

Embora o Visual FoxPro ainda suporte programação procedural padrão, novas extensões para a linguagem conferem a você o poder e a flexibilidade da programação orientada a objetos.

A criação e a programação orientadas a objetos representam uma mudança de enfoque em relação à programação procedural padrão. Em vez de pensar no fluxo do programa da primeira à última linha de código, você precisa pensar na criação de objetos: componentes independentes de um aplicativo que possuam uma funcionalidade particular, bem como uma funcionalidade que você pode expor ao usuário.

Este capítulo aborda:

- [Conhecendo os objetos no Visual FoxPro](#)
- [Conhecendo as classes no Visual FoxPro](#)
- [Combinando classe e tarefa](#)
- [Criando classes](#)
- [Adicionando classes a formulários](#)
- [Definindo classes por programação](#)

## Conhecendo os objetos no Visual FoxPro

No Visual FoxPro, [formulários](#) e [controles](#) são objetos que você inclui em seus aplicativos. Você manipula estes objetos através de [propriedades](#), [eventos](#) e [métodos](#).

As extensões de linguagem do Visual FoxPro orientadas a objetos conferem um controle muito maior sobre os objetos nos aplicativos. Estas extensões também facilitam a criação e manutenção de bibliotecas de códigos reutilizáveis, dando a você:

- Código mais compacto.
- Facilidade na incorporação de código em aplicativos sem esquemas de nomeação elaborados.
- Menos complexidade na integração de códigos a partir de arquivos diferentes em um aplicativo.

A programação orientada a objetos é uma maneira de compactar códigos para que eles possam ser reutilizados e mantidos mais facilmente. O pacote primário é denominado [classe](#).

## Classes e objetos:

### A base dos aplicativos

Classes e objetos estão estreitamente relacionados, mas não são a mesma coisa. Uma classe contém informações sobre como devem ser a aparência e o funcionamento de um objeto. Uma classe é a planta ou o esquema de um objeto. O esquema elétrico e o layout de um telefone, por exemplo, seriam semelhantes a uma classe. O objeto ou um exemplo da classe, seria um telefone.

**A classe determina as características do objeto.**

**Classe / Esquema**



**Objeto / Telefone**

## Propriedades dos objetos

Um objeto possui determinadas [propriedades](#) ou atributos. Por exemplo, um telefone tem uma determinada cor e tamanho. Quando você instala um telefone no seu escritório, ele ocupa uma determinada posição na mesa. O fone pode estar ou não no gancho.

Os objetos que você cria no Visual FoxPro também têm propriedades que são determinadas pela classe na qual o objeto está baseado. Estas propriedades podem ser definidas na [hora da criação](#) ou [em tempo de execução](#).

Por exemplo, a tabela a seguir lista algumas das propriedades que uma [caixa de verificação](#) pode ter:

Propriedade	Descrição
Caption	O texto descritivo ao lado da caixa de verificação.
Enabled	Se a caixa de verificação pode ser escolhida por um usuário.
ForeColor	A cor do texto da legenda.
Left	A posição do lado esquerdo da caixa de verificação.
MousePointer	A aparência do ponteiro do mouse quando ele está sobre a caixa de verificação.
Top	A posição da parte superior da caixa de verificação.
Visible	Se a caixa de verificação está visível.

## Eventos e métodos associados aos objetos

Cada objeto reconhece e pode responder a determinadas ações denominadas [eventos](#). Um evento é uma atividade específica e predeterminada, iniciada por um usuário ou pelo sistema. Os eventos, na maioria dos casos, são gerados por uma interação com o usuário. Por exemplo, com um telefone, um evento é disparado quando um usuário tira o fone do gancho. Também são disparados eventos quando o usuário pressiona as teclas de um telefone para fazer uma ligação.

No Visual FoxPro, as ações do usuário que disparam eventos incluem cliques, movimentos do mouse e pressionamentos de teclas. Inicializar um objeto e encontrar uma linha de código que está causando um erro são eventos iniciados pelo sistema.

Os [métodos](#) são procedimentos que são associados a um objeto. Eles são diferentes dos procedimentos normais do Visual FoxPro: os métodos estão inextricavelmente ligados a um objeto e são chamados de modo diferente de como são chamados os procedimentos normais do Visual FoxPro.

Os eventos podem ter métodos associados a eles. Por exemplo, se você escrever um código de método para o evento Click, esse código será executado quando o evento Click ocorrer. Os métodos também podem existir independentemente de qualquer evento. Esses métodos devem ser chamados de forma explícita no código.

O conjunto de eventos, embora amplo, é fixo. Você não pode criar novos eventos. O conjunto de métodos, contudo, pode ser estendido infinitamente.

A tabela a seguir lista alguns dos eventos associados a uma [caixa de verificação](#):

Evento	Descrição
<a href="#">Click</a>	O usuário clica na caixa de verificação.
<a href="#">GotFocus</a>	O usuário seleciona a caixa de verificação, com um clique ou através da tecla Tab.
<a href="#">LostFocus</a>	O usuário seleciona outro controle.

A tabela a seguir lista alguns dos métodos associados a uma caixa de verificação:

Método	Descrição
<a href="#">Refresh</a>	O valor da caixa de verificação é atualizado para refletir

qualquer alteração que tenha ocorrido na fonte de dados básica.

#### SetFocus

O destaque é definido para a caixa de verificação como se o usuário tivesse pressionado a tecla TAB até que a caixa de verificação fosse selecionada.

Consulte o capítulo 4, [Conhecendo modelos de eventos](#), para obter uma explicação sobre a ordem em que os eventos ocorrem.

## Conhecendo as classes no Visual FoxPro

Todas as [propriedades](#), [eventos](#) e [métodos](#) para um objeto são especificados na definição de classe. Além disso, as classes apresentam as características a seguir, que as tornam úteis na criação de código reutilizável e de fácil manutenção:

- Encapsulamento
- Subclasses
- Herança

## Ocultando a complexidade desnecessária

Quando você instala um telefone no seu escritório, provavelmente não se preocupa em como ele recebe uma chamada interna, inicia ou finaliza as conexões com o PBX ou traduz as teclas pressionadas em sinais eletrônicos. Tudo o que você precisa é saber que pode retirar o fone do gancho, discar o número apropriado e conversar com a pessoa desejada. A complexidade desta conexão está oculta. O benefício de poder ignorar os detalhes internos de um objeto para poder focar os aspectos do objeto que você necessita é chamado [abstração](#).

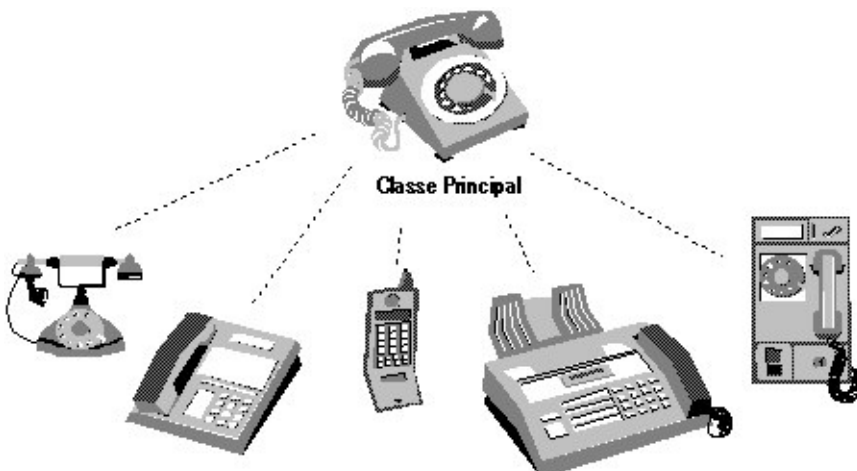
### A complexidade interna pode ser ocultada

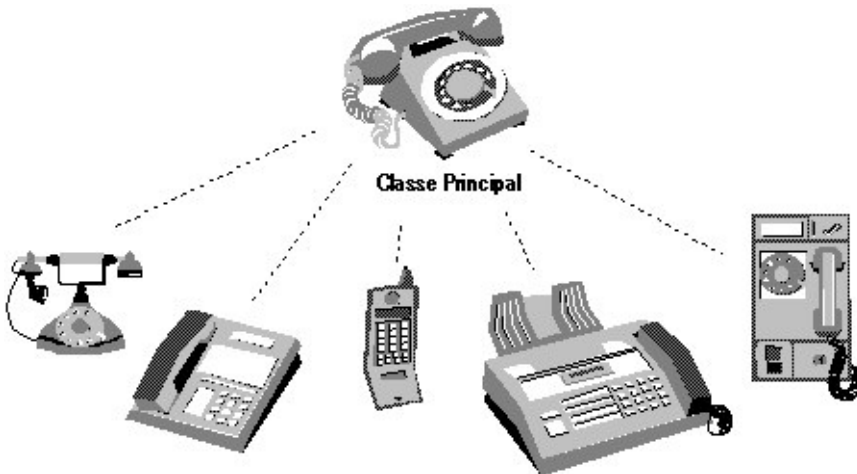


O [encapsulamento](#), que envolve empacotamento do código da propriedade e do método juntos em um objeto, contribui para a abstração. Por exemplo, as propriedades que determinam os itens de uma [caixa de listagem](#) e o código que é executado quando você escolhe um item na lista, podem ser encapsulados em um único [controle](#) que você adiciona a um [formulário](#).

## Multiplicando o poder das classes já existentes

Uma [subclasse](#) pode ter toda a funcionalidade de uma classe já existente, mais qualquer controle ou funcionalidade que você queira acrescentar. Se sua classe for um telefone básico, você poderá ter subclasses com toda a funcionalidade do telefone original e qualquer recurso especializado que você deseje dar a elas.





A criação de subclasses é uma maneira de diminuir a quantidade de código que você tem que escrever. Você pode começar com a definição de um objeto que seja semelhante ao que você deseja e personalizá-lo.

## Tornando eficiente a manutenção do código

Com a [herança](#), se você fizer uma alteração em uma classe, essa alteração se refletirá em todas as subclasses baseadas na classe. Esta atualização automática economiza tempo e trabalho. Por exemplo, se um fabricante de telefones quiser mudar o estilo dos telefones, de disco para teclas, ele economizará muito trabalho se alterar o esquema principal e fizer com que todos os telefones fabricados anteriormente com base nesse esquema principal herdem automaticamente essa nova característica, em vez de modificar individualmente todos os telefones já existentes.

**A herança facilita a manutenção do seu código.**

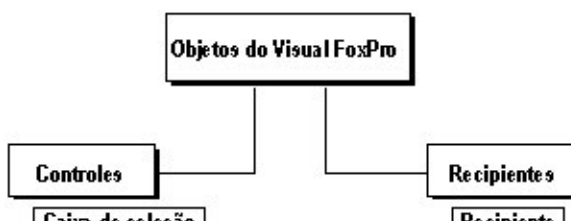


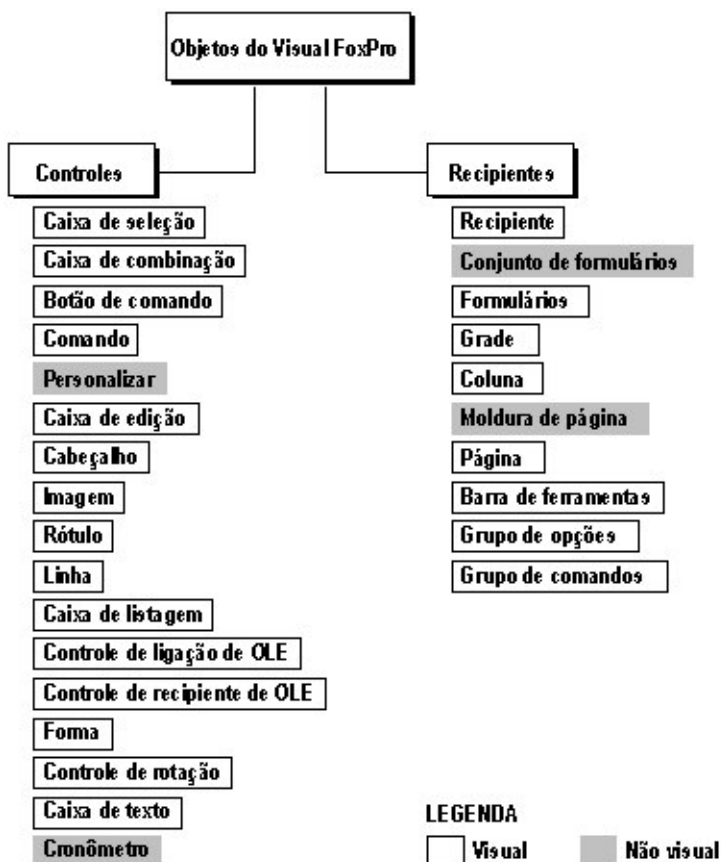
A herança não funciona com o hardware, mas você tem essa capacidade em relação ao software. Se você descobrir um problema em uma classe, em vez de ir a cada subclasses e alterar o código, conserte-o uma única vez na classe e a alteração se propagará por todas as subclasses.

## Hierarquia de classes do Visual FoxPro

Ao criar classes definidas pelo usuário, isto ajudará você a compreender a hierarquia de classes do Visual FoxPro.

**Hierarquia de classes do Visual FoxPro**

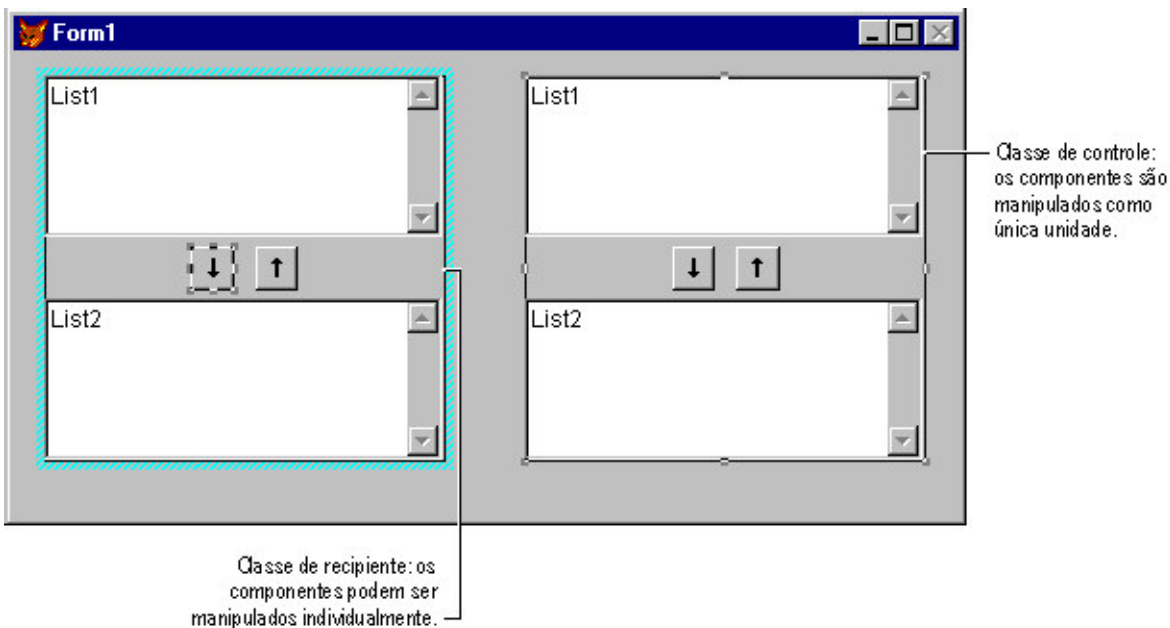




## Recipientes e não-recipientes

Os dois tipos primários de classes do Visual FoxPro e, por extensão, os objetos do Visual FoxPro são: classes recipientes e classes de controle.

### Classes recipientes e classes de controle



## Classes recipientes

Os recipientes podem conter outros objetos e permitem o acesso aos objetos contidos neles. Por exemplo, se você criar uma classe recipiente constituída de duas [caixas de listagem](#) e dois [botões de comando](#) e depois adicionar um objeto baseado nessa classe a um formulário, cada objeto poderá ser manipulado individualmente em [tempo de execução](#) e na [hora da criação](#). Você pode alterar facilmente as posições das caixas de listagem ou as legendas dos botões de comando. Você pode também adicionar objetos ao [controle](#) na hora da criação; por exemplo, você pode adicionar rótulos para identificar as caixas de listagem.

A tabela a seguir lista o que cada classe recipiente pode conter:

Recipiente	Pode conter
Grupos de botões de comando	Botões de comando
Recipiente	Qualquer controle
Controle	Qualquer controle
Personalizado	Qualquer controle, moldura de página, recipiente, controle personalizado
<a href="#">Conjunto de formulários</a>	Formulários, <a href="#">barras de ferramentas</a>
<a href="#">Formulários</a>	Molduras de página, qualquer controle, recipientes, controles personalizados
Colunas de grade	Cabeçalhos de colunas, qualquer objeto exceto formulários, conjuntos de formulários, barras de ferramentas, <a href="#">cronômetros</a> e outras colunas
<a href="#">Grades</a>	Colunas de grade
Grupos de botões de opção	Botões de opção
Molduras de página	Páginas
Páginas	Qualquer controle, recipientes, controles personalizados
Barras de ferramentas	Qualquer controle, moldura de página, recipiente

## Classes de controle

As classes de controle são mais encapsuladas do que as classes recipientes, mas, por essa razão, podem ser menos flexíveis. As classes de controle não têm um [método AddObject](#).

## Combinando classe e tarefa

Você quer utilizar as classes em muitos contextos diferentes. Um planejamento adequado permitirá que você decida melhor quais são as classes a serem criadas e qual funcionalidade será incluída na classe.

## Decidindo quando criar classes

Você pode criar uma classe para cada [controle](#) e cada [formulário](#) utilizado, mas essa não é a forma mais eficiente de criar seus aplicativos. Provavelmente, você terá várias classes que fazem a mesma coisa, mas que devem ser mantidas de forma separada.

## Encapsulando funcionalidades genéricas

Você pode criar uma classe de controle para funcionalidade genérica. Por exemplo, os [botões de comando](#) que permitem ao usuário mover o ponteiro de registro em uma tabela, um botão para

fechar um formulário e um botão de ajuda, podem ser salvos como classes e adicionados a formulários sempre que você quiser que os formulários tenham esta funcionalidade.

Você pode exibir [propriedades](#) e [métodos](#) em uma classe para que o usuário possa integrá-los ao [ambiente de dados](#) específico de um formulário ou conjunto de formulários.

## Fornecendo ao aplicativo aparência e comportamento consistentes

Você pode criar um [conjunto de formulários](#), um [formulário](#) e [classes de controle](#) com uma aparência específica para que todos os componentes do seu aplicativo tenham a mesma aparência. Por exemplo, você pode adicionar gráficos e padrões de cor específicos a uma classe de formulários e utilizar isso como um modelo para todos os formulários criados. Também é possível criar uma classe de [caixa de texto](#) com uma aparência específica, como um efeito de sombreado, e utilizar essa classe em todo o aplicativo sempre que quiser adicionar uma caixa de texto.

## Decidindo o tipo de classe a criar

O Visual FoxPro permite que você crie vários tipos de classes, cada qual com suas próprias características. Especifique o tipo de classe que deseja criar na [caixa de diálogo Nova Classe](#) ou na cláusula AS, no comando [CREATE CLASS](#).

## Classes principais do Visual FoxPro

Você pode criar subclasses a partir das classes principais do Visual FoxPro no [Criador de classes](#).

### Classes principais do Visual FoxPro

<a href="#">CheckBox</a>	<a href="#">EditBox</a>	<a href="#">ListBox</a>	<a href="#">Shape</a>
<a href="#">Column</a> <sup>1</sup>	<a href="#">Form</a>	<a href="#">OLEBoundControl</a>	<a href="#">Spinner</a>
<a href="#">CommandButton</a>	<a href="#">FormSet</a>	<a href="#">OLEContainerControl</a>	<a href="#">TextBox</a>
<a href="#">CommandGroup</a>	<a href="#">Grid</a>	<a href="#">OptionButton</a> <sup>1</sup>	<a href="#">Timer</a>
<a href="#">ComboBox</a>	<a href="#">Header</a> <sup>1</sup>	<a href="#">OptionGroup</a>	<a href="#">ToolBar</a>
<a href="#">Container</a>	<a href="#">Image</a>	<a href="#">Page</a> <sup>1</sup>	
<a href="#">Control</a>	<a href="#">Label</a>	<a href="#">PageFrame</a>	
<a href="#">Custom</a>	<a href="#">Line</a>	<a href="#">Separator</a>	

<sup>1</sup> Estas classes são parte integrante de um recipiente pai e não é possível criar subclasses para elas no [Criador de classes](#). Todas as classes principais do Visual FoxPro reconhecem o conjunto mínimo de eventos a seguir.

Evento	Descrição
<a href="#">Init</a>	Ocorre quando o objeto é criado.
<a href="#">Destroy</a>	Ocorre quando o objeto é liberado da memória.
<a href="#">Error</a>	Ocorre sempre que houver um erro nos procedimentos de evento ou de método da classe.

Todas as classes principais do Visual FoxPro apresentam o conjunto mínimo de propriedades a seguir.

Propriedade	Descrição
<a href="#">Class</a>	Qual é o tipo da classe.
<a href="#">BaseClass</a>	A classe principal da qual ela foi derivada, como Form, Commandbutton, Custom e assim por diante.
<a href="#">ClassLibrary</a>	A biblioteca de classes a que a classe pertence.
<a href="#">ParentClass</a>	A classe da qual a classe atual foi derivada. Se a classe tiver sido derivada diretamente de uma classe



principal do Visual FoxPro, a propriedade ParentClass será igual à propriedade BaseClass.

## Estendendo as classes principais do Visual FoxPro

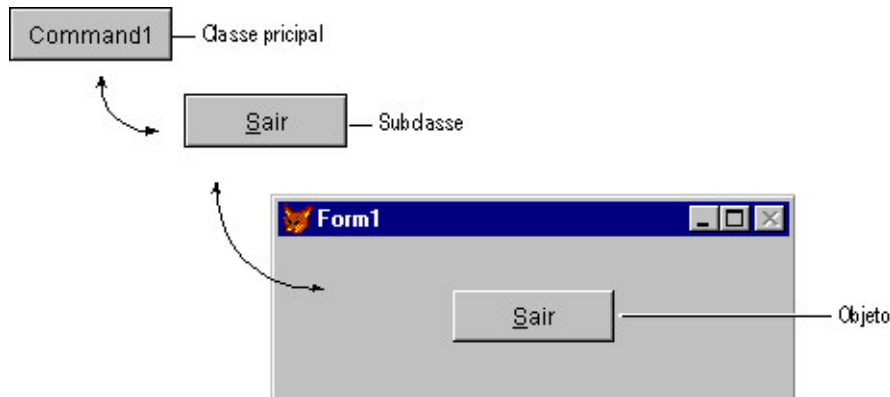
Essas classes podem ser divididas em subclasses para que você possa definir suas próprias propriedades de controle padrão. Por exemplo, se quiser que os nomes padrão dos controles adicionados aos formulários em seus aplicativos reflitam automaticamente suas convenções de nomenclatura, poderá criar classes com base nas classes principais do Visual FoxPro. Você pode criar classes de formulário com aparência ou comportamento personalizados para servir como modelos para todos os formulários criados.

As classes principais do Visual FoxPro também podem ser divididas em subclasses para criar controles com funcionalidade encapsulada. Se você quiser que um botão libere formulários quando for clicado, poderá criar uma classe com base na classe de botão de comando do Visual FoxPro, definir a propriedade Caption para "Sair" e incluir o seguinte comando no evento Click:

```
THISFORM.Release
```

Esse novo botão pode ser adicionado a qualquer formulário no seu aplicativo.

### Botão de comando personalizado adicionado a um formulário



## Criando controles com vários componentes

As subclasses não se limitam a classes principais. Você pode adicionar vários controles a uma única definição de classe recipiente. Muitas das classes no exemplo de biblioteca de classes do Visual FoxPro estão nesta categoria. Por exemplo, a classe VCR em BUTTONS.VCX, localizada em VFP\SAMPLES\CLASSES, contém quatro [botões de comando](#) para navegar os registros em uma tabela.

## Criando classes não-visuais

Uma classe com base na classe personalizada do Visual FoxPro não possui um elemento visual em tempo de execução. Você pode criar [métodos](#) e [propriedades](#) para sua classe personalizada, utilizando o ambiente do **Criador de classes**. Por exemplo, você pode criar uma classe personalizada denominada StrMethods e incluir uma série de métodos para manipular seqüências de caracteres. Essa classe pode ser adicionada a um formulário com uma [caixa de edição](#) e os métodos podem ser chamados conforme necessário. Se você tiver um método denominado WordCount, poderá chamá-lo quando necessário.

```
THISFORM.txtCount.Value = ;
THISFORM.StrMethods.WordCount(THISFORM.edtText.Value)
```

As classes não-visuais (como o controle personalizado e o controle de cronômetro) possuem uma representação visual somente na [hora da criação](#) no **Criador de formulários**. Defina a propriedade da Picture da classe personalizada como o arquivo .BMP que você deseja exibir no **Criador de formulários** quando a classe personalizada for adicionada a um formulário.



## Criando classes

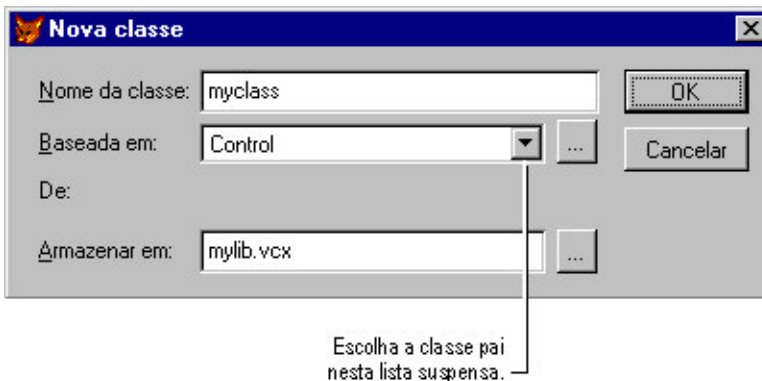
Você pode criar novas classes no **Criador de classes** e ver como cada objeto aparecerá para o usuário durante a sua criação.

### ► Para criar uma nova classe

- No **Gerenciador de projetos**, selecione a guia **Classes** e escolha **Novo**.
  - Ou –
- No menu **Arquivo**, escolha **Novo**, selecione **Classe** e, em seguida, **Novo Arquivo**.
  - Ou –
- Utilize o comando **CREATE CLASS**.

A caixa de diálogo **Nova classe** permite que você especifique como chamar a nova classe, a classe em que ela será baseada e a biblioteca onde armazená-la.

### Criando uma nova classe



## Modificando uma definição de classe

Depois de criar uma classe, você pode modificá-la. As alterações feitas em uma classe afetam todas as subclasses e todos os objetos baseados nesta classe. Você pode aprimorar uma classe ou corrigir um erro em uma classe e todas as subclasses e objetos baseados nessa classe herdarão a alteração.

### ► Para modificar uma classe no Gerenciador de projetos

- 1 Selecione a classe que deseja modificar.
- 2 Selecione **Modificar**.

O **Criador de classes** será aberto.

Também é possível modificar uma definição de classe visual com o comando **MODIFY CLASS**.

**Importante** Não altere a propriedade Name de uma classe se ela já estiver sendo utilizada em qualquer outro componente do aplicativo. Caso contrário, o Visual FoxPro não conseguirá localizar a classe quando for necessário.

## Dividindo uma definição de classe em subclasses

Para criar uma subclasse de uma classe definida pelo usuário, siga um dos dois procedimentos abaixo:

### ► Para criar uma subclasse de uma classe definida pelo usuário

- 1 Na caixa de diálogo **Nova classe** clique no botão de diálogo à direita da caixa **Baseada em**.
- 2 Na caixa de diálogo **Abrir**, selecione a classe em que deseja basear a nova classe.

– Ou –

- Utilize o comando **CREATE CLASS**.

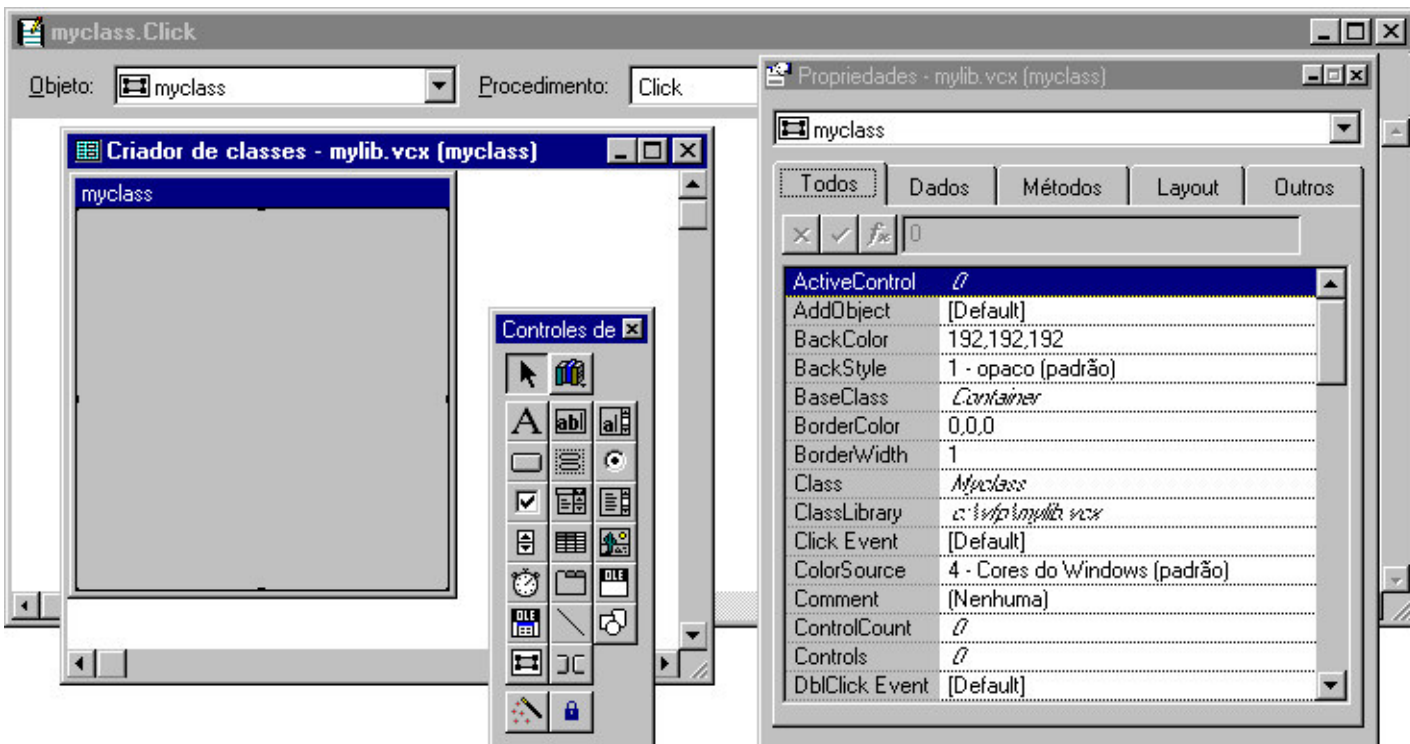
Por exemplo, para basear uma nova classe, x, na `classepai` em `MINHABIB.VCX`, utilize o código abaixo:

```
CREATE CLASS x OF y AS parentclass;  
FROM minhabib
```

## Utilizando o Criador de classes

Quando você especificar qual a classe em que a nova classe será baseada e a biblioteca que armazenará a classe, o **Criador de classes** será aberto.

### Criador de classes



O **Criador de classes** fornece a mesma interface que o **Criador de formulários**, permitindo que você veja e edite as propriedades da sua classe na **janela Propriedades**. Janelas de edição de código permitem que você escreva código a ser executado quando ocorrerem eventos ou quando métodos forem chamados.

### Adicionando objetos a um controle ou classe recipiente

Se você basear a nova classe em uma classe recipiente ou de controle, você poderá adicionar controles a ela da mesma maneira que adiciona controles no **Criador de formulários**: selecione um botão de controle na **Barra de ferramentas controles de formulário** e arraste para dimensioná-lo no **Criador de classes**.

Qualquer que seja o tipo de classe em que a nova classe esteja baseada, você poderá definir propriedades e escrever o código de método. Você poderá também criar novas propriedades e métodos para a classe.

### Adicionando propriedades e métodos a uma classe

Você pode adicionar quantas [propriedades](#) e [métodos](#) novos que desejar à nova classe. As propriedades têm um valor; os métodos têm um código procedural a ser executado quando você chamar o método.

## Criando novas propriedades e métodos

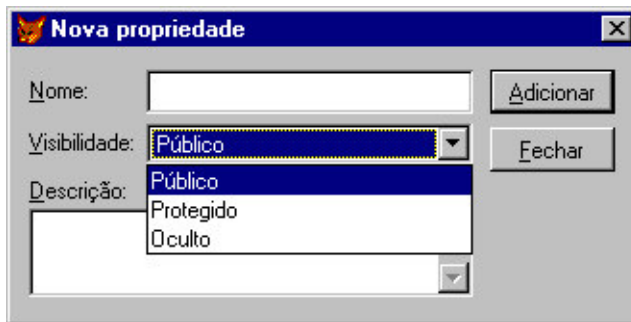
Quando você cria novas propriedades e métodos para as classes, o escopo das propriedades e métodos é o mesmo da classe e não dos componentes individuais da classe.

### ► Para adicionar uma nova propriedade a uma classe

- 1 No menu **Classe**, selecione **Nova propriedade**.
- 2 Na caixa de diálogo **Nova propriedade**, digite o nome da propriedade.
- 3 Especifique a visibilidade: **Público**, **Protegido** ou **Oculto**.

Uma propriedade Public pode ser acessada em qualquer parte do seu aplicativo. As propriedades Protected e Hidden e os métodos são abordados em “[Protegendo e ocultando membros da classe](#)” posteriormente neste capítulo.

#### Caixa de diálogo Nova propriedade



- 4 Selecione **Adicionar**.

Você pode também incluir uma descrição da propriedade que será exibida na parte inferior da janela **Propriedades** no **Criador de classes** e no **Criador de formulários** quando o controle for adicionado a um formulário.

**Solucionando Problemas** Ao adicionar uma propriedade a uma classe para que ela possa ser definida pelo usuário, o usuário pode digitar uma definição inválida para a propriedade, o que poderá gerar erros em tempo de execução. Você precisa documentar explicitamente as definições válidas para a propriedade. Se for possível definir a propriedade como 0, 1 ou 2, por exemplo, informe isso na caixa Descrição da **caixa de diálogo Nova propriedade**. Pode ser que você também queira verificar o valor da propriedade no código que faça referência a ela.

### ► Para criar uma propriedade de matriz

- Na caixa **Nome** da caixa de diálogo **Nova propriedade**, especifique o nome, o tamanho e as dimensões da matriz.

Por exemplo, para criar uma propriedade de matriz denominada `myarray` com dez linhas e duas colunas, digite o seguinte na caixa Nome:

```
myarray[10,2]
```

A propriedade de matriz é somente para leitura na [hora da criação](#) e é exibida somente na janela **Propriedades** em itálico. A propriedade de matriz pode ser administrada e redimensionada em [tempo de execução](#). Para obter um exemplo de como utilizar uma propriedade de matriz, consulte “Gerenciando múltiplas instâncias de um formulário” no capítulo 9, [Criando formulários](#).

### ► Para adicionar um novo método a uma classe

- 1 No menu **Classe**, selecione **Novo método**.
- 2 Na caixa de diálogo **Novo método**, digite o nome do novo método.

3 Especifique a visibilidade: **Público**, **Protegido** ou **Oculto**.

Você pode também incluir uma descrição do método.

## Protegendo e ocultando membros da classe

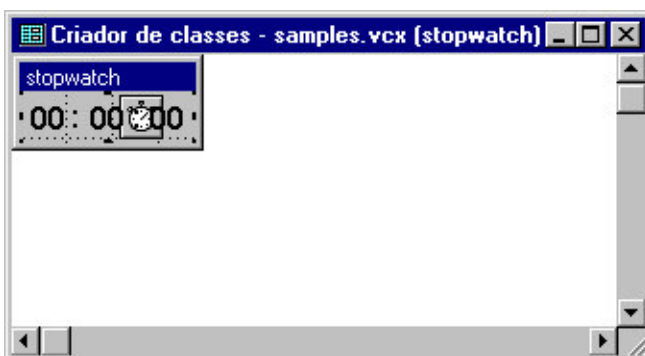
[Propriedades](#) e [métodos](#) em uma definição de classe são **Públicos** como padrão: código em outras classes ou procedimentos pode definir propriedades ou chamar os métodos. As propriedades e métodos designados como **Protegidos** podem ser acessados somente por outros métodos na definição de classe ou nas subclasses da classe. As propriedades e métodos designados como **Ocultos** podem ser acessados somente por outros membros na definição de classe. As subclasses da classe não podem “ver” ou fazer referência a membros ocultos.

Para garantir o funcionamento correto em algumas classes, é preciso evitar que os usuários utilizem a linguagem de programação para alterar as propriedades ou chamar o método de fora da classe.

O exemplo a seguir ilustra como utilizar propriedades e métodos protegidos em uma classe.

A classe Stopwatch incluída em SAMPLES.VCX em VFP\SAMPLES\CLASSES inclui um [cronômetro](#) e cinco [rótulos](#) para exibir o tempo decorrido:

### A classe Stopwatch em SAMPLES.VCX



A classe Stopwatch contém rótulos e um cronômetro.

### Definições de propriedade para a classe Stopwatch

Controle	Propriedade	Definição
lblSeconds	<a href="#">Caption</a>	00
lblColon1	Caption	:
lblMinutes	Caption	00
lblColon2	Caption	:
lblHours	Caption	00
tmrSWatch	<a href="#">Interval</a>	1000

Essa classe possui também três propriedades protegidas, nSec, nMin e nHour e um método protegido, UpdateDisplay. Os outros três métodos personalizados da classe, Start, Stop e Reset, não estão protegidos.

**Dica** Selecione Informações sobre a classe no menu **Classe** para exibir a visibilidade de todas as propriedades e métodos de uma classe.

As propriedades protegidas são utilizadas em cálculos internos no método UpdateDisplay e no evento Timer. O método UpdateDisplay define as legendas dos rótulos para refletir o tempo decorrido.

### O Método UpdateDisplay

Código	Comentários
--------	-------------

```
cSecDisplay = ALLTRIM(STR(THIS.nSec))
cMinDisplay = ALLTRIM(STR(THIS.nMin))
cHourDisplay = ALLTRIM(STR(THIS.nHour))
```

```
THIS.lblSeconds.Caption = ;
    IIF(THIS.nSec < 10, ;
        "0", "") + cSecDisplay
THIS.lblMinutes.Caption = ;
    IIF(THIS.nMin < 10, ;
        "0", "") + cMinDisplay
THIS.lblHours.Caption = ;
    IIF(THIS.nHour < 10, ;
        "0", "") + cHourDisplay
```

Converte as propriedades numéricas em tipo Caractere para exibição nas legendas dos rótulos.

Define as legendas dos rótulos, mantendo o 0 à esquerda se o valor da propriedade numérica for menor que 10.

A tabela a seguir lista o código no evento `tmrSWatch.Timer`:

### O Evento Timer

Código	Comentários
<pre>THIS.Parent.nSec = THIS.Parent.nSec + 1  IF THIS.Parent.nSec = 60     THIS.Parent.nSec = 0     THIS.Parent.nMin = ;     THIS.Parent.nMin + 1 ENDIF  IF THIS.Parent.nMin = 60     THIS.Parent.nMin = 0     THIS.Parent.nHour = ;     THIS.Parent.nHour + 1 ENDIF THIS.Parent.UpdateDisplay</pre>	<p>Incrementa a propriedade <code>nSec</code> sempre que o evento Timer é disparado: a cada segundo.</p> <p>Se <code>nSec</code> atingir 60, redefine para 0 e incrementa a propriedade <code>nMin</code>.</p> <p>Se <code>nMin</code> atingir 60, redefine para 0 e incrementa a propriedade <code>nHour</code>.</p> <p>Chama o método <code>UpdateDisplay</code> quando os novos valores da propriedade são definidos.</p>

A classe Stopwatch possui três métodos que não estão protegidos: Start, Stop e Reset. O usuário poderá chamar estes métodos diretamente do controle Stopwatch.

O método Start contém a linha de código a seguir:

```
THIS.tmrSWatch.Enabled = .T.
```

O método Stop contém a linha de código a seguir:

```
THIS.tmrSWatch.Enabled = .F.
```

O método Reset define as propriedades protegidas como 0 e chama o método protegido:

```
THIS.nSec = 0
THIS.nMin = 0
THIS.nHour = 0
THIS.UpdateDisplay
```

O usuário não pode definir estas propriedades diretamente ou chamar este método, mas o código no método Reset pode.

### Especificando o valor padrão de uma propriedade

Quando você cria uma nova propriedade, a definição padrão é falso (.F.). Para especificar uma definição padrão diferente para uma propriedade, utilize a [janela Propriedades](#). Na guia **Outros**, clique na propriedade e defina-a com o valor desejado. Esta será a definição inicial da propriedade quando a classe for adicionada a um [formulário](#) ou [conjunto de formulários](#).

Também é possível definir qualquer propriedade das classes principais no [Criador de classes](#). Quando um objeto baseado na classe é adicionado ao formulário, ele reflete as suas definições de propriedades e não as definições de propriedades das classes principais do Visual FoxPro.

**Dica** Se você quiser transformar a definição padrão de uma propriedade em uma sequência vazia, selecione a definição na caixa de edição de propriedade e pressione a tecla BACKSPACE.

## Especificando a aparência na hora de criação

Você pode especificar o ícone da barra de ferramentas e o ícone do recipiente para a sua classe na caixa de diálogo [Informações sobre a classe](#).

### ► Para definir um ícone da barra de ferramentas para uma classe

- 1 No [Criador de classes](#), selecione **Informações sobre a classe** no menu **Classe**.
- 2 Na caixa de diálogo [Informações sobre a classe](#), digite o nome e o caminho do arquivo .BMP na caixa **Ícone da barra de ferramentas**.

**Dica** O bitmap (arquivo .BMP) de ícone da barra de ferramentas deve ter 15 por 16 [pixels](#). Se a figura for maior ou menor, será dimensionada para 15 por 16 pixels e poderá não ficar como você gostaria.

O ícone da barra de ferramentas que você especificar será exibido na [Barra de ferramentas controles de formulário](#) quando você ocupar a barra de ferramentas com as classes da sua [biblioteca de classes](#).

Você pode também especificar o ícone a ser exibido para a classe no [Gerenciador de projetos](#) e no [Pesquisador de classes](#) definindo o ícone do recipiente.

### ► Para definir um ícone do recipiente para uma classe

- 1 No [Criador de classes](#), selecione **Informações sobre a classe** no menu **Classe**.
- 2 Na caixa **Ícone do recipiente**, digite o nome e o caminho do arquivo .BMP a ser exibido no botão da [Barra de ferramentas controles de formulário](#).

## Utilizando arquivos da bibliotecas de classes

Cada classe criada visualmente é armazenada em uma biblioteca de classes com a extensão de arquivo .VCX.

### Criando uma biblioteca de classes

Para criar uma biblioteca de classes, siga um dos procedimentos abaixo.

#### ► Para criar uma biblioteca de classes

- Ao criar uma classe, especifique o arquivo da nova biblioteca de classes na caixa **Armazenar em** da caixa de diálogo [Nova classe](#).

– Ou –

- Utilize o comando [CREATE CLASS](#), especificando o nome da nova biblioteca de classes.  
Por exemplo, a instrução abaixo cria uma nova classe denominada `myclass` e uma nova biblioteca de classes denominada `new_lib`:

```
CREATE CLASS myclass OF new_lib AS CUSTOM
```

– Ou –

- Utilize o comando [CREATE CLASSLIB](#).

Por exemplo, digite o comando abaixo na [janela Comando](#) para criar uma biblioteca de classes denominada `new_lib`:

```
CREATE CLASSLIB new_lib
```

### Copiando e removendo classes de bibliotecas de classes

Depois que você adicionar uma biblioteca de classes a um projeto, será fácil copiar classes de uma biblioteca para outra ou simplesmente remover classes das bibliotecas.

#### ► Para copiar uma classe de uma biblioteca para outra

- 1 Certifique-se de que as duas bibliotecas estejam em um projeto (não necessariamente no mesmo projeto).

- 2 No **Gerenciador de projetos**, selecione a guia **Classes**.
- 3 Clique no sinal de adição (+) à esquerda da biblioteca de classes em que a classe se encontra no momento.
- 4 Arraste a classe da biblioteca anterior e solte-a na nova biblioteca.

**Dica** Para fins de conveniência e velocidade, talvez você queira manter uma classe e todas as subclasses nela baseadas, em uma mesma biblioteca de classes. Caso uma classe contenha elementos de várias bibliotecas de classes diferentes, todas estas bibliotecas deverão estar abertas; assim, levará um pouco mais de tempo para carregar inicialmente a classe em [tempo de execução](#) e na [hora da criação](#).

► **Para remover uma classe de uma biblioteca**

- Selecione a classe no **Gerenciador de projetos** e escolha **Remover**.
  - Ou –
- Utilize o comando **REMOVE CLASS**.

Para alterar o nome de uma classe em uma biblioteca de classes, utilize **RENAME CLASS**. Lembre-se, porém, de que ao alterar o nome de uma classe, os formulários que contêm a classe e subclasses em outros arquivos .VCX continuarão fazendo referência ao nome anterior e não funcionarão corretamente.

O Visual FoxPro inclui um **Pesquisador de classes** para facilitar o uso e gerenciamento de classes e bibliotecas de classes. Para obter maiores informações, consulte a janela **Pesquisador de classes**.

## Adicionando classes a formulários

Você pode arrastar uma classe do **Gerenciador de projetos** para o **Criador de formulários** ou para o **Criador de classes**. Você pode também registrar as suas classes para que elas possam ser exibidas diretamente na **Barra de ferramentas controles de formulário** no **Criador de classes** ou no **Criador de formulários** e para possam ser adicionadas a recipientes da mesma maneira que os controles padrão são adicionados.

► **Para registrar uma biblioteca de classes**

- 1 No menu **Ferramentas**, selecione **Opções**.
- 2 Na caixa de diálogo **Opções**, selecione a guia **Controles**.
- 3 Selecione **Bibliotecas de classes visuais** e escolha **Adicionar**.
- 4 Na caixa de diálogo **Abrir**, selecione uma biblioteca de classes para adicionar ao registro e selecione **Abrir**.
- 5 Selecione **Definir como padrão** para que a biblioteca de classes fique disponível na **Barra de ferramentas controles de formulário** em sessões futuras do Visual FoxPro.

Você pode também adicionar a sua biblioteca de classes à **Barra de ferramentas controles de formulário**, selecionando **Adicionar** no submenu do botão **Visualizar classes**. Para fazer com que estas classes fiquem disponíveis na **Barra de ferramentas controles de formulário** em sessões futuras do Visual FoxPro, você deve definir o padrão na [caixa de diálogo Opções](#).

## Alterando definições de propriedades padrão

Ao adicionar objetos baseados em uma classe definida pelo usuário a um [formulário](#), você pode alterar as definições de todas as [propriedades](#) da classe que não estejam protegidas, substituindo as definições padrão. Se, mais tarde, você alterar as propriedades da classe no **Criador de classes**, as definições do objeto no formulário não serão afetadas. Se você não tiver alterado a definição de uma propriedade no formulário e alterar a definição da propriedade na classe, a alteração também ocorrerá no objeto.



Por exemplo, o usuário pode adicionar um objeto baseado na sua classe a um formulário e alterar a propriedade BackColor de branco para vermelho. Se você alterar a propriedade BackColor da classe para verde, o objeto no formulário do usuário ainda terá o vermelho como cor de segundo plano. Se, por outro lado, o usuário não alterar a propriedade BackColor do objeto e você alterar a cor de segundo plano da classe para verde, a propriedade BackColor do objeto no formulário herdar a alteração e também será verde.

## Chamando o código de método da classe pai

Um objeto ou classe com base em outra classe herda a funcionalidade do original automaticamente. No entanto, você poderá substituir facilmente o código de método herdado. Por exemplo, você pode escrever código novo para o [evento Click](#) de uma classe depois de dividi-lo em subclasse ou depois de adicionar um objeto com base na classe a um recipiente. Em ambos os casos, o novo código será executado em [tempo de execução](#); o código original não será executado.

Mais freqüentemente, no entanto, você vai querer adicionar funcionalidade à nova classe ou objeto e manter a funcionalidade original. Na verdade, uma das principais decisões a ser tomada na programação orientada a objetos, é qual a funcionalidade a ser incluída no nível de classe, nível de subclasse e nível de objeto. Você pode otimizar a criação da sua classe, utilizando a função [DODEFAULT\( \)](#) ou o [operador de resolução de escopo \(::\)](#) para adicionar código em níveis diferentes na hierarquia de classes ou recipientes.

## Adicionando funcionalidade a subclasses

Você pode chamar o código de classe pai de uma subclasse, utilizando a função [DODEFAULT\( \)](#).

Por exemplo, `cmdOK` é um botão de comando armazenado na classe `BUTTONS.VCX`, localizado em `VFP\SAMPLES\CLASSES`. O código associado ao evento Click de `cmdOk` libera o formulário no qual está o botão. `cmdCancel` é uma subclasse de `cmdOk` na mesma biblioteca de classe. Para adicionar funcionalidade a `cmdCancel` para desconsiderar alterações, por exemplo, você pode adicionar o código a seguir ao evento Click:

```
IF USED( ) AND CURSORGETPROP("Buffering") != 1
    TABLEREVERT(.T.)
ENDIF
DODEFAULT( )
```

Devido as alterações serem gravadas em uma tabela de buffer como padrão quando a tabela é fechada, você não precisa adicionar o código [TABLEUPDATE\( \)](#) a `cmdOk`. O código adicional em `cmdCancel` reverte as alterações para a tabela antes de chamar o código em `cmdOk`, `ParentClass`, para liberar o formulário.

## Hierarquias de classes e recipientes

A hierarquia de classe e recipiente são duas entidades separadas. O Visual FoxPro procura código de evento na hierarquia de classe, enquanto os objetos são referenciados na hierarquia de recipiente. A seção a seguir, “Fazendo referência a objetos na hierarquia de recipiente”, discute hierarquias de recipiente. As hierarquias de classe são explicadas posteriormente neste capítulo, na seção [Chamando o código de evento através da hierarquia de classes](#).

## Fazendo referência a objetos na hierarquia de recipientes

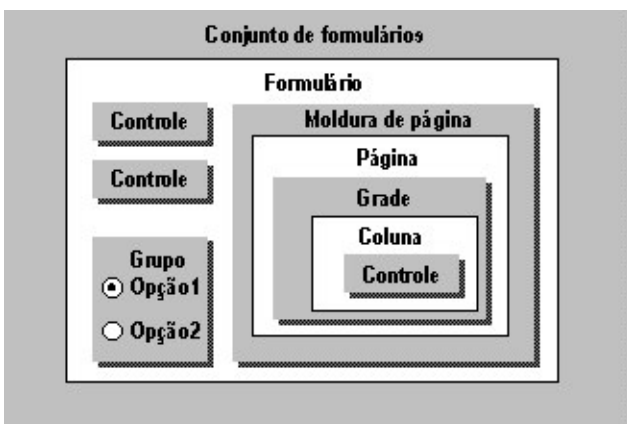
Para manipular um objeto, você precisa identificá-lo em relação à hierarquia de recipiente. Por exemplo, para manipular um [controle](#) em um [formulário](#) em um [conjunto de formulários](#), você precisa fazer referência ao conjunto de formulários, ao formulário e, em seguida ao controle.

Você pode comparar o processo de fazer referência a um objeto em sua hierarquia de recipiente e o de fornecer ao Visual FoxPro um endereço para o objeto. Quando você descreve a localização de uma casa para alguém fora da sua estrutura de referência imediata, você precisa indicar o país, o estado ou a região, a cidade, a rua ou apenas o número da rua da casa, dependendo da distância.

Caso contrário, pode haver confusão.

A ilustração a seguir mostra uma possível situação de aninhamento de recipientes.

### Recipientes aninhados



Para desativar o controle na coluna de grade, você precisa fornecer o seguinte endereço:

```
Formset.Form.PageFrame.Page.;  
Grid.Column.Control.Enabled = .F.
```

A [propriedade ActiveForm](#) do objeto do aplicativo (\_VFP) permite que você manipule o formulário ativo mesmo que você não saiba seu nome. Por exemplo, a linha de código a seguir altera a cor de fundo do formulário ativo, independentemente do conjunto de formulários ao qual pertence.

```
_VFP.ActiveForm.BackColor = RGB(255,255,255)
```

Da mesma forma, a [propriedade ActiveControl](#) permite que você manipule o controle ativo no formulário ativo. Por exemplo, a expressão a seguir, digitada na [janela Observar](#) exibe o nome do controle ativo em um formulário à medida que você escolhe de forma interativa os vários controles:

```
_VFP.ActiveForm.ActiveControl.Name
```

### Referência relativa

Quando você faz referência a objetos a partir da hierarquia de recipiente (por exemplo, no evento Click de um botão de comando em um formulário de um conjunto de formulários), pode utilizar alguns atalhos para identificar o objeto que você deseja manipular. A tabela a seguir lista as propriedades ou palavras-chave que facilitam a referência a um objeto a partir da hierarquia de objeto.

Propriedade ou palavra-chave	Referência
<a href="#">Parent</a>	O recipiente imediato do objeto
<a href="#">THIS</a>	O objeto
<a href="#">THISFORM</a>	O formulário que contém o objeto
<a href="#">THISFORMSET</a>	O conjunto de formulários que contém o objeto

**Observação** Você pode utilizar THIS, THISFORM e THISFORMSET somente no código de método ou evento.

A tabela a seguir fornece exemplos de como utilizar THISFORMSET, THISFORM, THIS e Parent para definir propriedades de objeto:

Comando	Onde incluir o comando
<code>THISFORMSET.frm1.cmd1.Caption = "OK"</code>	No código de evento ou método de qualquer controle em qualquer

```
THISFORM.cmd1.Caption = "OK"
```

```
THIS.Caption = "OK"
```

```
THIS.Parent.BackColor = RGB(192,0,0)
```

formulário no conjunto de formulários.

No código de evento ou método de qualquer controle no mesmo formulário onde `cmd1` estiver.

No código de evento ou método do controle cuja legenda você deseja alterar.

No código de evento ou método de um controle em um formulário. O comando altera a cor de fundo do formulário para vermelho escuro.

## Definindo propriedades

Você pode definir as propriedades de um objeto em [tempo de execução](#) ou na [hora da criação](#).

### ► Para definir uma propriedade

- Utilize esta sintaxe:

*Recipiente.Objeto.Propriedade = Valor*

Por exemplo, as instruções a seguir definem várias propriedades de uma [caixa de texto](#) denominada `txtDate` em um formulário denominado `frmPhoneLog`:

```
frmPhoneLog.txtDate.Value = DATE( ) && Exibe a data atual
frmPhoneLog.txtDate.Enabled = .T. && O controle está ativado
frmPhoneLog.txtDate.ForeColor = RGB(0,0,0) && texto preto
frmPhoneLog.txtDate.BackColor = RGB(192,192,192) && fundo cinza
```

Para as definições de propriedades dos exemplos anteriores, `frmPhoneLog` é o objeto recipiente de nível mais elevado. Se `frmPhoneLog` estivesse contido em um [conjunto de formulários](#), você precisaria também incluir um conjunto de formulários no caminho pai:

```
frsContacts.frmPhoneLog.txtDate.Value = DATE( )
```

## Definindo várias propriedades

A estrutura `WITH ... ENDWITH` simplifica a definição de várias propriedades. Por exemplo, para definir várias propriedades de uma coluna em uma [grade](#) em um [formulário](#) de um [conjunto de formulários](#), você pode utilizar a sintaxe a seguir:

```
WITH THISFORMSET.frmForm1.grdGrid1.grcColumn1
.Width = 5
.Resizable = .F.
.ForeColor = RGB(0,0,0)
.BackColor = RGB(255,255,255)
.SelectOnEntry = .T.
ENDWITH
```

## Chamando métodos

Uma vez criado o objeto, você pode chamar os [métodos](#) deste objeto de qualquer lugar do seu aplicativo.

### ► Para chamar um método

- Utilize esta sintaxe:

*Pai.Objeto.Método*

As instruções a seguir chamam os métodos para exibir um [formulário](#) e definir o foco para uma [caixa de texto](#):

```
frsFormSet.frmForm1.Show
```

```
frsFormSet.frmForm1.txtGetText1.SetFocus
```

Os métodos que retornam valores e são utilizados em [expressões](#), devem terminar em parênteses de abertura e fechamento. Por exemplo, a instrução a seguir define a [legenda](#) de um [formulário](#) para o valor retornado do método definido pelo usuário `GetNewCaption`:

```
Form1.Caption = Form1.GetNewCaption( )
```

**Observação** Os [parâmetros](#) passados para os métodos devem ser incluídos entre parênteses depois do nome do método; por exemplo, `Form1.Show(nStyle)`. passa `nStyle` para o código de método `Show` do `Formulario1`.

## Respondendo a eventos

O código que você inclui em um procedimento de evento é executado quando o [evento](#) ocorre. Por exemplo, o código incluído no procedimento do evento `Click` de um [botão de comando](#) é executado quando o usuário clica sobre o botão de comando.

Por meio da linguagem de programação, você pode causar os eventos [Click](#), [DbClick](#), [MouseMove](#) e [DragDrop](#) com o comando [MOUSE](#), ou utilizar o comando [ERROR](#) para gerar eventos `Error` e o comando [KEYBOARD](#) para gerar eventos `KeyPress`. Não é possível causar qualquer outro evento por meio da linguagem de programação, mas o procedimento associado ao evento pode ser chamado. Por exemplo, a instrução a seguir causa a execução do código no [evento Activate](#) de `frmPhoneLog`, mas não ativa o formulário:

```
frmPhoneLog.Activate
```

Se você quiser ativar o formulário, utilize o [método Show](#) do formulário. Ao chamar o método `Show`, o formulário é exibido e ativado e, neste ponto, o código no evento `Activate` também será executado:

```
frmPhoneLog.Show
```

## Definindo classes por programação

Você pode definir classes visualmente no [Criador de classes](#) e no [Criador de formulários](#) ou utilizando a linguagem de programação nos arquivos `.PRG`. Esta seção descreve como escrever definições de classes. Para obter informações sobre comandos, funções e operadores específicos, consulte a [Ajuda](#). Para obter maiores informações sobre formulários, consulte o capítulo 9, [Criando formulários](#).

Em um arquivo de programa, você pode ter um código de programa antes das definições de classes, mas não depois delas, da mesma maneira que um código de programa não pode vir depois dos procedimentos em um programa. A shell básica para a criação de uma classe tem a sintaxe a seguir:

```
DEFINE CLASS NomeClasse1 AS cClassePrincipal1
    [[PROTECTED NomePropriedade1, NomePropriedade2...]
    NomePropriedade = eExpressão...]
    [[HIDDEN NomePropriedade1, NomePropriedade2...]
    NomePropriedade = eExpressão...]
    [ADD OBJECT [PROTECTED] NomeObjeto AS NomeClasse[NOINIT]
    [WITH cListaPropriedades]]
    [[PROTECTED| HIDDEN] FUNCTION | PROCEDURE Nome
    cInstruções
    [ENDFUNC | ENDPROC]]
ENDDFINE
```

## Protegendo e ocultando membros da classe

Você pode proteger ou ocultar [propriedades](#) e [métodos](#) em uma definição de classe com as palavras-chave `PROTECTED` e `HIDDEN` do comando [DEFINE CLASS](#).

Por exemplo, se você criar uma classe contendo informações sobre os funcionários e não quiser que os usuários alterem as datas de contratação, você poderá proteger a propriedade `HireDate`. Se os usuários precisarem saber quando um funcionário foi contratado, você poderá incluir um método

para retornar a data de contratação.

```
DEFINE CLASS employee AS CUSTOM
    PROTECTED HireDate
        First_Name = ""
        Last_Name = ""
        Address = ""
        HireDate = {}/{}

PROCEDURE GetHireDate
    RETURN This.HireDate
ENDPROC
ENDDEFINE
```

## Criando objetos a partir de classes

Depois de salvar uma classe visual, você poderá criar um objeto baseado nesta classe com a função [CREATEOBJECT\( \)](#). O exemplo a seguir demonstra como executar um formulário salvo como uma definição de classe no arquivo da biblioteca de classes FORMS.VCX:

### Criando e exibindo um objeto Form cuja classe foi criada no Criador de formulários

Código	Comentários
<pre>SET CLASSLIB TO Forms ADDITIVE</pre>	Define a biblioteca de classes como o arquivo .VCX em que a definição do formulário foi salva. A palavra-chave ADDITIVE impede que este comando feche qualquer outra biblioteca de classes que por acaso esteja aberta.
<pre>frmTest = CREATEOBJECT("TestForm")</pre>	Este código considera que o nome da classe de formulário salva na biblioteca de classes é TestForm.
<pre>frmTest.Show</pre>	Exibe o formulário.

## Adicionando objetos a um recipiente

Você pode utilizar a cláusula ADD OBJECT no comando [DEFINE CLASS](#) ou o [método AddObject](#) para adicionar objetos a um recipiente.

Por exemplo, a definição de classe a seguir é baseada em um formulário. O comando ADD OBJECT adiciona dois botões de comando ao formulário:

```
DEFINE CLASS myform AS FORM
    ADD OBJECT cmdOK AS COMMANDBUTTON
    ADD OBJECT PROTECTED cmdCancel AS COMMANDBUTTON
ENDDEFINE
```

Utilize o método AddObject para adicionar objetos a um recipiente depois que o objeto recipiente tiver sido criado. Por exemplo, as linhas de código a seguir criam um objeto formulário e adicionam dois botões de comando a ele:

```
frmMessage = CREATEOBJECT("FORM")
frmMessage.AddObject("txt1", "TEXTBOX")
frmMessage.AddObject("txt2", "TEXTBOX")
```

Você pode também utilizar o método AddObject no código de método de uma classe. Por exemplo, a definição de classe a seguir utiliza AddObject no código associado ao [evento Init](#) para adicionar um [controle](#) a uma coluna de grade.

```
DEFINE CLASS mygrid AS GRID
    ColumnCount = 3
    PROCEDURE Init
        THIS.Column2.AddObject("cboClient", "COMBOBOX")
        THIS.Column2.CurrentControl = "cboClient"
    ENDPROC
ENDPROC
```

## Adicionando e criando classes em código de método

Através da linguagem de programação, você pode adicionar [objetos](#) a um recipiente com o método AddObject. Também é possível criar objetos com a função [CREATEOBJECT\( \)](#) no método Load, Init ou em qualquer outro método da classe.

Quando um objeto é adicionado com o método AddObject, ele torna-se membro do recipiente. A [propriedade Parent](#) do objeto adicionado refere-se ao recipiente. Quando um objeto baseado na classe recipiente ou de controle é liberado da memória, o objeto adicionado também é liberado.

Quando um objeto é criado com a função [CREATEOBJECT\( \)](#), o seu escopo é o mesmo de uma propriedade da classe ou uma [variável](#) do método que chama esta função. A propriedade Parent do objeto é indefinida.

## Atribuindo códigos de métodos e eventos

Além de escrever códigos para os [métodos](#) e [eventos](#) de um [objeto](#), você pode estender o conjunto de métodos em subclasses das classes principais do Visual FoxPro. Aqui estão as regras para escrever códigos de eventos e de métodos:

- O conjunto de eventos para as classes principais do Visual FoxPro é fixo e não pode ser estendido.
- Cada classe reconhece um conjunto de eventos padrão fixos, cujo conjunto mínimo inclui os eventos [Init](#), [Destroy](#) e [Error](#).
- Quando você criar um [método](#) em uma definição de classe com o mesmo nome de um [evento](#) que a [classe](#) possa reconhecer, o código no método será executado quando o evento ocorrer.
- Você pode adicionar métodos às suas classes criando um [procedimento](#) ou [função](#) na definição de classe.

## Chamando o código de evento através da hierarquia de classes

Quando você criar uma [classe](#), ela herdará todas as [propriedades](#), [métodos](#) e [eventos](#) da classe pai automaticamente. Se um código for escrito para um evento na classe pai, ele será executado quando o evento ocorrer com respeito a um objeto baseado na subclasse. No entanto, você pode sobrescrever o código da classe pai escrevendo um código para o evento na subclasse.

Para chamar explicitamente o código de evento em uma classe pai quando a subclasse tiver um código escrito para o mesmo evento, utilize a função [DODEFAULT\( \)](#).

Por exemplo, você pode ter uma classe denominada cmdBottom baseada na classe principal de botão de comando que tem o código a seguir no [evento Click](#):

```
GO BOTTOM  
THISFORM.Refresh
```

Ao adicionar um [objeto](#) baseado nesta classe a um [formulário](#), denominado, por exemplo, cmdBottom1, você pode decidir que também queira exibir uma mensagem para informar o usuário de que o ponteiro do registro está no final da tabela. Você pode adicionar o código a seguir ao evento Click do objeto para exibir a mensagem:

```
WAIT WINDOW " No Final da Tabela " TIMEOUT 1
```

Contudo, quando você executar o formulário, a mensagem será exibida, mas o ponteiro do registro não se moverá porque o código no evento Click da classe pai nunca será executado. Para garantir que o código no evento Click da classe pai seja executado, inclua as linhas de código a seguir no procedimento do evento Click do objeto:

```
DODEFAULT( )  
WAIT WINDOW " No Final da Tabela " TIMEOUT 1
```

**Observação** Você pode utilizar a função [AClass\( \)](#) para determinar todas as classes na hierarquia de classes de um objeto.

## Impedindo a execução do código da classe principal

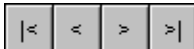
Talvez você queira evitar o funcionamento padrão da classe principal em um [evento](#) ou [método](#). Isto pode ser feito incluindo-se a palavra-chave NODEFAULT no código de método escrito. Por exemplo, o programa a seguir utiliza a palavra-chave NODEFAULT no [evento KeyPress](#) de uma [caixa de texto](#) para evitar que os caracteres digitados sejam exibidos na caixa de texto:

```
frmKeyExample = CREATEOBJECT("test")
frmKeyExample.Show
READ EVENTS
DEFINE CLASS test AS FORM
    ADD OBJECT text1 AS TEXTBOX
    PROCEDURE text1.KeyPress
        PARAMETERS nKeyCode, nShiftAltCtrl
        NODEFAULT
        IF BETWEEN(nKeyCode, 65, 122) && entre 'A' e 'z'
            This.Value = ALLTRIM(This.Value) + "*"
            ACTIVATE SCREEN      && envia a saída para a janela principal do Visual FoxPro
            ?? CHR(nKeyCode)
        ENDIF
    ENDPROC
    PROCEDURE Destroy
        CLEAR EVENTS
    ENDPROC
ENDDEFINE
```

## Criando um conjunto de botões de navegação na tabela

Um recurso comum de vários aplicativos é uma série de botões de navegação que permitem aos usuários movimentarem-se em uma tabela. Isto inclui, tipicamente, botões para movimentar o ponteiro do registro para o registro seguinte ou anterior, bem como para o primeiro ou o último registro na tabela.

### Botões de navegação na tabela



### Criando os botões de navegação

Os botões terão algumas características e funcionalidades em comum; portanto, é recomendável criar uma classe de botões de navegação. Assim, os botões individuais poderão facilmente derivar desta aparência e funcionalidade comuns. Essa classe pai é Navbutton, definida posteriormente nesta seção.

Uma vez definida a classe pai, as subclasses a seguir definirão a funcionalidade e a aparência específica de cada um dos quatro botões de navegação: navTop, navPrior, navNext, navBottom .

Finalmente, uma [classe Container](#) vcr, será criada e cada botão de navegação será adicionado à classe recipiente. O recipiente poderá ser adicionado a um [formulário](#) ou a uma [barra de ferramentas](#) para conferir funcionalidade à navegação na tabela.

### Definição da classe NAVBUTTON

Para criar a classe Navbutton, salve as seis definições de classe a seguir (Navbutton, navTop, navBottom, navPrior, navNext e vcr) em um arquivo de programa como NAVCLASS.PRG.

### Definição da classe commandbutton de navegação genérica

Código	Comentários
DEFINE CLASS Navbutton AS COMMANDBUTTON  Height = 25	Define a classe pai dos botões de navegação.



<pre>Width = 25 TableAlias = ""</pre>	<p>Inclui uma <a href="#">propriedade</a> personalizada, TableAlias, para armazenar o nome do <a href="#">alias</a> no qual navegar.</p>
<pre>PROCEDURE Click   IF NOT EMPTY(This.TableAlias)     SELECT (This.TableAlias)   ENDIF ENDPROC</pre>	<p>Se TableAlias tiver sido definida, este procedimento de classe pai selecionará o alias antes que o código de navegação real nas subclasses, seja executado. De outro modo, assumirá que o usuário quer navegar pela tabela na área de trabalho selecionada atualmente.</p>
<pre>PROCEDURE RefreshForm   _SCREEN.ActiveForm.Refresh ENDPROC</pre>	<p>O uso de _SCREEN.ActiveForm.Refresh em vez de THISFORM.Refresh permite que você adicione a classe a um <a href="#">formulário</a> ou uma <a href="#">barra de ferramentas</a> e que ela funcione igualmente bem.</p>
<pre>ENDDEFINE</pre>	<p>Encerra a definição de classe.</p>

Todos os botões específicos de navegação estão baseados na classe Navbutton. O código a seguir define o botão **Superior** para o conjunto de botões de navegação. Os três botões restantes são definidos na tabela a seguir. As quatro definições de classe são similares, de modo que somente a primeira apresenta comentários extensivos.

#### Definição da classe do botão de navegação superior

Código	Comentários
<pre>DEFINE CLASS navTop AS Navbutton   Caption = "&lt;" PROCEDURE Click    DODEFAULT( )    GO TOP    THIS.RefreshForm  ENDPROC</pre>	<p>Define a classe do botão de navegação <b>Superior</b> e a <a href="#">propriedade Caption</a>.</p> <p>Cria o código de método a ser executado quando ocorrer o <a href="#">evento Click</a> para o <a href="#">controle</a>. Chama o código do evento Click na classe pai, Navbutton, para que o <a href="#">alias</a> apropriado possa ser selecionado se a propriedade TableAlias tiver sido definida.</p> <p>Inclui o código para colocar o ponteiro do registro no primeiro registro da tabela: GO TOP.</p> <p>Chama o método RefreshForm na classe pai. Não é necessário utilizar o <a href="#">operador de resolução de escopo (::)</a> neste caso porque não há nenhum <a href="#">método</a> na subclasse com o mesmo nome do método na classe pai. Por outro lado, a classe pai e a subclasse têm um código de método para o evento Click.</p> <p>Encerra o procedimento Click.</p>
<pre>ENDDEFINE</pre>	<p>Encerra a definição de classe.</p>

Os outros botões de navegação possuem definições de classe similares.

#### Definição das outras classes de botão de navegação

Código	Comentários
<pre>DEFINE CLASS navNext AS Navbutton   Caption = "&gt;" PROCEDURE Click   DODEFAULT( )</pre>	<p>Define a classe do botão de navegação <b>Próximo</b> e a <a href="#">propriedade Caption</a>.</p>

<pre> SKIP 1 IF EOF( )     GO BOTTOM ENDIF THIS.RefreshForm ENDPROC ENDDEFINE </pre>	<p>Inclui o código para colocar o ponteiro do registro no próximo registro da tabela.</p>
<pre> DEFINE CLASS navPrior AS Navbutton     Caption = "&lt;" PROCEDURE Click     DODEFAULT( )     SKIP -1     IF BOF( )         GO TOP     ENDIF     THIS.RefreshForm ENDPROC ENDDEFINE </pre>	<p>Encerra a definição de classe.</p> <hr/> <p>Define a classe do botão de navegação <b>Anterior</b> e a propriedade Caption.</p>
<pre> DEFINE CLASS navBottom AS Navbutton     Caption = "&gt; " PROCEDURE Click     DODEFAULT( )     GO BOTTOM     THIS.RefreshForm ENDPROC ENDDEFINE </pre>	<p>Inclui o código para colocar o ponteiro do registro no registro anterior da tabela.</p>
<pre> DEFINE CLASS navBottom AS Navbutton     Caption = "&gt; " PROCEDURE Click     DODEFAULT( )     GO BOTTOM     THIS.RefreshForm ENDPROC ENDDEFINE </pre>	<p>Encerra a definição de classe.</p> <hr/> <p>Define a classe do botão de navegação <b>Inferior</b> e a propriedade Caption.</p>
<pre> PROCEDURE Click     DODEFAULT( )     GO BOTTOM     THIS.RefreshForm ENDPROC ENDDEFINE </pre>	<p>Inclui o código para colocar o ponteiro do registro no último registro da tabela.</p>
<pre> ENDPROC ENDDEFINE </pre>	<p>Encerra a definição de classe.</p>

A definição de classe a seguir contém todos os quatro botões de navegação, para que eles possam ser adicionados como uma unidade a um formulário. A classe também inclui um método para definir a propriedade TableAlias dos botões.

#### Definição da classe de controle de navegação de um tabela

Código	Comentários
<pre> DEFINE CLASS vcr AS CONTAINER     Height = 25     Width = 100     Left = 3     Top = 3 </pre>	<p>Começa a definição de classe. A <a href="#">propriedade Height</a> é definida para a mesma altura dos botões de comando que irá conter.</p>
<pre>     ADD OBJECT cmdTop AS navTop ;     WITH Left = 0     ADD OBJECT cmdPrior AS navPrior ;     WITH Left = 25     ADD OBJECT cmdNext AS navNext ;     WITH Left = 50     ADD OBJECT cmdBot AS navBottom ;     WITH Left = 75 </pre>	<p>Adiciona os botões de navegação.</p>
<pre> PROCEDURE SetTable(cTableAlias)     IF TYPE("cTableAlias") = 'C'         THIS.cmdTop.TableAlias = ;         cTableAlias         THIS.cmdPrior.TableAlias = ;         cTableAlias         THIS.cmdNext.TableAlias = ;         cTableAlias         THIS.cmdBot.TableAlias = ; </pre>	<p>Este método é utilizado para definir a propriedade TableAlias dos botões. TableAlias é definida na classe pai Navbutton.</p> <p>Você pode também utilizar o <a href="#">método SetAll</a> para definir esta propriedade:</p> <pre> IF TYPE ("cTableAlias") = 'C' </pre>

```

ENDIF
ENDPROC

```

```

This.SetAll("TableAlias",
"cTableAlias")
ENDIF

```

Contudo, isto causará um erro caso seja adicionado à classe um objeto que não apresente a propriedade TableAlias.

```

ENDDEFINE

```

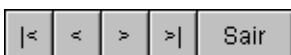
Encerra a definição de classe.

Uma vez definida a classe, você poderá criar subclasses para ela ou adicioná-la a um [formulário](#).

## Criando uma subclasse baseada na nova classe

Você pode também criar subclasses baseadas em `vcr` com botões adicionais como Procurar, Editar, Salvar e Sair. Por exemplo, `vcr2` inclui um botão **Sair**:

### Botões de navegação na tabela com um botão para fechar o formulário



### Definição da subclasse de controle de navegação de uma tabela

Código	Comentários
<pre> DEFINE CLASS vcr2 AS vcr ADD OBJECT cmdQuit AS COMMANDBUTTON WITH ;     Caption = "Sair",;     Height = 25, ;     Width = 50 Width = THIS.Width + THIS.cmdQuit.Width cmdQuit.Left = THIS.Width - ;     THIS.cmdQuit.Width PROCEDURE cmdQuit.CLICK     RELEASE THISFORM ENDPROC </pre>	<p>Define uma classe baseada em <code>vcr</code> e adiciona um botão de comando a ela.</p>
<pre> ENDPROC </pre>	<p>Quando o usuário clica sobre <code>cmdQuit</code>, este código libera o <a href="#">formulário</a>.</p>
<pre> ENDDEFINE </pre>	<p>Encerra a definição de classe.</p>

`Vcr2` tem tudo que `vcr` tem, mais o novo botão de comando e você não precisa reescrever nenhum dos códigos existente.

## Alterações em VCR refletidas na subclasse

Por causa da [herança](#), as alterações na classe pai são refletidas em todas as subclasses baseadas na classe pai. Por exemplo, você pode informar ao usuário que o final da tabela foi alcançado, alterando a instrução `IF EOF( )` em `navNext.Click` para o seguinte:

```

IF EOF( )
    GO BOTTOM
    SET MESSAGE TO "Final da tabela"
ELSE
    SET MESSAGE TO
ENDIF

```

Você pode informar ao usuário que o início da tabela foi alcançado, alterando a instrução `IF BOF( )` em `navPrior.Click` para o seguinte:

```

IF BOF( )
    GO TOP
    SET MESSAGE TO "Início da tabela"
ELSE
    SET MESSAGE TO

```

ENDIF

Se estas alterações forem feitas nas classes `navNext` e `navPrior` elas também se aplicarão automaticamente aos botões apropriados em `vcr` e `vcr2`.

## Adicionando vcr a uma classe de formulário

Uma vez definido como um [controle](#), `vcr` poderá ser adicionado à definição de um recipiente. Por exemplo, o código a seguir adicionado a `NAVCLASS.PRG` define um [formulário](#) com os botões de navegação adicionados:

```
DEFINE CLASS NavForm AS Form
    ADD OBJECT oVCR AS vcr
ENDDEFINE
```

## Executando o formulário que contém VCR

Uma vez definida a subclasse do formulário, você pode exibi-lo facilmente com os comandos apropriados.

### ► Para exibir o formulário

- 1 Carregue a definição de classe:

```
SET PROCEDURE TO navclass ADDITIVE
```

- 2 Crie um [objeto](#) baseado na classe `navform`:

```
frmTest = CREATEOBJECT("navform")
```

- 3 Chame o método [Show](#) do formulário:

```
frmTest.Show
```

Se você não chamar o método `SetTable` de `oVCR` (o objeto `vcr` em `NavForm`) quando o usuário clicar sobre os botões de navegação, o ponteiro do registro se movimentará na tabela da área de trabalho selecionada atualmente. Você pode chamar o método `SetTable` para especificar em qual tabela vai se movimentar.

```
frmTest.oVCR.SetTable("customer")
```

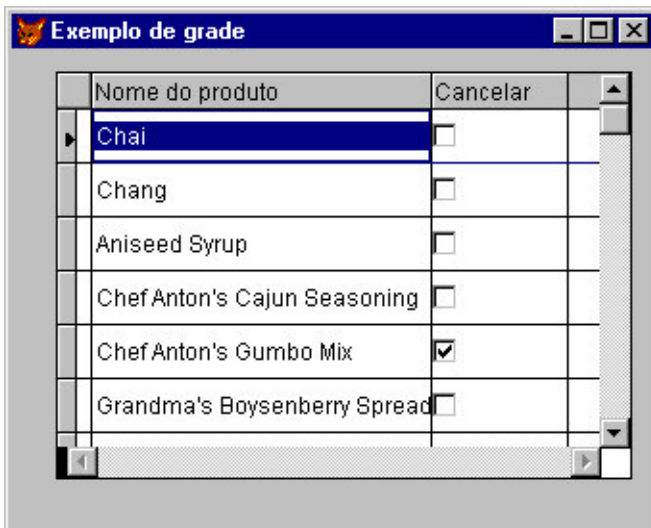
**Observação** Quando o usuário fechar o formulário, `frmTest` será definido como um [valor nulo](#) (.NULL.). Para liberar a variável de objeto da memória, utilize o comando [RELEASE](#). As variáveis de objeto criadas em arquivos de programa serão liberadas da memória quando o programa for completado.

## Definindo um controle de grade

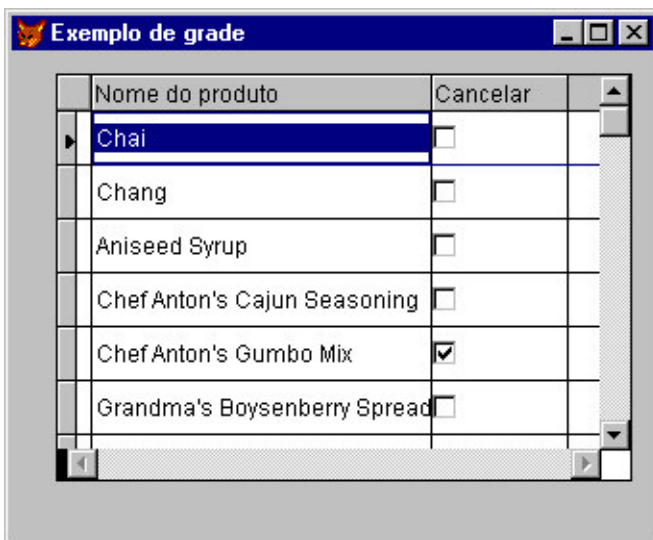
Uma [grade](#) contém colunas que, por sua vez, podem conter cabeçalhos e qualquer outro [controle](#). O controle padrão contido em uma coluna é uma [caixa de texto](#), de modo que a funcionalidade padrão da grade é semelhante à da janela [Pesquisar](#). Contudo, a arquitetura básica da grade amplia esta funcionalidade ao máximo.

O exemplo a seguir cria um formulário que contém um objeto `Grade` com duas colunas. A segunda coluna contém uma [caixa de verificação](#) para exibir os valores em um campo lógico de uma tabela.

### Controle de grade com uma caixa de verificação em uma coluna



Nome do produto	Cancelar
Chai	<input type="checkbox"/>
Chang	<input type="checkbox"/>
Aniseed Syrup	<input type="checkbox"/>
Chef Anton's Cajun Seasoning	<input type="checkbox"/>
Chef Anton's Gumbo Mix	<input checked="" type="checkbox"/>
Grandma's Boysenberry Spread	<input type="checkbox"/>



### Definição de uma classe de grade com uma caixa de verificação em uma coluna de grade

Código	Comentários
<pre> DEFINE CLASS grdProducts AS Grid     Left = 24     Top = 10     Width = 295     Height = 210     Visible = .T.     RowHeight = 28     ColumnCount = 2 </pre>	<p>Inicia a definição de classe e define as <a href="#">propriedades</a> que determinam a aparência da grade.</p> <p>Ao definir a propriedade ColumnCount como 2, você adiciona duas colunas à grade. Cada coluna contém um cabeçalho com o nome Header1. Além disso, cada coluna tem um grupo independente de propriedades que determinam sua aparência e funcionamento.</p>
<pre> Column1.ControlSource = "prod_name" Column2.ControlSource = "discontinuu" </pre>	<p>Quando você define ControlSource de uma coluna, esta exibe os valores desse campo para todos os registros da tabela. Discontinuu é um campo lógico.</p>
<pre> Column2.Sparse = .F. </pre>	<p>A Column2 conterá a caixa de verificação. Defina a <a href="#">propriedade Sparse</a> da coluna como .F. para que a caixa de verificação fique visível em todas as linhas e não apenas na célula selecionada.</p>
<pre> Procedure Init     THIS.Column1.Width = 175     THIS.Column2.Width = 68     THIS.Column1.Header1.Caption = ;         "Nome do Produto"     THIS.Column2.Header1.Caption = ;         "Suspenso"      THIS.Column2.AddObject ("chk1", ;         "checkbox")     THIS.Column2.CurrentControl = ;         "chk1"     THIS.Column2.chk1.Visible = .T.     THIS.Column2.chk1.Caption = "" ENDPROC </pre>	<p>Define as larguras das colunas e as legendas dos cabeçalhos.</p> <p>O <a href="#">método AddObject</a> permite que você adicione um objeto a um recipiente — neste caso, uma caixa de verificação denominada chk1.</p> <p>Define o <a href="#">CurrentControl</a> da coluna como a</p>

caixa de verificação para que esta seja exibida.  
Assegura que a caixa de verificação fique visível.  
Define a legenda como uma sequência vazia para que a [legenda](#) padrão "chk1" não fique exibida.

ENDDEFINE

Encerra a definição de classe.

A definição de classe a seguir é o formulário que contém a grade. As duas definições de classe podem ser incluídas no mesmo arquivo de programa.

### Definição de uma classe de formulário que contém a classe de grade

Código	Comentários
DEFINE CLASS GridForm AS FORM Width = 330 Height = 250 Caption = "Exemplo de Grade" ADD OBJECT grid1 AS grdProducts	Cria uma classe de formulário e adiciona a ela um <a href="#">objeto</a> , baseado na classe de grade.
PROCEDURE Destroy CLEAR EVENTS ENDPROC	O programa que cria um objeto baseado nesta classe utilizará <a href="#">READ EVENTS</a> . A inclusão de CLEAR EVENTS no <a href="#">evento Destroy</a> do formulário permite que o programa seja encerrado quando o usuário fechar o formulário.
ENDDEFINE	Encerra a definição de classe.

O programa a seguir abre a tabela com os campos a serem exibidos nas colunas de grade, cria um objeto baseado na classe GridForm e executa o comando [READ EVENTS](#).

```
CLOSE DATABASE
OPEN DATABASE (SYS(2004) + "samples\data\testdata.dbc")
USE products
frmTest= CREATEOBJECT("GridForm")
frmTest.Show
READ EVENTS
```

Este programa poderá ser incluído no mesmo arquivo das definições de classe se for colocado no início do arquivo. Você pode também utilizar o comando [SET PROCEDURE TO](#) para especificar o programa com as definições de classe e incluir este código em um programa separado.

## Criando referências de objetos

Em vez de fazer uma cópia de um [objeto](#), você pode criar uma referência a esse objeto. Uma referência utiliza menos memória do que um objeto adicional, pode ser facilmente passada entre os [procedimentos](#), e pode ajudar a escrever um código genérico.

### Retornando uma referência a um objeto

É possível que você queira manipular um [objeto](#) através de uma ou mais referências ao mesmo. Por exemplo, o programa a seguir define uma classe, cria um objeto baseado na classe e retorna uma referência ao objeto.

```
*--NEWINV.PRG
*--Retorna uma referência a um novo formulário de fatura
frmInv = CREATEOBJECT("InvoiceForm")
RETURN frmInv
```

```
DEFINE CLASS InvoiceForm AS FORM
    ADD OBJECT txtCompany AS TEXTBOX
```

```
* código p/ definir propriedades, adicionar outros objetos, etc
ENDDEFINE
```

O programa a seguir estabelece uma referência ao objeto criado em NEWINV.PRG. A [variável](#) de referência pode ser manipulada exatamente da mesma maneira que a variável do objeto.

```
frmInvoice = NewInv() && armazena a referência de objeto em uma variável
frmInvoice.SHOW
```

Você pode também criar uma referência a um objeto em um [formulário](#), como no exemplo a seguir.

```
txtCustName = frmInvoice.txtCompany
txtCustName.Value = "Usuário do Fox"
```

**Dica** Uma vez criado um objeto, você pode utilizar o comando **DISPLAY OBJECTS** para exibir a hierarquia de classes de objeto, as definições de propriedades, os objetos recipientes e os métodos e eventos disponíveis. Você pode preencher uma matriz com as propriedades (não as definições de propriedade), eventos, métodos e objetos recipientes de um objeto com a função **AMEMBERS( )**.

## Liberando objetos e referências da memória

Se uma referência a um [objeto](#) existir, a liberação do objeto não elimina o objeto da memória. Por exemplo, o comando a seguir libera o objeto original, `frmInvoice`:

```
RELEASE frmInvoice
```

Contudo, como uma referência a um objeto pertencente a `frmInvoice` ainda existe, o objeto não será liberado da memória até que `txtCustName` seja liberado com o comando a seguir:

```
RELEASE txtCustName
```

## Verificando se um objeto existe

Você pode utilizar as funções **TYPE( )** e **ISNULL( )** para determinar se um objeto existe. Por exemplo, as linhas de código a seguir verificam se o objeto denominado `oConnection` existe:

```
IF TYPE("oConnection") = "O" AND NOT ISNULL(oConnection)
* O objeto existe
ELSE
* O objeto não existe
ENDIF
```

**Observação** **ISNULL( )** é necessária porque `.NULL.` é armazenado na variável de objeto do formulário quando um usuário fecha um formulário, mas o tipo da variável permanece "O".

## Criando matrizes de membros

Você pode definir membros de classes como [matrizes](#). No exemplo a seguir, `choices` é uma matriz de [controles](#).

```
DEFINE CLASS MoverListBox AS CONTAINER
DIMENSION choices[3]
ADD OBJECT lstFromListBox AS LISTBOX
ADD OBJECT lstToListBox AS LISTBOX
ADD OBJECT choices[1] AS COMMANDBUTTON
ADD OBJECT choices[2] AS COMMANDBUTTON
ADD OBJECT choices[3] AS CHECKBOX
PROCEDURE choices.CLICK
PARAMETER nIndex
DO CASE
CASE nIndex = 1
* código
CASE nIndex = 2
* código
CASE nIndex = 3
* código
ENDCASE
```



```
ENDPROC
ENDDDEFINE
```

Quando o usuário clica um controle em uma matriz de controles, o Visual FoxPro passa o número de índice do controle para o procedimento do evento Click. Neste procedimento, você pode utilizar uma instrução [CASE](#) para executar um código diferente, dependendo de qual botão foi clicado.

## Criando matrizes de objetos

Você pode também criar [matrizes](#) de [objetos](#). Por exemplo, `MyArray` contém cinco botões de comandos:

```
DIMENSION MyArray[5]
FOR x = 1 TO 5
    MyArray[x] = CREATEOBJECT("COMMANDBUTTON")
ENDFOR
```

Alguns pontos que devem ser considerados com referência às matrizes de objetos:

- Você não pode atribuir um objeto a uma matriz inteira com um comando. É preciso atribuir o objeto a cada membro da matriz, individualmente.
- Você não pode atribuir um valor a uma [propriedade](#) de uma matriz inteira. O comando a seguir resultará em um erro:  
`MyArray.Enabled = .F.`
- Quando você redimensionar uma matriz de objeto para que fique maior do que a matriz original, os novos elementos serão inicializados com o valor falso (.F.), como é o caso de todas as matrizes do Visual FoxPro. Quando você redimensionar uma matriz de objeto para que fique menor do que a matriz original, os objetos com um índice maior do que o maior índice novo serão liberados.

## Utilizando objetos para armazenar dados

Em linguagens orientadas a objetos, uma [classe](#) oferece um veículo útil e conveniente para armazenar dados e [procedimentos](#) relacionados a uma entidade. Por exemplo, você pode definir uma classe de cliente para conter informações sobre um cliente, assim como um [método](#) para calcular a idade dele:

```
DEFINE CLASS customer AS CUSTOM
    LastName = ""
    FirstName = ""
    Birthday = { / / }
    PROCEDURE Age
        IF !EMPTY(THIS.Birthday)
            RETURN YEAR(THIS.Birthday) - YEAR(THIS.Birthday)
        ELSE
            RETURN 0
        ENDIF
    ENDPROC
ENDDEFINE
```

Contudo, os dados que são armazenados nos objetos baseados na classe de cliente são armazenados apenas na memória. Se estes dados estivessem em uma tabela, ela seria armazenada em disco. Se você tivesse mais de um cliente para controlar, a tabela daria acesso a todos os comandos e funções de gerenciamento de banco de dados do Visual FoxPro. Como resultado, você poderia localizar informações rapidamente, classificá-las, agrupá-las, executar cálculos com elas, criar relatórios e consultas baseadas nelas e assim por diante.

Armazenar e manipular dados em [bancos de dados](#) e [tabelas](#) é o que o Visual FoxPro faz de melhor. Contudo, talvez você queira armazenar dados em [objetos](#). Normalmente, os dados serão significativos apenas enquanto o aplicativo estiver sendo executado e irão se referir a uma única entidade.

Por exemplo, em um aplicativo que inclui um sistema de segurança, você teria, tipicamente, uma

tabela de usuários com acesso ao aplicativo. A tabela incluiria a identificação do usuário, a senha e o nível de acesso. Uma vez que um usuário tenha efetuado um logon, você não precisará de todas as informações da tabela. Você precisará apenas de informações sobre o usuário atual e elas poderão ser facilmente armazenadas e manipuladas em um objeto. A definição de classe a seguir, por exemplo, inicia um logon quando um objeto baseado na classe é criado.

```
DEFINE CLASS NewUser AS CUSTOM
    PROTECTED LogonTime, AccessLevel
    UserId = ""
    PassWord = ""
    LogonTime = { / / : : }
    AccessLevel = 0
    PROCEDURE Init
        DO FORM LOGON WITH ; && assumindo que você criou este formulário
            This.UserId, ;
            This.PassWord, ;
            This.AccessLevel
        This.LogonTime = DATETIME( )
    ENDPROC
* Cria métodos para retornar valores de propriedade protegidos.
    PROCEDURE GetLogonTime
        RETURN This.LogonTime
    ENDPROC
    PROCEDURE GetAccessLevel
        RETURN This.AccessLevel
    ENDPROC
```

ENDDEFINE

No programa principal do seu aplicativo, você pode criar um [objeto](#) baseado na classe NewUser:

```
oUser = CREATEOBJECT('NewUser')
oUser.Logon
```

No seu aplicativo, quando precisar de informações sobre o usuário atual, você poderá obtê-las do objeto oUser. Por exemplo:

```
IF oUser.GetAccessLevel( ) >= 4
    DO ADMIN.MPR
ENDIF
```

## Integrando objetos e dados

Na maioria dos aplicativos, você poderá utilizar melhor o poder do Visual FoxPro, integrando objetos e dados. A maioria das classes do Visual FoxPro apresenta [propriedades](#) e [métodos](#) que permitem a você integrar o poder de um gerenciador de banco de dados relacional e um sistema completo orientado a objetos.

### Propriedades para integrar classes do Visual FoxPro e dados de banco de dados

Classe	Propriedade dos dados
<a href="#">Grid</a>	<a href="#">RecordSource</a> , <a href="#">ChildOrder</a> , <a href="#">LinkMaster</a>
Todos os outros <a href="#">controles</a>	<a href="#">ControlSource</a>
<a href="#">List box</a> e <a href="#">combo box</a>	<a href="#">ControlSource</a> , <a href="#">RowSource</a>
<a href="#">Form</a> e <a href="#">form set</a>	<a href="#">DataSession</a>

Como estas propriedades de dados podem ser alteradas na [hora da criação](#) ou em [tempo de execução](#), você pode criar controles genéricos com funcionalidade encapsulada que operam em diversos dados.

Para obter maiores informações sobre a integração de dados e objetos, consulte o capítulo 9, [Criando Formulários](#), e o capítulo 10, [Utilizando Controles](#).

