

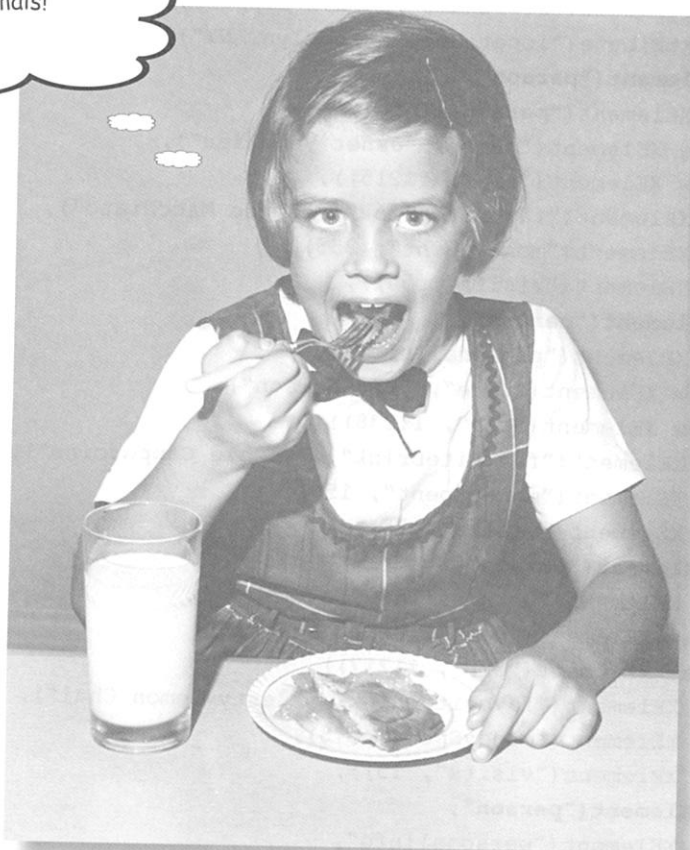
Apendice I: sobras *



Os 5 principais tópicos que gostaríamos de ter incluído neste livro



Ainda quero mais!



A diversão está apenas começando!

Mostramos a você muitas ferramentas ótimas para desenvolver **programas poderosos** em C#. Mas não seria possível incluir **cada uma das ferramentas, tecnologias e técnicas** neste livro - simplesmente não existem páginas o suficiente. Tivemos que fazer algumas escolhas realmente difíceis a respeito sobre o que incluir e o que deixar de fora. Aqui apresentamos alguns itens que não sobreviveram aos cortes. Ainda que não possamos nos aprofundar neles, acreditamos que eles são **importantes e úteis**, e queremos dar a você uma pequena introdução a respeito deles.

1 LINQ para XML

XML - Extensible Markup Language, linguagem extensível de marcação - é um formato para arquivos e streams de dados que representam dados complexos usando texto. O Framework .NET dá a você algumas ferramentas muito poderosas para criar, carregar e salvar arquivos XML. E uma vez que você tenha em suas mãos os dados XML, pode usar LINQ para fazer consultas. Adicione `"using System.Xml.Linq;"` no início do arquivo e digite o seguinte método - ele gera um documento XML para armazenar os dados de fidelidade de clientes do Starbuzz.

```
private static XDocument GetStarbuzzData() {
    XDocument doc = new XDocument(
        new XDeclaration("1.0", "utf-8", "yes"),
        new XComment("Starbuzz Customer Loyalty Data"),
        new XElement("starbuzzData",
            new XAttribute("storeName", "Park Slope"),
            new XAttribute("location", "Brooklyn, NY"),
            new XElement("person",
                new XElement("personalInfo",
                    new XElement("name", "Janet Venutian"),
                    new XElement("zip", 11215)),
                new XElement("favoriteDrink", "Choco Macchiato"),
                new XElement("moneySpent", 255),
                new XElement("visits", 50)),
            new XElement("person",
                new XElement("personalInfo",
                    new XElement("name", "Liz Nelson"),
                    new XElement("zip", 11238)),
                new XElement("favoriteDrink", "Double Cappuccino"),
                new XElement("moneySpent", 150),
                new XElement("visits", 35)),
            new XElement("person",
                new XElement("personalInfo",
                    new XElement("name", "Matt Franks"),
                    new XElement("zip", 11217)),
                new XElement("favoriteDrink", "Zesty Lemon Chai"),
                new XElement("moneySpent", 75),
                new XElement("visits", 15)),
            new XElement("person",
                new XElement("personalInfo",
                    new XElement("name", "Joe Ng"),
                    new XElement("zip", 11217)),
                new XElement("favoriteDrink", "Banana Split in a Cup"),
                new XElement("moneySpent", 60),
                new XElement("visits", 10)),
            new XElement("person",
                new XElement("personalInfo",
                    new XElement("name", "Sarah Kalter"),
                    new XElement("zip", 11215)),
                new XElement("favoriteDrink", "Boring Coffee"),
                new XElement("moneySpent", 110),
                new XElement("visits", 15)))));
    return doc;
}
```

Salve e Abra arquivos XML

Você pode exibir um objeto `XDocument` no console ou salvá-lo num arquivo, e pode carregar um arquivo XML nele:

```
XDocument doc = GetStarbuzzData();
Console.WriteLine(doc.ToString());
doc.Save("starbuzzData.xml");
XDocument anotherDoc = XDocument.Load("starbuzzData.xml");
```

O objeto `XDocument` tem métodos `Load()` e `Save()` para ler e escrever arquivos XML. É este método `ToString()` transforma tudo dentro do arquivo num grande documento XML.

Consulte seus dados

Eis aqui uma consulta LINQ simples que pesquisa nos dados do Starbuzz usando seu `XDocument`:

```
var data = from item in doc.Descendants("person")
select new { drink = item.Element("favoriteDrink").Value,
moneySpent = item.Element("moneySpent").Value,
zipCode = item.Element("personalInfo").Element("zip").Value };
foreach (var p in data)
Console.WriteLine(p.ToString());
```

O método `Descendants()` (descendentes) retorna uma referência para um objeto que você pode incluir diretamente dentro da sua consulta LINQ.

Você já sabe que a LINQ permite que você chame métodos e use isso como parte da consulta, o que funciona muito bem com o método `Element()`.

E você pode fazer consultas mais complexas também:

```
var zipcodeGroups = from item in doc.Descendants("person")
group item.Element("favoriteDrink").Value
by item.Element("personalInfo").Element("zip").Value
into zipcodeGroup
select zipcodeGroup;
foreach (var group in zipcodeGroups)
Console.WriteLine("{0} favorite drinks in {1}",
group.Distinct().Count(), group.Key);
```

`Element()` retorna um objeto `Xelement`, e você pode usar suas propriedades para procurar por valores específicos no seu documento XML.

Leia dados de um canal RSS

Você pode fazer algumas coisas muito poderosas com LINQ para XML. Eis aqui uma consulta simples para ler artigos do nosso blog:

```
XDocument ourBlog = XDocument.Load("http://www.stellman-greene.com/feed");
Console.WriteLine(ourBlog.Element("rss").Element("channel").Element("title").Value);
var posts = from post in ourBlog.Descendants("item")
select new { Title = post.Element("title").Value,
Date = post.Element("pubDate").Value };
foreach (var post in posts)
Console.WriteLine(post.ToString());
```

O método `Load()` de `XDocument` tem muitos construtores sobrecarregados. Este aqui recupera dados XML de uma URL.

Coloque um botão num formulário, certifique-se de estar usando `System.Xml.Linq`; no início do arquivo, digite esta consulta num tratador de evento, e veja o que ela imprime no console.

Usamos a URL do nosso blog, *Building Better Software* (desenvolvendo programas melhores): <http://www.stellman-greene.com/>

2 Refatoração

Refatoração quer dizer alterar a forma como seu código está estruturado sem mudar seu comportamento. Sempre que você escrever um método complexo, deve parar uns minutos e pensar como você poderia modificá-lo para que ficasse mais fácil de entender. Por sorte, o IDE tem algumas ferramentas de refatoração muito úteis. Existe todo o tipo de refatoração que pode ser feita - aqui estão alguns exemplos que usamos com frequência.

Extrair um método

Quando estávamos escrevendo o renderizador baseado em controles do Capítulo 13, originalmente incluímos o seguinte laço foreach:

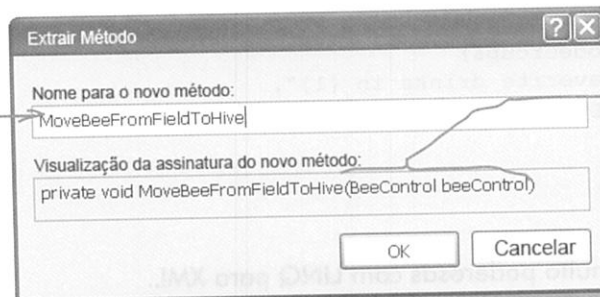
```
foreach (Bee bee in world.Bees) {
    beeControl = GetBeeControl(bee);
    if (bee.InsideHive) {
        if (fieldForm.Controls.Contains(beeControl)) {
            fieldForm.Controls.Remove(beeControl);
            beeControl.Size = new Size(40, 40);
            hiveForm.Controls.Add(beeControl);
            beeControl.BringToFront();
        } else if (hiveForm.Controls.Contains(beeControl)) {
            hiveForm.Controls.Remove(beeControl);
            beeControl.Size = new Size(20, 20);
            fieldForm.Controls.Add(beeControl);
            beeControl.BringToFront();
        }
    }
    beeControl.Location = bee.Location;
}
```

Estas quatro linhas movem um BeeControl do formulário do campo para o da colmeia.

Estas quatro movem o BeeControl da colmeia para o campo.

Um dos nossos revisores técnicos, Joe Albahari, destacou que isso era um pouco difícil de ler. Ele sugeriu que **extraíssemos esses dois blocos de quatro linhas e os transformássemos em métodos**. Então selecionamos o primeiro bloco, clicamos nele com o botão direito, e selecionamos "Refactor>>Extract Method..." - e a seguinte janela foi exibida:

Digitamos o nome do novo método. Decidimos chamá-lo MoveBeeFromFieldToHive() porque isso em suma descreve o que o código faz.



O IDE examina o código que está selecionado e determina que ele use uma variável BeeControl chamada beeControl, então ele a adicionou como parâmetro do método.

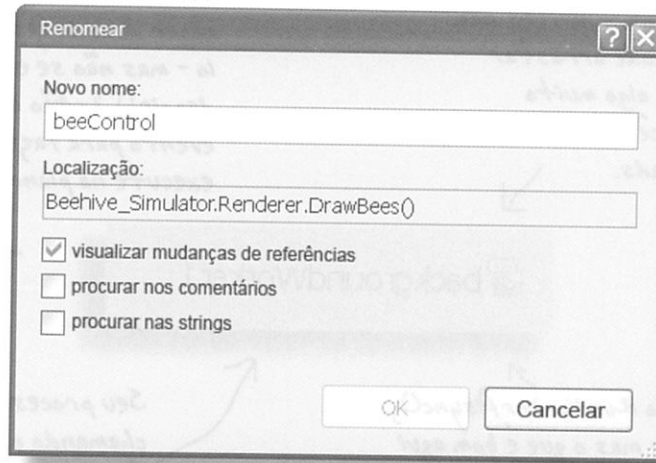
Então fizemos a mesma coisa com o segundo bloco de quatro linhas, extraíndo-o para um método que chamamos de MoveBeeFromHiveToField(). Eis como o laço foreach ficou no final - muito mais fácil de ler:

```
foreach (Bee bee in world.Bees) {
    beeControl = GetBeeControl(bee);
    if (bee.InsideHive) {
        if (fieldForm.Controls.Contains(beeControl))
            MoveBeeFromFieldToHive(beeControl);
        } else if (hiveForm.Controls.Contains(beeControl))
            MoveBeeFromHiveToField(beeControl, bee);
    }
    beeControl.Location = bee.Location;
}
```

Renomear uma variável

Lá atrás no Capítulo 3, explicamos como pode-se deixar seu código muito mais fácil de entender ao escolher nomes intuitivos para suas classes, métodos, campos e variáveis. O IDE pode realmente ajudar quando se trata de nomear coisas no código. Basta clicar com o botão direito em qualquer classe, variável, campo, propriedade, namespace, constante - basicamente qualquer coisa que se possa nomear - e escolher "Refactor >> Rename". Fizemos isso com "beeControl" no código do simulador. Eis a janela que apareceu:

Esta janela permite que você escolha um novo nome para o item. Se você tivesse escolhido, digamos, Bobbo, então o IDE iria vasculhar todo o código e mudar cada ocorrência para o novo nome.



O IDE faz um trabalho muito minucioso no que se refere a renomear. Se você fizer isso com uma classe, ele mudará cada comando ou instância que a utilize. Você pode clicar em qualquer ocorrência do nome, em qualquer lugar no código, e o IDE vai fazer a alteração no restante do programa.

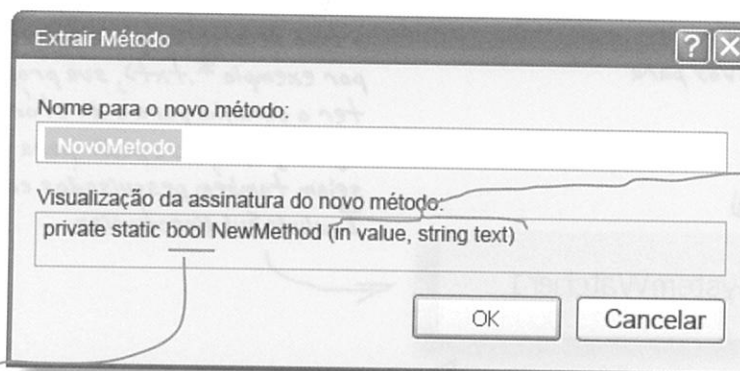
Consolidando uma expressão condicional

Eis aqui uma forma muito interessante de usar a característica "Extrair Método". Abra qualquer programa, acrescente um botão, insira o seguinte código no tratador de evento:

```
private void button1_Click(object sender, EventArgs e) {
    int value = 5;
    string text = "Hi there";
    if (value == 36 || text.Contains("there"))
        MessageBox.Show("Pow!");
}
```

Selecione tudo dentro do comando if: `value == 36 || text.Contains("there")`. Então clique com o botão direito e escolha "Refactor >> Extract Method...". Aqui está o que você vai ver:

Todas as expressões condicionais resultam num booleano, então o IDE vai criar um método que retorne um para substituir o teste condicional com uma chamada para esse método.



Não apenas isso fará o código mais fácil de ler, mas agora você tem um novo método que pode reutilizar em outro lugar!

A expressão usa dois variáveis chamadas value e text, então o IDE adiciona parâmetros ao método usando esses nomes.

3 Alguns de nossos componentes favoritos na caixa de ferramentas

Este é um livro a respeito de aprender C#, não sobre vantagens e desvantagens a respeito de componentes que vem com o Framework .NET. Ainda assim, temos nossos favoritos, e gostaríamos de compartilhar um pouco com você.

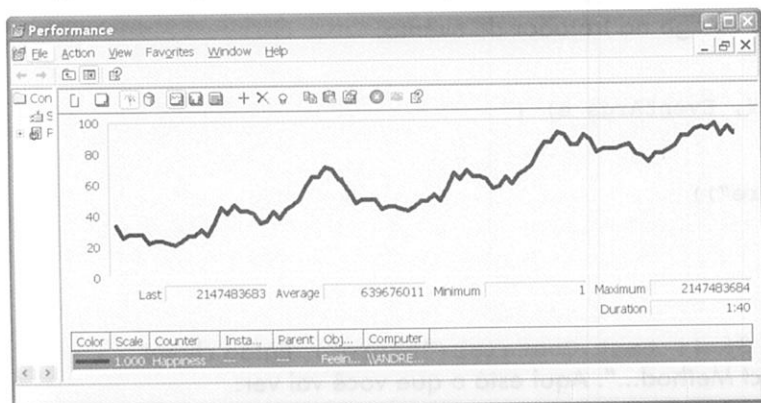
BackgroundWorker é um desses componentes não visuais (como o *Timer*) que você pode arrastar para seus formulários, e ele faz algo muito interessante - permite que você facilmente crie aplicativos com múltiplas threads.

Basta arrastar um para seu formulário e clicar duas vezes nele (ou, se quiser, instanciá-lo - mas não se esqueça de descartá-lo depois!). Então adicione um tratador de evento para fazer o que quer que você queira que execute no plano de fundo.



Quando você chamar seu método *RunWorkerAsync()*, ele dispara o método *DoWork()* - mas o que é bom aqui é que ele faz isso numa outra thread. Isso quer dizer que ele vai executar em paralelo com as demais tarefas que seu programa estiver realizando.

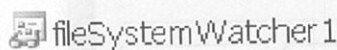
Seu processo pode relatar progresso chamando o método *ReportProgress()* do *BackgroundWorker* - isso faz com que ele levante o evento *ProgressChanged*.



Criamos esse contador *Happiness* (felicidade) na categoria *Feelings* (sentimentos), e demos a ele dados usando *PerformanceCounter*.

O *FileSystemWatcher* faz o que o nome indica - ele vigia o sistema de arquivos para ver se alguma coisa mudou.

Você pode configurar sua propriedade *Filter* para o tipo de arquivo que ele deve procurar (como por exemplo **.txt*), sua propriedade *Path* deve ter o caminho para o diretório que deve ser vigiado, e pode-se pedir para que os subdiretórios sejam também pesquisados com a propriedade *IncludeSubdirectories*.




Uma vez que tenha sido configurado, ele vigia o diretório procurando por qualquer arquivo novo ou alterado. Assim que um arquivo é adicionado, alterado ou apagado, ele dispara seu evento *Changed*. Ele também tem os eventos *Created*, *Deleted* e *Renamed* para fazer um controle mais específico.

Quando o *FileSystemWatcher* dispara um evento, ele passa informações sobre o arquivo que foi alterado (ou adicionado, apagado, etc) usando um objeto *FileSystemEventArgs*.

Quando você estiver escrevendo um programa que fique executando continuamente, é realmente útil monitorá-lo. É o Windows vem com uma ferramenta útil chamada Monitor de Performance (perfmon.exe) que permite que se monitorem processos.

O componente `PerformanceCounter` (contador de performance) permite que você disponibilize informações sobre seu programa para o sistema de monitoramento de performance do Windows. Use `Increment()` e `Decrement()` ou atribua `RawValue` (valor bruto, literalmente valor cru) diretamente. Assim que você fizer isso, poderá ver os dados no Monitor de Performance.

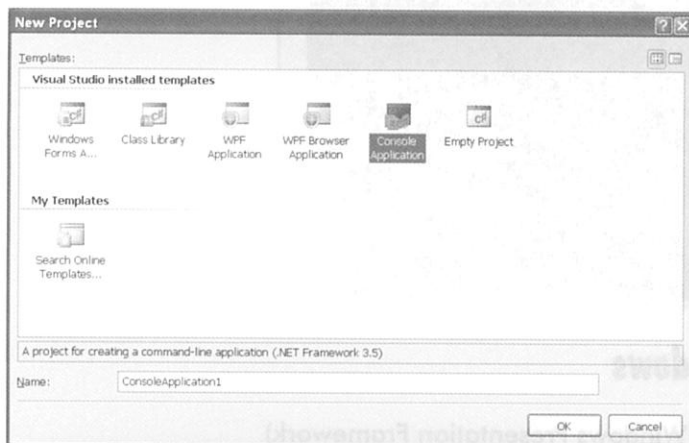
 performanceCounter1

O Windows divide seus contadores de performance em categorias, então você vai precisar criar uma - existem métodos para isso em `System.Diagnostics`. Depois basta associar seu `PerformanceCounter` com a categoria criada, e você pode começar a enviar informações de diagnóstico, para seu deleite!

4 Aplicativos de Console

Muitos livros sobre C# começam com aplicativos de console. Achamos que isso é chato. É muito mais satisfatório desenvolver programas que parecem, bem... com qualquer coisa que seja. E é exatamente isso que um programa de console não é. Mas algumas vezes você vai precisar escrever um aplicativo de linha de comando. Por sorte, isso é bastante fácil. Veja como:

- 1 Crie um projeto de Aplicativo de Console**
Qualquer projeto pode ser um aplicativo de console. Entre em qualquer um, selecione "Properties" no menu Project, e altere "Output type" (tipo de saída) para "Console Application". Mas é mais fácil criar uma a partir do zero.



- 2 O IDE vai adicionar apenas um arquivo - Program.cs**
E ele terá um ponto de entrada vazio... e é tudo.

```
class Program
{
    static void Main(string[] args)
    {
    }
}
```

Aqui vai um pequeno projeto para você: pegue o descarregador hexadecimal que você desenvolveu no Capítulo 9 e o transforme num aplicativo de console. Faça com que ele leia um arquivo passado na linha de comando e o imprima como uma descarga hexa. Faça com que ele aceite entradas direto da linha de comando (usando `Console.ReadLine`) e imprima essa entrada como uma descarga hexa. Então dê uma pesquisada a respeito do comando `od` do Unix e veja se você pode reproduzi-lo em C#.

então é por isso que essa classe é chamada *Console*

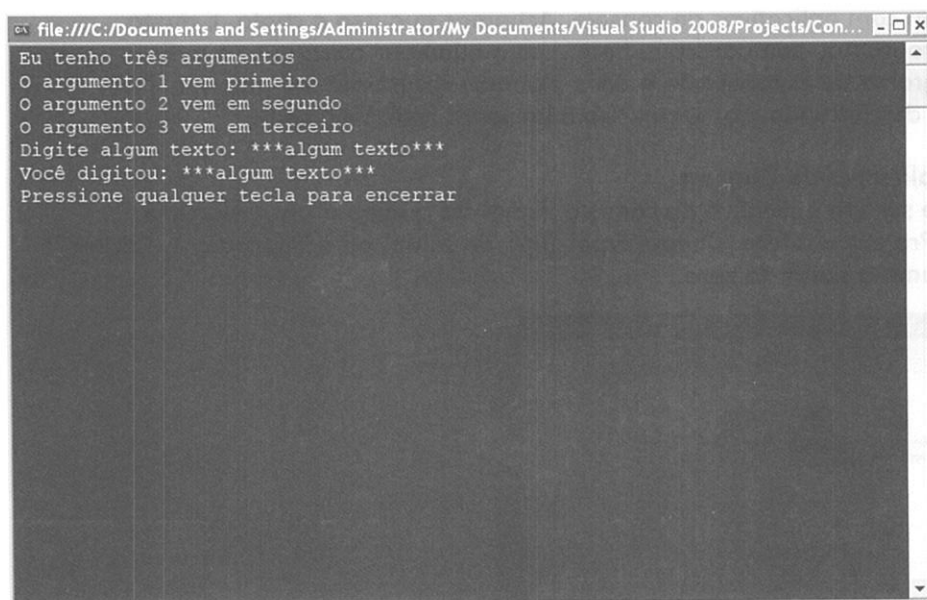
3 Use o parâmetro args para obter os argumentos de linha de comando

Seu ponto de entrada tem um único parâmetro, uma matriz de strings chamada args, que conterá os argumentos de linha de comando. Você já sabe como usar o método Console.WriteLine() - e existem alguns outros métodos úteis do console, incluindo ReadLine() e ReadKey() (ler tecla).

```
class Program {
    static void Main(string[] args) {
        Console.WriteLine("I got {0} arguments", args.Length);
        for (int i = 1; i <= args.Length; i++)
            Console.WriteLine("Argument #{0} is {1}", i, args[i - 1]);
        Console.Write("Enter some text: ");
        string input = Console.ReadLine();
        Console.WriteLine("You entered: {0}", input);
        Console.WriteLine("Press any key to end...");
        Console.ReadKey();
    }
}
```

4 Depure seu programa numa janela de console

Quando você depurar seu programa, o IDE mostra uma janela de console. Os métodos ReadLine() e ReadKey() obtêm seus resultados dessa mesma janela - basta digitar nela. E ao invés de escrever na janela Output, um aplicativo de console escreve em vez disso também nessa janela. Você pode alterar os argumentos de linha de comando na página "Debug" da janela de propriedades de projeto.



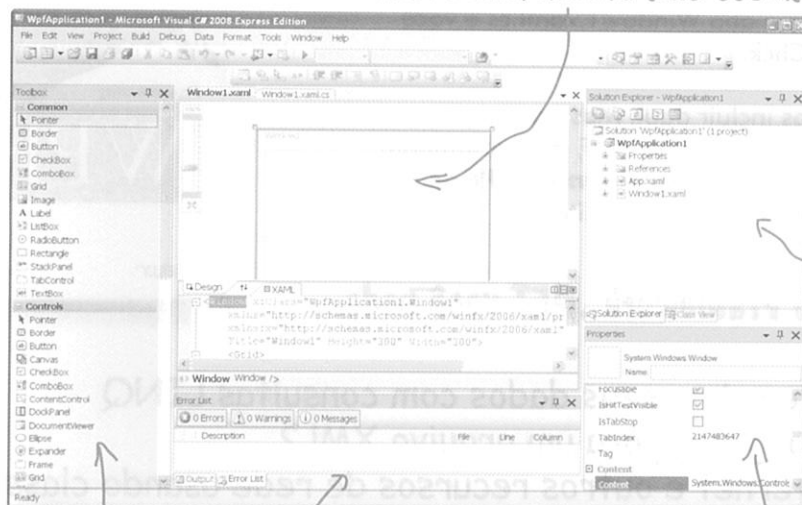
5 Framework de Apresentação do Windows

O Framework de Apresentação do Windows (WPF, Windows Presentation Framework) é a plataforma de próxima geração da Microsoft para desenvolver aplicativos visuais. Ela é extraordinária - tem um layout baseado em XML, controles escaláveis, um sistema totalmente novo para controles, gráficos e animações 2D e 3D, fluxo de texto e formatação de documentos, e mesmo um plugin para navegadores para qualquer plataforma que a utiliza.

Infelizmente, embora o WPF seja realmente legal e uma tecnologia de alta capacidade, não é uma ferramenta particularmente boa para ensinar C#. E esse era nosso objetivo - enfiar conceitos de C# na sua cabeça o mais rápido e fácil possível.

Reserve uns momentos e crie um novo aplicativo WPF. Basta criar um novo projeto usando o IDE, mas não crie um de Windows Forms. Em vez disso, **selecione Aplicativo WPF**. Você imediatamente vai notar uma diferença no IDE.

A maior diferença que você vai perceber é que o designer de formulário não se parece nada com o que você está acostumado. Examinaremos isso melhor num minuto.



Os aplicativos WPF ainda usam classes, como qualquer outro programa C# ou .NET, logo o Navegador de Solução é o mesmo.

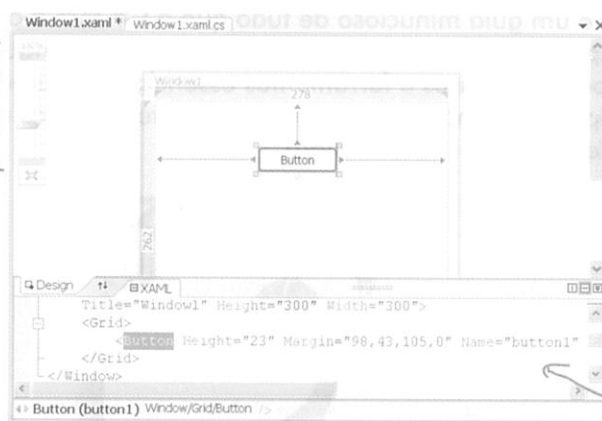
Examine atentamente a caixa de ferramentas - ela tem todo um novo conjunto de controles.

Éia as janelas de lista de erros e resultados já familiares que você tem usado.

As propriedades parecem totalmente diferentes. Isso é porque você usa essa janela para alterar atributos num arquivo XAML, e não propriedades de objetos.

Arraste um botão da caixa de ferramentas para o formulário. Se esse fosse um aplicativo Windows Forms, o IDE adicionaria código em Form1.Designer.cs para adicionar um controle ao objeto Form1. Mas com o WPF é diferente - ele usa uma linguagem baseada em XML chamada XAML para definir como a interface de usuário é apresentada.

Arraste o cursor deste controle para cima e para baixo para aumentar ou diminuir o zoom. Mesmo quando você amplia muito, sua interface continua boa - ela não fica pixelada.



XAML é abreviação de Linguagem de Marcação Extensível para Aplicativos (extensible application markup language), e é a linguagem baseada em XML que aplicativos WPF usam para determinar onde todos os controles e outros elementos de UI aparecem.

O IDE tem um editor de XML realmente poderoso otimizado para trabalhar com XAML.

Entre no editor XML e adicione um segundo botão digitando a linha em negrito abaixo. Você vai perceber que o IntelliSense do IDE faz um bom trabalho auxiliando você a digitar todas as tags XML.

```
<Grid>
  <Button Height="23" Margin="98,43,105,0" Name="button1"
    VerticalAlignment="Top" Click="button1_Click">Button</Button>
  <Button Height="23" Margin="5,5,100,20" Name="button2"
    VerticalAlignment="Top" Click="button2_Click">Another button</Button>
</Grid>
```

Quando você chegar na parte da linha que diz "Click='button2_Click'", não digite o nome do tratador de evento. Em vez disso, use a janela do IntelliSense que aparece para dizer ao IDE para adicionar um novo método. Assim que você tiver terminado a linha, verá um novo botão aparecer no designer. Mude para a aba Window1.xaml.cs e você encontrará um novo método button2_Click.

Isso é todo o WPF e XAML que pudemos incluir aqui. Mas agora que você tem as ferramentas para começar a aprender sobre WPF, nós certamente recomendamos que você dê uma olhada em **Programando WPF** de Chris Sells e Ian Griffiths. Ele está disponível no site da sua editora.



Você sabia que com o C# e o Framework .NET você pode...

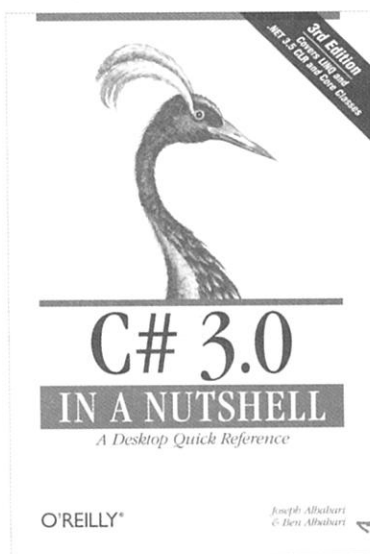
- ★ Ter muito mais controle sobre seus dados com consultas LINQ avançadas? Serializar objetos para um arquivo XML?
- ★ Acessar páginas de Internet e outros recursos de rede usando classes nativas?
- ★ Adicionar criptografia avançada e segurança a seus programas?
- ★ Criar aplicativos complexos com múltiplas linhas de execução?
- ★ Distribuir suas classes para que outras pessoas possam usá-las?
- ★ Usar expressões regulares para fazer busca avançada de texto?

Eu não tinha idéia! Onde eu posso aprender mais?



Existe um livro excelente que explica isso tudo!

É chamado C# 3.0 In a Nutshell, de Joseph Albahari e Ben Albahari, e é um guia minucioso de tudo que o C# tem a oferecer. Você vai aprender sobre características avançadas da linguagem, lerá sobre todas as classes e ferramentas essenciais do Framework .NET, e aprenderá mais sobre o que realmente acontece nos bastidores do C#. Verifique no site da editora.



Joseph Albahari nos ajudou muito mesmo fazendo neste livro uma revisão técnica realmente minuciosa. Agradecemos muito por toda a sua ajuda, Joe!