro qualquer. Vejamos alguns recursos necessários a esse trabalho:

Método Show

Esse método é usado para carregar e exibir um formulário. Sua sintaxe é:

```
<formulário>.Show <modalidade>
```

Onde <formulário> é o nome do formulário, e <modalidade> indica se o formulário será exibido como modal ou não. Um formulário modal é aquele que deve ser obrigatoriamente encerrado para que o aplicativo continue: enquanto ele estiver ativo, nenhuma outra janela ou aplicativo pode ser acessado. Para indicar se o formulário deve ser exibido como modal usamos a constante de sistema vbModal:

```
frmMeuForm.Show vbModal
```

Se a modalidade for omitida, será assumida como não-modal.

Método Refresh

Durante a execução do sistema, às vezes acontece uma situação que impede a exibição correta do formulário. Normalmente, isso ocorre quando alteramos um objeto de um formulário que já está sendo exibido em uma procedure externa a ele, ou então quando estamos executando ações que demandam uma certa quantidade de recursos do sistema, como na abertura de arquivos. O método Refresh corrige esse problema simplesmente provocando a reconstrução imediata do formulário. Outros objetos, como os controles Data, também aceitam o método Refresh.

Comando Unload

O comando Unload fecha um formulário e devolve o controle do sistema ao formulário que o chamou. Sua sintaxe é muito simples: basta indicar o nome do formulário. Exemplo:

```
Unload frmMeuForm
```

Eliminando formulários do projeto

Para excluir um formulário do projeto, basta clicar com o botão direito do mouse sobre o nome do formulário na janela Projeto e escolher a opção *Remove <nome do formulário>* no menu de contexto.

Formulários MDI

Um formulário MDI (*Multiple-Document Interface*) funciona como uma janela que contém outros formulários a ela subordinadas (chamados de **MDIChild**), atuando em *background* para uma aplicação. Para criar um formulário MDI você deve escolher a opção *Add MDI Form* no menu *Project* do VB. Por suas características, o formulário principal de uma aplicação é freqüentemente criado como MDI. Veja as principais:

- Uma aplicação pode conter apenas um formulário MDI, mas pode conter vários *MDI Child Forms*;
- Se o formulário MDIChild contém uma estrutura de menus, a barra de menu do formulário MDI é automaticamente atualizada e o menu exibido passa a ser o do MDIChild ativo. Falaremos mais sobre menus logo adiante;
- Se o formulário MDI é minimizado, todos os formulários MDIChild abertos também são;
- Se um formulário MDIChild é minimizado, aparece como um ícone no rodapé do MDI;
- A maioria das propriedades, métodos e eventos dos formulários MDI são os mesmos de um formulário comum, mas um MDI não pode ser aberto como modal;
- Um formulário MDI só pode conter menus e o controle *PictureBox*, ou então controles customizados preparados para isso. Você pode usar outros controles no formulário MDI apenas se criar uma *PictureBox* e então inserir esses controles dentro dela, como se fosse uma “moldura”;
- Um formulário MDIChild não aceita a propriedade StartUpPosition.

Propriedade MDIChild

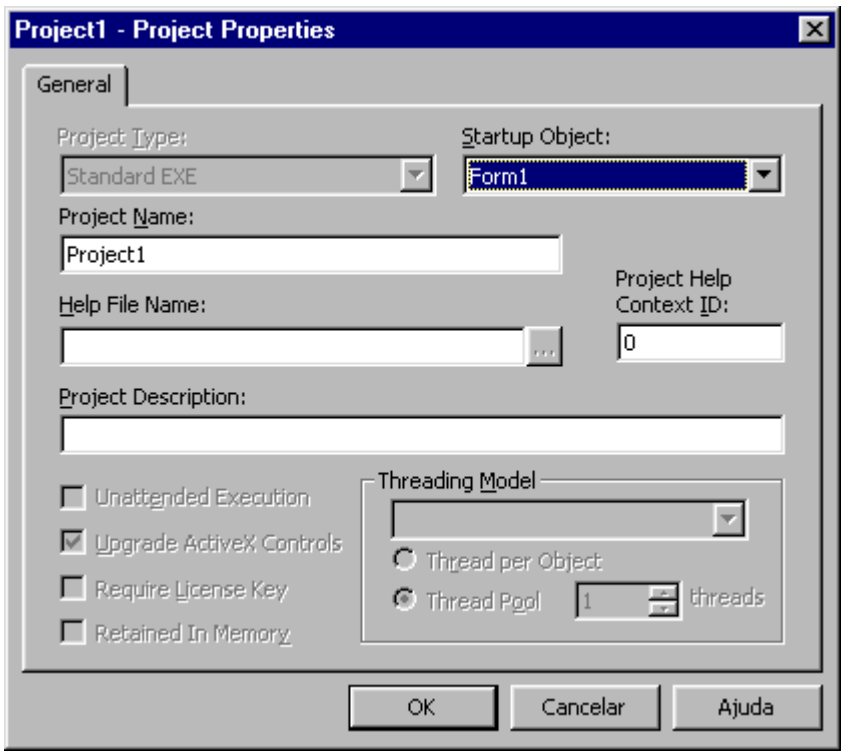
Essa propriedade indica se um formulário será exibido como MDIChild ou não. A propriedade MDIChild só tem efeito quando existe no projeto um formulário MDI. Nesse caso, qualquer outro formulário que não o MDI (obviamente) poderá ser configurado como MDIChild.

Quando o valor de MDIChild é True, o formulário poderá ser maximizado, minimizado ou movido dentro do formulário principal (MDI). Assim como o MDI, um formulário MDIChild não pode ser aberto como modal. MDIChild não pode ser alterada em modo de execução, e o valor padrão é False.

Startup Object

O objeto de abertura (ou *Startup Object*) é o primeiro objeto a ser executado quando iniciamos um aplicativo. Quando criamos um novo projeto, o formulário Form1 é definido como objeto de abertura, por ser o primeiro objeto a fazer parte dele. Mas qualquer formulário pode ser configurado como *Startup*. Para alterar o objeto de abertura, siga as instruções a seguir:

- Abra o menu *Project*;
- Escolha a opção *Project Properties*: será exibida a janela *Project Properties*:



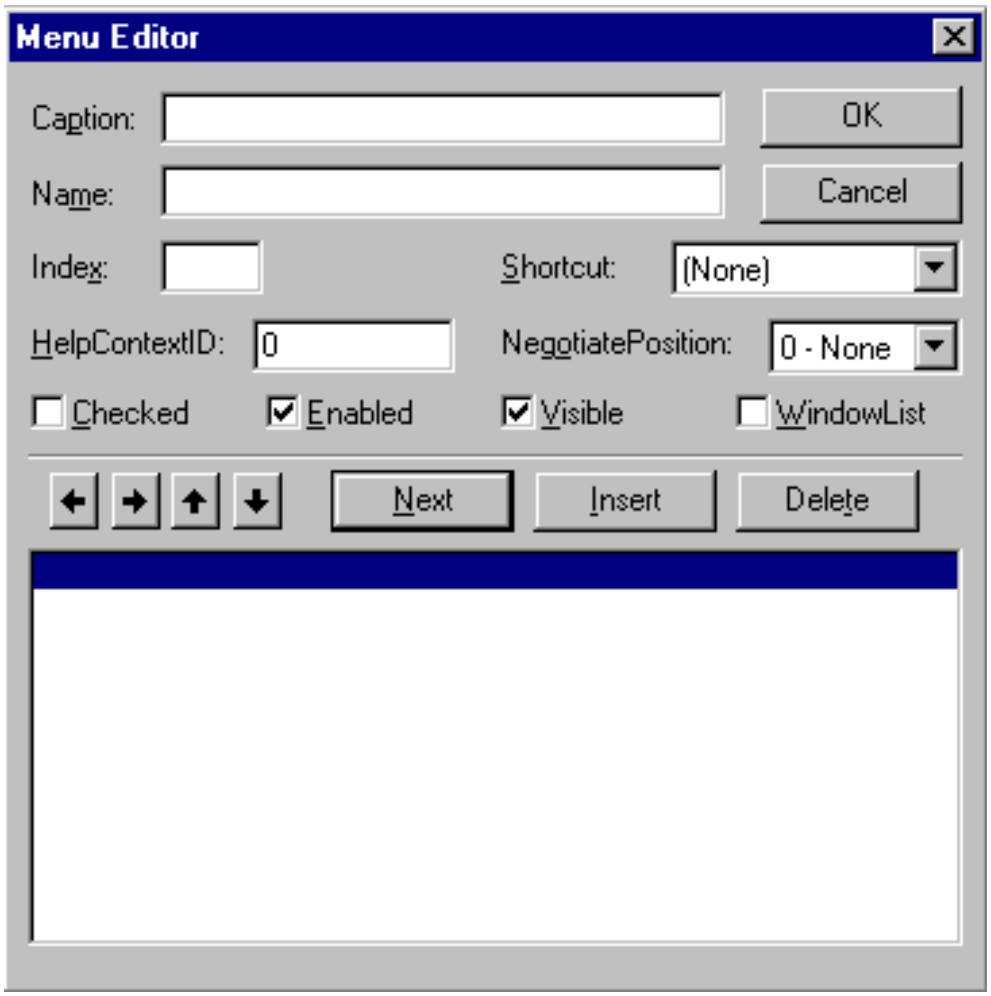
- A caixa de combinação *Startup Object* contém os objetos que podem ser configurados como *Startup*: note que só são exibidos os formulários contidos no projeto (além da sub Main, que estudaremos mais adiante);
- Indique na combo qual será o objeto de abertura e clique em Ok.

Menus

A barra de menus é um dos mais importantes componentes de um aplicativo Windows. O VB dispõe de uma poderosa e prática ferramenta para o projeto de menus: o Editor de Menu (*Menu Editor* – veja a figura da próxima página). Com ele, em poucos passos você pode criar menus que funcionam exatamente como nos aplicativos mais conhecidos.

Existem algumas regras de padronização que devem ser seguidas na criação de menus para os aplicativos Windows:

- O usuário já sabe o que esperar em determinados menus, quando presentes em um aplicativo. No menu Editar, por exemplo, ele espera encontrar ao menos as opções Copiar, Recortar e Colar. Outros menus também possuem opções padronizadas: Abrir, Salvar e Imprimir são freqüentemente encontradas no menu Arquivo;
- As opções da barra de menus devem permitir o acesso através da combinação Alt+Letra Sublinhada;
- Para as opções mais comuns é interessante que existam teclas de atalho. Mas, cuidado: algumas teclas de atalho também são padronizadas no Windows. Por exemplo: Ctrl+C é o atalho para copiar algo para a área de transferência, não importa qual seja o aplicativo;
- Itens relacionados devem ser agrupados e separados dos demais por barras separadoras. Exemplo: as opções Localizar, Localizar Próximo e Substituir do menu Editar do WordPad;



A janela Menu Editor

- Quando o item leva o usuário a uma caixa de diálogo, seu rótulo deve ser seguido de reticências (**...**). Quando leva a um submenu, deve ser seguido de uma seta ▶;
- Itens selecionáveis, como a opção Régua do menu Exibir do WordPad, devem ser precedidos pela marca de seleção √ quando estiverem ativos;
- Quando algum pré-requisito para o funcionamento de uma opção não estiver sendo cumprido, ela deve aparecer desabilitada. Por exemplo: se não houver texto selecionado, as opções Copiar e Recortar não devem estar disponíveis.

Para acessar a janela *Menu Editor*, o formulário deve estar selecionado. Escolha então a opção no menu *Tools* do VB, acione a combinação Ctrl+E ou escolha o botão *Menu Editor* na barra de ferramentas.

Vamos descrever alguns componentes da janela Menu Editor:

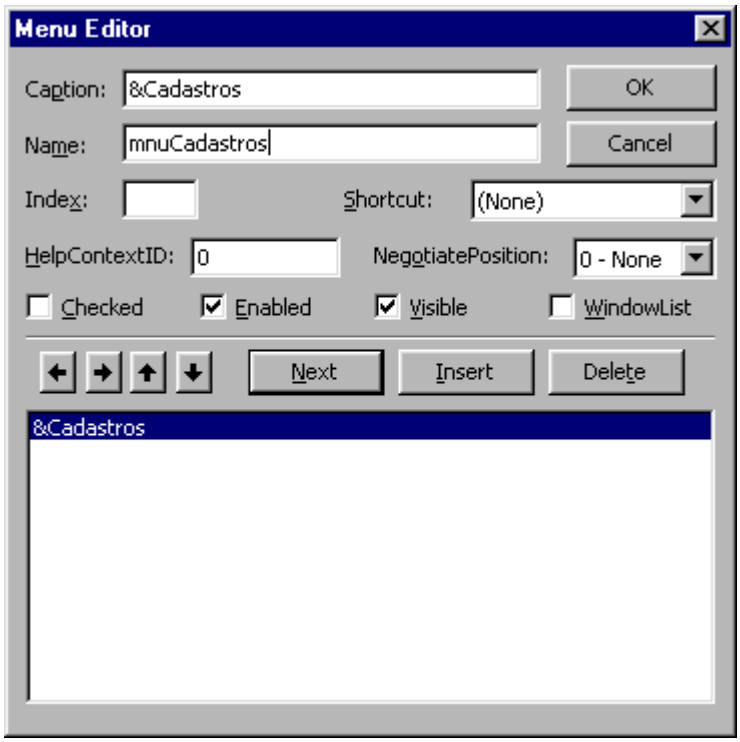
- **Caption**: neste campo deve ser digitado o rótulo que aparecerá na barra de menus ou dentro do menu: corresponde à propriedade Caption de um controle;
- **Name**: neste campo informamos o valor da propriedade Name do menu;
- **ShortCut**: permite a definição de uma tecla de atalho;
- **Checked**: permite a definição da propriedade Checked (marca de seleção);
- **Enabled e Visible**: o mesmo que as propriedades Enabled e Visible dos controles;
- **Index**: quando o menu fizer parte de uma matriz, permite indicar seu índice;
- **Botões Up e Down**: permitem a movimentação do menu selecionado para cima ou para baixo, mudando sua posição na barra;
- **Botões Left e Right**: permitem a mudança de nível do menu, ou seja: se a opção aparecerá na barra de menu ou se fará parte de um submenu. Podem ser criados até quatro níveis;

- **Botão Next**: move a seleção para a próxima linha, se houver. Se não houver, insere uma nova linha no final da lista;
- **Botão Insert**: insere uma nova linha acima da selecionada;
- **Botão Delete**: exclui da lista a linha selecionada.

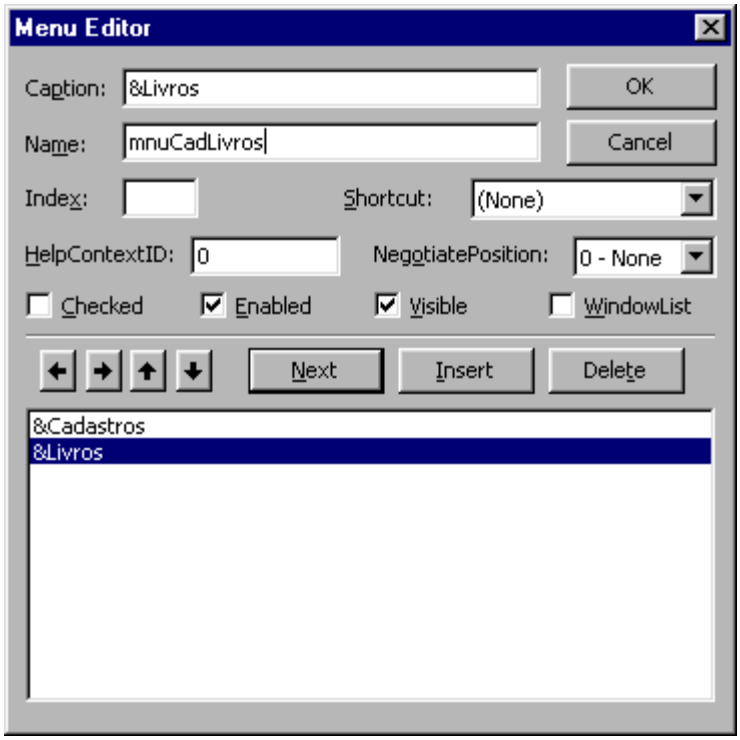
O Aplicativo Bibliotecário

Para exercitar os vários assuntos que veremos nessa lição, vamos iniciar a construção no nosso sistema, que chamaremos de Bibliotecário:

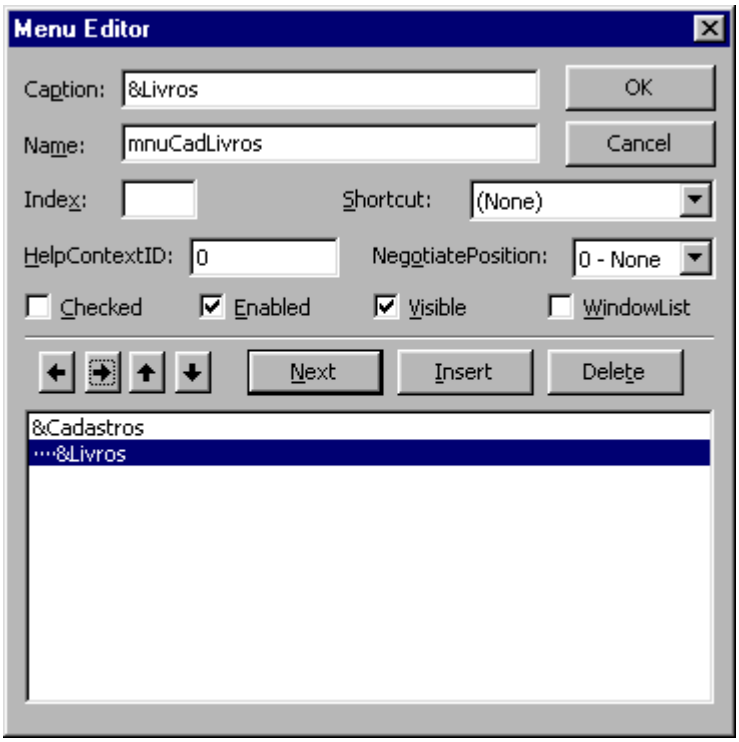
1. Inicie um novo projeto;
2. Remova a formulário Form1 do projeto;
3. Adicione ao projeto um formulário MDI, e altere suas seguintes propriedades:
   - *Name*: frmBiblio
   - *Caption*: Bibliotecário – Controle de Acervo de Bibliotecas
   - *BackColor*: Amarelo Claro (ou outra de sua preferência – mas não escolha uma cor muito forte para não ofuscar o coitado do usuário);
   - *Icon*: Writing\Books01.ICO;
   - *WindowState*: 2 - Maximized
4. Configure-o como *Startup Object*. Aproveite que já está na janela *Project Properties* e altere também a propriedade *Project Name* para **Bibliotecário**. Essa propriedade equivale a um rótulo a ser exibido na janela *Project* para identificação do projeto, mas sua definição não é obrigatória;
5. Salve o formulário com o nome de **FrmBiblio** e o projeto como **Bibliotecario**;
6. Abra a janela *Menu Editor*;
7. Defina os campos para a primeira opção do nosso menu: Caption = &Cadastros e Name = mnuCadastros;
8. Clique no botão Next para passar para a próxima linha:



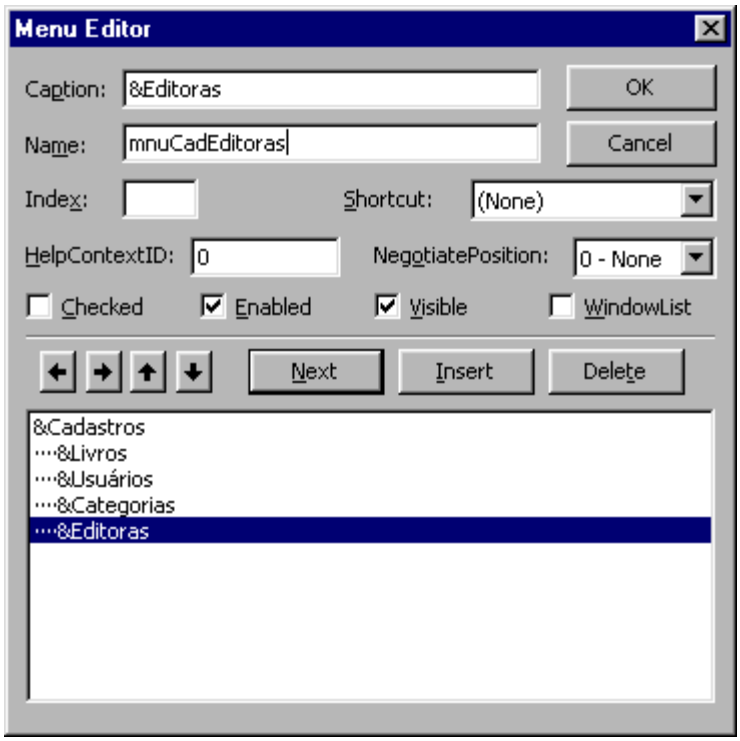
9. O próximo item é a opção Livros do menu Cadastros:
   - *Caption*: &Livros
   - *Name*: mnuCadLivros



10. Clique no botão Ok e veja o resultado. Note que as opções estão no mesmo nível, pois a opção Livros está ao lado de Cadastros. O correto seria que Livros fosse um item do menu Cadastros;
11. Acione novamente o Menu Editor e selecione a linha que contém o menu Livros;
12. Clique na seta para a direita. O nível do menu Novo foi alterado, e agora está abaixo do menu Cadastro:



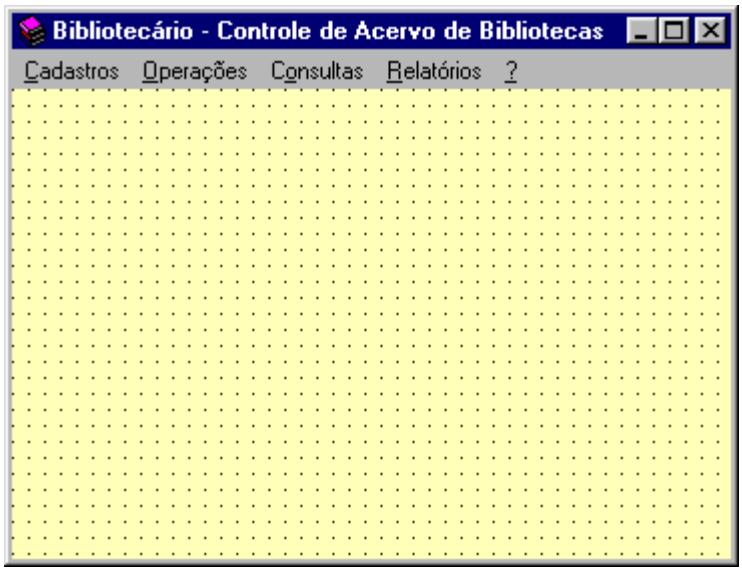
13. Clique no botão Ok e observe se o resultado agora é o esperado;
14. Inclua mais dois três opções ao menu Cadastros:
    - *Caption*: &Usuários — *Name*: mnuCadUsuarios
    - *Caption*: &Categorias — *Name*: mnuCadCategorias
    - *Caption*: &Editoras — *Name*: mnuCadEditoras



15. Vamos incluir agora uma barra separadora. Selecione a opção Editoras e clique no botão Next;
16. Indique para a propriedade Caption um hífen (-), e altere Name para BS1 (de Barra Separadora nº 1). Clique em Next;
17. Para terminar o menu Cadastros, vamos incluir nele uma opção para sair do sistema:
    - *Caption*: &Sair do Sistema
    - *Name*: mnuSair
18. Clique no botão Ok e confira o menu. Faça correções, se necessário;
19. Adicione as demais opções do menu de acordo com a tabela abaixo:

Caption:	***Name:***
&Operações	mnuOperacoes
....&Empréstimo de Livros	mnuEmprestimos
....&Devolução de Livros	mnuDevolucoes
C&onsultas	mnuConsultas
....&Livros	mnuConLivros
........&Todos	mnuConTodos
........-	BS2
........por A&utor	mnuLivrosPorAutor
........por &Categoria	mnuLivrosPorCategoria
........por E&ditora	mnuLivrosPorEditora
........-	BS3
........&Emprestados	mnuConEmprestados
........Em &Atraso	mnuConAtrasados
....&Usuários	mnuConUsuarios
....&Categorias	mnuConCategorias
....&Editoras	mnuConEditoras
&Relatórios	mnuRelatórios
....&Livros	mnuRelLivros
....&Usuários	mnuRelUsuarios
....&Categorias	mnuRelCategorias
....&Editoras	mnuRelEditoras
&?	mnuHelp
....&Sobre o Bibliotecário	mnuSobre

20. Verifique o funcionamento do menu. Note que na barra devem constar cinco opções: Cadastros, Operações, Consultas, Relatórios e ?. Além disso, nos menus Consultas e Relatórios, o item Livros leva a submenus. Seu formulário agora deve estar parecido com o do exemplo a seguir:



21. Faça ajustes, se necessário;
22. Salve o formulário como **frmBiblio** e o projeto como **Bibliotecario**.

Associando Codificação a um Menu

Um menu se comporta como outro objeto qualquer do VB, quanto à codificação. Só um evento é associado a um menu: o Click. Para exemplificar, vamos fazer a codificação do menu mnuSair:

1. Em tempo de projeto, clique sobre o menu Cadastros para abri-lo;
2. Clique sobre a opção Sair (não é necessário o duplo-click);
3. A janela Código é aberta. Escreva a procedure correspondente:

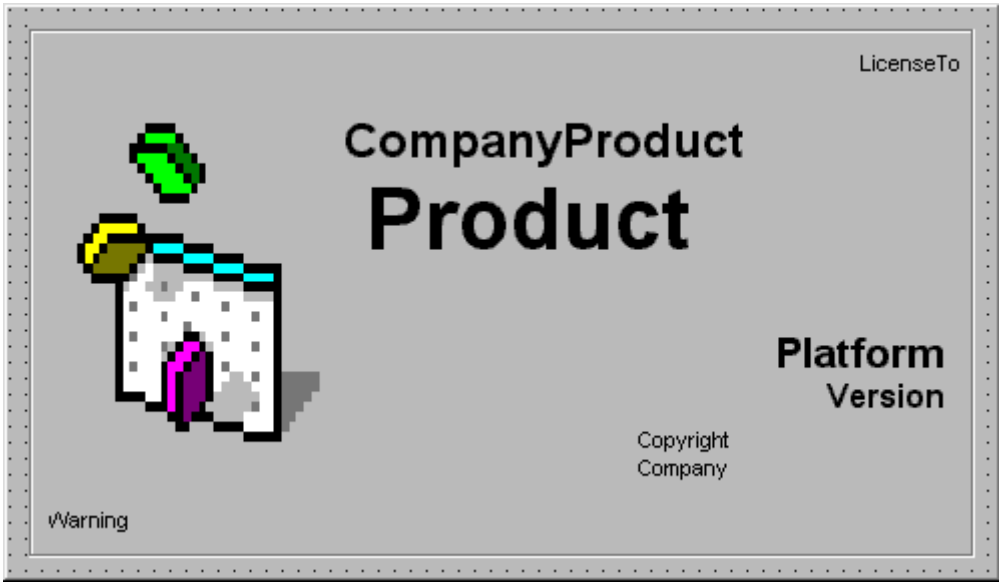
```
Private Sub mnuSair_Click()
Dim vOk as Integer
      vOk = MsgBox("Confirma o encerramento do sistema?", _
            vbYesNo + vbQuestion + vbApplicationModal, "Saída")
      If vOk = vbYes Then End
End Sub
```

4. Execute o aplicativo e verifique o funcionamento da procedure.

Formulários Splash

A maioria dos aplicativos para Windows (como o Word e o próprio VB) exibe uma janela com diversas informações sobre o produto enquanto ele é carregado. Essa janela é chamada de *Splash Window*, ou tela de Splash. O VB 6 permite a criação dessas telas com muita facilidade. Para esses casos, o VB nos oferece um modelo de formulário pronto, configurado e com a programação necessária para funcionar como aqueles dos aplicativos comerciais. Vamos então criar um formulário Splash em nosso aplicativo:

1. Abra a janela *Add Form*;
2. Lembra-se que na aba *New* há vários tipos disponíveis de formulários? Um deles é chamado de *Splash Screen*. É essa opção que usaremos para criar nossa tela de Splash. Escolha *Splash Screen*, e o formulário abaixo será adicionado ao projeto:



3. Você pode alterar o tamanho do formulário e seus objetos à vontade, incluir novos labels e figuras, apagar os que não forem necessários, enfim, deixar o formulário ao seu gosto. Veja como ficou o meu:

Obs.: os formulários padronizados, como os que estamos usando, quando inseridos no projeto normalmente já trazem algum tipo de programação pré-definida. Deve-se então verificar essa programação e alterá-la, ou apagá-la se não for necessária, para evitar erros da execução.

4. Grave o novo formulário como **frmSplash**.

O Controle Timer

Existe um problema em nosso projeto: não existe em frmSplash nenhum objeto que possa ser usado para acionar o frmBiblio, como um botão de comando. Mas o VB oferece um controle cujo funcionamento se encaixa perfeitamente em casos como esse: o controle Timer, que é usado em situações de execução de um ou mais comandos a intervalos de tempo regulares. Por exemplo: atualizar os registros exibidos em uma tela de consulta de um sistema multiusuário a cada 10 segundos. O controle Timer só responde ao evento Timer.

Propriedade Interval

A propriedade Interval do controle Timer determina o intervalo de tempo, medido em milisegundos, da ocorrência do evento Timer. É do tipo Integer, portanto o maior valor possível é 65.535, o que equivale a pouco mais de 1 minuto. O valor default é 1.000 milisegundos, ou 1 segundo. Esse controle pode ser posicionado em qualquer lugar do formulário, pois não é exibido durante a execução do aplicativo.

Em nosso sistema, usaremos um Timer para provocar uma pausa de 5 segundos, durante a qual será exibida nossa tela de Splash:

1. Adicione um Timer ao frmSplash;
2. Altere o valor da propriedade Interval do Timer para 5000;
3. Dê um duplo click sobre o controle Timer e faça a codificação do evento Timer conforme segue:

```
Private Sub Timer1_Timer()
      Unload Me
      frmBiblio.Show
End Sub
```

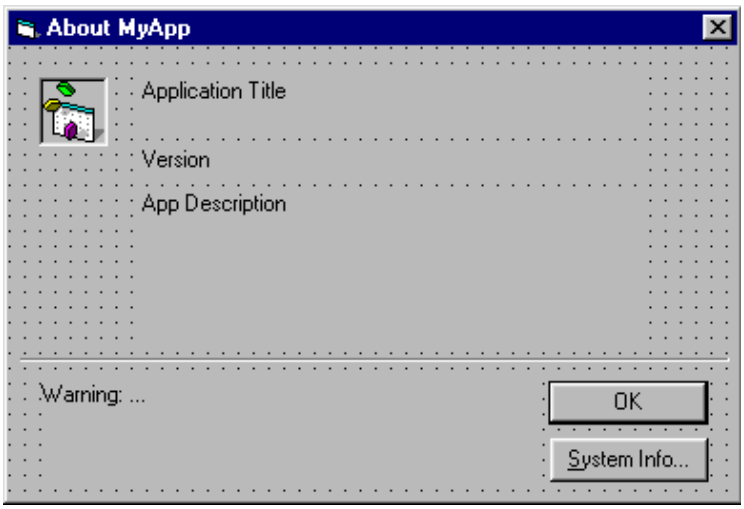
   Obs.: "Me" é equivalente ao nome do formulário em uso. Assim, ao invés de ficarmos escrevendo o nome do formulário para acessar suas propriedades e métodos durante a codificação, podemos escrever simplesmente "Me" em seu lugar que o resultado será o mesmo.

4. Execute o projeto e veja: seu aplicativo agora tem uma tela de Splash!
5. Grave seu trabalho.

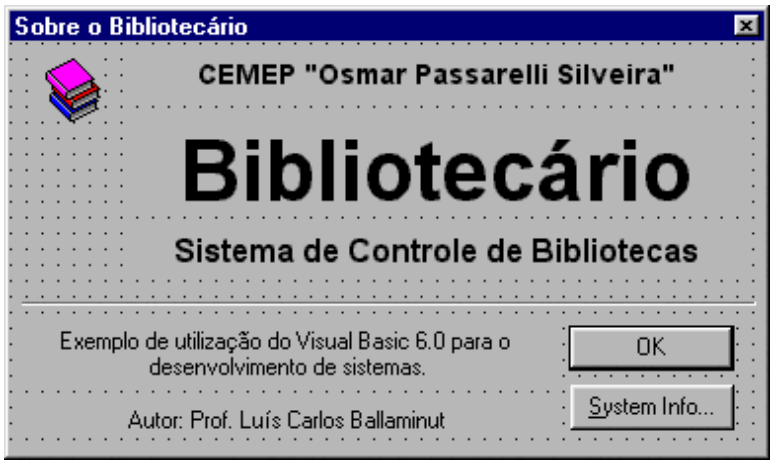
Formulários About

Outro modelo de formulário pré-definido que o VB nos oferece é o *About Form*, usado para a confecção daquelas janelas "Sobre...". Para adicioná-lo o processo é o mesmo do Splash. Vamos inserir no projeto um formulário do tipo About:

1. Abra a janela *Add Form* e escolha na aba *New* a opção *About Dialog*;
2. O formulário a seguir será inserido em seu projeto. Note os botões de comando: *Ok* fecha o formulário e *System Info* abre a janela de Informações do Sistema do Windows:



3. Modifique-o ao seu gosto, mas cuidado com o evento Load: ele altera o rótulo de alguns labels. Se você mudar esses labels, apague as linhas correspondentes a eles em Form_Load. Veja como ficou o meu:



4. Grave o formulário como **frmAbout**;

5. Abra o formulário frmBiblio e escreva o comando para exibir frmAbout em mnuSobre_Click:

```
Private Sub mnuSobre_Click()
    frmAbout.Show vbModal
End Sub
```

6. Execute o aplicativo. No formulário principal, escolha a opção Sobre no menu e veja se o formulário frm-About está funcionando corretamente;

7. Grave seu trabalho. Continuaremos o desenvolvimento do sistema na próxima lição.

Lição 7: Controles Customizados

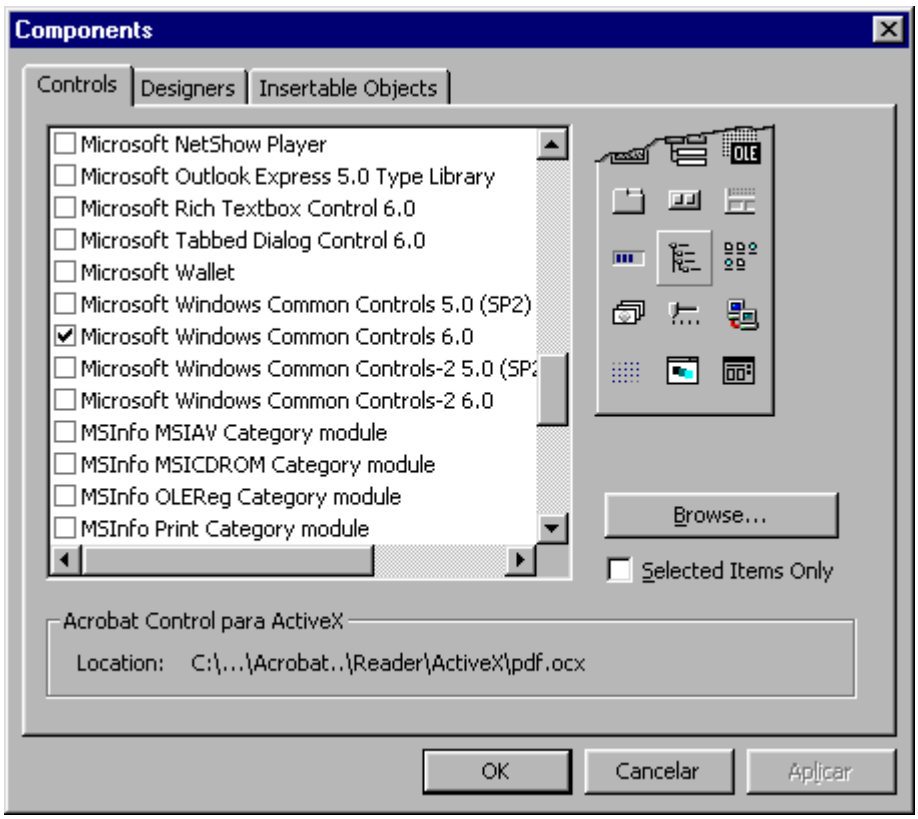
Até agora só usamos em nosso projeto controles padronizados, aqueles que aparecem na caixa de ferramentas quando iniciamos o VB. Mas não são apenas esses os controles disponíveis para uso em nossos projetos. Muito pelo contrário, existe uma infinidade de outros controles no mercado, muitos deles comerciais e desenvolvidos por empresas especializadas. Alguns desses controles adicionais, chamados de **customizados**, são fornecidos junto com o pacote do VB e podem ser usados livremente na construção de aplicativos.

Os controles customizados não aparecem na caixa de ferramentas: eles devem ser adicionados ao projeto para serem usados. Uma vez que isso é feito, porém, eles se comportam da mesma maneira que qualquer controle padronizado. Mas não vá adicionando controles à vontade em seu projeto, pois, como são arquivos OCX separados, para cada um deles que você usar seu aplicativo vai "engordar" alguns Kbytes. Assim sendo, só adicione aqueles controles que realmente irá usar.

Adicionando um Controle Customizado

Vamos então adicionar um controle customizado ao nosso aplicativo. Se ainda não o fez, inicie o VB e abra o projeto Bibliotecario:

1. No menu *Project* do VB, escolha a opção *Components*, ou então tecle Ctrl+T: a janela *Components* será aberta: ela permite que controles customizados sejam adicionados ao projeto:



2. Procure e marque a opção *Microsoft Windows Commom Controls 6.0*;
3. Clique no botão Aplicar: repare que alguns ícones novos, como os da figura ao lado, foram inseridos na caixa de ferramentas;
4. Clique em Ok. A partir de agora, você poderá usar qualquer um dos *Microsoft Windows Commom Controls* em seu projeto como qualquer outro controle.

Criando Uma Barra de Ferramentas

Sem dúvida um dos recursos mais comuns nos aplicativos para Windows é a barra de botões ou de ferramentas - o próprio VB as tem em profusão. A principal finalidade dessas barras é a de facilitar o acesso do usuário aos comandos mais comuns do aplicativo.

O Controle ToolBar

Um dos controles customizados que acabamos de adicionar é o ToolBar, que permite a criação de barras de botões com muita facilidade. Ele responde ao evento Click, e a identificação do botão clicado é feita pelo seu número de índice.

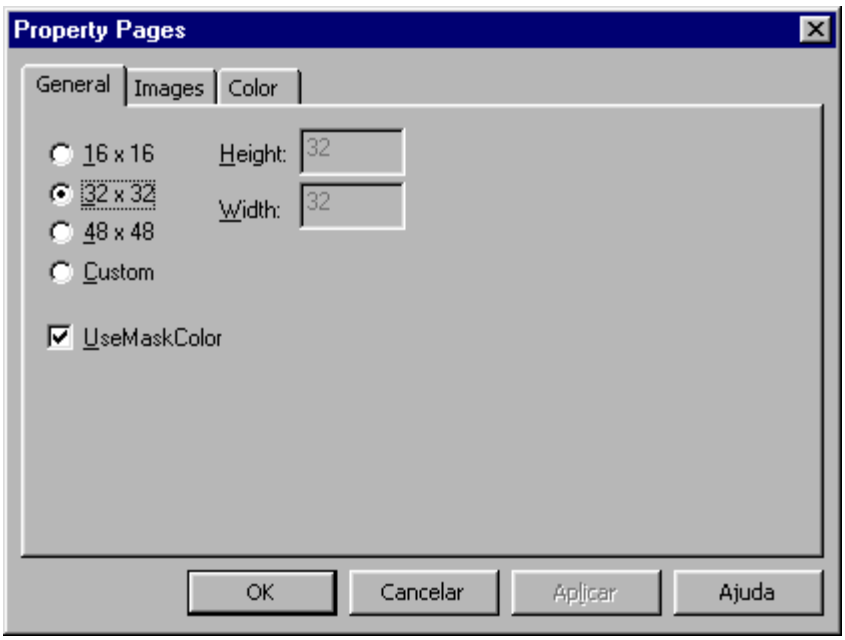
Para criar os botões, ToolBar trabalha em conjunto com outro controle:

O Controle ImageList

O controle ImageList também faz parte dos *Microsoft Windows Commom Controls* e, como seu próprio nome indica, serve para criar uma lista de imagens que será associada a um outro controle, como uma ToolBar. Esse controle não responde a nenhum evento e não aparece no formulário durante a execução do aplicativo. Depois que a ImageList é associada a um controle, não pode mais ser alterada, a menos que a associação seja excluída.

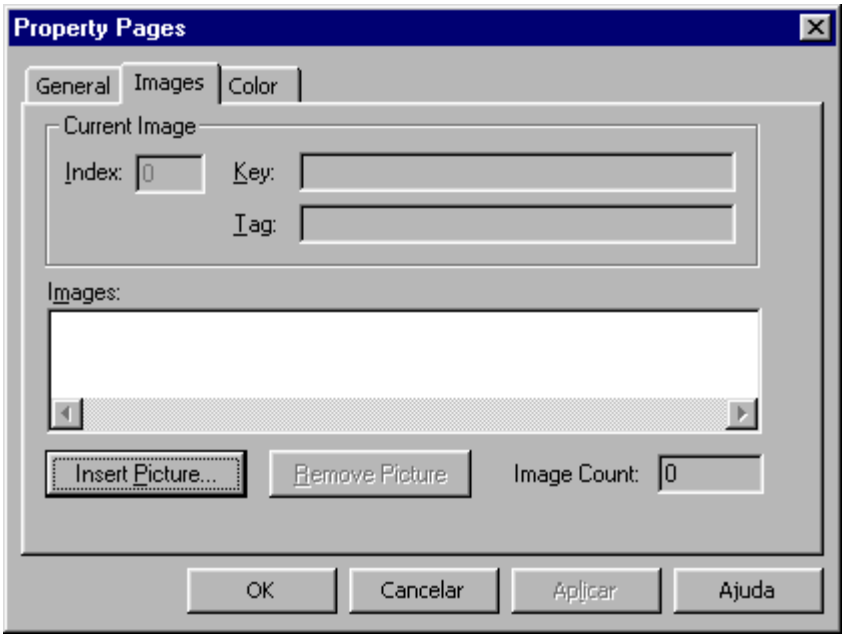
Vamos então criar uma barra de botões em nosso projeto:

1. Em tempo de projeto, abra o formulário frmBiblio;
2. Adicione ao formulário uma ImageList. Não se preocupe com a posição em que ela vai ficar, pois como já disse ela não aparecerá durante a execução do aplicativo;
3. Clique com o botão direito do mouse sobre a ImageList;
4. No menu de contexto que se abre, escolha a opção *Properties*. A janela de propriedades da ImageList será exibida:

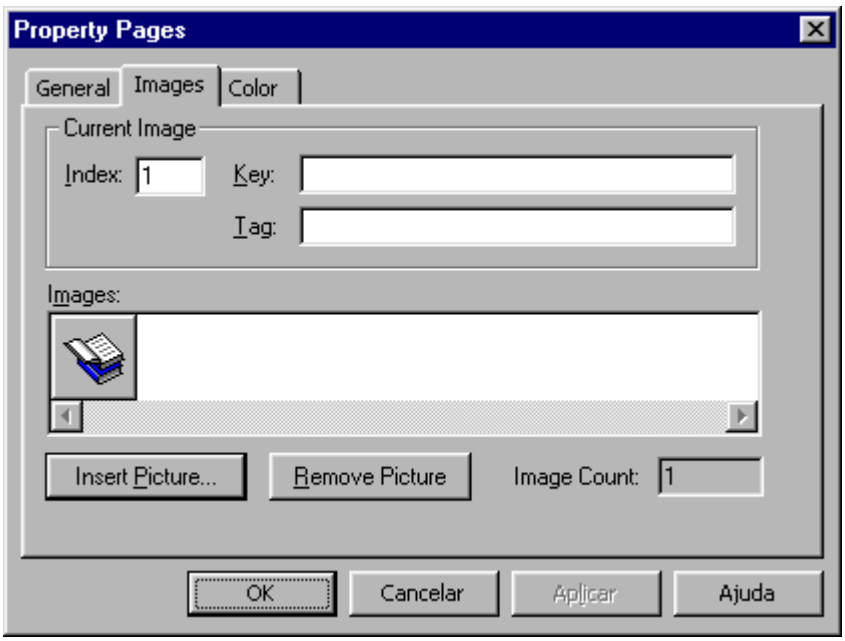


5. Na aba *General* definimos o tamanho em pixels das imagens da lista. Compare os tamanhos:
   - 16 x 16: 
   - 32 x 32: 
   - 48 x 48: 

6. Na aba *Images* inserimos as imagens que farão parte da lista:

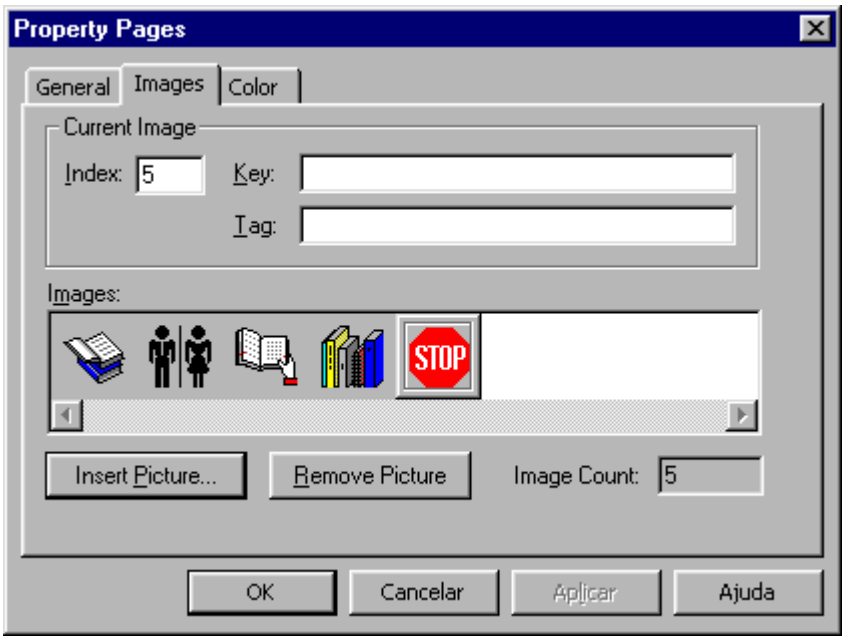


7. Para adicionar imagens, clicamos no botão *Insert Picture* e para excluir uma imagem no botão *Remove Picture*. Clique em *Insert Picture* e adicione o ícone Writing\Books02.ICO:

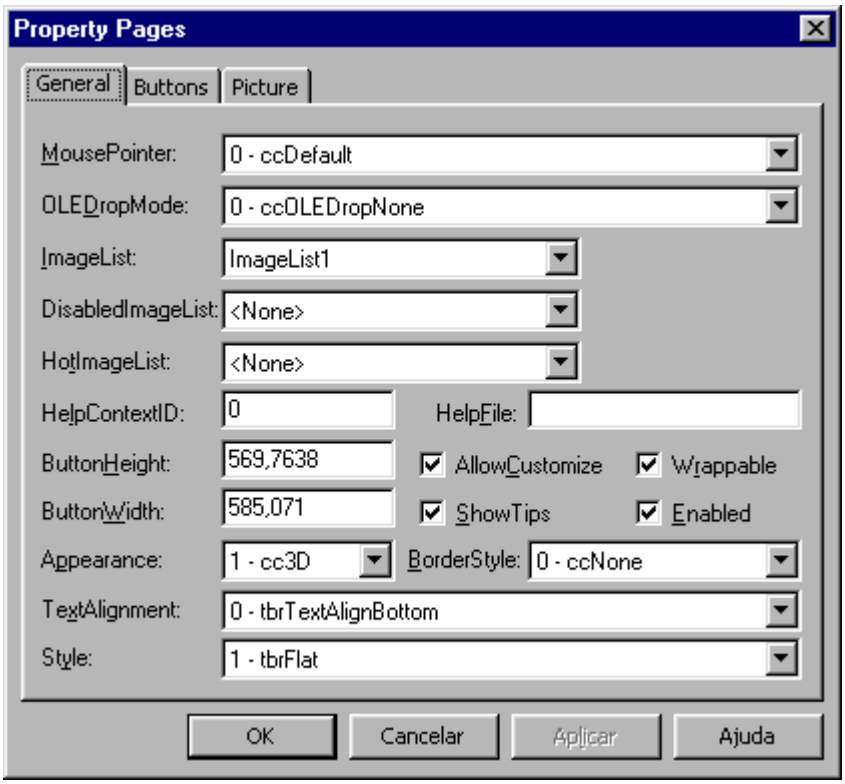


Obs.: note que, após adicionar o ícone, o campo *Index* passou a indicar o valor 1. Esse é o número de índice da imagem na lista, através do qual fazemos a associação da imagem com o botão da ToolBar.

8. Adicione as seguintes imagens na lista. Procure respeitar a ordem na inclusão, para que o índice corresponda corretamente à imagem desejada quando estivermos criando a barra de botões:
   - Misc\Misc28.ICO (Index 2)
   - Writing\Book04.ICO (Index 3)
   - Writing\Books04.ICO (Index 4)
   - Traffic\Trffc14.ICO (Index 5)
9. Se você fez tudo corretamente, sua ImageList deve estar parecida com o exemplo a seguir:



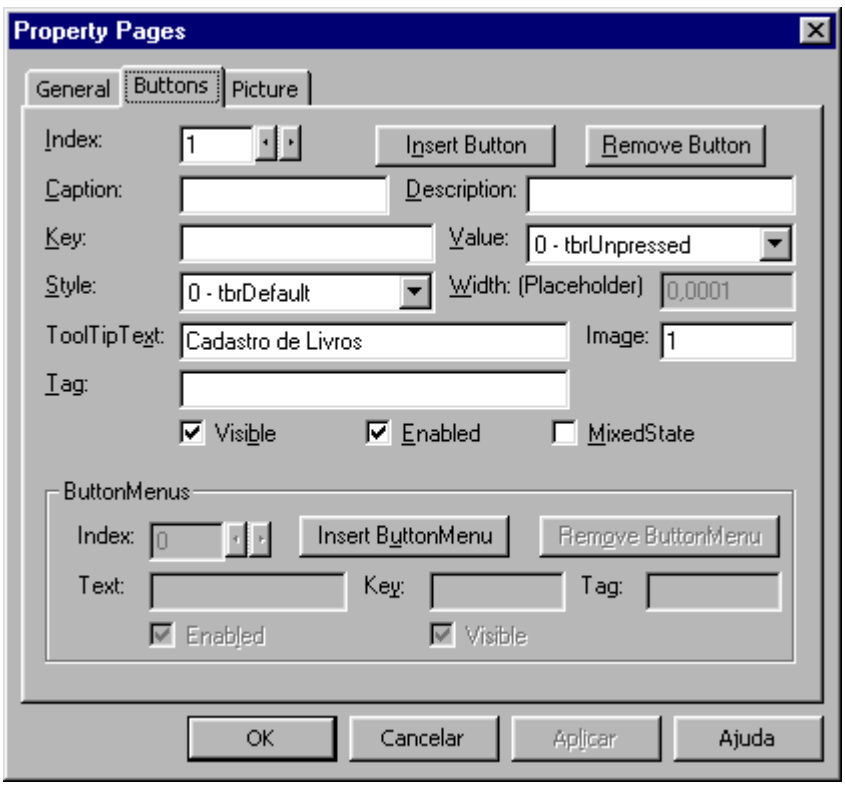
10. Clique no botão Ok para retornar ao formulário;
11. Adicione ao formulário uma ToolBar. Note que esse controle é automaticamente posicionado no topo da área de trabalho do formulário, logo abaixo da barra de menus;
12. A configuração da ToolBar é feito como na ImageList: clique com o botão direito do mouse sobre o controle e escolha a opção *Properties*;
13. Na aba *General* definimos as propriedades que determinam a aparência da Toolbar. As principais são:



- **ImageList**: indica o nome da ImageList que será usada na criação da barra de botões. Escolha a opção "ImageList1", que deve ser a única disponível;
- **Appearance**: determina a aparência da barra, que pode ser em perspectiva 3D (opção **1 - cc3D**) ou integrada à área de trabalho do formulário (opção **0 – ccFlat**). Escolha 3D;
- **BorderStyle**: indica se a barra terá borda (opção **1 – ccFixedSingle**) ou não (opção **0 – ccNone**). A aparência final da borda depende da propriedade Appearance. Escolha None;

- **Style**: indica o estilo dos botões da barra: pode ser **0 – tbrStandard**, em que eles serão apresentados como pequenos botões de comando, ou **1 – tbsFlat**, em que os botões terão a mesma aparência das barras de ferramentas do VB. Escolha Flat.

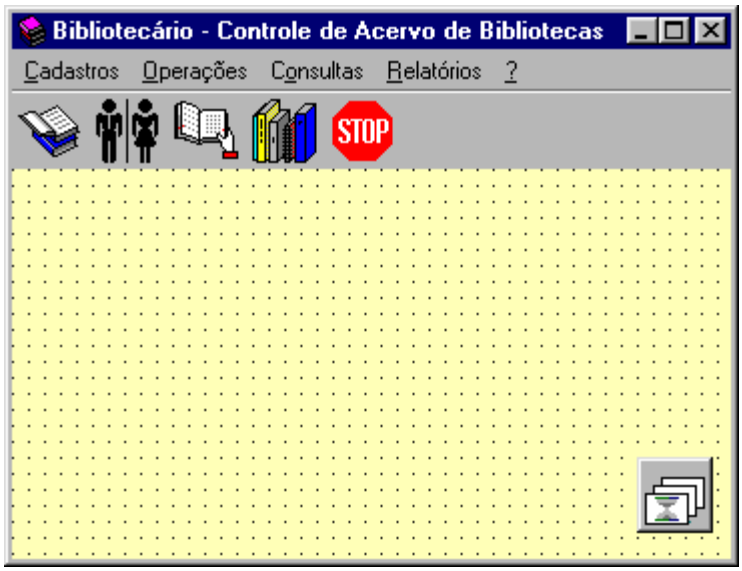
14. Na aba *Buttons* configuramos cada um dos botões da barra. Para inserir um botão, clicamos em *Insert Button*, e para remover clicamos em *Remove Button*. Clique no botão *Insert Button*;
15. Vamos definir as propriedades do botão que inserimos. As principais propriedades dos botões são:
    - **Index**: indica o índice do botão na barra, pelo qual ele será identificado quando for clicado. Normalmente não alteramos o valor essa propriedade;
    - **Caption**: define um rótulo para o botão. Deixe em branco, assim só será exibido o ícone no botão;
    - **ToolTipText:** texto a ser exibido em uma pequena etiqueta explicativa quando o ponteiro do mouse ficar parado por alguns instantes sobre o botão. Escreva "Cadastro de Livros" nessa propriedade (obs.: quase todos os controles do VB possuem essa propriedade);
    - **Image**: indica o número da imagem da ImageList a ser exibida por esse botão. Indique o valor 1, que corresponde ao primeiro ícone da ImageList;
    - **Visible** e **Enabled**: o mesmo que as propriedades Visible e Enabled dos demais controles.
16. Sua configuração do botão 1 deve estar assim:



17. Insira mais quatro botões na barra, conforme a tabela a seguir:

Botão	***Propriedades = Valor***
2	ToolTipText = Cadastro de Usuários Image = 2
3	ToolTipText = Empréstimo de Livros Image = 3
4	ToolTipText = Devolução de Livros Image = 4
5	ToolTipText = Sai do Sistema Image = 5

18. Clique no botão Ok. Seu formulário deve estar parecido com o do exemplo a seguir:



19. Salve seu trabalho.

Programando a Barra de Ferramentas

A programação dos botões da barra de botões é muito simples: como a barra só responde ao evento Click, basta que identifiquemos qual botão foi clicado e fazer a codificação correspondente. Como já vimos, a identificação do botão é feita pela propriedade Index. Vamos então programar o botão Sair:

1. Dê um duplo click sobre a barra de botões para abrir a janela Código no evento Click da barra;
2. Note que a procedure recebe um parâmetro, chamado Button, que indica em que botão aconteceu o click. Testaremos então o índice desse parâmetro (propriedade Index) para saber qual foi o botão clicado – em breve estudaremos a propriedade Index mais detalhadamente:

```
Private Sub Toolbar1_ButtonClick(ByVal Button As MSComctlLib.Button)
      If Button.Index = 5
            'Foi clicado o botão Sair:
            mnuSair_Click
      End If
End Sub
```

3. Repare que, se o botão for o número 5, chamaremos a subrotina mnuSair_Click, ou seja, estamos desviando a execução para o evento Click do menu Sair, pois a programação em ambos os casos é a mesma. Isso pode ser feito à vontade, pois os eventos dos controles também são subrotinas como quaisquer outras, e portanto podem ser executadas sempre que necessárias;
4. Teste o aplicativo e verifique se o botão está funcionando como previsto;
5. Salve seu trabalho. Os demais botões da barra serão programados conforme os formulários a que eles se referem forem sendo construídos nas próximas lições.

Lição 8: Bancos de Dados

Sempre que trabalhamos com qualquer linguagem temos a intenção de guardar as informações geradas para a obtenção de algum resultado, ou então para consultá-las quando forem necessárias. Assim, elas devem ser gravadas em arquivos criados para esse fim, chamados de Bancos de Dados, ou serão perdidas após o encerramento do programa que as gerou. Existem vários softwares específicos para o gerenciamento desses arquivos no mercado, e cada um deles possui padrões próprios de armazenamento de dados. Exemplos de Sistemas Gerenciadores de Bancos de Dados (SGDB) são o SQL Server, o Dataflex, o FoxPro, o Oracle, o Access, etc.

O VB pode acessar arquivos de dados de vários formatos diferentes, incluindo os citados, mas vamos utilizar o MDB do Access. Esse é formato nativo do VB, tanto que o utilitário para a criação de bancos de dados que o acompanha, chamado *Visual Data Manager*, é uma versão reduzida do Access. Aliás, bastante reduzida: suas funções limitam-se à criação e manipulação de bancos de dados e de suas tabelas.

Nesse texto, assumimos que você já conhece o Access o suficiente para a criação do banco de dados, das tabelas e das consultas que vamos utilizar. Caso não o possua instalado em sua máquina, use então o *Visual Data Manager*, acessando-o pelo menu *Add-Ins* do VB.

Métodos de Acesso

Basicamente, um método de acesso é o meio pelo qual associamos um tipo especial de variável chamado de **variável objeto** ao banco de dados. Elas são chamadas de objeto porque possuem propriedades e respondem a métodos, permitindo assim a manipulação dos dados. O método de acesso usado tradicionalmente pelo VB é o DAO (ou *Data Access Object*), que consiste na abertura de suas tabelas e execução de comandos através das variáveis objeto. Porém, a partir da versão 6 do VB a Microsoft introduziu um novo método, chamado de ADO (ou *ActiveX Data Objects*), pelo qual o aplicativo se comunica com o gerenciador de banco de dados através de um provedor em uma conexão de rede, mesmo que a aplicação e o SGBD estejam rodando localmente. As operações de inclusão, exclusão, alteração e seleção de dados são executadas a partir de comandos SQL enviados pelo aplicativo ao SGBD através do provedor (veremos o que é SQL e como usá-la nas próximas lições). Em nosso projeto, usaremos o método ADO.

Objeto ADO Connection

Connection é a variável objeto que representa a conexão propriamente dita. Normalmente é declarada como pública (portanto, em um módulo de programação) com o tipo **ADODB.Connection**. Porém, antes de ver como criar uma conexão ADO, devemos estudar alguns conceitos básicos:

Propriedade ConnectionString

Essa propriedade do objeto Connection é uma string formada pelo nome do provedor usado e pelos parâmetros necessários para estabelecer a conexão com o SGBD.

O Provedor

A função do provedor é, como já disse, permitir a conexão do aplicativo com o banco de dados. Essa conexão é do tipo OLEDB, ou *Object Linking and Enbedding Database*. Portanto, na verdade o que acontece quando nos conectamos ao banco de dados via ADO é a vinculação de um objeto ao banco, que passa a responder às propriedades e métodos do objeto a ele vinculado. Esse vínculo se assemelha a uma conexão de rede, ou seja: o banco de dados não é um arquivo que pode ser manipulado diretamente, mas mantém um canal de comunicação com seu aplicativo pela qual as operações serão executadas. É como uma ligação telefônica: você pode conversar com as outras pessoas, mas não pode abraçá-las.

Existe um provedor específico para cada SGBD. No nosso caso, como usaremos uma base Access, o provedor será o **Microsoft Jet OLEDB Provider**. São necessários alguns parâmetros para se estabelecer a conexão, que variam de acordo com o provedor, mas no caso do Jet os parâmetros necessários são três:

- **Data Source**: nome e o caminho completo (*path*) do arquivo MDB a ser conectado;
- **User ID**: identificação do usuário;

- **Password:** senha do usuário.

Obviamente, os dois últimos podem ser omitidos se o banco não estiver protegido por senha.

Abrindo e fechando a conexão

Após definir a ConnectionString, podemos estabelecer a conexão aplicando o método Open ao objeto Connection. Veja um exemplo de conexão ADO usando o Jet:

```
Dim cnnTeste As New ADODB.Connection
cnnTeste.ConnectionString = "Provider=Microsoft.Jet.OLEDB.4.0;" & _
                            "Data Source=C:\Sistema\Dados.mdb;"
cnnTeste.Open
```

Obs.: 4.0 é a versão do Jet para o Access 2000.

Para fechar a conexão existe o método Close, mas ao invés de usá-lo é aconselhável a remoção da variável da memória usando a palavra-chave Nothing e o comando Set:

```
Set cnnTeste = Nothing
```

Objeto ADO Command

Um comando ADO é uma variável objeto que contém a definição do comando a ser executado pelo provedor no banco de dados. Ele é necessário para qualquer operação ou manipulação dos dados do banco (uma seleção de registros, por exemplo), pois é nele que definimos as configurações exigidas pelo provedor para executar o comando SQL correspondente. Entre essas configurações, devemos destacar as seguintes:

Propriedade ActiveConnection

Contém o nome da conexão ADO ativa a ser usada para execução do comando.

Propriedades CommandText e CommandType

CommandText é uma string que contém o comando a ser executado pelo objeto command, e CommandType indica seu tipo, ou melhor, como CommandText será avaliado em tempo de execução pelo objeto Command. CommandType pode ser um dos seguintes:

- **adCmdUnknown:** indefinido (default).
- **adCmdText:** string que representa um comando SQL válido;
- **adCmdTableDirect:** nome de uma tabela;
- **adCmdStoredProc:** nome de uma stored procedure (o equivalente a uma consulta ação do Access) ou de uma view (que equivale a uma consulta seleção);
- **adExecuteNoRecords:** indica que CommandText é um comando ou stored procedure que não retorna nenhum registro. Por exemplo, um comando de inserção de registros em uma tabela. Caso algum registro seja retornado, ele será desprezado. Sempre deve ser combinado com adCmdText or adCmdStoredProc.

Executando o comando

A execução do comando ADO é feita pelo método Execute. Veja um exemplo:

```
Dim cnnComando As New ADODB.Command
With cnnComando
      .ActiveConnection = cnnTeste
      .CommandType = adCmdStoredProc
      .CommandText = "spAtualizaSaldo"
      .Execute
End With
```

Note o uso das propriedades: como o tipo informado em CommandType é *adCmdStoredProc*, o provedor interpreta o texto contido em CommandText (“spAtualizaSaldo”) como sendo o nome de uma stored procedure, que será executada.

Objetos Recordset

Um *Recordset* é a representação lógica de um conjunto de registros, daí seu nome. Pode receber todos os registros de uma tabela, o resultado de uma consulta, ou ainda uma seleção de dados feita por um comando SQL. O recordset também é uma variável objeto, assim, para fazer a associação com o conjunto de registros desejado deve ser usado o comando Set e o método Execute em um comando ADO. Veja um exemplo:

```
Dim cnnComando As New ADODB.Command
Dim rsSelecao As New ADODB.Recordset
With cnnComando
      .ActiveConnection = cnnTeste
      .CommandType = adCmdTableDirect
      .CommandText = "Clientes"
      Set rsSelecao = .Execute
End With
```

Em nosso exemplo, o recordset rsSelecao será associado à tabela Clientes, e assim todos os seus registros e campos poderão ser acessados através de rsSelecao. Note que informamos que “Clientes” é o nome de uma tabela através do valor *adCmdTableDirect* em CommandType.

Para se referir a um campo de um *Recordset*, devemos usar o caractere “!”. Por exemplo: para atribuir o valor do campo Preço de um *Recordset* chamado rsProdutos a uma variável, poderíamos usar o seguinte comando:

```
vPreço = rsProdutos!Preço
```

Para fechar um Recordset, novamente usamos o comando Set e a palavra chave Nothing, da mesma maneira que fizemos com o Command:

```
Set rsSelecao = Nothing
```

Veja a seguir as principais propriedades e métodos do *Recordset*:

Propriedades BOF e EOF

Essas propriedades indicam se o apontador de registros está posicionado no início ou no fim da tabela. BOF retorna True se o apontador estiver no início da tabela, e EOF retorna True se estiver no final da tabela. Em qualquer outro caso, ambas retornam False.

Propriedade AbsolutePosition

Essa propriedade retorna o número do registro corrente na tabela.

Propriedade RecordCount

Retorna o número de registros que contidos no *Recordset*.

Métodos Move (MoveFirst, MoveLast, MoveNext e MovePrevious)

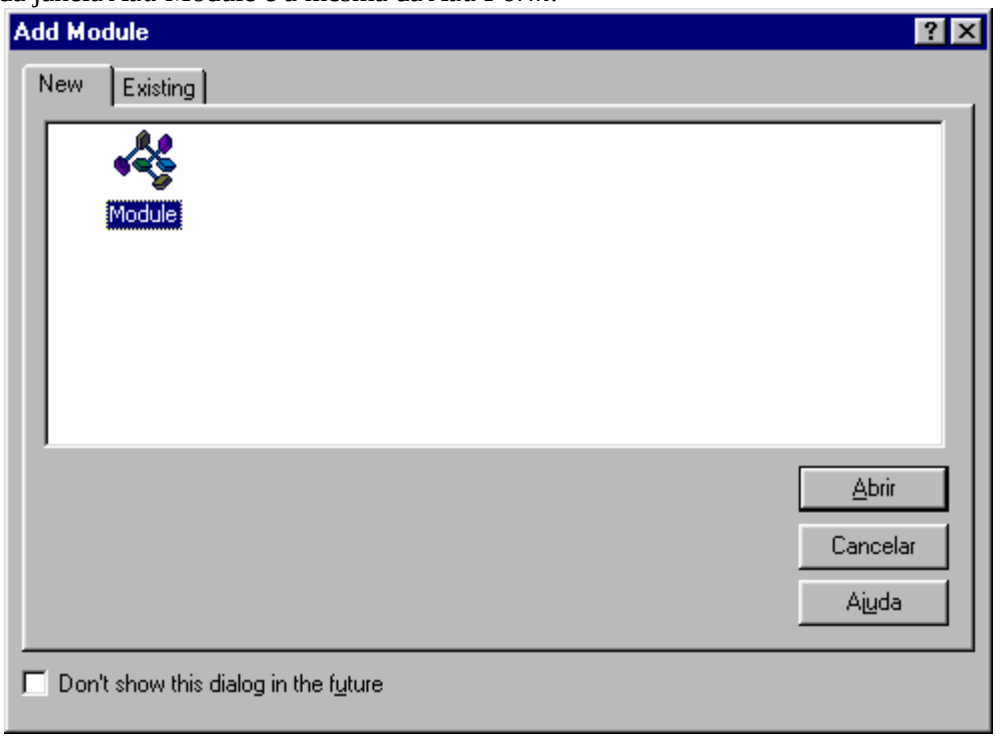
Esses métodos fazem a movimentação seqüencial do apontador de registros dentro do *Recordset*:

- **MoveFirst** move o apontador para o primeiro registro do *Recordset*;
- **MoveLast** move o apontador para o último registro do *Recordset*;
- **MoveNext** move o apontador para o próximo registro;
- **MovePrevious** move o apontador para o registro anterior.

Módulos de Programação

Como vimos na lição anterior, existem vários tipos de objetos que podem ser adicionados ao projeto. Um dos mais usados é o módulo de programação, que é um arquivo texto com extensão BAS contendo procedures, funções e declarações de uso geral no sistema. Um uso muito freqüente é a declaração de variáveis públicas. Variáveis Connection são freqüentemente declaradas como públicas, para que não seja necessária a criação de uma conexão com o banco a cada vez que precisarmos usar um command – se a conexão é pública, podemos usar sempre a mesma.

Para adicionar um módulo de programação ao projeto, usamos a opção *Add Module* do menu *Project* do VB. A operação da janela *Add* Module é a mesma da *Add Form*:



Usando o Banco de Dados

Para começar, criaremos um novo banco de dados no Access com o nome de **Biblio**. Ele conterá quatro tabelas, que devem conter as estruturas descritas a seguir (siga fielmente os nomes das tabelas e os nomes e propriedades dos campos, para que a codificação do projeto funcione corretamente):

Tabela: **Categorias**

Nome do Campo	*Tipo*	*Propriedades*
CodCategoria (Chave primária)	Número	Tamanho = Inteiro Longo
NomeCategoria	Texto	Tamanho = 35

Tabela: **Editoras**

Nome do Campo	*Tipo*	*Propriedades*
CodEditora (Chave primária)	Número	Tamanho = Inteiro Longo
NomeEditora	Texto	Tamanho = 35

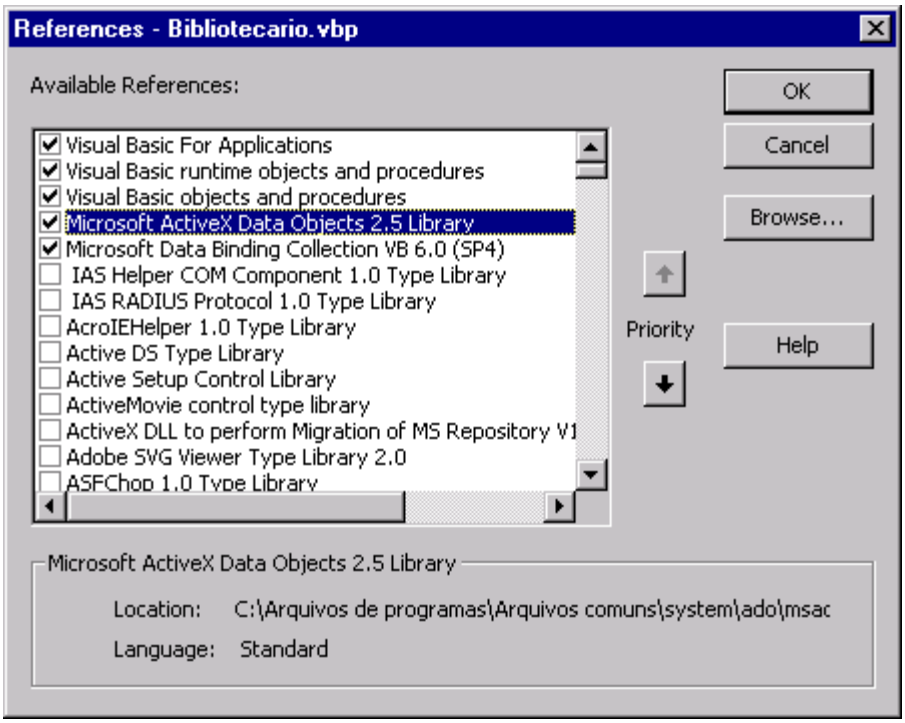
Tabela: **Livros**

Nome do Campo	*Tipo*	*Propriedades*
CodLivro (Chave primária)	Número	Tamanho = Inteiro Longo
Titulo	Texto	Tamanho = 50
Autor	Texto	Tamanho = 35
CodEditora	Número	Tamanho = Inteiro Longo
CodCategoria	Número	Tamanho = Inteiro Longo
AcompCD	Sim/Não	-
AcompDisquete	Sim/Não	-
Idioma	Número	Tamanho = Byte
Observacoes	Memorando	-
Emprestado	Sim/Não	-
CodUsuario	Número	Tamanho = Inteiro Longo
DataEmprestimo	Data/Hora	Formato = Data Abreviada
DataDevolucao	Data/Hora	Formato = Data Abreviada

Tabela: **Usuarios**		
Nome do Campo	*Tipo*	*Propriedades*
CodUsuario (Chave primária)	Número	Tamanho = Inteiro Longo
NomeUsuario	Texto	Tamanho = 35
Endereco	Texto	Tamanho = 60
Cidade	Texto	Tamanho = 25
Estado	Texto	Tamanho = 2
CEP	Texto	Tamanho = 9
Telefone	Texto	Tamanho = 15

Após a criação do banco de dados, vamos criar a conexão no sistema Bibliotecário:

1. Abra o VB e o projeto Bibliotecario, se ainda não o fez;
2. Precisamos adicionar às referências do projeto a biblioteca que permite a utilização dos objetos ADO. Abra o menu *Project* do VB e escolha a opção *References*:



3. O VB abre uma janela chamada *References*. Note que já existem algumas referências marcadas. Para adicionar uma nova, basta clicar na *checkbox* ao lado de seu nome. Para remover uma referência, remova a marca de seleção (obs.: o VB não permite a remoção de uma biblioteca que esteja em uso);
4. A biblioteca que usaremos é a *Microsoft ActiveX Data Objects Library* Existem várias versões dessa biblioteca, dependendo da versão do Jet que estiver instalada em sua máquina. Se houver mais de uma, procure sempre utilizar a mais recente (na figura está sendo usada a versão 2.5). Clique em Ok;
5. Vamos declarar a variável Connection como pública. Para isso, precisamos de um módulo de programação. Abra o menu *Project* e escolha a opção *Add Module*;
6. Após adicionar o módulo abra-o na janela Código, e na seção Declarations escreva o comando:

```
Public cnnBiblio As New ADODB.Connection
```

7. Grave o módulo de programação com o nome de **Biblio.BAS**;
8. Vamos abrir a conexão com o banco de dados no evento Timer do formulário de Splash. Abra o formulário frmSplash na janela Código;
9. Altere a procedure Timer1_Timer para que fique como a seguir:

```
Private Sub Timer1_Timer()
      On Error GoTo errConexao
      cnnBiblio.ConnectionString =  "Provider=Microsoft.Jet.OLEDB.3.51;" & _
                                    "Data Source=" & App.Path & "\Biblio.mdb;"
      cnnBiblio.Open
```

```
      Unload Me
      frmBiblio.Show
      Exit Sub

errConexao:
      With Err
            If .Number <> 0 Then
                  MsgBox "Houve um erro na conexão com o banco de dados." & _
                        vbCrLf & "O sistema será encerrado.", _
                        vbCritical + vbOKOnly + vbApplicationModal, _
                        "Erro na conexão"
                  .Number = 0
                  Set cnnBiblio = Nothing
                  End
            End If
      End With
End Sub
```

Obs.: note que no trecho acima usamos a versão 3.51 do Jet. Note também o uso de App.Path em Data Source: App é o objeto *Application*, e *Path* uma propriedade que retorna a pasta onde a aplicação está instalada. Portanto, App.Path informa ao provedor que o arquivo Biblio.MDB está localizado na mesma pasta da aplicação.

10. É interessante fechar a conexão quando encerramos o sistema. Abra então o formulário frmBiblio na janela Código e escreva o evento Unload, que ocorre quando o formulário é fechado:

```
Private Sub MDIForm_Unload(Cancel As Integer)
      Set cnnBiblio = Nothing
End Sub
```

11. Grave seu trabalho.

Repare que ainda não usamos nenhum comando ADO, só nos conectamos ao banco de dados. Antes de fazer isso, vamos estudar um pouco de SQL, que é o assunto da próxima lição.

Lição 9: Manipulação de Registros

Nessa lição, vamos estudar a programação necessária para executar as operações básicas de manipulação de registros, que são: seleção, inclusão, alteração e exclusão. Usaremos como base para nosso estudo a tabela de Usuários, pois como ela possui apenas um campo numérico, e todos os demais são do tipo texto, é a mais simples de trabalhar com os controles que conhecemos até aqui.

É importante ressaltar que não constará nessa apostila os formulários e a programação referentes às tabelas Editoras e Categorias, pois serão praticamente iguais àquelas que vamos elaborar para a de Usuários. Ficará para você, portanto, desenvolver essas rotinas como exercício de fixação.

Um Pouco de SQL

Antes de começar a trabalhar no formulário, precisamos conhecer um pouco da linguagem SQL. A SQL (*Structure Query Language* – Linguagem de Consulta Estruturada) é uma linguagem comum de gerenciamento de bancos de dados relacionais criada para padronizar o tratamento dos dados em todos os sistemas que obedeçam ao padrão SQL vigente. Isso quer dizer que os mesmos comandos SQL que você usa no Access podem ser aplicados na maioria dos SGBDs de arquitetura cliente-servidor existentes no mercado. É importante destacar, porém, que a SQL não é uma linguagem de programação comum, usada para a criação de sistemas e aplicativos, e sim uma linguagem voltada exclusivamente para aplicações em bancos de dados relacionais.

Não vamos abordar SQL a fundo nesse texto, pois não é esse o objetivo do curso. Veremos apenas os comandos principais, que permitam à aplicação que estamos construindo executar as operações básicas de manutenção de bancos de dados.

Comando SELECT

SELECT é o comando para selecionar dados no banco de dados e retornar as informações solicitadas como um conjunto de registros. Certamente é o comando SQL mais utilizado. Sua sintaxe básica é:

```
SELECT * FROM <tabela> [WHERE <condição>] [ORDER BY <ordem>];
```

Ou então:

```
SELECT campo1, campo2, campo3, ..., campoN FROM <tabela>
[WHERE <condição>] [ORDER BY <ordem>];
```

Onde:

- *****: indica que todos os campos da tabela devem ser selecionados;
- **Lista de campos**: indica que somente os campos especificados devem ser selecionados;
- **Tabela**: nome da tabela que contém os dados a serem selecionados;
- **Condição** (opcional): expressão lógica que indica que somente os registros que atendam à ela devem ser selecionados pelo comando SELECT;
- **Ordem** (opcional): lista de campos, separados por vírgulas, que indicam a ordem de classificação dos registros selecionados.

Ao executar esta operação, o provedor procura no banco de dados a tabela especificada, extrai as colunas escolhidas, seleciona as linhas que satisfazem a condição e finalmente classifica as linhas resultantes na ordem especificada.

É importante destacar que, apesar de WHERE e ORDER BY serem opcionais e podermos usá-los em separado, no caso de ambos serem utilizados obrigatoriamente WHERE deverá vir antes de ORDER BY.

Veja alguns exemplos:

- Para selecionar todos os campos da tabela de funcionários:

  ```
  SELECT * FROM Funcionarios;
  ```

- Se quiser ordenar os regitros em ordem alfabética, poderia usar:

  ```
  SELECT * FROM Funcionarios ORDER BY Nome;
  ```

- Para selecionar apenas os campos Nome e Departamento da mesma tabela:

```
SELECT Nome, Departamento FROM Funcionarios;
```

- Para filtrar apenas os funcionários do departamento RH:

```
SELECT Codigo, Nome FROM Funcionarios WHERE Departamento = “RH”;
```

- Usando WHERE e ORDER BY juntos:

```
SELECT Codigo, Nome FROM Funcionarios
WHERE Departamento = “RH” ORDER BY Nome;
```

Em um primeiro momento, usaremos o comando SELECT para selecionar apenas um registro por vez. Nas próximas lições, principalmente quando falarmos de consultas, estaremos estudando-o com um pouco mais de profundidade.

Comando INSERT

INSERT é o comando SQL usado para adicionar registros a uma tabela. Sua sintaxe é a seguinte:

```
INSERT INTO <tabela> (campo1, campo2, campo3, ..., campoN)
VALUES (valor1, valor2, valor3, ..., ValorN);
```

Onde:

- **Tabela**: é o nome da tabela em que o novo registro deve ser inserido;
- **Lista de campos**: os nomes dos campos nos quais os dados devem ser gravados;
- **Lista de valores**: dados a serem gravados nos campos, sendo que cada valor é inserido no campo correspondente à posição do valor na lista: valor1 em campo1, valor2 em campo2, e assim por diante.

Não é necessário indicar valores para todos os campos da tabela. Quando um dado não forem designado para um campo qualquer, o valor padrão definido nas propriedades dos campos omitidos será inserido pelo Jet. Se não existir um valor padrão, então será inserido Null (nulo) nas colunas omitidas. Nesse caso, o campo deve suportar o valor nulo. Os registros são sempre adicionados no final da tabela.

O ponto-e-vírgula faz parte da sintaxe de todo comando SQL. É chamado de terminador, e serve para indicar o fim do comando. Para alguns provedores (como o Jet) ele é dispensável.

Veja um exemplo que cria um novo registro numa tabela chamada Funcionários:

```
INSERT INTO Funcionários (Codigo, Nome, Sobrenome, Titulo)
VALUES (73, 'André', 'Pereira', 'Estagiário');
```

Comando UPDATE

UPDATE faz a atualização de registros de uma tabela, de acordo com uma condição. Devemos indicar quais campos serão atualizadados. Sua sintaxe é:

```
UPDATE <tabela> SET <campo> = <valornovo> [WHERE <condição>];
```

Onde:

- **Tabela**: nome da tabela cujos registros devem ser alterados:
- **Campo**: nome do campo onde deve ser gravado o novo valor. Podem ser alterados vários campos ao mesmo tempo;
- **Valornovo**: valor ou expressão cujo resultado será gravado no campo indicado;
- **Condição** (opcional): expressão lógica que indica quais registros devem ser afetados pelo comando Update. Mas cuidado! Se não for especificada uma condição, Update vai alterar TODOS os registros da tabela.

Veja alguns exemplos:

- Para alterar o campo nome do registro cujo campo código é igual a 15 na tabela Funcionários:

```
UPDATE Funcionários SET Nome = 'José Lima' WHERE Codigo = 15;
```

- Para calcular o preço de venda de todos os registros de uma tabela chamada Produtos:

```
UPDATE Produtos SET PrecoVenda = (PrecoCusto * 1.25);
```

Comando DELETE

Realiza a exclusão de registros da tabela indicada. Veja a sintaxe:

```
DELETE FROM <tabela> [WHERE <condição>];
```

Onde:

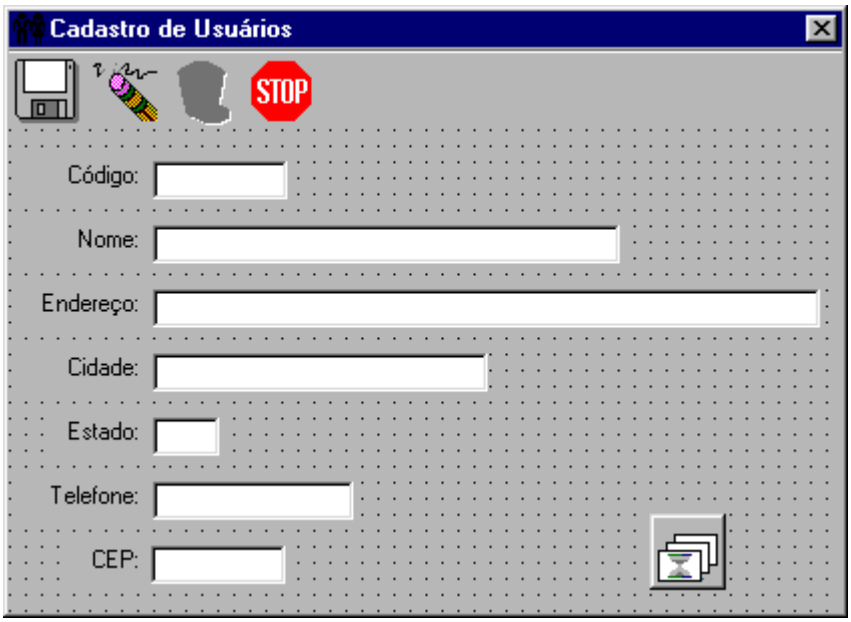
- **Tabela**: nome da tabela da qual os registros são excluídos;
- **Condição** (opcional): expressão lógica que indica quais registros devem ser excluídos. Apesar de ser opcional, é extremamente importante especificar uma condição, pois em caso contrário TODOS os registros da tabela serão excluídos.

Veja um exemplo:

```
DELETE FROM Funcionários WHERE Titulo = 'Estagiário';
```

Esse exemplo exclui todos os registros da tabela Funcionários cujo título seja “Estagiário”.

Bem, agora que já vimos os comandos SQL que vamos precisar, começemos a trabalhar. Inclua um novo formulário no projeto e formate-o como no exemplo a seguir:



Altere as propriedades dos objetos conforme a tabela:

Objeto	***Nome***	***Outras Propriedades = Valor***
Formulário	frmCadUsuarios	Caption = Cadastro de Usuários BorderStyle = 1 – Fixed Single Icon = Misc\Misc28.ICO KeyPreview = True MDIChild = True
Caixa de texto	txtCodUsuario	Text = “” MaxLength = 5
Caixa de texto	txtNomeUsuario	Text = “” MaxLength = 35
Caixa de texto	txtEndereco	Text = “” MaxLength = 60
Caixa de texto	txtCidade	Text = “” MaxLength = 25
Caixa de texto	txtEstado	Text = “” MaxLength = 2
Caixa de texto	txtTelefone	Text = “” MaxLength = 15
Caixa de texto	txtCEP	Text = “” MaxLength = 9

Objeto	***Nome***	***Outras Propriedades = Valor***
ImageList	ImageList1	Tamanho das imagens: 32 x 32 pixels Imagens: 1 = Computer\Disk04.ICO 2 = Writing\Erase02.ICO 3 = Computer\Trash01.ICO 4 = Traffic\Trffc14.ICO
ToolBar	ToolBar1	ImageList = ImageList1 Appearance = 1 – cc3D BorderStyle = 0 – ccNone Style = 1 – tbrFlat Botão 1: ToolTipText = Grava o registro atual Image = 1 Botão 2: ToolTipText = Limpa os dados digitados Image = 2 Botão 3: ToolTipText = Exclui o registro atual Image = 3 Enabled = False (desmarcar) Botão 4: ToolTipText = Retorna ao menu principal Image = 4

O funcionamento básico do formulário será o seguinte:

- Após a digitação do código do usuário, será feita uma pesquisa para verificar se o código existe ou não na tabela. Se existir, os dados do registro serão exibidos nos campos do form e poderão ser alterados. Caso contrário, será considerado como um código novo, ou seja, uma operação de inclusão;
- O botão Excluir só deverá estar habilitado quando o registro estiver em modo de alteração;
- Quando o botão Gravar for clicado, deverá ser feita uma crítica nos dados digitados antes da gravação, para identificar possíveis erros de digitação e impedir a gravação nesse caso;
- Quando o botão Limpar for clicado, qualquer dado constante nos campos do formulário será apagado.

Vamos agora escrever a codificação. Siga os passos:

1. Inicialmente, escreveremos as procedures referentes aos eventos do formulário. Abra a janela código a partir do formulário frmCadUsuarios e programe os eventos Load e KeyPress conforme abaixo:

```
Private Sub Form_Load()
    'Centraliza o formulário na área de trabalho do MDI:
    Me.Left = (frmBiblio.ScaleWidth - Me.Width) / 2
    Me.Top = (frmBiblio.ScaleHeight - Me.Height) / 2
End Sub

Private Sub Form_KeyPress(KeyAscii As Integer)
    'Se a tecla Enter foi pressionada, passa o foco para o próximo controle na
    'seqüência de TabIndex:
    If KeyAscii = vbKeyReturn Then
        SendKeys "{Tab}"
        KeyAscii = 0
    End If
End Sub
```

   Obs.: como um formulário MDIChild não aceita a propriedade StartUpPosition, escrevemos no evento Load a codificação necessária para centralizar o form dentro da área de trabalho do MDI. As propriedades ScaleWidth e ScaleHeight retornam, respectivamente, a largura e a altura da área de trabalho do formulário.

2. Vamos iniciar a programação das operações de manutenção de registros. Nossa primeira preocupação deve ser como identificar se estamos fazendo uma inclusão ou uma alteração. A maneira mais simples é criando uma variável privada que informe se o registro existe ou não na tabela. Abra a janela Código na seção Declaration e declare a variável:

```
Dim vInclusao As Boolean
```

3. Agora vamos pesquisar na tabela o código digitado pelo operador. Faremos isso no evento LostFocus do campo txtCodUsuario:

```
Private Sub txtCodUsuario_LostFocus()
Dim cnnComando As New ADODB.Command
Dim rsSelecao As New ADODB.Recordset
    On Error GoTo errSelecao
    'Verifica se foi digitado um código válido:
    If Val(txtCodUsuario.Text) = 0 Then
        MsgBox "Não foi digitado um código válido, verifique.", _
            vbExclamation + vbOKOnly + vbApplicationModal, "Erro"
        Exit Sub
    End If
    Screen.MousePointer = vbHourglass
    With cnnComando
        .ActiveConnection = cnnBiblio
        .CommandType = adCmdText
        'Monta o comando SELECT para selecionar o registro na tabela:
        .CommandText = "SELECT * FROM Usuarios WHERE CodUsuario = " & _
            txtCodUsuario.Text & ";"
        Set rsSelecao = .Execute
    End With
    With rsSelecao
        If .EOF And .BOF Then
            'Se o recordset está vazio, não retornou registro com esse código:
            LimparDados
            'Identifica a operacao como Inclusão:
            vInclusao = True
        Else
            'Senão, atribui aos campos os dados do registro:
            txtNomeUsuario.Text = !NomeUsuario
            txtEndereco.Text = !Endereco
            txtCidade.Text = !Cidade
            txtEstado.Text = !Estado
            txtCEP.Text = !CEP
            txtTelefone.Text = Empty & !Telefone
            'Identifica a operacao como Alteração:
            vInclusao = False
            'Habilita o botão Excluir:
            Toolbar1.Buttons(3).Enabled = True
        End If
    End With
    'Desabilita a digitação do campo código:
    txtCodUsuario.Enabled = False

Saida:
    'Elimina o command e o recordset da memória:
    Set rsSelecao = Nothing
    Set cnnComando = Nothing
    Screen.MousePointer = vbDefault
    Exit Sub

errSelecao:
    With Err
        If .Number <> 0 Then
            MsgBox "Houve um erro na recuperação do registro solicitado.", _
            vbExclamation + vbOKOnly + vbApplicationModal, "Aviso"
            .Number = 0
            GoTo Saida
        End If
    End With
End Sub
```

Repare como foi montado o comando SELECT: informando à cláusula WHERE o código a pesquisar por concatenação. Mas continuemos a programação do formulário:

4. "LimparDados" é uma subrotina para apagar os campos do formulário (menos o código):

```
Private Sub LimparDados()
    'Apaga o conteúdo dos campos do formulário:
    txtNomeUsuario.Text = Empty
    txtEndereco.Text = Empty
    txtCidade.Text = Empty
    txtEstado.Text = Empty
    txtCEP.Text = Empty
    txtTelefone.Text = Empty
End Sub
```

5. Já podemos fazer a codificação da barra de botões. Note as chamadas a subrotinas:

```
Private Sub Toolbar1_ButtonClick(ByVal Button As MSComctlLib.Button)
      'Verifica qual foi o botão clicado:
      Select Case Button.Index
            Case 1
                  'Botão Gravar:
                  GravarDados
            Case 2
                  'Botão Limpar:
                  LimparTela
            Case 3
                  'Botão Excluir:
                  ExcluirRegistro
            Case 4
                  'Botão Retornar:
                  Unload Me
      End Select
End Sub
```

6. "GravarDados", como o próprio nome indica, é a subrotina de gravação dos dados digitados. Nela vamos identificar o tipo de operação que devemos executar (inclusão ou alteração) e montar o comando SQL correspondente:

```
Private Sub GravarDados()
Dim cnnComando As New ADODB.Command
Dim vConfMsg As Integer
Dim vErro As Boolean
    On Error GoTo errGravacao
    'Inicializa as variáveis auxiliares:
    vConfMsg = vbExclamation + vbOKOnly + vbSystemModal
    vErro = False
    'Verifica os dados digitados:
    If txtNomeUsuario.Text = Empty Then
        MsgBox "O campo Nome não foi preenchido.", vConfMsg, "Erro"
        vErro = True
    End If
    If txtEndereco.Text = Empty Then
        MsgBox "O campo Endereço não foi preenchido.", vConfMsg, "Erro"
        vErro = True
    End If
    If txtCidade.Text = Empty Then
        MsgBox "O campo Cidade não foi preenchido.", vConfMsg, "Erro"
        vErro = True
    End If
    If txtEstado.Text = Empty Then
        MsgBox "O campo Estado não foi preenchido.", vConfMsg, "Erro"
        vErro = True
    End If
    If txtCEP.Text = Empty Then
        MsgBox "O campo CEP não foi preenchido.", vConfMsg, "Erro"
        vErro = True
    End If
    'Se aconteceu um erro de digitação, sai da sub sem gravar:
    If vErro Then Exit Sub
```

```
    Screen.MousePointer = vbHourglass
    With cnnComando
        .ActiveConnection = cnnBiblio
        .CommandType = adCmdText
        'Verifica a operação e cria o comando SQL correspondente:
        If vInclusao Then
            'Inclusão:
            .CommandText = "INSERT INTO Usuarios " & _
                "(CodUsuario, NomeUsuario, Endereco, Cidade, " & _
                "Estado, CEP, Telefone) VALUES ('" & _
                txtCodUsuario.Text & "','" & _
                txtNomeUsuario.Text & "','" & _
                txtEndereco.Text & "','" & _
                txtCidade.Text & "','" & _
                txtEstado.Text & "','" & _
                txtCEP.Text & "','" & _
                txtTelefone.Text & "');"
        Else
            'Alteração:
            .CommandText = "UPDATE Usuarios SET " & _
                "NomeUsuario = '" & txtNomeUsuario.Text & "'," & _
                "Endereco = '," & txtEndereco.Text & "'," & _
                "Cidade = '," & txtCidade.Text & "'," & _
                "Estado = '," & txtEstado.Text & "'," & _
                "CEP = '," & txtCEP.Text & "'," & _
                "Telefone = '," & txtTelefone.Text & "' " & _
                "WHERE CodUsuario = " & txtCodUsuario.Text & ";"
        End If
        .Execute
    End With
    MsgBox "Gravação concluída com sucesso.", _
            vbApplicationModal + vbInformation + vbOKOnly, _
            "Gravação OK"
    'Chama a sub que limpa os dados do formulário:
    LimparTela

Saida:
    Screen.MousePointer = vbDefault
    Set cnnComando = Nothing
    Exit Sub

errGravacao:
    With Err
        If .Number <> 0 Then
            MsgBox "Houve um erro durante a gravação dos dados na tabela.", _
                    vbExclamation + vbOKOnly + vbApplicationModal, "Erro"
            .Number = 0
            GoTo Saida
        End If
    End With
End Sub
```

7. “LimparTela” também é uma subrotina, que limpa totalmente o formulário para uma nova digitação:

```
Private Sub LimparTela()
      'Chama a sub LimparDados para limpar os campos do formulário:
      LimparDados
      'Desabilita o botão Excluir:
      Toolbar1.Buttons(3).Enabled = False
      'Apaga o conteúdo do campo CodUsuario e lhe passa o foco:
      txtCodUsuario.Text = Empty
      txtCodUsuario.SetFocus
End Sub
```

8. Falta ainda a subrotina de exclusão de registros. Essa é um pouco mais simples:

```
Private Sub ExcluirRegistro()
Dim cnnComando As New ADODB.Command
Dim vOk As Integer
    On Error GoTo errExclusao
```

```
    'Solicita confirmação da exclusão do registro:
    vOk = MsgBox("Confirma a exclusão desse registro?", _
            vbApplicationModal + vbDefaultButton2 + vbQuestion + vbYesNo, _
            "Exclusão")
    If vOk = vbYes Then
        Screen.MousePointer = vbHourglass
        With cnnComando
            .ActiveConnection = cnnBiblio
            .CommandType = adCmdText
            'Cria o comando SQL:
            .CommandText = "DELETE FROM Usuarios WHERE CodUsuario = " & _
                txtCodUsuario.Text & ";"
            .Execute
        End With
        MsgBox "Registro excluído com sucesso.", _
                vbApplicationModal + vbInformation + vbOKOnly, _
                "Exclusão OK"
        'Chama a sub que apaga todos os campos do formulário:
        LimparTela
    End If

Saida:
    Screen.MousePointer = vbDefault
    Set cnnComando = Nothing
    Exit Sub

errExclusao:
    With Err
        If .Number <> 0 Then
            MsgBox "Houve um erro durante a exclusão do registro.", _
                    vbExclamation + vbOKOnly + vbApplicationModal, "Erro"
            .Number = 0
            GoTo Saida
        End If
    End With
End Sub
```

9. Para terminar a codificação desse formulário falta apenas codificar o evento LostFocus dos campos cujo conteúdo queremos formatar antes de gravar:

```
Private Sub txtNomeUsuario_LostFocus()
      'Converte a primeira letra de cada palavra digitada em maiúscula,
      'e as demais em minúsculas:
      txtNomeUsuario.Text = StrConv(txtNomeUsuario.Text, vbProperCase)
End Sub

Private Sub txtCidade_LostFocus()
      txtCidade.Text = StrConv(txtCidade.Text, vbProperCase)
End Sub

Private Sub txtEndereco_LostFocus()
      txtEndereco.Text = StrConv(txtEndereco.Text, vbProperCase)
End Sub

Private Sub txtEstado_LostFocus()
      'Converte a sigla do Estado digitada em letras maiúsculas:
      txtEstado.Text = UCase(txtEstado.Text)
End Sub

Private Sub txtCEP_LostFocus()
      'Formata o CEP digitado:
      txtCEP.Text = Format(txtCEP.Text, "00000-000")
End Sub
```

10. Finalmente, abra o formulário frmBiblio na janela Código e codifique o evento Click do menu mnuCadUsuarios:

```
Private Sub mnuCadUsuarios_Click()
    frmNovoArtista.Show
End Sub
```

11. Altere o evento Click da ToolBar para chamar esse mesmo formulário quando for clicado o botão nº 1:

```
Private Sub Toolbar1_ButtonClick(ByVal Button As MSComctlLib.Button)
      If Button.Index = 1 Then
            frmCadLivros.Show
      ElseIf Button.Index = 5 Then
            mnuSair_Click
      End If
End Sub
```

12. Teste o aplicativo. No caso de algum erro, verifique cuidadosamente a construção dos comandos SQL, pois uma simples vírgula fora do lugar ou faltando faz com que um comando não funcione. Aproveite para incluir alguns usuários em seu banco de dados;

13. Grave esse formulário com o nome de **frmCadUsuarios**;

14. Baseado na programação vista nessa lição, crie os formulários para manutenção dos cadastros de Categorias e de Editoras.

Lição 10: Mais Controles

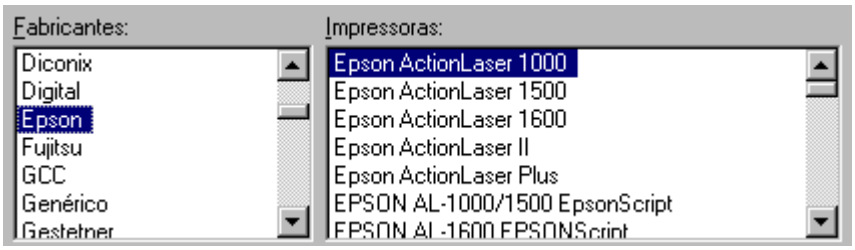
Além de caixas de textos e *labels*, podemos usar para alimentar os campos das tabelas quaisquer outros controles que permitam a entrada de dados. Em nosso caso, vendo a estrutura da tabela de Livros, encontraremos alguns campos que se encaixam nessa situação. Vejamos:

- Os campos CodEditora e CodCategoria estão relacionados, respectivamente, às tabelas Editoras e Categorias, e portanto podemos montar caixas de combinação onde o operador possa fazer sua escolha;
- Os campos AcompCD e AcompDisquete são do tipo Sim/Não, e portanto podemos usar caixas de verificação para marcar e desmarcar essas opções;
- O campo Idioma, do tipo Byte, receberá um valor entre 0 e 2 que indicará o idioma em que está escrito o livro, sendo que 0 = Português, 1 = Inglês e 2 = Outro. Como não podem ser marcadas dois ou mais idiomas ao mesmo tempo, podemos montar para esse campo um grupo de botões de opção.

Nessa lição estudaremos esses novos controles, enquanto elaboramos mais alguns formulários do projeto Bibliotecário. Adicionalmente, veremos como abrir consultas criadas no Access em um projeto VB.

Caixas de Listagem

Uma caixa de listagem oferece ao usuário uma lista de opções entre as quais ele deve obrigatoriamente efetuar sua escolha. Se o número de itens não couber no espaço ocupado pela caixa no formulário, uma barra de rolagem vertical será automaticamente exibida. Um exemplo de caixa de listagem é encontrado numa das telas do assistente para instalação de impressoras do Windows: o usuário deve escolher uma das impressoras apresentadas na lista:



Caixas de listagem para escolha de impressora do Windows

Caixas de Combinação

As caixas de combinação, também chamadas simplesmente de *combo*, têm esse nome por serem uma mistura de caixa de texto com caixa de listagem. O usuário tanto pode escolher um item da lista como digitar um valor diretamente na caixa de texto. As principais diferenças entre uma *ComboBox* e uma *ListBox* são duas, dependendo do tipo de caixa de combinação usado: a combo pode permitir ao usuário escolher uma opção que não consta da caixa e ocupar menos espaço no formulário que uma caixa de listagem:

A propriedade que determina o tipo da caixa de combinação é a Style. Existem três tipos diferentes:

- **0 – DropDown Combo**: é o tipo mais comum: permite tanto a escolha de um item da lista como a digitação diretamente na caixa de texto, inclusive de um dado não constante da lista. A lista fica oculta e é exibida quando se clica no botão ao lado da caixa de texto. É o estilo do nosso exemplo;
- **1 – Simple Combo**: nesse estilo a caixa de listagem é permanentemente exibida. O usuário pode digitar diretamente na caixa de texto, mas a vantagem da economia de espaço é perdida.

- **2 – DropDown List**: é bastante parecida com uma caixa de listagem comum. O usuário não pode digitar na caixa de texto, que serve apenas para exibir o item selecionado. Como não aceita digitação, não pode ser escolhida uma opção que não conste da lista.

As demais propriedades e métodos das caixas de listagem e de combinação são comuns entre elas. Vejamos as principais:

Propriedade List

A propriedade List contém um *array* cujos elementos são os itens da caixa de listagem ou de combinação. Ela pode fornecer um determinado item através de seu índice, que inicia-se em 0. Sua sintaxe é:

```
NomeDaCaixa.List(<índice>)
```

Onde <índice> é o número do índice do item desejado.

Propriedade ListCount

Essa propriedade, que também não está disponível em tempo de projeto, é usada para se obter o número de itens da caixa de listagem ou de combinação.

Propriedade ListIndex

A propriedade ListIndex retorna o número do índice do item atualmente selecionado da caixa de listagem ou de combinação. Se não foi selecionado nenhum item, ela retorna o valor –1.

Propriedade Text

A propriedade Text também pode ser usada para se obter o item atualmente selecionado da caixa de listagem ou de combinação, só que, como o próprio nome indica, ela retorna o texto do item selecionado, ao invés do índice.

Propriedade ItemData

Eis uma propriedade muito interessante: para entender sua utilidade, imagine a seguinte situação: no cadastro de Livros temos um campo chamado CodEditora, que, como já dissemos, será obtido a partir de uma caixa de combinação no formulário que vamos construir. Acontece que (como você deve ter reparado) o campo é numérico e o item exibido pela combo é um texto, que portanto não poderá ser gravado nesse campo. ItemData serve exatamente para resolver esse problema, pois ela armazena dados numéricos associados aos itens da combo. Assim, quando adicionarmos uma editora à caixa, aproveitaremos para gravar o código dessa editora na propriedade ItemData, e quando uma editora for selecionada gravaremos no registro o conteúdo de ItemData. A sintaxe dessa propriedade é a seguinte:

```
<expressão> = NomeDaCaixa.Itemdata.<índice>
```

Onde <expressão> é uma variável ou campo tipo Long, e <índice> é o número do índice do item desejado.

Propriedade Sorted

Essa propriedade determina se os itens serão exibidos em ordem alfabética ou não. Os valores possíveis são True (a lista será exibida em ordem alfabética) ou False (os itens serão exibidos na mesma ordem em que foram incluídos na lista).

Método AddItem

Para adicionarmos itens a uma caixa de listagem ou combinação, usamos o método AddItem. Veja a sintaxe:

```
NomeDaCaixa.AddItem <item>[,<índice>]
```

Onde:

- **Item** é o nome do item a ser adicionado à lista. Se for um texto deve vir entre aspas;
- **Índice** é opcional: se for especificado, o item será adicionado à lista com o número de índice indicado. O valor 0 indica a primeira posição, 1 o segundo, e assim por diante. Se não for informado, o item será adicionado ao final da lista. É muito útil quando trabalhamos com a propriedade ItemData.

Veja alguns exemplos:

- Para adicionar o item “Melancia” a uma caixa de listagem chamada lstFrutas:

```
lstFrutas.AddItem “Melancia”
```

- Para adicionar o mesmo item na primeira posição da caixa de listagem:

```
lstFrutas.AddItem “Melancia”,0
```

- Para adicionar o texto contido em uma caixa de texto à caixa de listagem:

```
lstFrutas.AddItem txtFruta.Text
```

Método RemoveItem

O método RemoveItem exclui um item de uma caixa de listagem ou combinação. Veja sua sintaxe:

```
NomeDaCaixa.RemoveItem <índice>
```

Onde <índice> é o índice do item que deve ser excluído da lista. Por exemplo: para excluir o item com índice 4 da caixa de listagem lstFrutas, o comando seria o seguinte:

```
lstFrutas.RemoveItem 4
```

Caixas de Verificação

Esse tipo de controle normalmente é usado para habilitar e desabilitar opções não exclusivas. Ou seja: quando você inclui mais de uma caixa de verificação em um formulário, você pode marcar uma, duas, três, todas, ou nenhuma. Sua principal propriedade é a **Value**, que indica se a caixa está ou não habilitada, e pode assumir três valores: **0 – Unchecked** (desabilitado), **1 – Checked** (habilitado), e **2 – Grayed**. Essa última ocorre quando a caixa está marcada apenas para uma parte de uma seleção de texto, por exemplo.

Botões de Opção

Já usamos esse controle no nosso primeiro aplicativo, lembra-se? Era ele que indicava nosso “humor matinal” no programa Hello, que elaboramos na lição 1. Ele também serve para indicar opções, só que dessa vez exclusivas, ou seja, somente um botão de opção pode estar habilitado de cada vez. A propriedade **Value** também é a mais importante para esse controle, e pode assumir os valores True e False. Os botões de opção podem ser agrupados de modo a conseguirmos determinar várias opções com eles ao mesmo tempo: somente um botão pode estar habilitado em um determinado momento, mas se temos em um formulário mais de um grupo de botões, em cada um deles pode existir um botão ligado. Esse efeito é conseguido com o uso de molduras.

Molduras

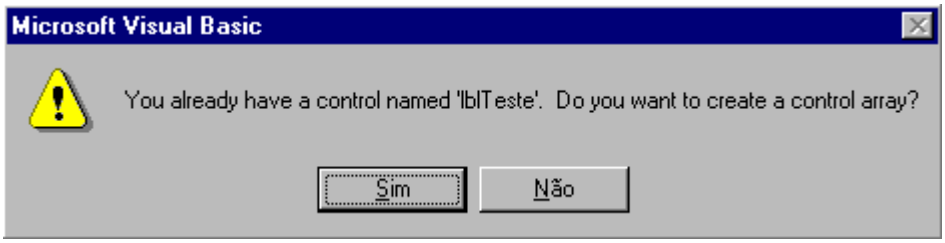
Eis um controle curioso: basicamente, uma moldura (ou *Frame*) não serve para nada além de criar “bordas” nas quais outros controles são inseridos, agrupando-os. Mas está justamente aí sua grande utilidade: quando os controles são agrupados em *frames* eles passam a funcionar de maneira integrada. Por exemplo: se você criar uma moldura, inserir nela alguns *labels*, e depois disso movê-la para outra área do formulário, os *labels* serão movidos junto com a moldura, pois estão subordinados a ela. Mas, cuidado! Para que isso aconteça, é imprescindível que os controles sejam desenhados **dentro** da moldura. Nada de dar duplo click em um controle e depois movê-lo para dentro da *frame*. Ele não será incorporado pela moldura, e continuará sendo um controle independente. A incorporação também não funcionará se uma das bordas do controle for desenhada para fora da moldura, mesmo que a maior parte esteja dentro dela.

A Propriedade Index

Como vimos no início do curso, cada objeto em um formulário deve possuir um nome, sendo que não podem existir dois objetos com o mesmo nome. Mas podemos criar um *control array* (vetor de controles) que é um conjunto de controles do mesmo tipo com o mesmo nome, que são acessados através de um índice atribuído a cada um deles. Por exemplo: quando usamos botões de opção da mesma maneira que no aplicativo Hello, temos que escrever uma procedure Click para cada um dos botões, pois eles são objetos independentes entre si (lembra-se?). Se ao invés disso criássemos um vetor com esse grupo de botões, todos possuiriam o mesmo nome, e a procedure Click seria comum a todos eles. Bastaria então identificar em qual deles o usuário

clicou, através da propriedade Index. Foi o que fizemos com os botões da ToolBar nos formulários do projeto Bibliotecário.

Para testar o uso do *control array*, inicie um novo projeto e insira no formulário um *label* com o nome de lblTeste. Logo em seguida, insira outro label e atribua a ele o mesmo nome, lblTeste. O VB exibirá a seguinte caixa de diálogo:



Traduzindo: "Você já possui um controle chamado 'lblTeste'. Você quer criar um vetor de controles?". Responda "Sim", e o VB renomeará o primeiro label para lblTeste(0) e o segundo para lblTeste(1). O valor entre parênteses é o da propriedade Index de cada um dos controles: 0 e 1, respectivamente. A partir desse momento, quando o objeto for referenciado você deverá indicar o índice do label que está querendo usar. Por exemplo: para mudar a propriedade Caption do primeiro label, poderíamos escrever o comando como a seguir:

```
lblTeste(0).Caption = "Esse é o primeiro objeto do vetor"
```

Quando trabalhamos com vetores, os eventos do *control array* passam a receber um parâmetro chamado Index, do tipo Integer, que recebe o valor da propriedade Index do controle que gerou o evento. Por exemplo: abra a janela Código no evento Click de um dos labels. Vamos escrever um comando que mostre uma mensagem no label que foi clicado. Para isso, vamos identificá-lo através do parâmetro Index:

```
Private Sub lblTeste_Click(Index As Integer)
    If Index = 0 Then
        lblTeste(0).Caption = "Você clicou aqui"
        lblTeste(1).Caption = Empty
    Else
        lblTeste(0).Caption = Empty
        lblTeste(1).Caption = "Você clicou aqui"
    End If
End Sub
```

Uma outra maneira de usar o parâmetro Index:

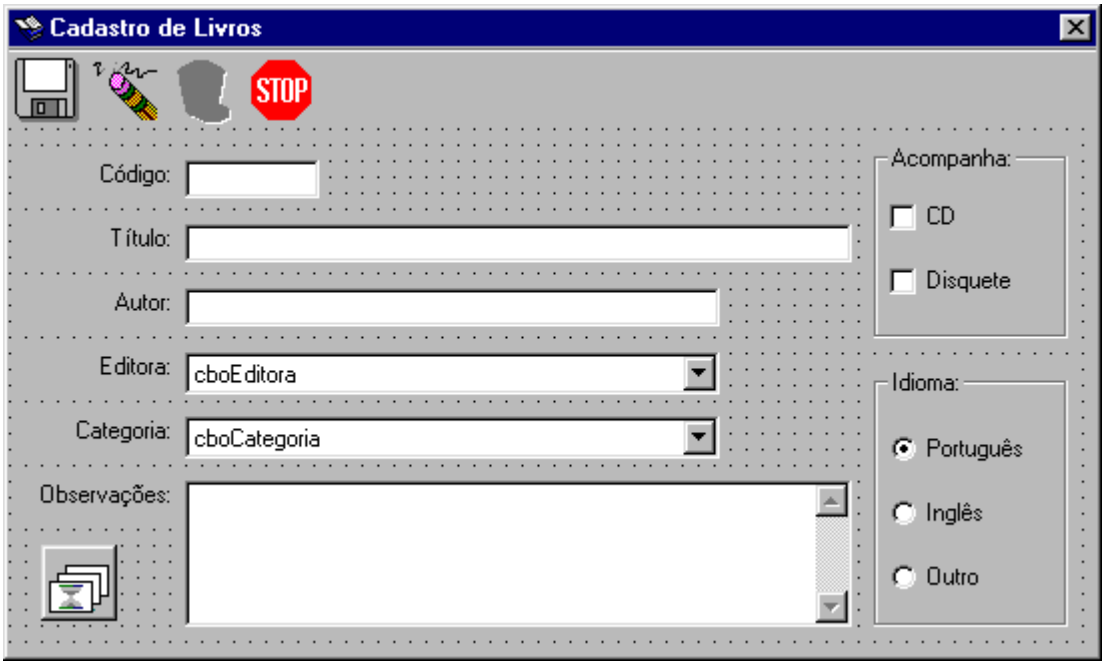
```
Private Sub lblTeste_Click(Index As Integer)
    lblTeste(Index).Caption = "Um texto qualquer"
End Sub
```

Por fim, não confunda o parâmetro Index, que nada mais é que uma variável recebida pela procedure, com a propriedade Index, que contém o número do índice do controle no vetor.

Observação: depois de criar um vetor, para fazer com que um de seus componentes volte a ser um controle independente, deve-se renomeá-lo e depois apagar o valor da propriedade Index.

Depois da apresentação dos controles que serão usados no formulário de cadastramento de Livros, podemos começar a trabalhar. Adicione um novo formulário ao projeto, formate-o como no modelo da próxima página e altere as propriedades dos objetos conforme a tabela a seguir (note que existe um *control array* para o idioma):

Objeto	***Nome***	***Outras Propriedades = Valor***
Formulário	frmCadLivros	Caption = Cadastro de Livros BorderStyle = 1 – Fixed Single Icon = Misc\Books02.ICO KeyPreview = True MDIChild = True
Caixa de texto	txtCodLivro	Text = "" MaxLength = 5
Caixa de texto	txtTitulo	Text = "" MaxLength = 50



Modelo do formulário do Cadastro de Livros

Objeto	***Nome***	***Outras Propriedades = Valor***
Caixa de texto	txtAutor	Text = “” MaxLength = 35
Caixa de Combinação	cboEditora	Style = 2 – DropDown List
Caixa de Combinação	cboCategoria	Style = 2 – DropDown List
Caixa de texto	txtObservacoes	Text = “” MultiLine = True ScroolBars = 2 - Vertical
Moldura	Frame1	Caption = Acompanha:
Caixa de Verificação	chkAcompCD	Caption = CD
Caixa de Verificação	chkAcompDisquete	Caption = Disquete
Moldura	Frame2	Caption = Idioma:
Botão de Opção	optIdioma	Caption = Português Index = 0 Value = True
Botão de Opção	optIdioma	Caption = Inglês Index = 1
Botão de Opção	optIdioma	Caption = Outro Index = 2
Image	ImageList1	Tamanho das imagens: 32 x 32 pixels
	Imagens	1 = Computer\Disk04.ICO 2 = Writing\Erase02.ICO 3 = Computer\Trash01.ICO 4 = TRAFFIC\TRFFC14.ICO
ToolBar	ToolBar1	ImageList = ImageList1 Appearance = 1 – cc3D BorderStyle = 0 – ccNone Style = 1 – tbrFlat
	Botão 1	ToolTipText = Grava o registro atual Image = 1
	Botão 2	ToolTipText = Limpa os dados digitados Image = 2
	Botão 3	ToolTipText = Exclui o registro atual Image = 3 Enabled = False (desmarcar)
	Botão 4	ToolTipText = Retorna ao menu principal Image = 4

Grave o formulário com o nome de **frmCadLivros**, e vamos então iniciar a programação. A maior parte dela segue a mesma lógica do cadastro de usuários, então começaremos pelos controles que estamos estudando agora:

1. Vamos precisar de algumas variáveis privadas para armazenar os valores das caixas de combinação e de verificação e dos botões de opção. Abra a janela Código na seção Declarations e escreva os comandos:

```
Dim vInclusao As Boolean
Dim vCodEditora, vCodCategoria As Long
Dim vAcompCD, vAcompDisquete As Boolean
Dim vIdioma As Byte
```

2. Escreva a procedure Form_KeyPress para permitir a mudança dos campos pela tecla Enter;

3. No evento Form_Load vamos inicializar as variáveis privadas:

```
Private Sub Form_Load()
    Me.Left = (frmBiblio.ScaleWidth - Me.Width) / 2
    Me.Top = (frmBiblio.Height - Me.Height) / 2
    'Inicializa as variáveis auxiliares:
    vCodEditora = 0
    vCodCategoria = 0
    vAcompCD = False
    vAcompDisquete = False
    vIdioma = 0
End Sub
```

4. A programação das caixas de verificação é muito simples: quando acontecer um click, verificamos se foi marcada ou desmarcada, e alteramos o valor da variável correspondente para True ou False, conforme o caso. Abra então a janela código dando um duplo-click sobre chkAcompCD:

```
Private Sub chkAcompCD_Click()
    'Verifica se chkAcompCD está ou não marcada, e atribui a vAcompCD o valor
    'correspondente:
    If chkAcompCD.Value = vbChecked Then
        vAcompCD = True
    Else
        vAcompCD = False
    End If
End Sub
```

5. Programe o evento Click da caixa de verificação chkAcompDisquete conforme a procedure acima;

6. A programação para os botões de opção é ainda mais fácil: basta atribuir à variável o valor do parâmetro Index:

```
Private Sub optIdioma_Click(Index As Integer)
    'Atribui à variável vIdioma o valor do parâmetro index, pois esse contém o
    'índice do botão que foi clicado: 0 = Português, 1 = Inglês e 2 = Outros:
    vIdioma = Index
End Sub
```

7. O próximo passo é fazer a programação das caixas de combinação. Vamos aproveitá-lo para aprender como criar recordsets a partir de consultas do Access. Por enquanto, grave seu trabalho e saia do VB;

Acionando Consultas do Banco de Dados

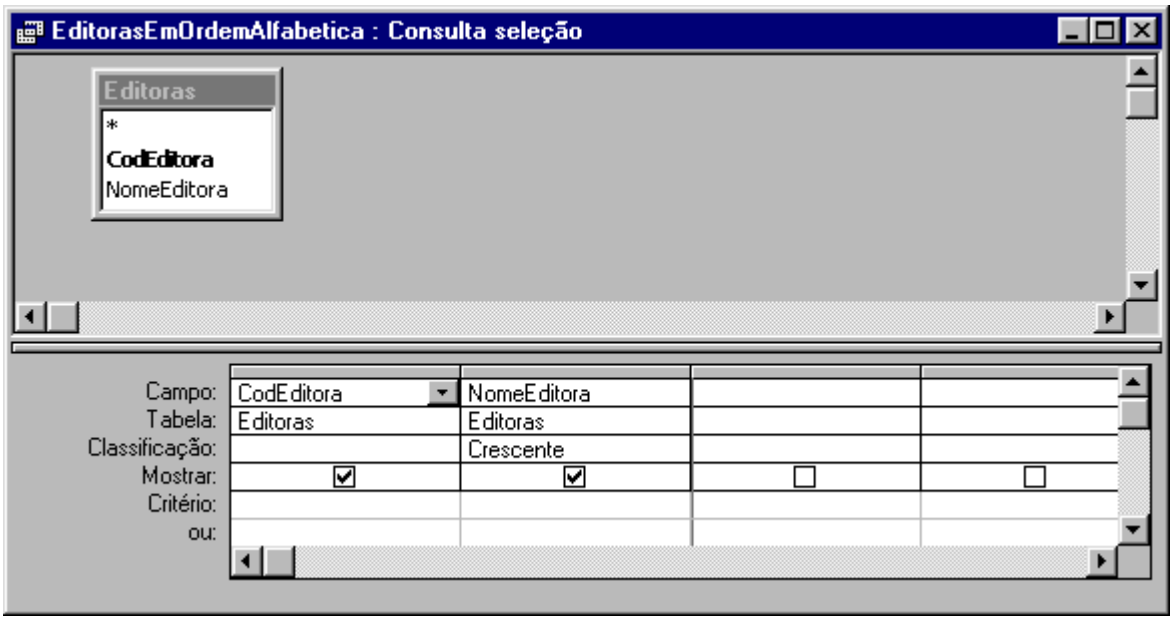
Muitas vezes utilizamos dados obtidos através de outros objetos do banco de dados que não sejam as tabelas, normalmente de consultas. No caso do Access, isso acontece muito quando precisamos de dados de tabelas relacionadas, ou com muitos critérios de seleção, pois é mais simples criar uma consulta que elaborar o comando SQL correspondente. A execução de uma consulta a partir da aplicação VB é muito simples:

1. Abra o Access e o banco de dados Biblio;

2. Crie uma nova consulta conforme o modelo da próxima página;

3. Salve a consulta com o nome de **EditorasEmOrdemAlfabetica**;

4. Aproveite para criar também a consulta **CategoriasEmOrdemAlfabetica,** com os campos CodCategoria e NomeCategoria;



A consulta EditorasEmOrdemAlfabetica

5. Saia do Access, abra o VB e o projeto Bibliotecario;

6. Vamos criar as procedures para carregar as combos no módulo de programação como públicas. Mas, se elas não serão criadas no formulário, como faremos para que reconheçam as combos, que são objetos locais? Simples: passaremos o nome da combo a ser carregada como parâmetro para a procedure. Abra a janela Código no módulo de programação Biblio;

7. Escreva a procedure ComboEditoras conforme segue:

```
Public Sub ComboEditoras(NomeCombo As ComboBox)
    Dim cnnComando As New ADODB.Command
    Dim rsTemp As New ADODB.Recordset
    Dim i As Integer
    On Error GoTo errComboEditoras
    'Executa a consulta EditorasEmOrdemAlfabetica:
    With cnnComando
        .ActiveConnection = cnnBiblio
        .CommandType = adCmdStoredProc
        .CommandText = "EditorasEmOrdemAlfabetica"
        Set rsTemp = .Execute
    End With
    With rsTemp
        'Verifica se existe alguma editora cadastrada:
        If Not (.EOF And .BOF) Then
            'Se existe, então posiciona o apontador no primeiro registro do rs:
            .MoveFirst
            'Inicializa a variável i que será usada como índice para a
            'propriedade ItemData:
            i = 0
            While Not .EOF
                'Adiciona um item à combo com o nome da editora:
                NomeCombo.AddItem !Descrição, i
                'Grava na propriedade ItemData desse o código da editora:
                NomeCombo.ItemData(i) = !Código
                'Vai para o próximo registro do rs:
                .MoveNext
                'Incrementa i:
                i = i + 1
            Wend
        End If
    End With

Saida:
    Set cnnComando = Nothing
    Set rsTemp = Nothing
```

```
    Exit Sub

errComboEditoras:
    With Err
        If .Number <> 0 Then
            MsgBox "Não foi possível a leitura da tabela de Editoras:", _
                    , vbInformation + vbOKOnly + vbApplicationModal, _
                    "Erro ao carregar ComboBox"
            .Number = 0
            GoTo Saida
        End If
    End With
End Sub
```

Obs.: NomeCombo é o parâmetro que vai receber a caixa de combinação a ser carregada. Assim, declaramos seu tipo como ComboBox. É dessa maneira que passamos objetos como parâmetros para subrotinas: indicando qual é o tipo desse objeto. Se você não souber o tipo do objeto, declare o parâmetro como Variant, dessa maneira o tipo será assumido dinamicamente na passagem do parâmetro.

8. Usando CarregaEditoras como base, crie um outra subrotina chamada CarregaCategorias executando a consulta CategoriasEmOrdemAlfabetica;

9. Abra o formulário frmCadLivros e acrescente a seguinte programação ao final do evento Form_Load, para fazer o carregamento das combos a partir das procedures que acabamos de criar:

```
'Chama a procedure ComboEditoras e passa a combo cboEditora como
'parâmetro:
ComboEditoras cboEditora
'O mesmo para cboCategorias:
ComboCategorias cboCategoria
'Garante que nenhum item esteja selecionado nas combos:
cboEditora.ListIndex = -1
cboCategoria.ListIndex = -1
```

10. Devemos agora programar o evento Click das combos para atribuir o código do item escolhido à variável correspondente. Comecemos por cboEditoras (atenção: o evento padrão de uma combo é o Change, mas a programação abaixo deve ser feita **no evento Click**, então cuidado para fazê-la corretamente):

```
Private Sub cboEditora_Click()
    With cboEditora
        'Verifica se foi selecionado um item da combo:
        If .ListIndex <> -1 Then
            'Se foi, atribui à variável vCodEditora o conteúdo da
            'propriedade ItemData:
            vCodEditora = .ItemData(.ListIndex)
        Else
            'Senão, zera a variável:
            vCodEditora = 0
        End If
    End With
End Sub
```

Obs.: como queremos armazenar na variável o código referente ao item selecionado, cujo índice é indicado por ListIndex, usamos essa propriedade como parâmetro para ItemData.

11. Baseado na procedure acima, escreva a programação para o evento Click da combo cboCategoria;

12. O próximo passo é apresentar os dados do registro selecionado nos campos do formulário em caso de alteração. Como no cadastro de usuários, isso será programado no evento LostFocus de txtCodLivro:

```
Private Sub txtCodLivro_LostFocus()
    Dim cnnComando As New ADODB.Command
    Dim rsSelecao As New ADODB.Recordset
    Dim vCod As Long
    Dim i As Integer
    On Error GoTo errSelecao
    'Converte o código digitado para pesquisa:
    vCod = Val(txtCodLivro.Text)
    'Se não foi digitado um código válido, sai da sub:
    If vCod = 0 Then Exit Sub
    Screen.MousePointer = vbHourglass
```

```
    'Tenta selecionar o registro na tabela de livros:
    With cnnComando
        .ActiveConnection = cnnBiblio
        .CommandType = adCmdText
        .CommandText = "SELECT * FROM Livros WHERE CodLivro = " & vCod & ";"
        Set rsSelecao = .Execute
    End With
    With rsSelecao
        If .EOF And .BOF Then
            'Se o recordset está vazio, não encontrou registro com esse código:
            LimparDados
            'Identifica a operacao como inclusão:
            vInclusao = True
        Else
            'Senão, atribui aos campos e variáveis auxiliares os dados do
            'registro:
            txtTitulo.Text = !Titulo
            txtAutor.Text = !Autor
            vCodEditora = !CodEditora
            vCodCategoria = !CodCategoria
            vAcompCD = !AcompCD
            vAcompDisquete = !AcompDisquete
            vIdioma = !Idioma
            'Como Observacoes não é um campo obrigatório, devemos impedir a
            'atribuição do valor nulo (se houver) à caixa de texto:
            txtObservacoes = Empty & !Observacoes
            'Exibe os dados das variáveis nos controles correspondentes:
            With cboEditora
                'Elimina a seleção atual:
                .ListIndex = -1
                'Como ListCount retorna o número de itens da combo,
                'ListCount - 1 é igual ao índice do último item. Portanto, o
                'loop abaixo será executado para todos os itens da combo
                'através de seu índice:
                For i = 0 To (.ListCount - 1)
                    If vCodEditora = .ItemData(i) Then
                        'Se ItemData for igual ao código atual,
                        'seleciona o item e sai do loop:
                        .ListIndex = i
                        Exit For
                    End If
                Next i
            End With
            With cboCategoria
                .ListIndex = -1
                For i = 0 To (.ListCount - 1)
                    If vCodCategoria = .ItemData(i) Then
                        .ListIndex = i
                        Exit For
                    End If
                Next i
            End With
            'Se vAcompCD = True, marca chkAcompCD, senão desmarca:
            chkAcompCD.Value = IIf(vAcompCD, vbChecked, vbUnchecked)
            chkAcompDisquete.Value = IIf(vAcompDisquete, vbChecked, vbUnchecked)
            'Marca o botão de opção correspondente ao idioma atual:
            optIdioma(vIdioma).Value = True
            'Habilita o botão Excluir:
            Toolbar1.Buttons(3).Enabled = True
            'Identifica a operação como Alteração:
            vInclusao = False
        End If
    End With
    'Desabilita a digitação do código:
    txtCodLivro.Enabled = False

Saida:
    'Elimina o command e o recordset da memória:
    Set rsSelecao = Nothing
```

```
    Set cnnComando = Nothing
    Screen.MousePointer = vbDefault
    Exit Sub

errSelecao:
    With Err
        If .Number <> 0 Then
            MsgBox "Erro na recuperação do registro solicitado:", _
                    vbExclamation + vbOKOnly + vbApplicationModal, "Aviso"
            .Number = 0
            GoTo Saida
        End If
    End With
End Sub
```

13. Na sub LimparDados eliminamos as seleções e marcações dos controles que estamos estudando:

```
Private Sub LimparDados()
    'Apaga o conteúdo dos campos do formulário:
    txtTitulo.Text = Empty
    txtAutor.Text = Empty
    txtObservacoes.Text = Empty
    'Elimina a seleção das combos:
    cboEditora.ListIndex = -1
    cboCategoria.ListIndex = -1
    'Desmarca as caixas de verificação:
    chkAcompCD.Value = vbUnchecked
    chkAcompDisquete.Value = vbUnchecked
    'Marca a opção Português em optIdioma:
    optIdioma(0).Value = True
    'Reinicializa as variáveis auxiliares:
    vCodEditora = 0
    vCodCategoria = 0
    vAcompCD = False
    vAcompDisquete = False
    vIdioma = 0
End Sub
```

14. Na sub GravarDados, passamos para os campos os dados das variáveis auxiliares:

```
Private Sub GravarDados()
    Dim cnnComando As New ADODB.Command
    Dim vSQL As String
    Dim vCod As Long
    Dim vConfMsg As Integer
    Dim vErro As Boolean
    On Error GoTo errGravacao
    'Converte o código digitado para gravação:
    vCod = Val(txtCodLivro.Text)
    'Verifica os dados digitados:
    vConfMsg = vbExclamation + vbOKOnly + vbApplicationModal
    vErro = False
    If vCod = 0 Then
        MsgBox "O campo Código não foi preenchido.", vConfMsg, "Erro"
        vErro = True
    End If
    If txtTitulo.Text = Empty Then
        MsgBox "O campo Título não foi preenchido.", vConfMsg, "Erro"
        vErro = True
    End If
    If txtAutor.Text = Empty Then
        MsgBox "O campo Autor não foi preenchido.", vConfMsg, "Erro"
        vErro = True
    End If
    If vCodEditora = 0 Then
        MsgBox "Não foi selecionada uma Editora.", vConfMsg, "Erro"
        vErro = True
    End If
    If vCodCategoria = 0 Then
        MsgBox "Não foi selecionada uma Categoria.", vConfMsg, "Erro"
```

```
        vErro = True
    End If
    'Se aconteceu um erro de digitação, sai da sub sem gravar:
    If vErro Then Exit Sub
    Screen.MousePointer = vbHourglass
    'Constrói o comando SQL para gravação:
    If vInclusao Then
        'Se é uma inclusão:
        vSQL = "INSERT INTO Livros (CodLivro, Titulo, Autor, CodEditora, " & _
            "CodCategoria, AcompCD, AcompDisquete, Idioma, Observacoes) " & _
            "VALUES (" & vCod & ",'" & _
            txtTitulo.Text & "','" & _
            txtAutor.Text & "'," & _
            vCodEditora & "," & _
            vCodCategoria & "," & _
            vAcompCD & "," & _
            vAcompDisquete & "," & _
            vIdioma & ",'" & _
            txtObservacoes.Text & "');"
    Else
        'Senão, alteração:
        vSQL = "UPDATE Livros SET Titulo = '" & txtTitulo.Text & _
            "', Autor = '" & txtAutor.Text & _
            "', CodEditora = " & vCodEditora & _
            ", CodCategoria = " & vCodCategoria & _
            ", AcompCD = " & vAcompCD & _
            ", AcompDisquete = " & vAcompDisquete & _
            ", Idioma = " & vIdioma & _
            ", Observacoes = '" & txtObservacoes.Text & _
            "' WHERE CodLivro = " & vCod & ";"
    End If
    'Executa o comando de gravação:
    With cnnComando
        .ActiveConnection = cnnBiblio
        .CommandType = adCmdText
        .CommandText = vSQL
        .Execute
    End With
    MsgBox "Gravação concluída com sucesso.", _
            vbApplicationModal + vbInformation + vbOKOnly, _
            "Gravação OK"
    'Chama a sub que limpa os dados do formulário:
    LimparTela

Saida:
    Screen.MousePointer = vbDefault
    Set cnnComando = Nothing
    Exit Sub

errGravacao:
    With Err
        If .Number <> 0 Then
            MsgBox "Erro durante a gravação dos dados no registro." & vbCrLf & _
                    "A operação não foi completada." , _
                    vbExclamation + vbOKOnly + vbApplicationModal, _
                    "Operação cancelada"
            .Number = 0
            GoTo Saida
        End If
    End With
End Sub
```

15. O restante da programação para esse formulário é semelhante àquela feita em frmCadUsuarios:

```
Private Sub Toolbar1_ButtonClick(ByVal Button As MSComctlLib.Button)
      Select Case Button.Index
            Case 1
                  GravarDados
            Case 2
                  LimparTela
```

```
            Case 3
                  ExcluirRegistro
            Case 4
                  Unload Me
      End Select
End Sub



Private Sub txtAutor_LostFocus()
    'Converte a primeira letra de cada palavra digitada em maiúscula,
    'e as demais em minúsculas:
    txtAutor.Text = StrConv(txtAutor.Text, vbProperCase)
End Sub



Private Sub txtTitulo_LostFocus()
    txtTitulo.Text = StrConv(txtTitulo.Text, vbProperCase)
End Sub



Private Sub ExcluirRegistro()
    Dim cnnComando As New ADODB.Command
    Dim vOk As Integer
    Dim vCod As Long
    On Error GoTo errExclusao
    'Solicita confirmação da exclusão do registro:
    vOk = MsgBox("Confirma a exclusão desse registro?", _
            vbApplicationModal + vbDefaultButton2 + vbQuestion + vbYesNo, _
            "Exclusão")
    If vOk = vbYes Then
        Screen.MousePointer = vbHourglass
        'Se confirmou, converte o código digitado:
        vCod = Val(txtCodLivro.Text)
        'Executa a operação:
        With cnnComando
            .ActiveConnection = cnnBiblio
            .CommandType = adCmdText
            .CommandText = "DELETE FROM Livros WHERE CodLivro = " & vCod & ";"
            .Execute
        End With
        MsgBox "Registro excluído com sucesso.", _
                vbApplicationModal + vbInformation + vbOKOnly, _
                "Exclusão OK"
        LimparTela
    End If

Saida:
    Screen.MousePointer = vbDefault
    Set cnnComando = Nothing
    Exit Sub

errExclusao:
    With Err
        If .Number <> 0 Then
            MsgBox "Erro durante a exclusão do registro." & vbCrLf & _
                    "A operação não foi completada.", _
                    vbExclamation + vbOKOnly + vbApplicationModal, _
                    "Operação cancelada"
            .Number = 0
            GoTo Saida
        End If
    End With
End Sub
```

```
Private Sub LimparTela()
      LimparDados
      Toolbar1.Buttons(3).Enabled = False
      txtCodLivro.Text = Empty
      txtCodLivro.SetFocus
End Sub
```

1. Faça no formulário principal a programação necessária para que o menu mnuCadLivros e o botão nº 1 da ToolBar abram o formulário frmCadLivros;

16. Grave seu trabalho;

17. Teste o funcionamento do formulário. Aproveite para adicionar alguns livros ao banco de dados.

Lição 11 – Datas e Relacionamentos

Uma das preocupações que todo programador deve ter é: como manter a integridade das informações do meu banco de dados? Por isso é tão importante a verificação dos dados digitados antes da gravação de um registro. Mas existem casos que essa verificação deve ser mais cuidadosa: na gravação de datas e quando existe um relacionamento entre as tabelas envolvidas. Nessa lição aprenderemos como tratar os dados nessas situações.

Gravação de Datas

Como já dissemos quando da apresentação dos tipos de dados admitidos pelo VB, para ele um dado tipo data nada mais é que um dado numérico com um formato especial que representa uma data. Devido a essa particularidade, para que possamos manipular datas corretamente no VB foram criadas algumas funções específicas. Vejamos as principais:

- **CDate**: nós já estudamos essa função quando falamos das funções de conversão. Ela converte uma string em uma data, se possível.
- **Date**: essa função retorna a data corrente do sistema, no formato especificado no Painel de Controle do Windows.
- **IsDate**: verifica se uma expressão pode ser convertida em uma data, e nesse caso retorna True, ou False caso contrário. A expressão pode ser um dado string, numérico, data ou variant. Sua sintaxe é:

  ```
  IsDate(<expressão>)
  ```

  Observação: também existem funções de verificação para outros tipos de dados. As principais são:

  - **IsNumeric**, para verificar se um dado é numérico,
  - **IsEmpty**, para verificar se uma string está vazia, e
  - **IsNull**, para verificar se uma expressão é nula.

Verificando os Relacionamentos

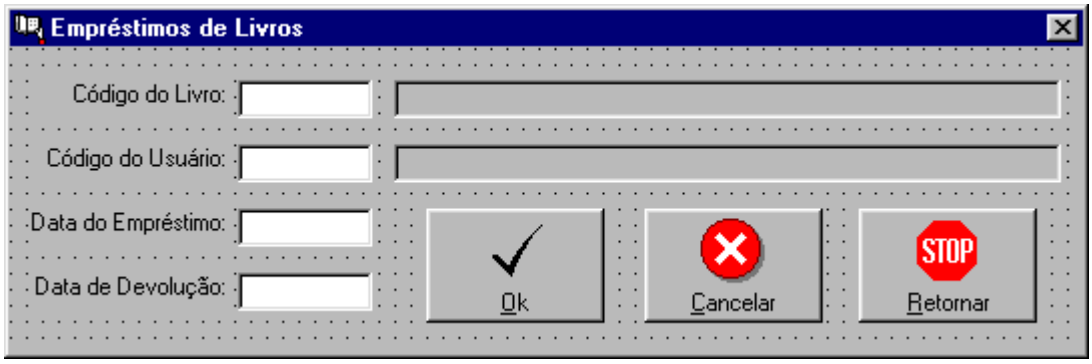
Os SGBD's atuais são em sua grande maioria do tipo relacional, ou seja, baseados no conceito de relacionamento entre tabelas para permitir a recuperação de informações impedindo redundância de dados. Um relacionamento é uma ligação entre a chave primária de uma tabela e uma chave estrangeira que contenha o mesmo dado em outra tabela. A tabela que contém a chave primária é chamada de *origem do relacionamento*, e a que contém a chave estrangeira é chamada de *destino do relacionamento*. O problema é que, para que um relaciomento funcione corretamente é preciso no destino não existam dados sem correspondência na tabela de origem. Isso implica que:

- Na inclusão ou alteração de registros na origem, a chave primária não pode ser repetida;
- Na inclusão ou alteração de registros no destino, deve-se verificar se existe o registro correspondente à chave estrangeira na origem;
- A alteração ou exclusão de registros na origem não deve ser permitida no caso de existirem registros relacionados no destino, a não ser que seja uma atualização em cascata (onde os registros correspondentes no destino também serão afetados pela operação executada na origem).

A esse conjunto de regras damos o nome de **Integridade Referencial**. Os SGBD's mais modernos – incluindo aí o Access – fazem a checagem de integridade referencial automaticamente, impedindo anomalias nas informações do banco. Mas isso não elimina a necessidade de checarmos os dados digitados. Por exemplo: em nosso sistema, no empréstimo de um livro, o operador informa o código do usuário que o está retirando. Devemos verificar se esse código existe na tabela de usuários, pois se enviarmos ao banco um código de usuário inválido, e os recursos de checagem de integridade referencial do banco estiverem ativados, a gravação não será realizada e o banco retornará um erro de execução.

Para aplicar esses conceitos, vamos criar o formulários de empréstimo de livros do sistema Bibliotecário. Note que nesse formulário será solicitada a digitação dos códigos do livro e do usuário, e portanto vamos precisar verificá-los antes de gravar o registro:

1. Adicione um novo formulário ao projeto e formate-o como no modelo:



A tabela abaixo contém as propriedades dos objetos:

Objeto	***Nome***	***Outras Propriedades = Valor***
Formulário	frmEmprestimos	Caption = Empréstimos de Livros BorderStyle = 1 – Fixed Single Icon = Misc\Book04.ICO KeyPreview = True MDIChild = True
Caixa de texto	txtCodLivro	Text = “” MaxLength = 5
Label	lblTitulo	Caption = “”
Caixa de texto	txtCodUsuario	Text = “” MaxLength = 5
Label	lblNomeUsuario	Caption = “”
Caixa de texto	txtDataEmprestimo	Text = “” MaxLength = 10
Caixa de texto	txtDataDevolucao	Text = “” MaxLength = 10
Botão de Comando	cmdOk	Caption = &Ok Style = 1 – Graphical Picture = Misc\Checkmrk.ICO
Botão de Comando	cmdCancelar	Caption = &Cancelar Style = 1 – Graphical Picture = Computer\W95mbx01.ICO
Botão de Comando	cmdRetornar	Caption = &Retornar Style = 1 – Graphical Picture = Traffic\Trffc14.ICO

2. Grave o novo formulário com o nome de **frmEmprestimos**;

3. Escreva as procedures Form_KeyPress e cmdRetornar_Click como já estudamos anteriormente;

4. No evento Load, vamos inicializar os campos txtDataEmprestimo e txtDataDevolucao. Note o uso da função Format:

```
Private Sub Form_Load()
    Me.Left = (frmBiblio.ScaleWidth - Me.Width) / 2
    Me.Top = (frmBiblio.Height - Me.Height) / 2
    'Atribui à caixa txtDataEmprestimo a data atual do sistema:
    txtDataEmprestimo.Text = Format(Date, "Short Date")
    'Atribui à caixa txtDataDevolucao a data atual do sistema + 15 dias:
    txtDataDevolucao.Text = Format((Date + 15), "Short Date")
End Sub
```

5. No evento LostFocus da caixa txtCodLivro, verificaremos se o código digitado é válido, e exibiremos o título do livro em lblTitulo:

```
Private Sub txtCodLivro_LostFocus()
    Dim cnnComando As New ADODB.Command
    Dim rsSelecao As New ADODB.Recordset
    Dim vCod As Long
    On Error GoTo errSelecao
```

```
    'Converte o código digitado:
    vCod = Val(txtCodLivro.Text)
    'Se não foi digitado um código válido, sai da sub:
    If vCod = 0 Then Exit Sub
    Screen.MousePointer = vbHourglass
    'Seleciona o livro solicitado:
    With cnnComando
        .ActiveConnection = cnnBiblio
        .CommandType = adCmdText
        .CommandText = "SELECT * FROM LIVROS WHERE CodLivro = " & vCod & ";"
        Set rsSelecao = .Execute
    End With
    With rsSelecao
        If .EOF And .BOF Then
            'Se não encontrou o livro, deixa o título do livro em branco:
            lblTitulo.Caption = Empty
        Else
            'Se encontrou, exibe o título do livro em lblTitulo:
            lblTitulo.Caption = !Titulo
            'Verifica se o livro já está emprestado:
            If !Emprestado Then
                'Se está, impede novo empréstimo:
                MsgBox "Esse livro já está emprestado, verifique.", _
                        vbExclamation + vbOKOnly + vbSystemModal, "Erro"
                'Volta o foco ao código para redigitação:
                txtCodLivro.SetFocus
            End If
        End If
    End With
Saida:
    Set rsSelecao = Nothing
    Set cnnComando = Nothing
    Screen.MousePointer = vbDefault
    Exit Sub

errSelecao:
    With Err
        If .Number <> 0 Then
            MsgBox "Houve um erro na recuperação do registro solicitado.", _
            vbExclamation + vbOKOnly + vbApplicationModal, "Aviso"
            .Number = 0
            GoTo Saida
        End If
    End With
End Sub
```

6. A codificação para o campo txtCodUsuario é um pouco mais simples:

```
Private Sub txtCodUsuario_LostFocus()
    Dim cnnComando As New ADODB.Command
    Dim rsSelecao As New ADODB.Recordset
    Dim vCod As Long
    On Error GoTo errUsuario
    vCod = Val(txtCodUsuario.Text)
    If vCod = 0 Then Exit Sub
    Screen.MousePointer = vbHourglass
    With cnnComando
        .ActiveConnection = cnnBiblio
        .CommandType = adCmdText
        .CommandText = "SELECT NomeUsuario FROM Usuarios " & _
            "WHERE CodUsuario = " & vCod & ";"
        Set rsSelecao = .Execute
    End With
    With rsSelecao
        If .EOF And .BOF Then
            MsgBox "Código do usuário inválido, verifique.", _
                    vbExclamation + vbOKOnly + vbSystemModal, "Erro"
            lblNomeUsuario.Caption = Empty
            txtCodUsuario.SetFocus
        Else
```

```
            'Se encontrou, exibe o nome do usuário em lblNomeUsuario:
            lblNomeUsuario.Caption = !NomeUsuario
        End If
    End With

Saida:
    Set rsSelecao = Nothing
    Set cnnComando = Nothing
    Screen.MousePointer = vbDefault
    Exit Sub

errUsuario:
    With Err
        If .Number <> 0 Then
            MsgBox "Houve um erro na recuperação do usuário solicitado.", & _
                  vbExclamation + vbOKOnly + vbApplicationModal, "Aviso"
            .Number = 0
            GoTo Saida
        End If
    End With
End Sub
```

7. No evento LostFocus de txtDataEmprestimo, antes de formatar a digitação, precisamos verificar se o texto do campo é ou não um data válida. Para isso usaremos IsDate:

```
Private Sub txtDataEmprestimo_LostFocus()
    Dim vDataDevolucao As Date
    With txtDataEmprestimo
        If .Text <> Empty Then
            'Verifica se o dado digitado pode ser uma data:
            If Not IsDate(.Text) Then
                'Se o dado digitado não pode ser uma data, assume a do sistema:
                .Text = Date
            Else
                'Senão, formata a data digitada:
                .Text = Format(.Text, "Short Date")
            End If
            'Atribui à caixa txtDataDevolucao a data digitada + 15 dias:
            vDataDevolucao = CDate(.Text) + 15
            txtDataDevolucao.Text = Format(vDataDevolucao, "Short Date")
        End If
    End With
End Sub
```

8. O mesmo vale para txtDataDevolucao:

```
Private Sub txtDataDevolucao_LostFocus()
    'Formata a data digitada:
    With txtDataDevolucao
        If .Text <> Empty Then
            If Not IsDate(.Text) Then
                'Se o dado digitado não pode ser uma data, assume
                'a data do sistema + 15 dias:
                .Text = Date + 15
            Else
                'Senão, formata a data digitada:
                .Text = Format(.Text, "Short Date")
            End If
        End If
    End With
End Sub
```

9. O botão Ok grava os dados do empréstimo no cadastro de livros:

```
Private Sub cmdOk_Click()
    Dim cnnComando As New ADODB.Command
    Dim vCodLivro, vCodUsuario As Long
    Dim vDataEmprestimo, vDataDevolucao As Date
    Dim vConfMsg As Integer
    Dim vErro As Boolean
    On Error GoTo errGravacao
```

```
    'Converte os dados digitados:
    vCodLivro = Val(txtCodLivro.Text)
    vCodUsuario = Val(txtCodUsuario.Text)
    vDataEmprestimo = CDate(txtDataEmprestimo.Text)
    vDataDevolucao = CDate(txtDataDevolucao.Text)
    'Verifica os dados digitados:
    vConfMsg = vbExclamation + vbOKOnly + vbSystemModal
    vErro = False
    If vCodLivro = 0 Then
        MsgBox "O campo Código do Livro não foi preenchido.", vConfMsg, "Erro"
        vErro = True
    End If
    If lblTitulo.Caption = Empty Then
        'Se lblTitulo estiver em branco, o código do livro informado não foi
        'encontrado na tabela:
        MsgBox "O campo Código do Livro não contém um dado válido.", _
            vConfMsg, "Erro"
        vErro = True
    End If
    If vCodUsuario = 0 Then
        MsgBox "O campo Código do Usuário não foi preenchido.", vConfMsg, "Erro"
        vErro = True
    End If
    If vDataEmprestimo >= vDataDevolucao Then
        MsgBox "A Data de Devolucao informada é menor ou igual à Data de " & _
            “Empréstimo, verifique.", vConfMsg, "Erro"
        vErro = True
    End If
    'Se aconteceu um erro de digitação, sai da sub sem gravar:
    If vErro Then Exit Sub
    Screen.MousePointer = vbHourglass
    'Executa a SP de devolução de livros:
    With cnnComando
        .ActiveConnection = cnnBiblio
        .CommandType = adCmdText
        .CommandText = "UPDATE Livros SET CodUsuario = " & vCodUsuario & _
            ", Emprestado = True, DataEmprestimo = " & vDataEmprestimo & _
            ", DataDevolucao = " & vDataDevolucao & " WHERE CodLivro = " & _
            vCodLivro & ";"
        .Execute
    End With
    MsgBox "Esse livro foi emprestado a " & lblNomeUsuario & " com sucesso.", _
            vbApplicationModal + vbInformation + vbOKOnly, "Empréstimo OK"
    'Chama a subrotina cmdCancelar_Click para limpar os campos do formulário:
    cmdCancelar_Click

Saida:
    Screen.MousePointer = vbDefault
    Set cnnComando = Nothing
    Exit Sub

errGravacao:
    With Err
        If .Number <> 0 Then
            MsgBox "Erro durante a gravação dos dados no registro." & _
                  vbCrLf & "A devolução desse livro não foi completada.", _
                  vbExclamation + vbOKOnly + vbApplicationModal, _
                  "Empréstimo cancelado"
            .Number = 0
            GoTo Saida
        End If
    End With
End Sub
```

10. Finalmente, o botão Cancelar apaga os dados do formulário sem gravá-los:

```
Private Sub cmdCancelar_Click()
    'Cancela a digitação atual e limpa todos os campos do formulário:
    txtCodLivro.Text = Empty
    lblTitulo.Caption = Empty
```

```
    txtCodUsuario.Text = Empty
    lblNomeUsuario.Caption = Empty
    txtDataEmprestimo.Text = Format(Date, "Short Date")
    txtDataDevolucao.Text = Format((Date + 15), "Short Date")
    txtCodLivro.SetFocus
End Sub
```

11. Abra o formulário frmBiblio e faça a programação necessária para que o menu mnuEmprestimos e o botão nº 3 da ToolBar abram o formulário frmEmprestimos;
12. Grave seu trabalho;
13. Teste a programação "emprestando" alguns livros.

Joins

Não haveria razão para relacionar tabelas se não pudéssemos recuperar as informações mais facilmente. Em SQL, usamos o comando SELECT e a cláusula JOIN para exibir dados de tabelas relacionadas. A sintaxe de SELECT com JOIN é a seguinte:

```
SELECT <lista de campo> FROM <destino>
<INNER ou RIGHT ou LEFT> JOIN <origem>
ON <chave estrangeira> = <chave primária>;
```

A primeira linha contém um comando SELECT comum. A segunda linha indica o tipo de JOIN que deve ser usado:

- **INNER**: serão selecionadas para exibição somente as linhas que contenham dados de ambas as tabelas;
- **RIGHT**: serão selecionadas para exibição todas as linhas da tabela que está a direita (right) do JOIN, ou seja, da origem do relacionamento, mesmo que não existam dados correspondentes no destino;
- **LEFT**: serão selecionadas para exibição todas as linhas da tabela que está a esquerda (left) do JOIN, ou seja, do destino do relacionamento, mesmo que não existam dados correspondentes na origem;

Por exemplo: a tabela de livros não contém o nome da editora. Mas podemos exibí-lo juntamente aos dados do livro, com o seguinte comando:

```
SELECT Livros.*, Editoras.NomeEditora FROM Livros INNER JOIN Editoras ON
Livros.CodEditora = Editoras.CodEditora;
```

Note que, como estamos trabalhando com mais de uma tabela, precisamos identificar de que tabela vem o campo desejado, através do ponto ".". Isso fica bem evidente no campo CodEditora, que existe em ambas as tabelas: para indicar o campo da tabela de Livros, usamos **Livros.CodEditora**. A indicação **Livros.*** indica todos os campos da tabela Livros.

Outro exemplo: o comando abaixo exibe todos os usuários cadastrados, e o título do livro quando o usuário estiver com algum emprestado:

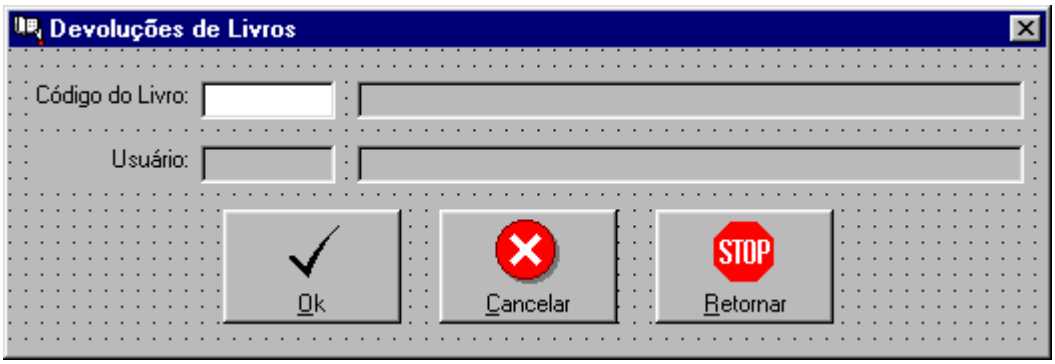
```
SELECT Usuarios.*, Livros.Titulo FROM Usuarios LEFT JOIN Livros ON
Usuarios.CodLivro = Livros.CodLivros;
```

É interessante destacar que JOIN pode ser combinado com outras cláusulas do comando SELECT, como WHERE e ORDER BY, sem nenhum problema. Como exemplo, vamos alterar o comando acima para ordenar o resultado pelo título do livro:

```
SELECT Livros.*, Editoras.NomeEditora FROM Livros INNER JOIN Editoras ON
Livros.CodEditora = Editoras.CodEditora ORDER BY Livros.Titulo;
```

Mas vamos aplicar o que aprendemos: o próximo passo em nosso sistema é a devolução de livros. Para isso vamos apagar as datas de devolução e empréstimo do registro correspondente ao livro que está sendo devolvido, e assim liberá-lo para outro empréstimo. Vejamos como fazer isso:

1. Adicione ao projeto um novo formulário, formate-o como o modelo da próxima página e defina as propriedades dos objetos conforme a tabela;
2. Grave o novo formulário com o nome de **frmDevolucoes**;
3. Escreva na procedure Form_Load os comandos para centralizar o formulário;
4. Crie as procedures Form_KeyPress e cmdRetornar_Click do novo formulário como estudamos nas lições anteriores;



Objeto	***Nome***	***Outras Propriedades = Valor***
Formulário	frmDevolucoes	Caption = Devoluções de Livros BorderStyle = 1 – Fixed Single Icon = Misc\Books04.ICO KeyPreview = True MDIChild = True
Caixa de texto	txtCodLivro	Text = “” MaxLength = 5
Label	lblTitulo	Caption = “”
Label	lblCodUsuario	Caption = “”
Label	lblNomeUsuario	Caption = “”
Botão de Comando	cmdOk	Caption = &Ok Style = 1 – Graphical Picture = Misc\Checkmrk.ICO
Botão de Comando	cmdCancelar	Caption = &Cancelar Style = 1 – Graphical Picture = Computer\W95mbx01.ICO
Botão de Comando	cmdRetornar	Caption = &Retornar Style = 1 – Graphical Picture = Traffic\Trffc14.ICO

5. A programação do campo txtCodLivro nesse formulário é um pouco mais complicada que no anterior, pois aqui devemos exibir também os dados do usuário, já que obviamente seu código não será digitado. Veja como ficou:

```
Private Sub txtCodLivro_LostFocus()
    Dim cnnComando As New ADODB.Command
    Dim rsLivro As New ADODB.Recordset
    Dim vCod As Long
    On Error GoTo errSelecao
    vCod = Val(txtCodLivro.Text)
    If vCod = 0 Then Exit Sub
    Screen.MousePointer = vbHourglass
    'Seleciona o livro solicitado:
    With cnnComando
        .ActiveConnection = cnnBiblio
        .CommandType = adCmdText
        .CommandText = "SELECT Livros.*, Usuarios.NomeUsuario " & _
            "FROM Livros LEFT JOIN Usuarios " & _
            "ON Livros.CodUsuario = Usuarios.CodUsuario " & _
            "WHERE Livros.CodLivro = " & vCod & ";"
        Set rsLivro = .Execute
    End With
    With rsLivro
        If .EOF And .BOF Then
            'Se não encontrou o livro, deixa os labels em branco:
            lblTitulo.Caption = Empty
            lblCodUsuario.Caption = Empty
            lblNomeUsuario.Caption = Empty
        Else
            'Se encontrou, exibe os dados do livro e do usuário:
            lblTitulo.Caption = !Titulo
```

```
            lblCodUsuario.Caption = Empty & !CodUsuario
            lblNomeUsuario.Caption = Empty & !NomeUsuario
            'Verifica se o livro está emprestado:
            If Not !Emprestado Then
                'Se não está, impede a devolução:
                MsgBox "Esse livro não está emprestado, verifique.", _
                        vbExclamation + vbOKOnly + vbSystemModal, "Erro"
                'Volta o foco ao código para redigitação:
                txtCodLivro.SetFocus
            End If
        End If
    End With

Saida:
    Set rsLivro = Nothing
    Set cnnComando = Nothing
    Screen.MousePointer = vbDefault
    Exit Sub

errSelecao:
    With Err
        If .Number <> 0 Then
            MsgBox "Erro na recuperação do registro solicitado." & vbCrLf & _
                   vbExclamation + vbOKOnly + vbApplicationModal, "Aviso"
            .Number = 0
            GoTo Saida
        End If
    End With
End Sub
```

6. O botão Ok, nesse formulário, apaga os dados gravados nos campos de empréstimo do cadastro de livros:

```
Private Sub cmdOk_Click()
    Dim cnnComando As New ADODB.Command
    Dim vCod As Long
    On Error GoTo errGravacao
    vCod = Val(txtCodLivro.Text)
    'Verifica o código digitado:
    If vCod = 0 Then
        MsgBox "O campo Código do Livro não foi preenchido.", _
                vbExclamation + vbOKOnly + vbSystemModal, "Erro"
        Exit Sub
    End If
    If lblTitulo.Caption = Empty Then
        MsgBox "O campo Código do Livro não contém um dado válido.", _
                vbExclamation + vbOKOnly + vbSystemModal, "Erro"
        Exit Sub
    End If
    Screen.MousePointer = vbHourglass
    With cnnComando
        .ActiveConnection = cnnBiblio
        .CommandType = adCmdText
        .CommandText = "UPDATE Livros SET Emprestado = False, " & _
            "CodUsuario = Null, " & _
            "DataEmprestimo = Null, " & _
            "DataDevolucao = Null " & _
            "WHERE CodLivro = " & vCod & ";"
        .Execute
    End With
    MsgBox "A devolução do livro " & lblTitulo & " foi efetuada com sucesso.", _
            vbApplicationModal + vbInformation + vbOKOnly, "Devolução OK"
    cmdCancelar_Click

Saida:
    Screen.MousePointer = vbDefault
    Set cnnComando = Nothing
    Exit Sub
```

```
errGravacao:
    With Err
        If .Number <> 0 Then
            MsgBox "Erro durante a gravação dos dados no registro." & vbCrLf & _
                    "A devolução desse livro não foi completada.", _
                    vbExclamation + vbOKOnly + vbApplicationModal, _
                    "Devolução cancelada"
            .Number = 0
            GoTo Saida
        End If
    End With
End Sub
```

7. Só faltou a procedure cmdCancelar_Click:

```
Private Sub cmdCancelar_Click()
    txtCodLivro.Text = Empty
    lblTitulo.Caption = Empty
    lblCodUsuario.Caption = Empty
    lblNomeUsuario.Caption = Empty
    txtCodLivro.SetFocus
End Sub
```

8. Grave o formulário;

9. Abra o frmBiblio e escreva a programação necessária para o menu mnuDevolucoes e para o botão nº 4 da ToolBar. Ambos devem abrir o formulário frmDevolucoes;

10. Teste o aplicativo e veja se está tudo funcionando como esperado.

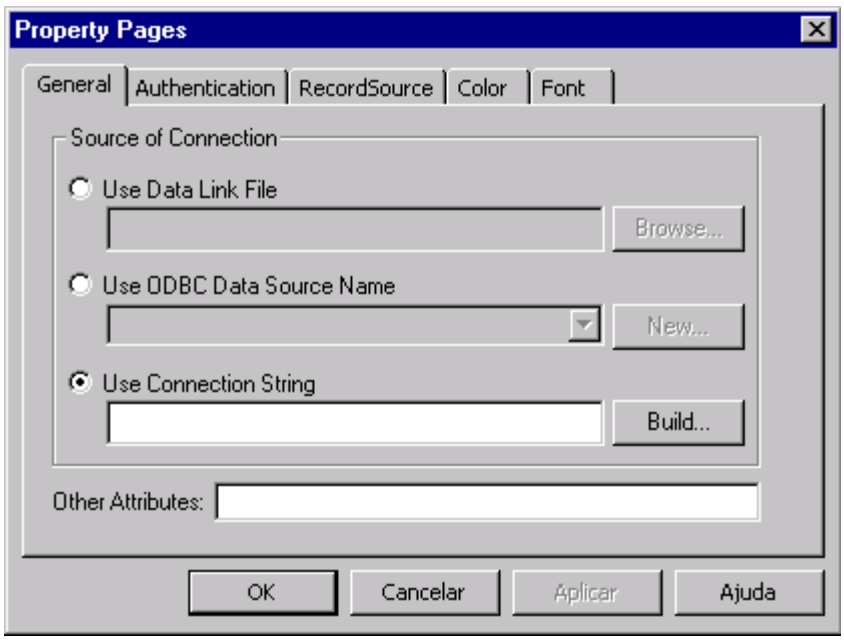
Lição 12: Controles de Dados

Nessa lição construiremos os formulários de consultas do sistema Bibliotecário. Para isso, aprenderemos a usar os controles de dados (ou *ADO Data Controls*), o *Microsoft DataGrid* e os *Microsoft DataList Controls*. Eles facilitam muito nossa vida quando precisamos fazer consultas à base de dados do sistema.

O Controle ADODC

O controle ADODC (ou *ADO Data Control*) permite a manipulação de bases de dados via ADO de maneira muito fácil e rápida, pois ele permite vincular controles como caixas de texto e labels diretamente aos campos do banco de dados. A grande vantagem de seu uso é a possibilidade de fazer a manutenção dos registros da base com bem pouca programação. A maior desvantagem é a pouca flexibilidade que oferece para tratamento dos dados. ADODC é um controle customizado: para utilizá-lo adicione ao projeto o componente *Microsoft ADO Data Control*.

A maneira mais simples de configurar o ADODC é pela janela de propriedades, acionada com o botão direito do mouse:



Um pequena descrição das abas:

- *General* e *Authentication*: contém parâmetros de configuração da conexão com o banco de dados;
- *RecordSource*: contém o tipo e a origem dos dados que serão vinculados ao ADODC;
- *Color* e *Font*: servem para customizar a aparência do controle.

As principais propriedades do ADODC são:

Propriedade ConnectionString (aba General)

Nessa propriedade criamos a string de conexão do ADODC com o banco de dados. Normalmente, é a mesma string usada na conexão principal. O botão *Build*, visível na figura acima, aciona um assistente que constrói a string a partir de algumas indicações, como por exemplo o nome do arquivo MDB.

Propriedades UserName e Password (aba Authentication)

Essas propriedades, como você já deve ter percebido, contém respectivamente o nome do usuário e a senha para conexão ao banco de dados. Podem ser informadas também pelo assistente *Build* da aba *General*.

Propriedade CommandType (aba RecordSource)

Informa o tipo do comando ADO que será criado pelo controle. Os tipos são os mesmos admitidos pelo objeto ADO Command.

Propriedade RecordSource (aba RecordSource)

Informa o texto do comando a ser executado pelo controle ADODC. Dependendo do tipo informado em CommandType, pode ser o nome de uma tabela ou de uma consulta, ou então um comando SQL.

Propriedades DataSource e DataField

Essas não são propriedades do controle ADODC, e sim dos controles que podem ser vinculados a ele. *DataSource* informa o nome do controle de dados ao qual será ligado, e *DataField* qual campo será acessado.

Método Refresh

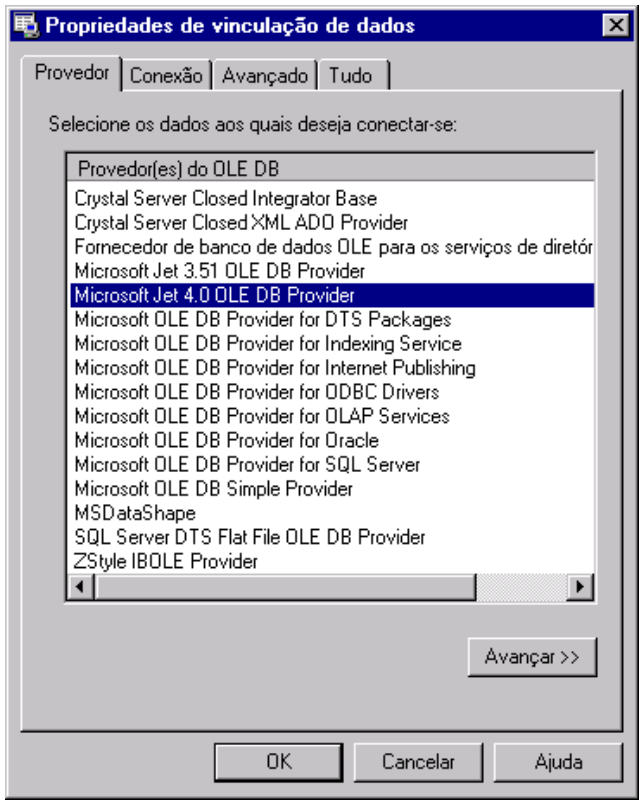
Existem casos em que é necessário alterar o RecordSource do controle de dados em tempo de execução, via programação. Acontece que a atualização do recordset do ADODC não é automática numa situação dessas. Para que o recordset seja atualizado e os dados sejam exibidos corretamente de acordo com o novo RecordSource, aplicamos ao ADODC o método Refresh.

Vamos criar um pequeno aplicativo para exemplificar o uso do controle ADODC:

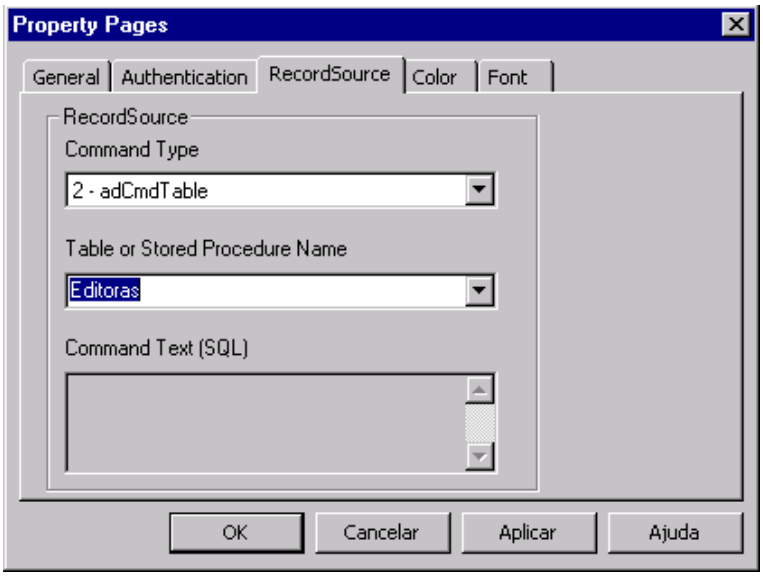
1. Inicie um novo projeto;
2. Crie o formulário conforme o modelo e a tabela de propriedades:

Objeto	***Nome***	***Outras Propriedades = Valor***
Formulário	frmTesteData	Caption = Tabela de Editoras BorderStyle = 1 – Fixed Single StartupPosition = 2 – Center Screen
Botão de comando	cmdFechar	Caption = &Fechar
ADODC	Adodc1	Caption = Editoras
Label	lblCodigo	Caption = “” BorderStyle = 1 – Fixed Single BackColor = Windows BackGround
Label	lblNome	Caption = “” BorderStyle = 1 – Fixed Single BackColor = Windows BackGround

3. Vamos usar a janela de propriedades para configurar a conexão do ADODC. Clique sobre ele com o botão direito do mouse e escolha a opção **ADODC Properties**;
4. Clique no botão Build (ao lado do campo Connection String). O VB abre a janela do assistente de conexão (veja a figura na próxima página);
5. Na aba Provedor, escolha a versão mais recente do Jet que estiver disponível (na figura a versão é a 4.0) e clique no botão Avançar;
6. Na aba Conexão, informe no item 1 o nome do banco de dados – Biblio.MDB, no nosso caso. Não se esqueça de informar o caminho completo (use o botão com as reticências, é mais simples);
7. Ainda na aba Conexão, o item 2 pede o nome do usuário (o assistente sugere Admin) e a senha. Como nosso banco não tem senha, deixe como está;



8. Clique no botão “Testar Conexão”. Se você receber a mensagem “Teste de conexão concluído com êxito”, a conexão está pronta;
9. Clique em Ok: o assistente informa a string de conexão para a propriedade ConnectionString, já com todos os parâmetros necessários;
10. Como nosso banco de dados não tem senha, não precisamos alterar nada em Authentication. Clique na aba RecordSource;
11. Em CommandType, selecione a opção **2 – adCmdTable**: a linha “Table or Stored Procedure Name” é habilitada;
12. Em “Table or Stored Procedure Name”, selecione a tabela de Editoras (veja o exemplo):



13. Clique em OK – a configuração do ADODC está pronta;
14. Selecione o label lblCodigo. Na janela de propriedades, escolha em DataSource o **Adodc1** e em DataField o campo **CodEditora**;
15. Faça o mesmo para o label lblNome, obviamente selecionando o campo **NomeEditora**;
16. Grave o formulário com o nome de **frmData** e o projeto como **TesteData**.

Sabe qual a programação que precisaremos fazer para esse aplicativo? Nenhuma! A não ser a do botão Fechar, é obvio. Execute o aplicativo e veja como ele funciona: basta clicar nos botões do contole de dados e os labels são atualizados de modo a exibir os campos do registro em que o apontador estiver posicionado:

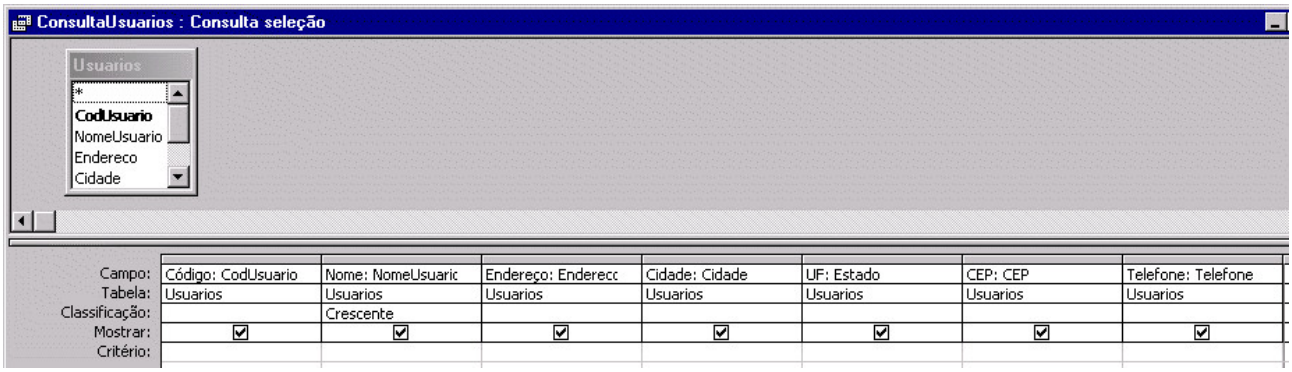
Você deve estar achando o *ADO Data Control* uma maravilha. Realmente, ele facilita muito as coisas, mas não se entusiasme demais: como já disse, ele não é muito flexível quando o assunto é a manutenção de uma base de dados em um aplicativo realmente sério, razão pela qual não é utilizado pelos programadores profissionalmente. Pelo menos não para isso: logo veremos uma aplicação realmente prática.

O Controle DataGrid

Esse controle ADO, que trabalha vinculado a um ADODC, lembra uma planilha em que linhas e colunas definem células. Cada linha é vinculada a um registro e cada coluna corresponde a um campo do recordset do ADODC. Também é um controle customizado, e portanto para poder utilizá-lo você deve adicionar o componente *Microsoft DataGrid Control*. Assim como o ADODC, é mais fácil configurá-lo pela janela de propriedades, mas nesse caso as opções da janela variam de acordo com o conjunto de dados que será apresentado. Um recurso muito interessante do DataGrid é a opção **Retrieve Fields** do menu de contexto (aquele menu acionado pelo botão direito do mouse). Após a vinculação do DataGrid ao ADODC, Retrieve Fields configura automaticamente as colunas do grid de acordo com os campos do recordset.

Vamos então elaborar uma consulta com o DataGrid. Antes, porém, criaremos uma consulta no Access, que será acessada pelo formulário de consulta ao cadastro de usuários:

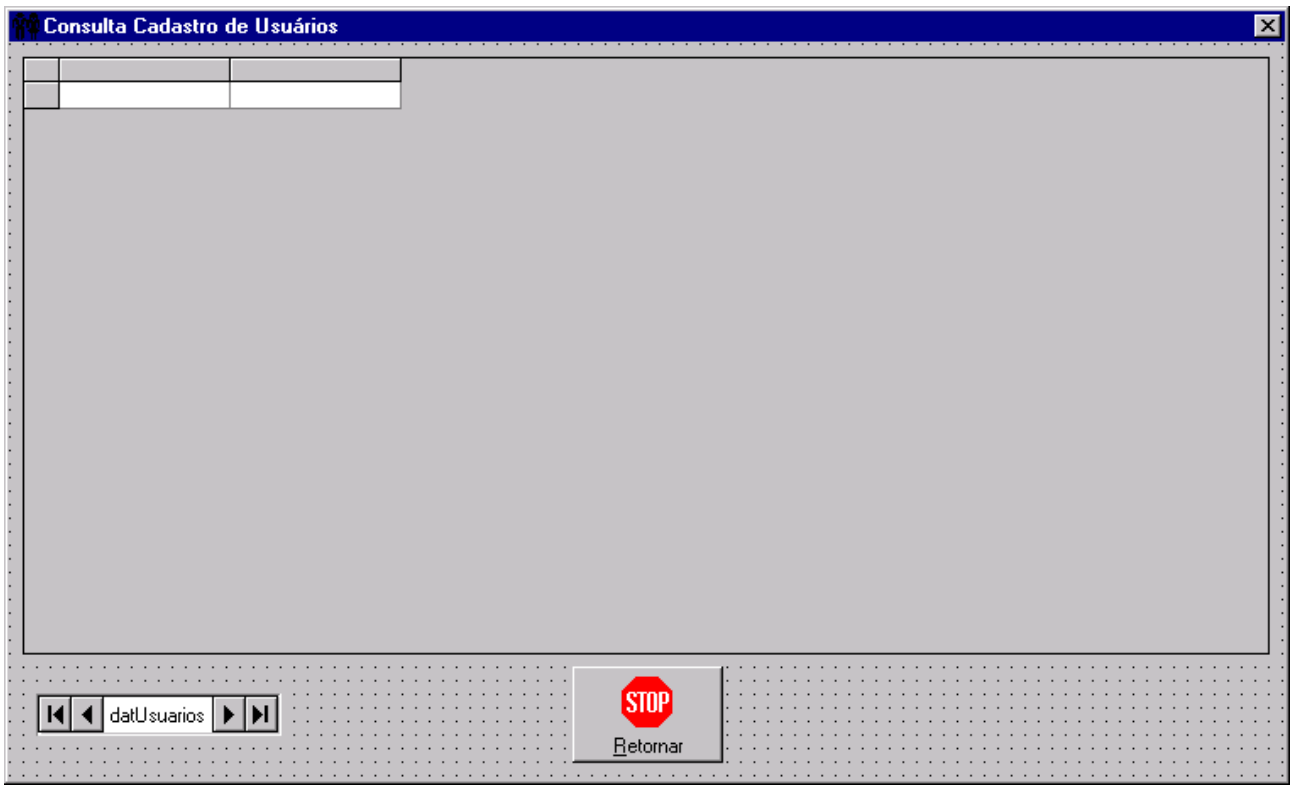
1. Entre no Access e abra o banco de dados Biblio;
2. Inicie a criação de uma nova consulta em modo estrutura;
3. Adicione a tabela Usuarios;
4. Mova todos os campos da tabela para o grid da consulta;
5. Altere a classificação do campo NomeUsuario para **Crescente**;
6. Mude os nomes das colunas indicadas (deixe as outras como estão):
   - **Código**, para o campo CodUsuario;
   - **Nome**, para o campo NomeUsuario;
   - **Endereço**, para o campo Endereco;
   - **UF**, para o campo Estado.
7. Compare sua consulta com o exemplo da próxima página;
8. Grave a consulta com o nome de **ConsultaUsuarios**;
9. Execute a consulta e verifique se está funcionando corretamente (veja o exemplo);
10. Aproveite para criar consultas semelhantes para as tabelas Categorias e Editoras;
11. Saia do Access, abra novamente o VB e o projeto Bibliotecario;



ConsultaUsuarios em modo estrutura

ConsultaUsuarios sendo executada no Access

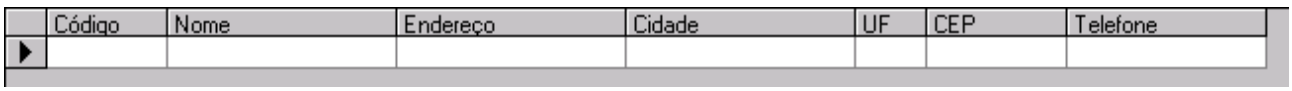
12. Adicione um novo formulário e formate-o conforme o modelo:



13. Altere as propriedades dos objetos:

Objeto	***Nome***	***Outras Propriedades = Valor***
Formulário	frmConUsuarios	Caption = Consulta Cadastro de Usuários BorderStyle = 1 – Fixed Single Icon = Misc\Misc28.ICO MDIChild = True
ADODC	datUsuarios	Caption = datUsuarios Visible = False ConnectionString = crie com o assistente CommandType = 2 – adCmdTable Table or Stored Procedure Name = ConsultaUsuarios
DataGrid	grdUsuarios	DataSource = datUsuarios
Botão de comando	cmdRetornar	Caption = &Retornar Style = 1 – Graphical Picture = Traffic\Trffc14.ICO

14. Clique com o botão direito do mouse sobre o DataGrid. No menu de contexto, escolha **Retrieve Fields**. O VB pede uma confirmação antes de fazer a configuração do grid: responda Ok, e seu DataGrid deve estar parecido com o da figura:

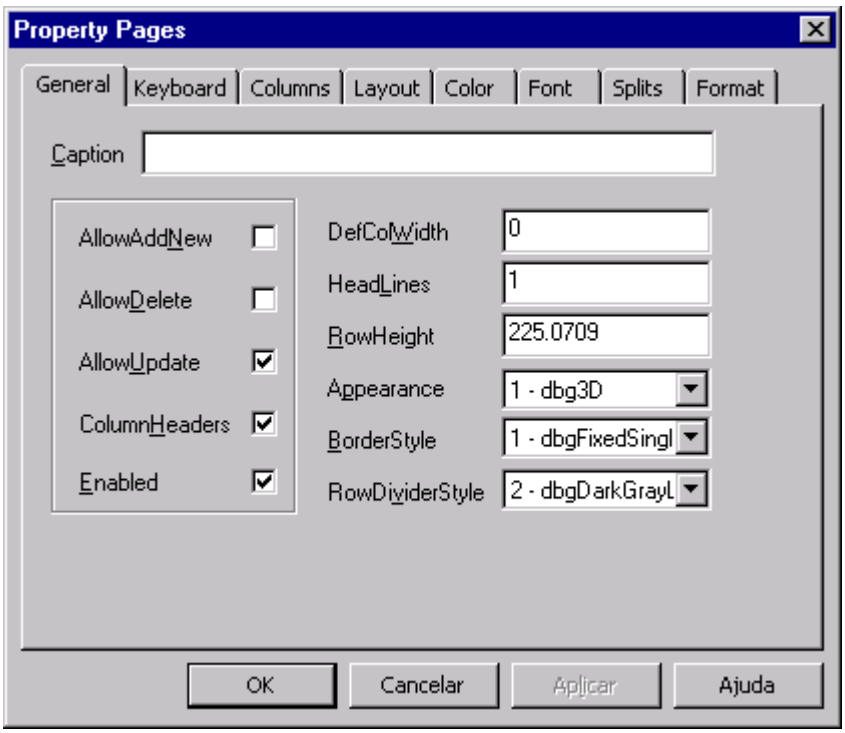


15. Clique novamente com o botão direito sobre grdUsuarios e escolha a opção **Properties** (veja a figura na próxima página);

16. A aba General contém opções de funcionamento e aparência do DataGrid. Desmarque a opção **AllowUpdate**, que permite a alteração de dados diretamente no grid, o que não é recomendável em uma consulta. As opções **AllowAddNew** e **AllowDelete** também devem estar desmarcadas, e servem, respectivamente, para permitir a inclusão e a exclusão de registros;

17. A aba Keyboard contém opções de navegação pelo teclado. Não será necessária nenhuma alteração;

18. A aba Columns permite a definição dos títulos das colunas do grid, e a qual campo do conjunto de dados a coluna deve ser vinculada. Retrieve Fields já faz a configuração dessas opções automaticamente, e portanto somente se quiser alterar o título de alguma coluna você precisará mudar algo;



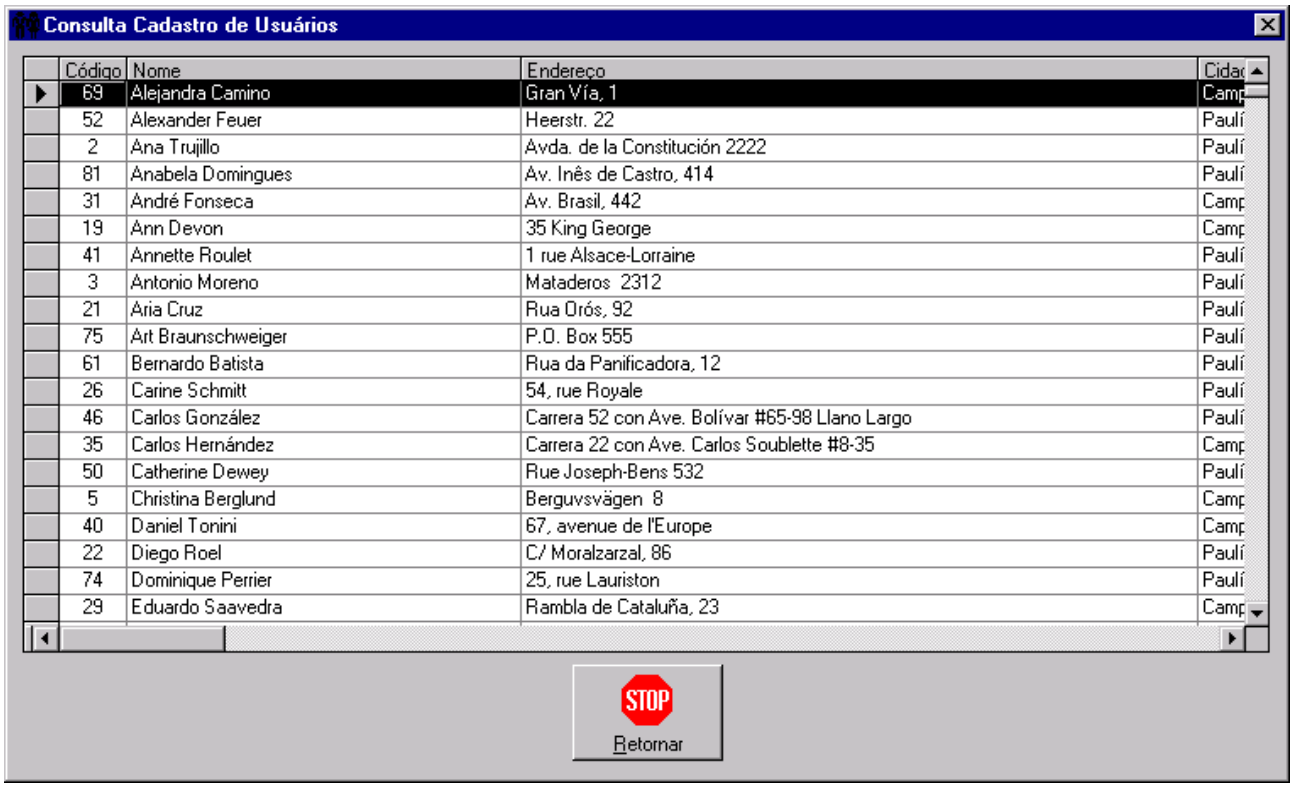
19. A aba Layout permite configurar opções de exibição para cada coluna. Na combo Column, escolha a coluna **0 – Código**;

20. Uma opção interessante é **Alignment**, que define o alinhamento do texto dentro da célula. As opções disponíveis são:
    - **0 – dbgLeft**: texto alinhado à esquerda;
    - **1 – dbgRight**: texto alinhado à direita;
    - **2 – dbgCenter**: texto centralizado no espaço da célula;
    - **3 – dbgGeneral**: usa o alinhamento definido como padrão nas opções do VB;

    Por default, todas as colunas são alinhadas à esquerda. Altere o alinhamento dessa coluna para **2 – dbgCenter**;
21. Faça o mesmo para as colunas 4 – UF, 5 – CEP e 6 – Telefone;
22. Outra opção muito usada é **Width**, que determina a largura da coluna. Retrieve Fields altera a largura de acordo com o valor retornado pela origem dos dados, mas com freqüência precisamos ajustá-la manualmente. Para isso, basta digitar em Width o novo valor desejado (em twips). Defina a largura das colunas como indicado:
    - 0 – Código: 600
    - 1 – Nome: 3500
    - 2 – Endereço: 6000
    - 3 – Cidade: 2500
    - 4 – UF: 350
    - 5 – CEP: 900
    - 6 – Telefone: 1500
23. Não vamos alterar nada nas abas Color e Font;
24. A aba Split define como será apresentado o foco nas linhas e colunas do grid. Uma opção interessante é **MarqueeStyle**, que altera a exibição do foco nas linhas. Altere para **4 – dbgHighlightRowRaiseCell**, para que toda a linha seja destacada;
25. Finalmente, a aba Format define a formatação dos dados das colunas. Retrieve Fields também altera essas definições conforme os dados de origem, mas é comum a necessidade de ajustes. Para alterar a formatação, devemos escolher a coluna na combo correspondente e depois indicar o formato desejado para essa coluna. Em nosso caso não será necessário nenhuma alteração.
26. Clique em Ok – a configuração do DataGrid está pronta;
27. Abra a janela Código no evento Form_Load. Além dos comandos de centralização do form, vamos alterar a propriedade ConnectionString do ADODC em tempo de execução. Isso é necessário devido a uma limitação do controle de dados: quando definimos a ConnectionString pelo assistente, indicamos o caminho e o nome do banco de dados a ser conectado, e essa informação é gravada na configuração do ADODC. Se o nome do arquivo MDB for alterado, ou trocarmos a pasta onde está gravado, o controle não será capaz de encontrá-lo, e será gerado um erro de execução. Isso é especialmente preocupante em um ambiente de rede, onde o caminho de um arquivo depende do mapeamento feito nas estações. Para evitar esse problema, indicamos ao ADODC que use para conectar-se ao banco a mesma string da conexão pricipal:

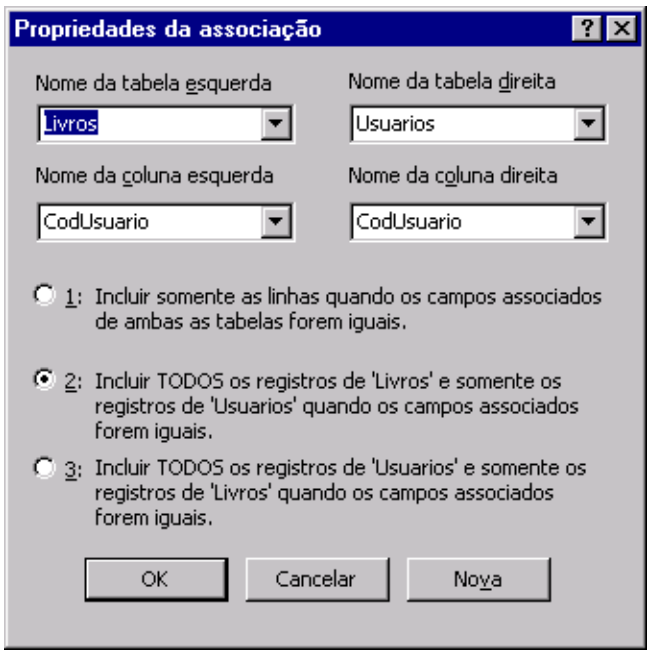
```
Private Sub Form_Load()
    Me.Left = (frmBiblio.ScaleWidth - Me.Width) / 2
    Me.Top = (frmBiblio.ScaleHeight - Me.Height) / 2
    datUsuarios.ConnectionString = cnnBiblio.ConnectionString
End Sub
```

28. Escreva a codificação necessária ao evento cmdRetornar_Click;
29. Grave o novo formulário com o nome **frmConUsuarios**;
30. Abra o formulário frmBiblio e programe a procedure de menu mnuConUsuarios_Click;
31. Teste a consulta. Você deve ter um resultado parecido com esse:

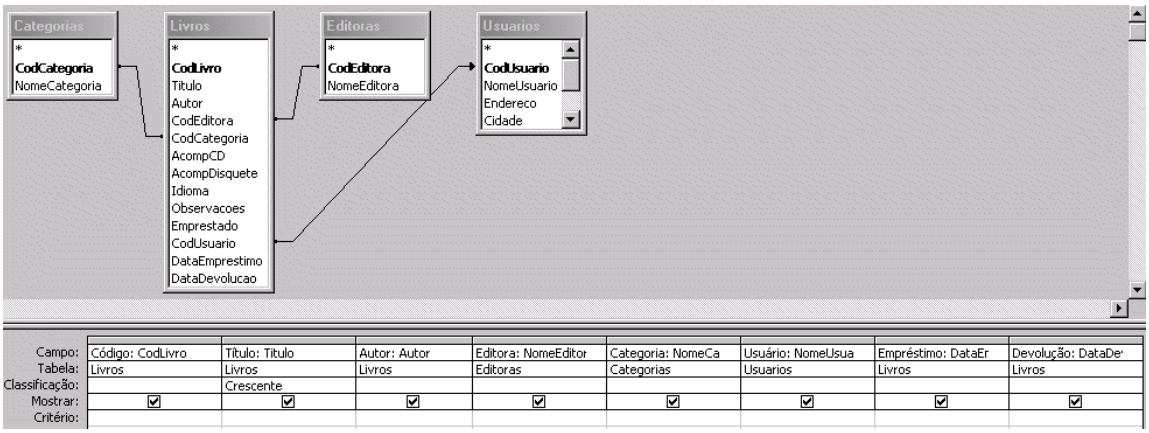


Bem, com o cadastro de usuários foi fácil. E para o cadastro de livros, em que existem códigos relacionados às outras tabelas do banco, qual o procedimento? Vejamos:

1. Entre novamente no Access e abra o banco de dados Biblio;
2. Inicie a criação de uma nova consulta em modo estrutura;
3. Adicione todas as tabelas do banco;
4. Relacione as tabelas:
   - Categorias e Livros pelo campo CodCategoria;
   - Editoras e Livros pelo campo CodEditora;
   - Usuarios e Livros pelo campo CodUsuario;
5. Clique com o botão direito do mouse sobre o relacionamento entre as tabelas Usuarios e Livros;
6. No menu de contexto, selecione a opção **Propriedades da Associação**;
7. Escolha a opção 2:



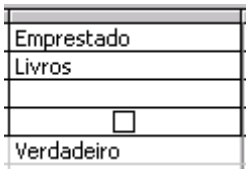
8. Adicione os campos CodLivro, Titulo, Autor, NomeEditora, NomeCategoria, NomeUsuario, DataEmprestimo e DataDevolucao ao grid;
9. Altere a classificação do campo Título para Crescente;
10. Altere os título das colunas:
    - **Código**, para o campo CodLivro;
    - **Título**, para o campo Titulo;
    - **Editora**, para o campo NomeEditora;
    - **Categoria**, para o campo NomeCategoria;
    - **Usuário**, para o campo NomeUsuario;
    - **Empréstimo**, para o campo DataEmprestimo;
    - **Devolução**, para o campo DataDevolucao.
11. Grave a consulta com o nome de **ConsultaTodosLivros**;
12. Sua consulta deve ter ficado parecida com a da figura:



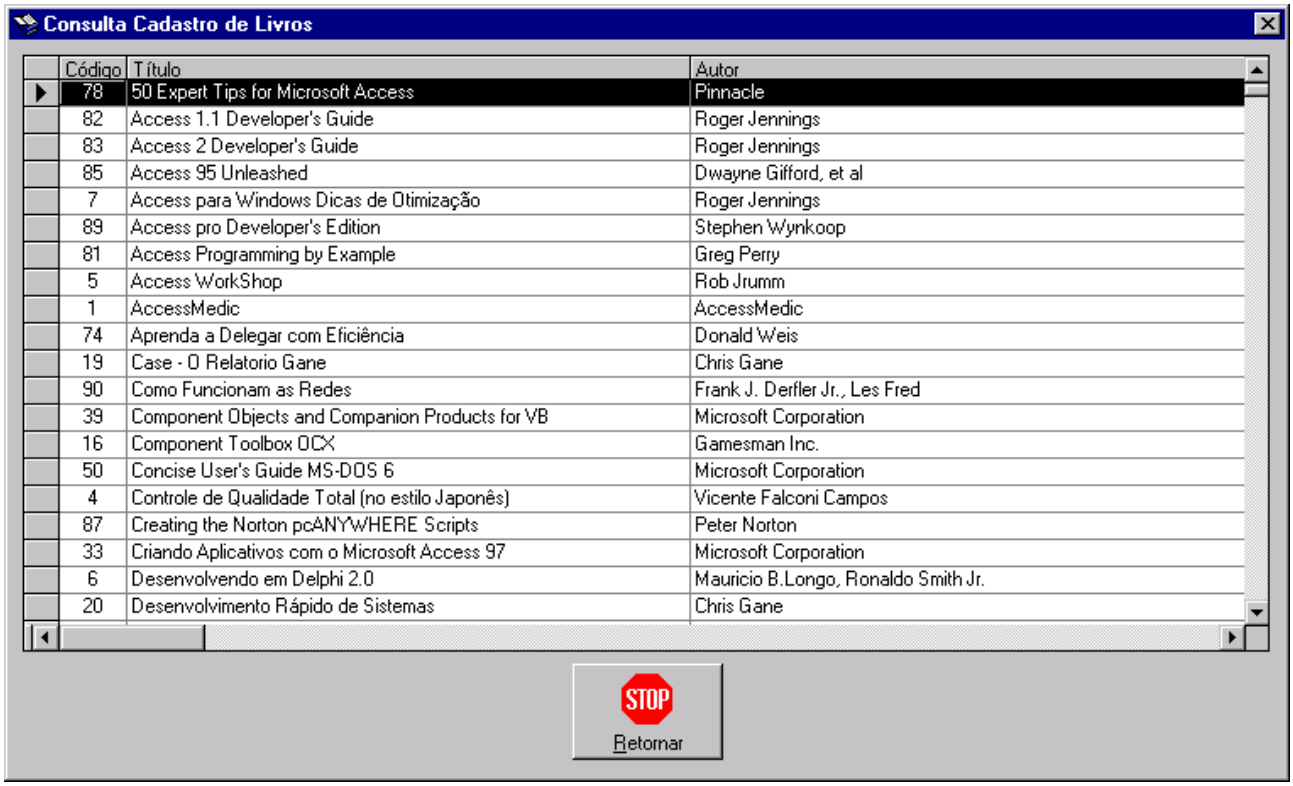
13. Execute a consulta e verifique se está funcionando corretamente;
14. Saia do Access e abra novamente o VB e o projeto Bibliotecario;
15. Adicione um novo formulário ao projeto;
16. Baseado na consulta de usuários, inclua os objetos no formulário e configure o ADODC e o DataGrid para executar a consulta **ConsultaTodosLivros**;
17. Escreva a procedure Form_Load conforme a do formulário frmConUsuarios. Não se esqueça do botão de saída do form;
18. Grave seu trabalho. Ao novo formulário dê o nome de **frmConLivros**;
19. No formulário principal programe a opção de menu mnuConTodosLivros para acionar o frmConLivros;
20. Execute o aplicativo e teste a consulta. Compare com o exemplo da próxima página.

Em nosso projeto temos duas opções no menu Consultas para as quais podemos paorveitar esse mesmo formulário: Livros Emprestados e Livros em Atraso. Para fazer isso, primeiramente vamos criar mais duas consultas no Access:

1. Abra o banco de dados Biblio;
2. Na aba Consultas, clique com o botão direito do mouse sobre a consulta ConsultaTodosLivros;
3. Cole com o nome de **ConsultaLivrosEmprestados**;
4. Abra a consulta ConsultaLivrosEmprestados em modo estrutura;
5. Adicione o campo Emprestado ao grid da consulta;
6. Desmarque a opção “Mostrar”;
7. Na linha “Critério” do campo Emprestado, escreva **Verdadeiro**. Isso fará com que apenas os livros emprestados sejam selecionados pela consulta. Veja na figura ao lado se fez tudo corretamente;



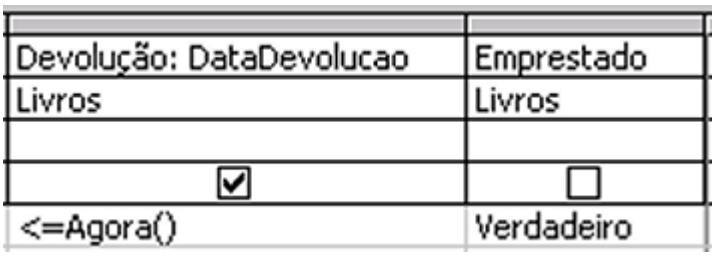
8. Grave a consulta e retorne à aba Consultas;



A consulta de Livros Cadastrados sendo executada

9. Selecione a ConsultaLivrosEmprestados, copie e cole com o nome de **ConsultaLivrosEmAtraso**;

10. Na linha "Critério" do campo DataDevolucao, escreva **<=Agora()**, para que sejam selecionados apenas os registros com data de devolução até hoje:



11. Verifique na figura se está tudo certo e grave a consulta:

12. Feche o Access e retorne ao projeto Biblioteario;

13. Para identificar qual consulta devemos abrir no formulário frmConLivros, criaremos uma variável pública no módulo de programação Biblio. Acrescente o seguinte comando na seção Declarations do módulo:

```
Public vConLivros As String
```

14. Altere a procedure mnuConTodosLivros_Click:

```
Private Sub mnuConTodosLivros_Click()
    'Atribui à variável vConLivros o nome da consulta correspondente à opção
    'de menu escolhida:
    vConLivros = "ConsultaTodosLivros"
    frmConLivros.Show
End Sub
```

15. Crie as procedures mnuConLivrosEmprestados e mnuConLivrosEmAtraso conforme segue:

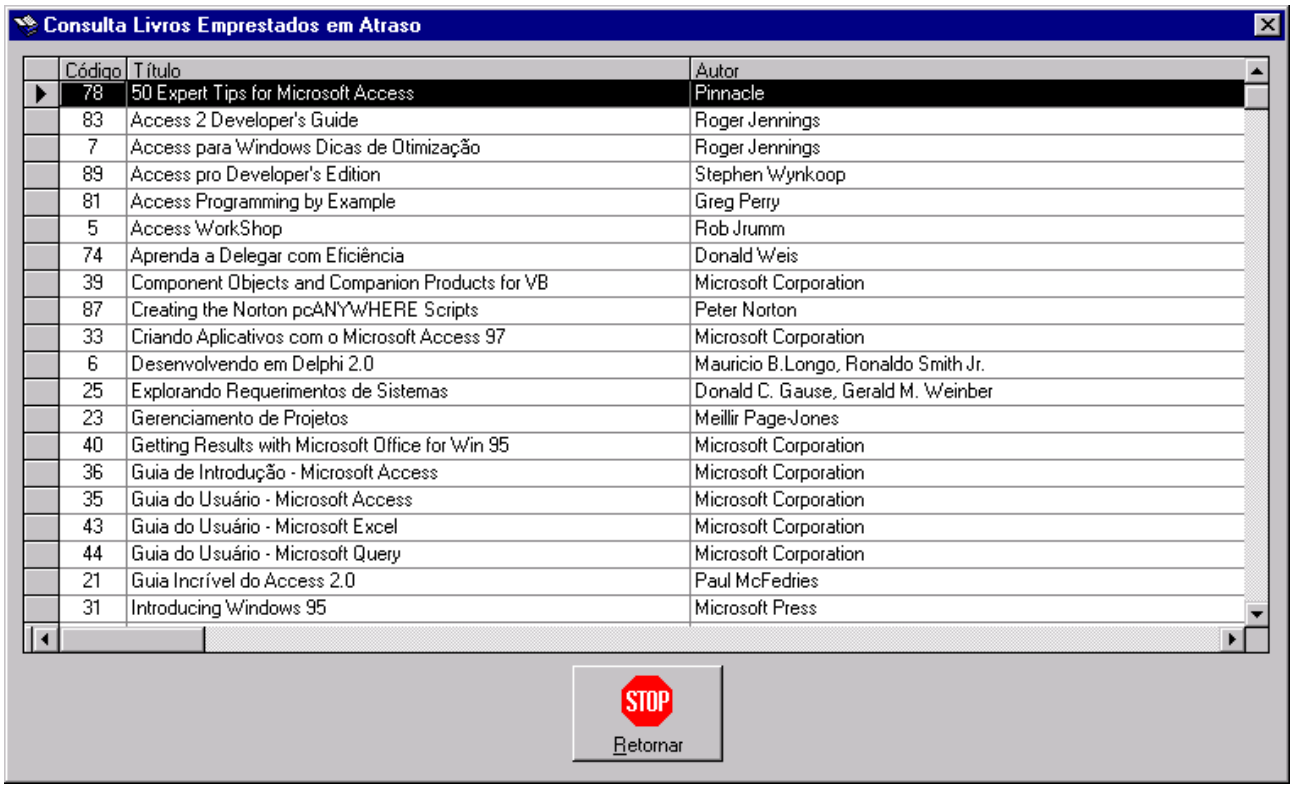
```
Private Sub mnuConLivrosEmAtraso_Click()
    vConLivros = "ConsultaLivrosEmAtraso"
    'Altera o título do formulário de consulta de livros:
    frmConLivros.Caption = "Consulta Livros Emprestados em Atraso"
    frmConLivros.Show
End Sub
```

```
Private Sub mnuConLivrosEmprestados_Click()
    vConLivros = "ConsultaLivrosEmprestados"
    frmConLivros.Caption = "Consulta Livros Emprestados"
    frmConLivros.Show
End Sub
```

16. Altere as seguintes linhas do evento Form_Load do formulário frmConLivros para que fique assim:

```
Private Sub Form_Load()
    Me.Left = (frmBiblio.ScaleWidth - Me.Width) / 2
    Me.Top = (frmBiblio.ScaleHeight - Me.Height) / 2
    datLivros.ConnectionString = cnnBiblio.ConnectionString
    'Altera o RecordSource do ADODC para usar como origem o contéudo da
    'variável vConLivros:
    datLivros.RecordSource = vConLivros
    'Atualiza o recordset de datLivros de acordo com o novo RecordSource:
    datLivros.Refresh
End Sub
```

17. Execute uma dessas consultas que acabamos de criar e veja se está funcionando corretamente. O exemplo a seguir é a consulta de livros em atraso:



18. Grave seu trabalho.

Os Controles DataList e DataCombo

Esses controles ADO são muito interessantes: DataList é uma caixa de listagem vinculada a um controle de dados. Seus itens serão preenchidos com os dados de um dos campos do recordset do controle de dados. DataCombo é uma caixa de combinação que usa o mesmo princípio. Ambos são controles customizados, e para adicioná-los à caixa de ferramentas marque a opção *Microsoft DataList Controls* na janela *Components*.

DataList e DataCombo têm as mesmas propriedades das caixas de listagem e combinação comuns e mais algumas específicas para trabalhar com o controle de dados:

Propriedades RowSource e ListField

Essas propriedades indicam, respectivamente, o nome do ADODC que deve ser vinculado à DataList ou DataCombo e o nome do campo que será usado como origem para os itens da caixa.

Propriedades DataSource e DataField

No caso específico de DataList e DataCombo, essas propriedades servem para atualizar o registro do recordset de um outro controle de dados, (que não é o mesmo vinculado às propriedades RowSource e ListField), de modo a sincronizar ambos os controles Data: assim, conforme um item é selecionado na caixa, os dois controles Data estarão apontando para o registro correspondente em seus respectivos recordsets.

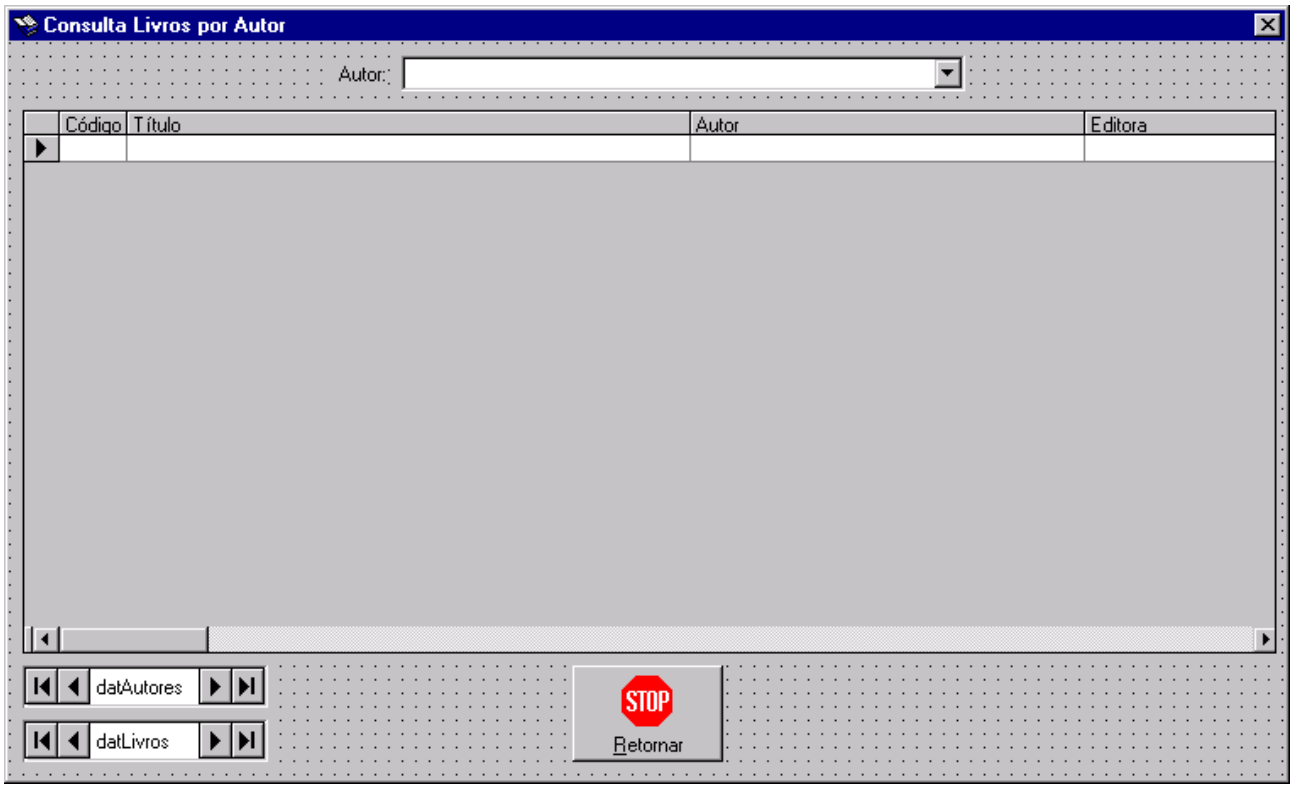
Propriedade BoundColumn

Define qual campo do recordset o controle retornará quando um item for selecionado. Por exemplo: se um recordset tem dois campos, digamos Código e Nome, com certeza você vai preferir que seja exibido na caixa o campo Nome, mas pode ser que vá utilizar nas operações de banco de dados o campo Código. Você pode então indicar para a propriedade ListField o campo Nome, e para BoundColumn o campo Código. Dessa maneira, quando o usuário escolher um item na caixa, o campo retornado por BoundColumn para aquele registro será Código, apesar do item estar exibindo o nome.

Propriedade BoundText

Contém o texto do item que está selecionado na caixa.

Mas vamos pôr esse conceitos em prática, elaborando as últimas consultas ao cadastro de livros. Vamos começar pela consulta de Livros por Autor. Veja o modelo do novo formulário:



As propriedades dos objetos são as seguintes:

Objeto	*Nome*	*Outras Propriedades = Valor*
Formulário	frmConLivrosPorAutor	Caption = Consulta Livros por Autor BorderStyle = 1 – Fixed Single Icon = Writing\Books02.ICO MDIChild = True
ADODC	datLivros	Caption = datLivros Visible = False ConnectionString = crie com o assistente CommandType = 1 – adCmdText CommandText = SELECT * FROM ConsultaTodosLivros;

Objeto	***Nome***	***Outras Propriedades = Valor***
ADODC	datAutores	Caption = datAutores Visible = False ConnectionString = crie com o assistente CommandType = 1 – adCmdText CommandText = SELECT DISTINCT Autor FROM Livros ORDER BY Autor;
DataCombo	cboAutores	RowSource = datAutores ListField = Autor BoundColumn = Autor Style = 2 – dbcDropdownList
DataGrid	grdLivros	DataSource = datLivros
Botão de comando	cmdRetornar	Caption = &Retornar Style = 1 – Graphical Picture = Traffic\Trffc14.ICO

Duas observações importantes:

- Na propriedade CommandType do datLivros definimos **1 – adCmdText** para poder usar um comando SQL que será criado em tempo de execução, de modo a selecionar apenas os livros do autor escolhido;
- Na propriedade CommandText do datAutores usamos a cláusula DISTINCT no comando SELECT, para eliminar linhas repetidas no resultado do comando SQL. DISTINCT faz com que, quando forem selecionados pelo comando SELECT dois ou mais registros exatamente iguais, apenas um será retornado, eliminando repetição. Fizemos isso para que não existam autores duplicados na combo.

O funcionamento dessa consulta será assim: na DataCombo serão exibidas os nomes dos autores, e conforme o operador for alternando entre eles, os livros do autor selecionado serão exibidos no grid. Será necessário, portanto, filtrar os registros da tabela de livros de acordo com o autor escolhido na combo. Vamos começar:

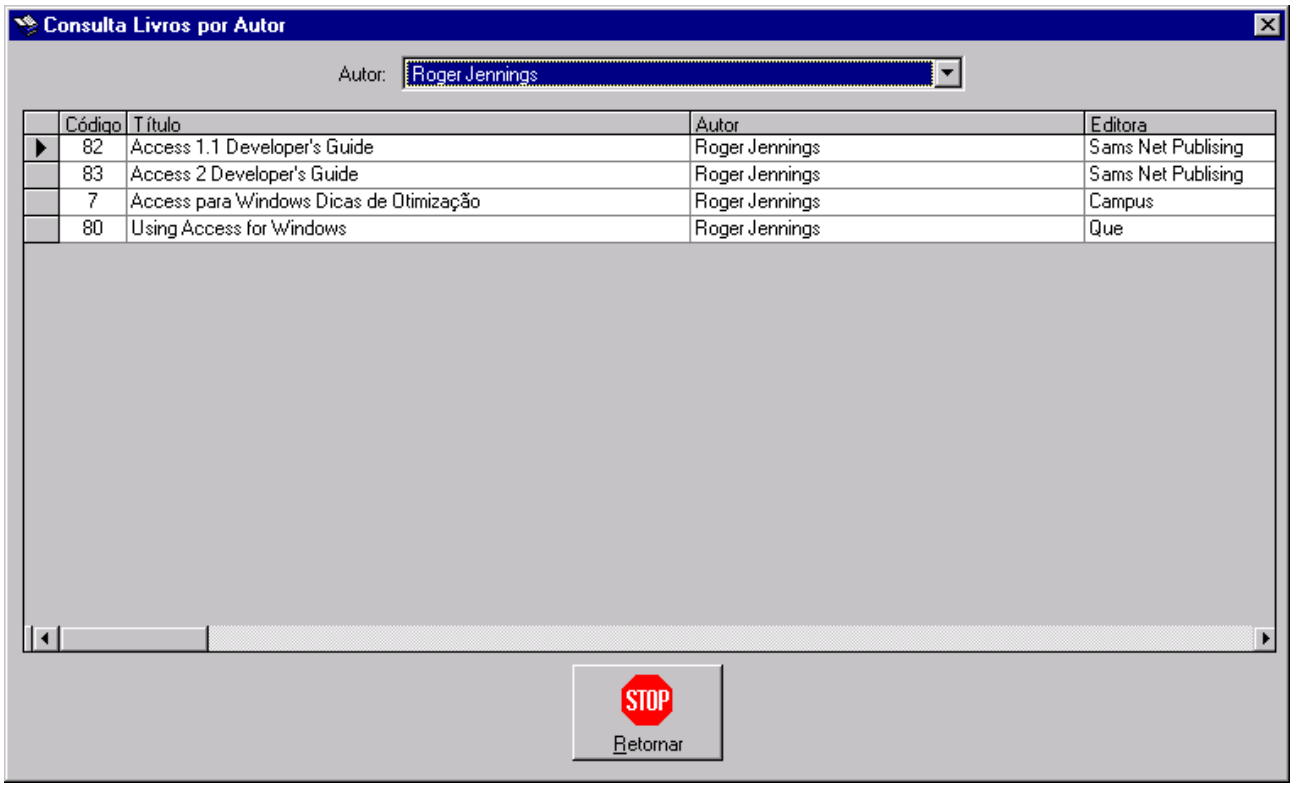
1. Use Retrieve Fields para formatar o grid. Lembre-se de alterar o tamanho das colunas e o alinhamento dos campos da mesma maneira que fizemos nas consultas anteriores;
2. Escreva a procedure para o evento Click do botão Retornar;
3. No evento Load do formulário, escreva os comandos para atribuir a ConnectionString aos controles de dados:

```
Private Sub Form_Load()
    Me.Left = (frmBiblio.ScaleWidth - Me.Width) / 2
    Me.Top = (frmBiblio.ScaleHeight - Me.Height) / 2
    datAutores.ConnectionString = cnnBiblio.ConnectionString
    datLivros.ConnectionString = cnnBiblio.ConnectionString
End Sub
```

4. O próximo passo é fazer com que o grid seja atualizado com os livros do autor selecionado. Para isso usaremos o eventos Click da combo:

```
Private Sub cboAutores_Click(Area As Integer)
    datLivros.RecordSource = "SELECT * FROM ConsultaTodosLivros " & _
        "WHERE Autor = '" & cboAutores.BoundText & "';"
    datLivros.Refresh
End Sub
```

5. Grave esse formulário com o nome de **frmConLivrosPorAutor**;
6. Abra o formulário frmBiblio e escreva a procedure para o evento Click do menu mnuLivrosPorAutor;
7. Teste a consulta. Veja que, conforme você escolhe um autor, o grid é automaticamente atualizado com seus respectivos livros. Caso não exista nenhum livro cadastrado para o autor, o grid não apresentará nenhum registro. Compare com a figura a seguir:



8. Seguindo o mesmo procedimento, crie as consultas de livros por Editora e por Categoria. Grave-os respectivamente com os nomes de **frmConLivrosPorEditora** e **frmConLivrosPorCategoria**;
9. Teste as consultas que você criou e veja se estão funcionando corretamente;
10. Grave seu trabalho.

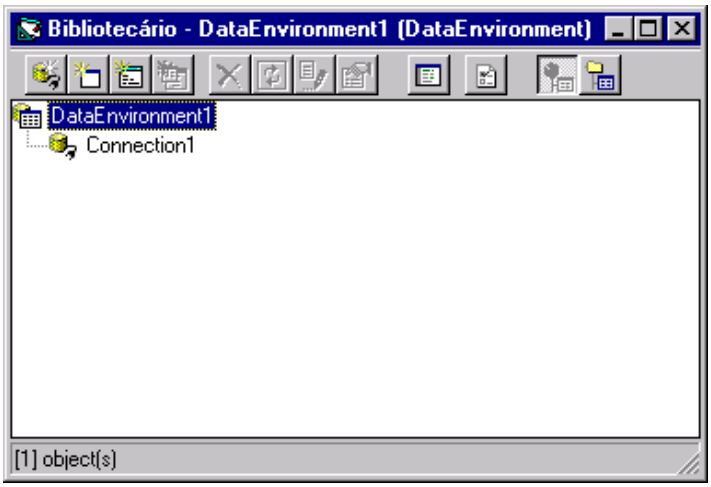
Lição 13: Relatórios

Quem já criou relatórios no Access sabe como uma ferramenta gráfica facilita esse tipo de trabalho. E vai se sentir em casa no VB: a partir da versão 6 do Visual Basic, a Microsoft introduziu duas novas ferramentas que juntas permitem a criação de relatórios de maneira visual, muito semelhante ao construtor de relatórios do Access. Nessa última lição, aprenderemos como usar essas ferramentas construindo um dos relatórios do nosso sistema de controle de bibliotecas.

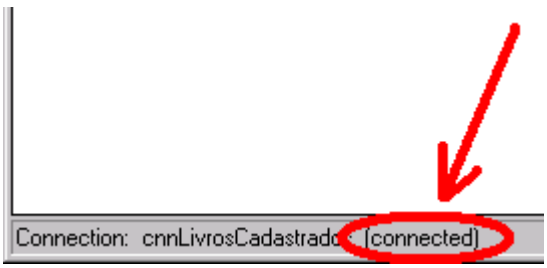
Usando o Data Environment

A primeira ferramenta que vamos estudar é o *Data Environment Designer*, cuja função é estabelecer conexões e comandos de acesso ADO a bancos de dados (*Data Environment* quer dizer "ambiente de dados") e que podem ser usados posteriormente por formulários e relatórios do projeto. Vejamos passo a passo como criar e configurar o Data Environment:

1. Se ainda não o fez, abra o VB e o projeto Bibliotecário;
2. No menu *Project*, escolha a opção *More ActiveX Designers;*
3. Escolha agora a opção *Data Environment.* O VB adiciona um Data Environment ao projeto, de nome DataEnvironment1. Você pode criar quantos forem necessários, cada um ligado a uma fonte de dados:

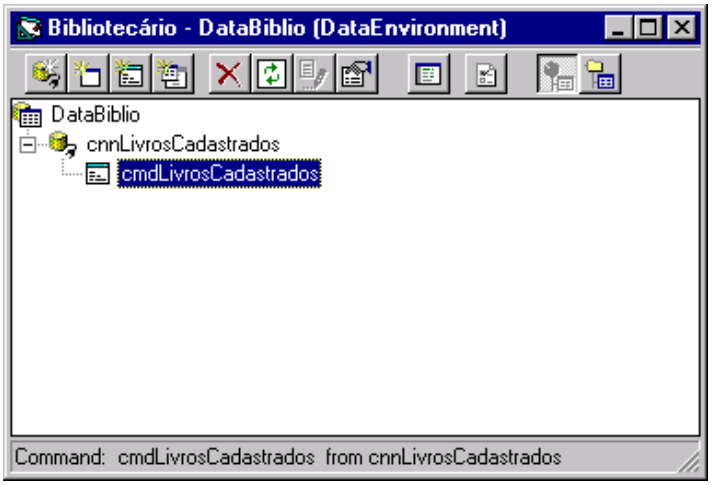


4. Vamos alterar o nome do Data Environment: na janela de propriedades, altere a propriedade Name para **DataBiblio**;
5. Vinculado ao Data Environment existe sempre ao menos um objeto Connection – veja na figura: o que acabamos de adicionar chama-se Connection1. Vamos renomeá-lo: clique com o botão direito do mouse sobre Connection1 e escolha a opção Rename;
6. Altere o nome para **cnnLivrosCadastrados**;
7. O próximo passo é criar a conexão com o banco de dados: clique novamente com o botão direito do mouse e escolha a opção **Properties**;
8. Crie a conexão da mesma forma que fizemos nas lições anteriores;
9. Clique mais uma vez com o botão direito do mouse e escolha a opção Refresh: se você fez tudo certo, a linha de status está indicando que o objeto Connection está conectado ao banco:

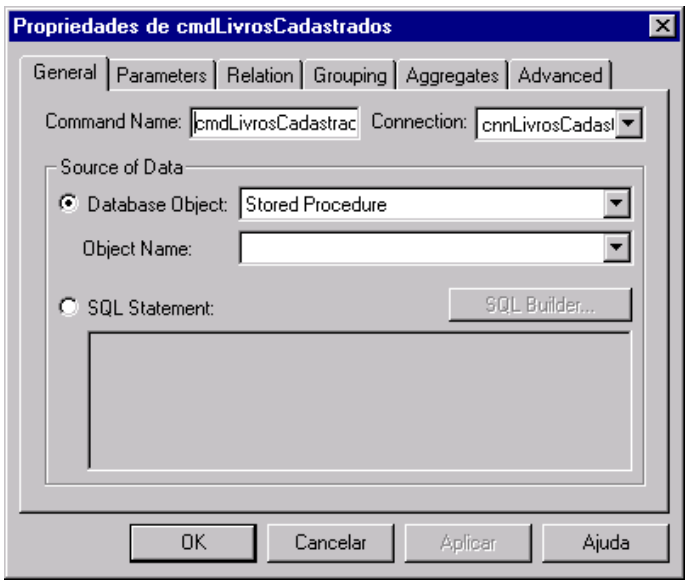


10. Devemos agora adicionar um objeto Command. Clique com o botão direito e escolha Add Command;

11. Altere o nome do comando para cmdLivrosCadastrados:



12. Clique com o botão direito sobre o comando e escolha a opção **Properties**:



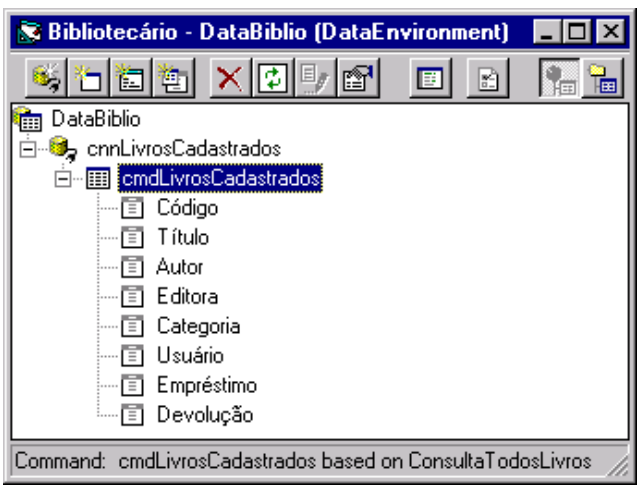
13. Na aba General definimos a origem dos dados: em *Database Object* informamos qual o tipo do objeto que será acessado pelo comando: *Table* (tabela), *View* (consulta) ou *Stored Procedure*, e também seu nome. A opção *SQL Statement* serve para usar um comando SQL;

14. Escolha **Database Object**, e o tipo **View**;

15. Em *Object Name*, escolha **ConsultaTodosLivros**;

16. Como não precisamos alterar nada nas outras abas, clique em Ok;

17. Clique no sinal "+" que aparece ao lado do nome do comando para visualizar as colunas da consulta:



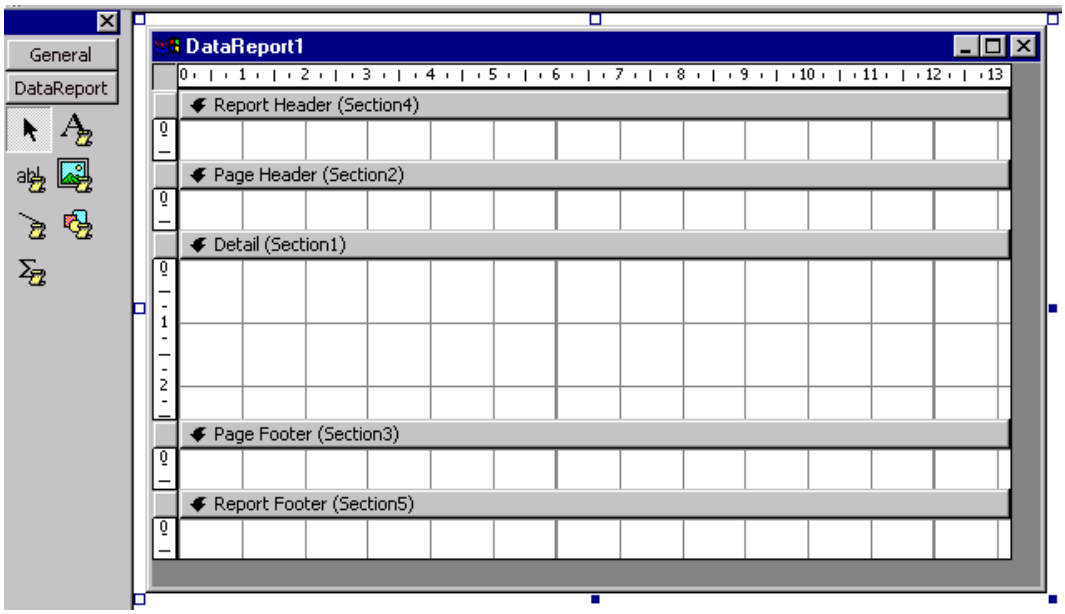
18. Grave seu trabalho. Ao Data Environment dê o nome de **DataBiblio**. Note que a extensão do arquivo é DSR (de *Designers*).

Bem, a primeira parte do serviço está pronta. Vejamos agora como criar o relatório propriamente dito:

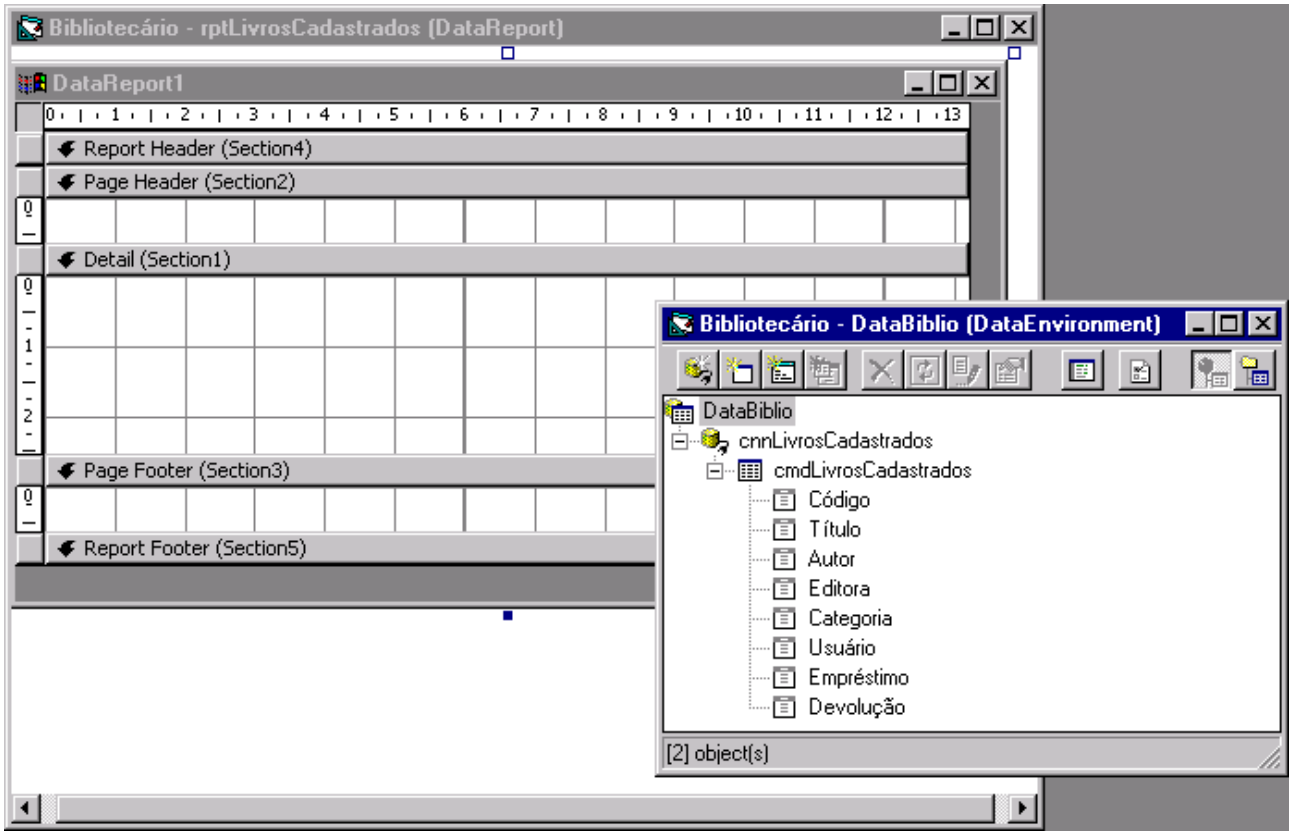
Usando o Data Report

A ferramenta para design de relatórios do VB é o *Data Report*. Sua operação é muito simples, baseada no recurso arrastar-e-soltar do Windows. Vejamos como usá-lo:

1. No menu *Project* escolha a opção *Add Data Report*;
2. O objeto DataReport1 é adicionado ao projeto, e a caixa de ferramentas passa a exibir os controles suportados pelo relatório:

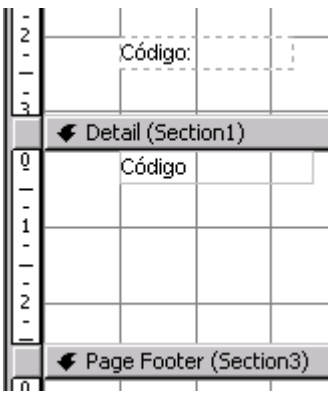


3. Note que o formulário está dividido em 5 partes (ou seções), chamadas de *Report Sections*:
   - ***Report Header***: cabeçalho a ser impresso no início do relatório;
   - ***Page Header***: cabeçalho a ser impresso no início de cada página;
   - ***Detail***: corpo do relatório, ou linha de detalhe: área onde serão inseridos os dados do relatório;
   - ***Page Footer***: rodapé a ser impresso no final de cada página;
   - ***Report Footer***: rodapé a ser impresso no final do relatório.
4. Cada seção tem propriedades e métodos que permitem formatar o relatório conforme necessário, além do próprio objeto DataReport – basta clicar na área que queremos formatar para ver suas propriedades na janela *Properties*. Clique no título da janela do DataReport para ver suas propriedades;
5. Altere a propriedade Name para **rptLivrosCadastrados**;
6. A propriedade DataSource indica o Data Environment que deve ser usado como origem dos dados. Escolha o **DataBiblio** (deve ser único disponível);
7. Em DataMember informamos o comando a ser executado pelo relatório: escolha o **cmdLivrosCadastrados**;
8. Agora clique com o botão direito do mouse sobre o DataReport e escolha a opção *Retrieve Structure*: note que o nome da seção *Detail* mudou para **cmdLivrosCadastrados_Detail;**
9. Feche as seções Report Header e Report Footer, pois não vamos usá-las. Para fazer isso arraste a barra que contém o nome da seção para cima (diminuindo a régua) até fechá-la.
10. Já podemos incluir campos em nosso relatório. Abra as janelas do Data Environment e do Data Report lado a lado, de forma a poder visualizar ambas ao mesmo tempo (mais ou menos como na figura a seguir):



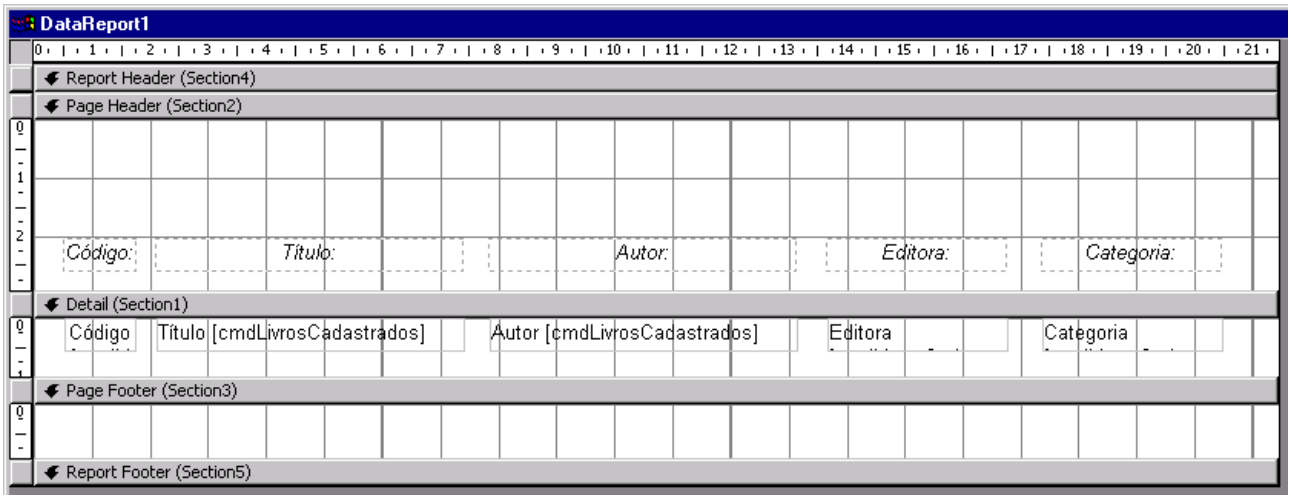
11. Arraste o campo Código da janela do Data Environment para dentro da seção Detail: note que, além do campo, o título da coluna também é adicionado ao relatório:

12. Aumente a altura da seção Page Header para uns 3 centrímetros (para isso arraste a barra para baixo);
13. Arraste o label que contém o título do campo (na figura é o da esquerda) para a seção Page Header:
14. Repita o processo para os campos Título, Autor, Editora e Categoria;
15. Acerte a largura dos campos. Para isso clique sobre o campo e altere a propriedade Width ou então arraste a borda do campo até a largura desejada;
16. Altere a largura do título de cada coluna para a mesma do campo correspondente;
17. Reposicione os campos e os títulos. Se necessário aumente a largura do relatório: para isso arraste a borda direita do relatório até a largura desejada, mas atenção à régua para não ultrapassar as medidas do papel (usaremos o tamanho Carta em modo Paisagem):
    - Tamanho Carta: 21,5 x 28 cm;
    - Tamanho A4: 21 x 29,7 cm;
    - Tamanho Ofício: 21,5 x 35,5 cm.

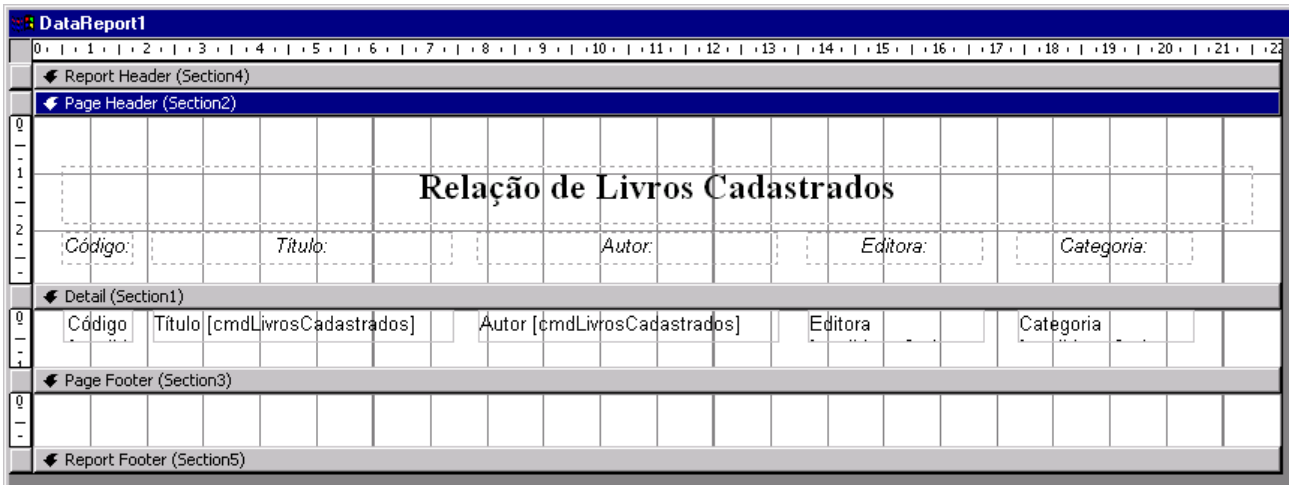


    Obs.: você pode melhorar o posicionamento dos campos clicando com o botão direito sobre o relatório e desmarcando a opção **Snap to Grid**;
18. Clique sobre o campo Codigo e altere a propriedade Aligment para **2 – rptJustifyCenter**, para centralizar o dado no campo;
19. Centralize também os títulos dos campos da mesma maneira;

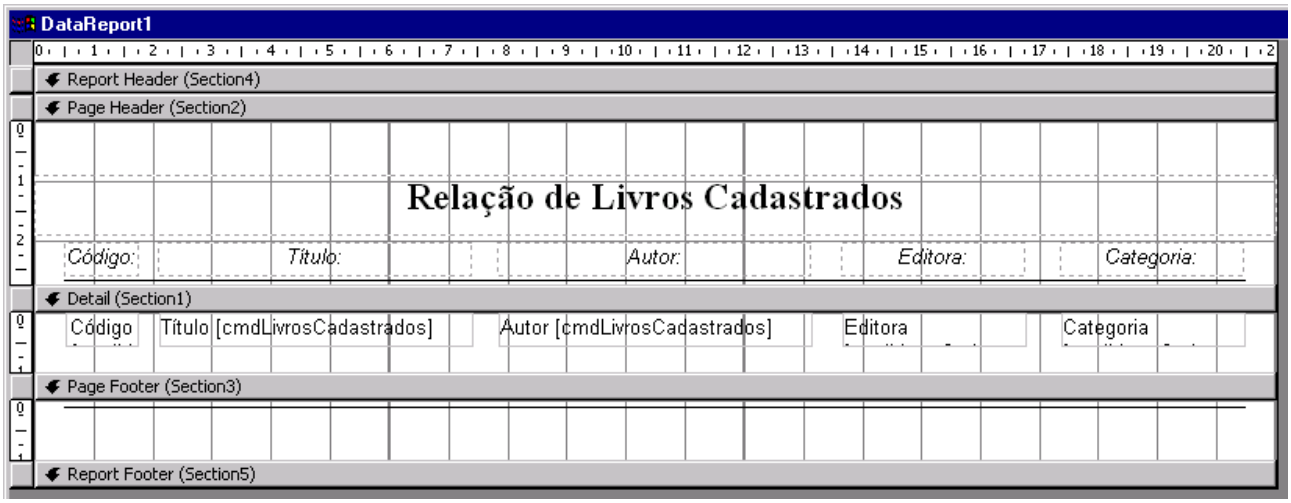
20. Altere a fonte (propriedade Font) dos títulos das colunas para Arial, itálico, tamanho 10;
21. Altere a fonte dos campos para Arial tamanho 10;
22. Seu relatório deve estar parecido com o do exemplo:



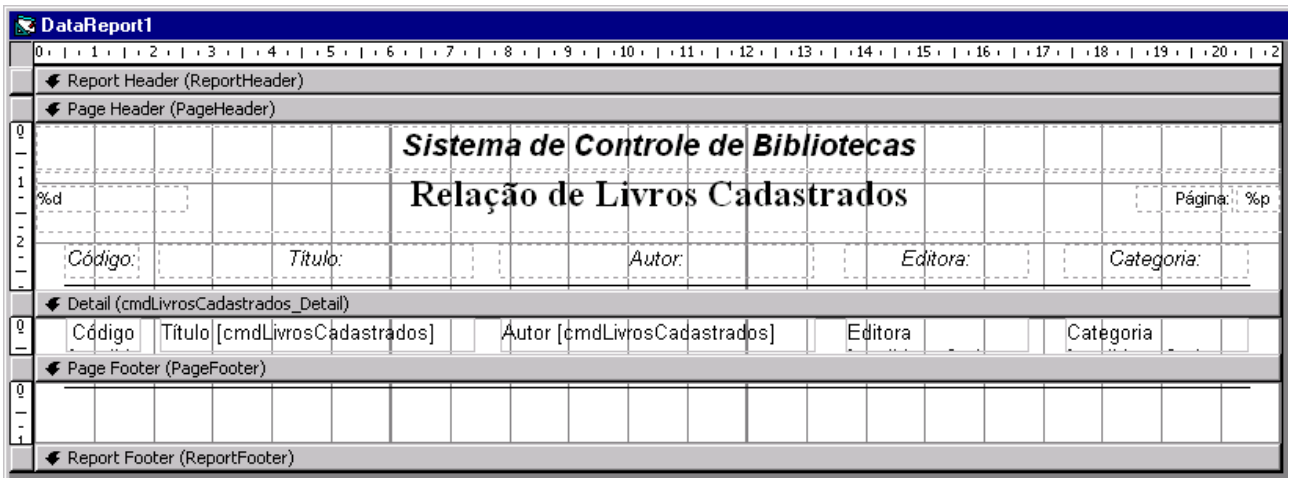
23. Procure na caixa de ferramentas o ícone rptLabel e insira um na seção Page Header – ele vai conter o título do nosso relatório;
24. Altere a propriedade caption do label para **Relação de Livros Cadastrados**;
25. Faça-o ocupar toda a largura do relatório;
26. Altere a fonte do label para Times New Roman, negrito, tamanho 16;
27. Confira seu relatório com o exemplo abaixo:



28. Procure agora o ícone do controle rptLine;
29. Desenhe uma linha na seção Page Header, abaixo dos títulos das colunas;
30. Desenhe outra linha, de mesmo comprimento, na seção Page Footer (aumente um pouco essa seção, se achar necessário);
31. Se desejar, você pode mudar o estilo da linha alterando a propriedade BorderStyle. Veja como ficou:
32. Existem alguns controles especiais, que são inseridos através do botão direito do mouse, como por exemplo a data do sistema. Vamos inserí-la: clique com o botão direito e escolha **Insert Control**;



33. Escolha agora a opção **Current Date (Short Format)**, que insere a data no formato curto do Windows;
34. Posicione o campo da data à esquerda, acima do título do campo código e alinhada com ele;
35. Outro controle interessante é o número da página. Acione novamente o menu Insert Control e escolha agora a opção *Current Page Number*;
36. Alinhe o número do página à direita do relatório, na mesma altura da data;
37. Diminua seu tamanho para um espaço de aproximadamente 3 caracteres;
38. Altere o alinhamento para centralizado;
39. Insira mais um label do lado esquerdo do número da página;
40. Altere a propriedade caption desse label para "**Página:**" (sem as aspas);
41. Altere o alinhamento do label para **1 – rptJustifyRight**;
42. Diminua a altura da seção Detail para a mesma dos campos nela contidos;
43. Se quiser incluir mais labels e formatações, esteja a vontade...
44. O layout do relatório está pronto. Veja como ficou o meu:



45. Grave o relatório com o nome de **rptLivrosCadastrados** – note que a extensão do arquivo é a mesma do Data Environment, ou seja, DSR;
46. Precisamos agora fazer nosso relatório funcionar. Abra o formulário frmBiblio;
47. Escreva a programação da opção de menu Relatórios – Livros para informar o orientação do papel e acionar o relatório:

```
Private Sub mnuRelLivros_Click()
    With rptLivrosCadastrados
        'A propriedade Orientation informa a orientação do papel para impressão
        'do relatório, e o valor rptOrientLandscape define modo Paisagem:
        .Orientation = rptOrientLandscape
```

```
        'Para exibir o resultado do Data Report usamos o método Show:
        .Show
    End With
End Sub
```

Obs.: para impressão em modo Retrato normalmente não é necessário alterar a propriedade Orientation, mas se precisar use a constante rptOrientPortrait.

48. Para evitar problemas de conexão, precisamos alterar a ConnectionString do DataEnvironment. Abra o formulário frmSplash;

49. Na procedure Timer1_Timer, localize o comando:

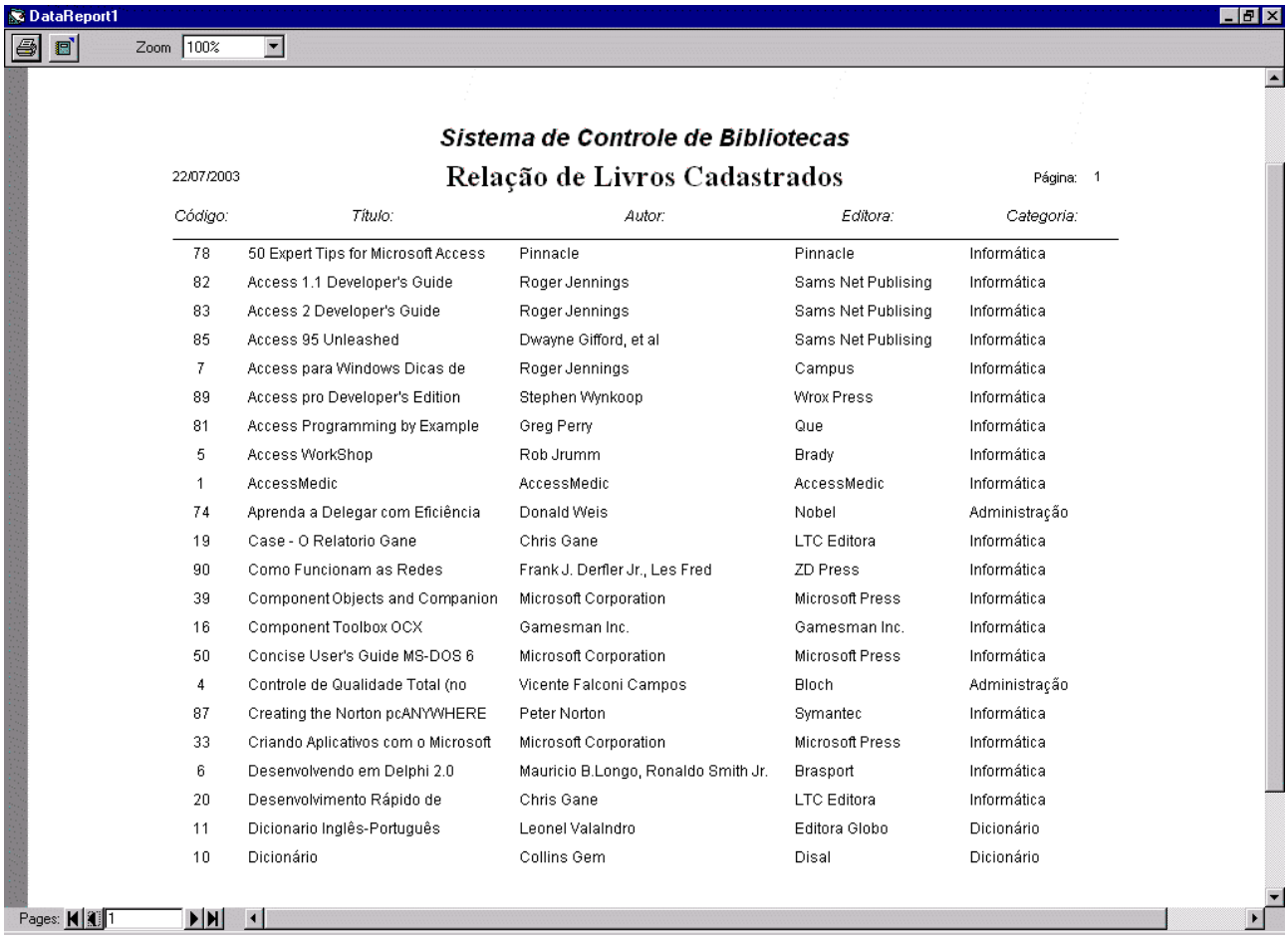
```
cnnBiblio.Open
```

50. Escreva abaixo dela a linha a seguir:

```
DataBiblio.cnnLivrosCadastrados.ConnectionString = cnnBiblio.ConnectionString
```

51. Grave seu trabalho;

52. Execute o programa e o relatório de Livros. Se estiver tudo Ok, o resultado será parecido com esse:



Código:	Título:	Autor:	Editora:	Categoria:
78	50 Expert Tips for Microsoft Access	Pinnacle	Pinnacle	Informática
82	Access 1.1 Developer's Guide	Roger Jennings	Sams Net Publising	Informática
83	Access 2 Developer's Guide	Roger Jennings	Sams Net Publising	Informática
85	Access 95 Unleashed	Dwayne Gifford, et al	Sams Net Publising	Informática
7	Access para Windows Dicas de	Roger Jennings	Campus	Informática
89	Access pro Developer's Edition	Stephen Wynkoop	Wrox Press	Informática
81	Access Programming by Example	Greg Perry	Que	Informática
5	Access WorkShop	Rob Jrumm	Brady	Informática
1	AccessMedic	AccessMedic	AccessMedic	Informática
74	Aprenda a Delegar com Eficiência	Donald Weis	Nobel	Administração
19	Case - O Relatorio Gane	Chris Gane	LTC Editora	Informática
90	Como Funcionam as Redes	Frank J. Derfler Jr., Les Fred	ZD Press	Informática
39	Component Objects and Companion	Microsoft Corporation	Microsoft Press	Informática
16	Component Toolbox OCX	Gamesman Inc.	Gamesman Inc.	Informática
50	Concise User's Guide MS-DOS 6	Microsoft Corporation	Microsoft Press	Informática
4	Controle de Qualidade Total (no	Vicente Falconi Campos	Bloch	Administração
87	Creating the Norton pcANYWHERE	Peter Norton	Symantec	Informática
33	Criando Aplicativos com o Microsoft	Microsoft Corporation	Microsoft Press	Informática
6	Desenvolvendo em Delphi 2.0	Mauricio B.Longo, Ronaldo Smith Jr.	Brasport	Informática
20	Desenvolvimento Rápido de	Chris Gane	LTC Editora	Informática
11	Dicionario Inglês-Português	Leonel Valaindro	Editora Globo	Dicionário
10	Dicionário	Collins Gem	Disal	Dicionário

Pronto! Agora só falta você criar os outros relatórios do sistema (Usuários, Categorias e Editoras) e ele estará terminado. Com isso chegamos ao final do nosso curso. Espero sinceramente que tenha gostado, e que aproveite bem tudo que aprendeu. Parabéns e boa sorte!!!

Luís Carlos Ballaminut