Migrando Clipper para Visual FoxPro

Trabalhando com tabelas do Clipper em Visual Foxpro.

Durante a migração de um aplicativo Clipper para Visual FoxPro pode tornar-se necessário utilizar o programa antigo em conjunto com o novo em VFP. Nesse caso, é necessário que a base de dados seja compartilhada. Um fator preocupante é a indexação dos arquivos. Os arquivos DBF podem não oferecer compatibilidade entre Clipper e Visual FoxPro, nessa caso seguem abaixo exemplos práticos que permitem trabalhar desta maneira.

1) Coloque no seu programa principal em Clipper os seguintes comandos:

```
REQUEST DBFCDX
RDDSETDEFAULT("DBFCDX")
```

Isto fará com que os índices criados passem a ser do tipo .IDX após a compilação.

Para compilar com a biblioteca DBFCDX.LIB faça um arquivo CL.BAT com os seguintes comandos:

```
clipper %1
if not error level 1 rtlink file %1 LIB CLIPPER, EXTEND, TERMINAL, DBFCDX
```

Após ter usado a rotina de re-criação de índices, observe que eles agora são do tipo IDX.

2) Suas tabelas no sistema antigo, agora podem ser compartilhadas entre o Clipper e Visual FoxPro. Para tanto, no Visual FoxPro, estas tabelas devem ser adicionadas como Tabelas Livres. Nunca reindexe os arquivos a partir do Visual FoxPro.

3) Assim que o seu banco de dados estiver projetado no novo sistema e estar certo que não haverá mais necessidade de usar o antigo programa, arraste cada tabela livre no Visual FoxPro para o seu banco de dados e a partir de então, aplique os recursos de integridade referencial e triggers. Ao abrir as tabelas pela primeira vez, o Visual FoxPro irá solicitar a página de código a ser aplicada, escolha a que melhor se adequar, a mais comum é 1252 – Windows ANSI.

As desvantagens desse processo de compartilhamento estão em não poder fazer uso dos seguintes recursos:

1. Relacionamentos, integridade e procedimentos armazenados. As tabelas são adicionadas ao VFP como livres, desta forma não dispõem deste recurso, mesmo porque o Clipper não os possui. Com isso, os seus formulários deverão ter códigos de validação, como no Clipper, isto pode prejudicar o tempo de aprendizagem do OOP.

2. Trava de registros e tabelas entre Clipper e Visual FoxPro. Os métodos usados pelo Visual FoxPro são diferentes dos adotados pelo Clipper. Os dados podem ser compartilhados.

Migrar os arquivos de dados será o ponto de partida para que seja possível transportar todo os seu sistema para o ambiente visual com orientação a objetos.

Orientação a Objetos

É importante destacar que o Visual FoxPro é uma ferramenta que atua com programação orientada a objetos. No entanto, apenas trabalhar com o Visual FoxPro não quer dizer que a programação está sendo orientada realmente a objetos.

Como o VFP assume compatibilidade com o padrão Xbase, muitos programadores o utilizam na realidade como uma ferramenta poderosa na criação de formulários, mantendo por trás o código escrito conforme vinha sendo feito no Clipper.

Entendendo a OOP

OOP - Object-Oriented Programming Language. (Programação Orientada a Objeto)

É uma metodologia de programação que permite uma hierarquia abstrata e do tipo modular por capacitar herança e encapsulamento.

O que é a Programação Orientada a Objetos ?

A programação orientada a objetos, é um estilo de programação que se baseia na construção de uma interface composta por elementos individuais inter-relacionados. Na programação tradicional (normalmente designada por programação procedimental ou seqüencial), o código do programa é executado numa seqüência pré-estabelecida. As aplicações orientadas por objetos (ou seja, as aplicações desenvolvidas usando programação orientada por objetos) são tipicamente mais User-friendly, ou seja, mais amigável para o usuário, por tentarem reproduzir o mundo real através do uso de objetos. São os menus típicos das interfaces gráficas, que nos permitem ver de imediato todas as opções neles existentes, sem nos obrigarem a escolher uma primeira opção de nível 1, seguida de outra de nível 2, e assim sucessivamente, até "achar" a opção que nos interessa (tal como é feito em aplicações antigas, como as primeiras versões do lotus 123).

A OOP admite três principios básicos:
O Encapsulamento, a herança e o polimorfismo. Vamos examinar rapidamente estes termos.

O **encapsulamento** determina que as informações sobre um objeto (suas propriedades) e os processos que são executados pelo objeto (seus métodos) estão todos contidos na definição do objeto. Um exemplo real de objeto é um automóvel. Você descreve um automóvel por suas propriedades, como um conversível vermelho ou um sedan preto de quatro portas. Cada característica, a cor, o número de portas, o fato de ser conversível ou não, todos são propriedades do carro.

Os **métodos** são as ações que um carro executa em resposta a um evento. Por exemplo, você inicia um evento quando gira a chave para dar partida no carro. O método de inicialização do carro assume o controle nesse momento, fornecendo instruções: como engatar a marcha inicial, ligar o motor de arranque, iniciar a injeção de combustível, enviar centelhas às velas e desativar o motor de arranque. Você não precisa informar ao carro como dar a partida ao motor.

A **herança**, significa que um determinado objeto pode se basear na descrição de outro objeto. Continuando com o exemplo do automóvel, é possível definir um carro como algo que tem quatro rodas, um motor e assentos para os passageiros. Em conseqüência disso, você pode definir um conversível como um carro que possui um teto retrátil. O conversível herda as propriedades do carro e acrescenta a elas uma nova propriedade, o teto retrátil. Não é necessário redefinir as propriedades do carro para o conversível. Por este motivo, dizemos que o conversível herda as propriedades do carro. Além das propriedades, os objetos também podem herdar métodos e eventos de outros objetos.

O **polimorfismo** indica que muitos objetos podem ter o mesmo método, e que a ação apropriada é executada para o objeto que chama o método. Por exemplo nos seus programas, você exibe texto na tela e promove a saída do texto na impressora. Cada um desses objetos (a tela e a impressora) pode possuir um método de impressão ou exibição que informe ao objeto que este deve inserir o texto em um determinado local. O método sabe o que fazer, de acordo com objeto que executa a chamada ao método.

O que a OOP pode fazer por você.

Os elementos fundamentais da OOP, com os quais você irá trabalhar, são componentes reutilizáveis conhecidos como controles ou classes que você irá criar. Os controles ou classes de objetos, que você usará na elaboração dos seus programas, são objetos que têm propriedades e métodos e que respondem a eventos. Você determina a aparência e o comportamento de um controle através de suas propriedades.

Por exemplo, você especifica qual será o aspecto do texto em uma caixa de texto, definindo suas propriedades **Font** e **Color**. Os controles têm métodos incorporados a eles e, ao utiliza-los, você é isolado de várias tarefas tediosas de programação. Vejamos mais uma vez o exemplo da caixa de texto: ela **"sabe"** como obter a informação do acionamento de uma tecla e exibir esta informação na região de edição da caixa, no formato apropriado. Não é necessário fornecer os detalhes dessa tarefa.

Abstração, Classes, Herança, Instância e Objetos.

Abstração/Encapsulamento

O Encapsulamento ou abstração, é a característica das classes que nos permite abstrair (ignorar) do funcionamento interno da classe. Não temos que conhecer em detalhes como é que as coisas são feitas internamente, desde que saibamos trabalhar com os objetos.

Tal como no mundo real, podemos ignorar ou abstrair-nos do que sucede "dentro" de um objeto, desde que saibamos interagir com ele. Ao fazer um sobre um botão sabemos que, graficamente, irá parecer que o botão desceu e depois voltou a subir para sua posição inicial e que, em seguida, será desencadeada a ação que está associada ao botão que carregamos; devido à caraterística de encapsulamento das classes, não é necessário nem importante sabermos o que deve de ser feito para criar essa ilusão de descida e volta à posição inicial, temos apenas que nos preocupar com o código que queremos que seja executado em resposta à referida ação de clique sobre o objeto.

Fazendo um paralelo com o mundo real, e usando um carro como objeto, se tivermos uma classe "carro", ou seja, um modelo para a criação de todos os carros, não é muito importante sabermos tudo o que acontece quando viramos a chave para ligar o motor, podemos dar como adquirido, que, nas condições adequadas, virar a chave irá ligar o motor.

O help do VFP diz que abstração é identificar as características distintivas de uma classe ou objeto sem precisar processar todas as informações sobre a classe ou o objeto.

Nesta definição, abstração está limitada a OOP, usada somente para classificar coisas em grupos e/ou tipos de coisas.

Colocamos as definições de classes e objetos logo abaixo, porém é importante continuar afirmando que em nossa concepção tudo que tem vida tem abstração. Quando uma árvore está crescendo, ela está fazendo o que é da natureza dela, com suas propriedades e seus métodos, coisas que vocês vão ver mais tarde.

Quanto mais subjetividade você tiver, melhor será sua análise orientada a objetos, isto do ponto de vista completamente abstrato. Ou seja, você agora vai pensar de forma abstrata na hora de construir seus aplicativos, pensado primeiramente em classes de objetos que seus aplicativos terão e serão reaproveitados em varias etapas da aplicação, como você fazia com suas procedures e funções.

Classes

É um modelo que serve de base a objetos. Como objetos que são, as classes têm propriedades, eventos e métodos. Como modelos, as classes têm características muito importantes. Uma classe seria no mundo real, um modelo. Uma classe "carro" seria o modelo de base no todos os objetos (todos os carros) seriam construídos. Depois de construídos, os objetos podem sofrer alterações relativamente à classe em que se baseiam; assim, por exemplo, do mesmo modelo de carro surgem alguns com diferentes cores e modelos.

Desde que nos entendemos por gente, estamos classificando as coisas ao nosso redor, por exemplo, carro (da classe transportes ou classe automóveis), mesa e cadeira (classe móveis), casa (da classe imóveis) e vai por ai a fora.

Um único objeto pode estar classificado em várias classes e subclasses, não vem ao caso estudar isso aqui.

Todos estes objetos citados, existiram antes na concepção de uma idéia, para depois existirem como objetos concretos, neste momento foram de fato instanciados, como uma casa que agora existe e antes estava somente na planta. A casa construída é uma das instâncias daquela planta da classe imóveis, e da mesma planta podemos instanciar (criar) outras casas do mesmo padrão.

No nosso caso, todos os formulários que construímos, são instâncias do objeto formulário da classe de formulários de VFP.

Observe a tabela abaixo com as classes de base do Visual FoxPro em ordem alfabética da esquerda para a direita.

CheckBox	Form	OleContainerControl
ComboBox	FormSet	OptionButton
CommandButton	Grid	OptionCroup

CommandGroup	Image	PageFrame
Container	Label	Sahpe
Control	Line	Spinner
Custom	ListBox	TextBox
EditBox	OleBoundControl	Timer e ToolBAr

O VFP subdivide essas classes em classes de container e de controle. Uma classe de container pode conter outros objetos. Por exemplo, um formulário pertence a uma classe de container porque você pode inserir nele outros objetos, como, por exemplo, caixas de verificação, de edição e de texto bem como linhas e outros objetos semelhantes. As classes de base container incluem o seguinte:

- Column Object
- Command Button Groups
- Custom
- Container
- Control
- Forms
- FormSets
- Grids
- Option Groups
- Page
- Page Frame
- Toolbars

As classes de controle não podem conter outros objetos. Por exemplo, você não pode inserir um objeto de linha em um objeto de caixa de texto. Depois de colocar um objeto de controle em um container, qualquer referência a um objeto tem que passar pelo container. Para fazer referência a uma propriedade de um controle armazenada em um container, use a seguinte sintaxe:

<nome do objeto de container>.<nome do objeto de controle>.<propriedade>

Você usa a classe básica custom do VFP, para criar uma classe definida pelo usuário, sem representação visual.

Herança

Característica das classes pela qual todas as alterações feitas a uma classe (às suas propriedades, eventos ou métodos), se propagam a todas as subclasses, subsubclasses, etc, direta ou indiretamente ligados, baseados ou não nesta classe.

Esta característica das classes de Visual FoxPro é muito útil: se usarmos um modelo para todos os botões (push buttons) e este usar a fonte Times New Roman, corpo 12 para o texto que surge escrito nos botões, todos os botões e subclasses que se baseiam nesse modelo irão usar estas mesmas propriedades: Fonte Times New Roman, corpo 12 pontos. Se alterarmos o modelo para a fonte Arial, corpo 8 pontos, todos os objetos e subclasses irão refletir a mudança que fizemos no modelo e passarão a usar também fonte Arial, 8 pontos. No entanto se, em um botão derivado da classe, tivéssemos alterado já a fonte para Courier New, esse botão não iria mudar para Arial, visto que sua fonte já não estava sendo baseada na fonte do modelo; mesmo assim, o tamanho da fonte mudaria para 8 pontos, caso esse valor fosse ainda o valor baseado no modelo.

Uma terminologia da programação orientada a objetos. A capacidade de uma subclasse assumir as características da classe em que está baseada. Se as características da classe pai se alterarem, a subclasse nela baseada herdará estas características. Por exemplo, se você adicionar

uma nova propriedade, IsBold, a um controle de edição, todas as subclasses baseadas neste controle também terão uma propriedade IsBold.

Você vai aprender a criar classes e objetos, modificar e criar propriedades e métodos desses objetos no intuito de aproveitar a herança de código e de dados, no momento de usar os objetos em quaisquer partes e sobre todos os sistemas que você irá desenvolver.

Instância

Uma terminologia da programação orientada a objetos. Um objeto criado a partir de uma definição de classe. Ao contrário de uma classe, que é apenas uma definição, uma instância existe de fato como um objeto que pode ser utilizado para executar tarefas. Por exemplo, uma caixa de texto em um formulário em execução é uma instância da classe TextBox.

Qualquer objeto criado é uma instância de sua classe principal. O processo de criação do objeto é, então, denominado instanciação. Se por exemplo você definir uma outra instância do objeto, ela será exatamente igual a primeira. Como exemplo, os comandos a seguir criam duas instâncias da classe form:

```
frmjan1=createobject("form")
frmjan2=createobject("form")
frmjan1.show
frmjan2.show
```

Observe que depois de você digitar esses comandos na janela de comando do VFP, serão exibidas duas janelas de igual tamanho com os títulos de form1 e form2, a partir daí, basta você atribuir as propriedades caption e width, para por exemplo, mudar a título e a largura de cada uma delas.

```
frmjan1.caption=”Primeira Janela”
frmjan1.width=300
```

Objetos

O help do VFP diz que objeto é uma instância de uma classe que combina dados e procedimentos, e prossegue dando como exemplo: um controle em um formulário em execução.

O VFP executa implicitamente um comando Create Object, e o objeto passa a existir (evento init) e estar em escopo (momento que pode ser refenciado) até sua “destruição” (evento destroy).

No evento init de um formulário eu posso fazer o seguinte teste

```
if ! logado
  return .f.
endif
```

Poderia estar testando se um usuário está com logon no sistema para ter acesso ao formulário em questão.

Propriedades, Métodos, Eventos e Encapsulamento de um objeto.

Propriedade

Propriedades são atributos de todos os objeto. Por exemplos:

Carro.cor = azul
Haroldo.altura = 1,74 m
Haroldo.peso = 74 kg

Atribuímos a um carro qualquer a sua cor como azul, e mostramos como o VFP falaria qual é a minha altura e qual é o meu peso. Lembre-se agora que quando falamos de instância você criou uma janela e mudou o título dela com a propriedade atribuindo o valor da propriedade caption frmjan1.caption="Primeira Janela". Só para lembrar, você sabe como foi o meu nascimento na versão do VFP, não ?

Haroldo=createobject("SerHumano"). Assim instanciei-me nessa classe que já teve de Hittler a Mozart. Uma vez eu e um amigo meu, sociólogo, depois de horas de conversas metafísicas, chegamos a conclusão que a humanidade era um projeto que não deu certo, se eu tivesse contato com ele hoje iria dizer que a classe está com BUG. Foi dito que Deus fez o homem à sua imagem e semelhança, Hitler conseguiu passar tão longe de qualquer semelhança divina, que se Deus quiser ninguém mais vai nascer com os bugs que ele apresentou.

E quando eu morrer (sair de escopo) : Haroldo.release (release é um método do objeto), no meu dicionário de inglês release quer dizer libertação, quem sabe é a hora de dar o fora dessa classe que traz muito mais decepções do que acréscimos espirituais. Bom vamos continuar estudando VFP, este exemplo embora meio louco, só procura passar para você a forma hierárquica de como estão os objetos na OOP.

Outra coisa que vai fazer parte da vida dos novatos em orientação a eventos e OOP, e a definição de propriedades em tempo de projeto e em tempo de execução. Quando você ler sobre uma propriedade no help, as vezes você verá: Disponível na hora da criação e em tempo de execução.

A propriedade visible dos formulários e dos objetos que podem ser colocados nos formulários podem ser definidas em tempo de projeto e em tempo de execução com .t. ou .f., como você quiser. No entanto para o "Haroldo" foi convencionado em tempo de projeto que ele deveria estar sempre visível como o resto da humanidade, haroldo.visible=.t., não podendo ser mudado em tempo de execução, desta forma o "Haroldo" não vai poder realizar seu sonho de entrar invisível em qualquer vestiário feminino e..., deixa pra lá.

Esse último exemplo você pode ter achado horrível, vamos melhorar isso com o que me disseram uma vez: "Deus não dá asas a cobra" (olha Deus novamente na OOP), imediatamente você pode imaginar em seu pensamento abstrato uma jibóia de quatro metros e asas enormes engolindo seres humanos pela cabeça. Ou então ver que estrago poderia fazer um ditador se tivesse o poder de ficar invisível, é melhor deixar toda a humanidade sem poder por herança divina, se tornar invisível, inclusive eu.

Agora veja o que o VFP diz no help sobre a propriedade BaseClass- a propriedade em que um objeto está baseado - Especifica o nome da classe principal do Visual FoxPro, na qual está baseado o objeto que se referencia. Disponível somente para leitura na hora da criação e em tempo de execução. Isso impede que você tente fazer algum tipo de loucura com a natureza de algumas coisas, como por exemplo, mudar a referencia da classe de uma combobox para textbox.

Se o "João" por exemplo, tivesse sido instanciado na classe "Homem", uma subclasse da classe "SerHumano", ele iria trazer as propriedades baseclass=homem e parentclass=SerHumano, e ambas disponíveis somente para leitura, mesmo que ele conseguisse o melhor cirurgião do planeta na tentativa de mudar de sexo.

Observe o seguinte:

formulário.show (show é um método para apresentar um objeto instanciado)

formulário.visible=.t. (aqui temos uma propriedade para apresentar um objeto instanciado)

O método hide para esconder objeto e a propriedade de visibilidade do objeto definida com false, possuem os mesmos resultados, no sentido de ocultar o formulário.

Lembre-se: as propriedades falam do estado e das características do objeto, os métodos falam do comportamento do objeto.

Método

Uma ação que um objeto tem capacidade para executar. Por exemplo, caixas de listagem possuem métodos denominados AddItem, RemoveItem e Clear para manter o conteúdo das listas.

Método é o nome generalizado para o código que você escreve para definir comportamentos e reações do objeto. Os objetos do VFP já vêm com vários métodos nativos que você não pode acessar ou modificar, e você os usa para controlar o comportamento do objeto

Exemplo de métodos nativos são: show, move (move um objeto visual), hide, additem (adciona um item a num controle combobox ou listbox). Para executar um método basta usar a sintaxe Nome_do Objeto.Nome_do_metodo.

exemplos:

```
frmFormulario.show
cboCaixacombo.move(5, 5, 10, 12)
```

Os parâmetros do método move correspondem às novas coordenadas do canto esquerdo superior, e a nova largura e altura do obejto.

Podemos também criar quantos métodos quisermos para um objeto.

Dessa forma vamos colocar nos nossos formulários, métodos que nos interessem e digitar neles os códigos que forem da nossa vontade. Os métodos criados são procedures, mas diferente de procedures e programas, eles ficam intrinsecamente ligados ao objeto.

Quando quiser programar a reação de um objeto para um evento que ele pode detectar, você escreve um método para o evento. O código do método será executado toda vez que o objeto detectar o evento.

Se há um método (escrito) para o evento click de um certo objeto, por exemplo, você pode programaticamente executar o método do evento sem que o evento tenha sido detectado. Use a mesma sintaxe para executar um método, como nesses exemplos:

```
frmFormulario.cmdBotao.click
frmFormulario.dbclick
```

Estes eventos só aconteceriam por intermédio do mouse ou no caso do botão, quando o mesmo tivesse o foco (quando o cursor chega ao objeto através do mouse ou teclado), no entanto através da sintaxe respeitando a hierarquia do objeto, podemos executar o método ou rotina do evento.

Outro exemplo seria supor que dentro do seu gride com as colunas de Quantidade, Preço Unitário e Preço Parcial, você quisesse sempre ao alterar a quantidade ou o preço unitário, o preço parcial fosse atualizado. Se O ControlSource da coluna preço parcial está como (preço unitário multiplicado por quantidade), basta você chamar o método refresh da coluna preço parcial, a cada

mudança de quantidade ou preço unitário, isto se faria no evento valid de text de preço unitário ou quantidade. Ficaria assim:

No campo text evento valid, de coluna quantidade de grid1

```
WITH This.Parent.Parent

    =TABLEUPDATE()

    .PrecoParcial.Refresh

ENDWITH
```

Essas palavras this.parent e tableupdate, não se preocupe com elas, você irá aprender na hora certa. Para ir adiantando, tableupdade() faz a atualização do registro com o formulário usando buffer e .parent retorna ao nível imediatamente acima do objeto, neste caso, ficamos no nível do gride para que o comando .PrecoParcial.refresh pudesse ser executado assim sem precisar de um caminho hierárquico desde de o formulário.

Evento

Um acontecimento em tempo de execução, reconhecido por um objeto para o qual você pode programaticamente definir uma reação, por exemplo, colocar um código no evento click de um botão.

Eventos não podem ser criados, modificados, eliminados ou guardados. Eventos podem ser disparados por uma ação do usuário, como um click do mouse, e também podem ser disparados pelo sistema, como por exemplo, um controle timer no formulário.

Você controla o comportamento do objeto no momento que ele detecta o evento, escrevendo um método para o evento. Assim, você tem a opção também de executar o método do evento, sem que o vento necessariamente ocorra, por exemplo, dar um refresh no formulário quando estiver executando o comando clique de um botão. Os exemplos dados quando falamos em método de escrever métodos para eventos, se encaixam aqui.

Todas as classes básicas do VFP reconhecem o seguinte conjunto mínimo de eventos:

Evento	Quando acontece.
Init	Quando o objeto é criado.
Destroy	Quando o objeto é liberado (released) da memória
Error	Quando um erro ocorre num evento ou método do objeto

Incluindo e além desses três eventos essenciais, a tabela a seguir, lista os eventos mais comuns do VFP, que são reconhecidos (detectados) pela maioria dos controles.

Eventos	Quanto acontece
Load	Formulário ou formset é carrego na memória
Unload	O Formulário ou formset é descarregado da memória
Init	Um objeto é criado
Destroy	Um objeto é liberado da memória
click	Usuário clica o botão esquerdo do mouse
dblclick	Usuário dá dupla-clicada sobre o objeto com botão esquerdo do mouse
Righ click	Usuário clica o botão direito do mouse sobre o objeto
GotFocus	O objeto “recebe o foco”, seja por ação do usuário usando o mouse ou o teclado (tab, shift+tab), ou programaticamente pela execução do método setfocus.

LostFocus O objeto “perde o foco”, seja por ação do usuário usando o mouse ou o teclado (tab, shift+tab), ou programaticamente pela execução do método setfocus. (selecionando outro objeto).
KeyPress O usuário pressiona uma tecla
MouseDown O usuário pressiona (sem soltar) o botão do mouse enquanto o ponteiro estiver visualmente sobre o objeto.
MouseMove O usuário move (não precisa clicar) o mouse sobre o objeto
MouseUp O usuário solta o botão enquanto o ponteiro está visualmente sobre o objeto.
InteractiveChange O valor do objeto é mudado interativamente.
Programmatic change O valor do objeto é mudado programaticamente

LOOP DE EVENTOS - Na programação de um aplicativo, depois de estabelecer o ambiente do aplicativo e exibir o menu e a tela inicial, você está pronto para iniciar o loop que irá abrir e manter a sessão interativa e deixar os acontecimentos sob controle do usuário. No VFP, você inicia o loop executando READ EVENTS. Esse comando inicia um loop que espera e detecta as ações do usuário, ou os eventos de sistema que você programou usando um controle Timer por exemplo. Quando o usuário decidir encerrar a sessão, ele terá a opção de executar o CLEAR EVENTS, programado no aplicativo.

Voltando ao Encapsulamento

Um objeto pode conter e esconder as informações sobre si próprio, como código e estruturas de dados internas. O encapsulamento isola a complexidade interna da operação do objeto do resto do aplicativo. Por exemplo, ao definir a propriedade Caption em um botão de comando, você não precisa saber como a seqüência será armazenada.

Outro exemplo: Todo mundo que é da classe Ser Humano dispara o evento da digestão independente da sua vontade e sem o conhecimento de como ele se processa.

O código abaixo mostra como adicionar um botão a um formulário através de forma programada:

```
Release all
frmjan1=createobject("form")
frmjan1.show
frmjan1.caption="Meu formulário com botão"
frmjan1.addobject("MeuBotao","CommandButton")
cmdButton1=frmjan1.MeuBotao
cmdButton1.visible=.t.
cmdButton1.top=20
cmdButton1.left=5
cmdButton1.width=100
cmdButton1.height=20
cmdButton1.caption="Exit"
wait window
```

Experimente colar esse código num programa na guia de códigos do container de projeto e executar para ver o resultado.

Obs: Ao digitar esse exemplo, o botão foi apresentado antes de seu tamanho, sua posição e seu nome serem definidos. Em um aplicativo, você provavelmente vai querer manter os objetos ocultos até tê-los definidos por completo.

Referindo-se a um objeto na hierarquia

Quando falamos de métodos você com certeza ficou viajando quando colocamos um exemplo, que tratava objetos dentro de uma hierarquia, pois é assim, para manipular um objeto, você o identifica no código em referência à hierarquia em que o objeto está inserido, da mesma forma que chega até uma determinada pasta ou diretório no caminho de um disco local ou da rede. Por exemplo, para se referir a um controle numa página (page) de uma formulário inserido num formset (conjunto de formulários), você escreve: Formset.Form.PageFrame.Page.Controle. Os nomes dos objetos são separados por um ponto.

Você [pode identificar um objeto numa hierarquia de objetos de duas maneiras: por referência absoluta (absolute referencing) , ou por referencia relativa (relative referencing).

Referenciação absoluta: Para identificar e manipular uma propriedade, método ou evento de um objeto, você identifica o objeto, seguido do nome da propriedade, método ou evento. Os exemplos geréricos a seguir, identificam um controle inserido na coluna de um grid, que por sua vez está inserido numa página de uma pageFrame, que está inserido num formulário que faz parte de um conjunto de formulários (formset):

*** Para valorizar a propriedade enable.
formSet.form.PageFrame.Page.Grid.Column.Control.Enable=.f.

*** Para executar um método show.

formSet.form.PageFrame.Page.Grid.Column.Control.show

Como esses exemplos são genéricos, no seu código você terá que fazer a substituição pelos nomes reais de seus objetos.

Referenciação Relativa: É quando nos referimos a um objeto dentro de uma hierarquia usando as propriedades de somente leitura: Parent, ActiveControl, ActiveForm, ActivePage e as palavras this, thisform e thisformset.

Propriedade/Palavra chave	Referência
Parent	O container imediato do objeto.
This	O objeto.
Thisform	O formulário que contem o objeto.
ThisformSet	O conjunto de formulários que contem o objeto.
ActiveControl	O controle que tem o foco no form atualmente ativo.
ActiveForm	O formulário atualmente ativo.
ActivePage	A página atualmente ativa no formulário ativo.

Exemplos de referências relativas:

*** No código de qualquer objeto em um formulário num conjunto de formulários.

Thisformset.Frml.Cmdl.Caption.=”OK”

*** No código de qualquer objeto dentro da mesma forma onde está o objeto Cmdl.

Thisform.Cmdl.Caption=”OK”

*** No código do objeto que você está definindo a propriedade caption

This.caption=”OK”

*** No código de um objeto dentro de um formulário. Neste caso, o comando define a cor de fundo da forma onde está o objeto.

```
This.Parent.backcolor=RGB(192,0,0)
```

A propriedade parent sempre retorna o nome do container acima do controle:

```
nome_do_container = nome_do_controle.parent
```

Definindo múltiplas propriedades com With ... Endwith

A linguagem OO dispõe do comando With...Endwith, que simplifica e agiliza a definição de múltiplas propriedades. O exemplo a seguir, define as propriedades de largura, redimensionável, cor de frente e de fundo, de uma coluna de um grid dentro de um formulário, dentro de um conjunto de formulários.

```
With frsFormSet1.frmForm1.grdGrid1.grcColumn1.

        .width = 5
        .resizable = .f.
        .forecolor = RGB(0,0,0)
        .backcolor = RGB(255,255,255)

Endwith
```

Fazendo a propriedade resizable de uma coluna =.t., permite que o usuário consiga redimensionar horizontalmente a coluna, clicando e arrastando na linha entre as colunas.

Integrando o Visual com Outras Ferramentas

A seguir um exemplo de como integrar as ferramentas Visual FoxPro e Excel. O objetivo é simular em um gráfico, usando-se de números aleatórios, como tem sido as inscrições mês a mês no grupo FoxBrasil.

```
mProg = "c:\arquivos de programas\microsoft office\office\excel.exe"
mProg = mProg + " /E"
RUN /N1 &mProg
nCanal1   = DDEInitiate('Excel', 'System')
cPlanilha = "C:\Graficos\InscriFoxBrasil.Xls"
cComando  = '[OPEN("' + AllTrim(cPlanilha) + '")]'

*Observe a configuração no Excel para números, caso o ponto decimal esteja representado por
*ponto, substitua o comando abaixo por set point to '.'
set point to ','
set date brit

IF nCanal1 >=0
        lExec   = DDEExecute(nCanal1, cComando)
        nCanal2 = DDEInitiate('Excel', 'InscriFoxBrasil.xls')
        IF nCanal2 >=0
                cComando = '[SELEC("L6C3:L17C4")]'
                lExec = DDEExecute(nCanal2, cComando)
                lExec = DDEExecute(nCanal2, '[LIMPAR(3)]')
                lSend = DDEPoke(nCanal2, 'L5C3', 'MÊS')
                lSend = DDEPoke(nCanal2, 'L5C4', 'QTD.')
                nLin = 6
```

```
                nVal = INT(100 * RAND(-1) + 1)
                FOR i = 1 to 12
                        cData = "01/" + alltrim(str(i,2)) + "/1999"
                        dData = CTOD(cData)
                        cMes = LEFT(CMonth(dData),3)
                        nVal = INT(100 * RAND() + 1)
                        cVal = STR(nVal,5,2)
                        cLinCol = "L" + AllTrim(STR(nLin)) + "C3"
                        lSend = DDEPoke(nCanal2, cLinCol, cMes)
                        cLinCol = "L" + AllTrim(STR(nLin)) + "C4"
                        lSend = DDEPoke(nCanal2, cLinCol, cVal)
                        nLin = nLin + 1
                        INKEY(0.5)
                ENDFOR
                lExec = DDEExecute(nCanal2, '[SELEC("L1C1")]')
                lExec = DDETerminate(nCanal2)
        ENDIF
        lExec = DDETerminate(nCanal1)
ENDIF
??chr(7)
```

Para poder visualizar o resultado, dentro do Excel, crie uma planilha e um gráfico baseado nas colunas C e D, sendo que na coluna C da linha 6 até a 17 coloque os meses e na coluna D das linhas 6 até a 17 os valores. Sobre estes dois crie um gráfico e o mantenha ao lado dos valores. Após criado o gráfico, pode apagar os valores usados como exemplo e execute a rotina acima.