**Namespaces**

Os namespaces ajudam a resolver esse problema criando um contêiner para

itens, como classes. Duas classes com o mesmo nome não serão confundidas se elas

estiverem em namespaces diferentes.

É uma boa prática definir todas as suas classes em namespaces, e o ambiente do

Visual Studio 2013 segue essa recomendação utilizando o nome do seu projeto como

o namespace de nível mais alto. A biblioteca de classes do .NET Framework também

segue essa recomendação: toda classe no .NET Framework está situada em um namespace.

Por exemplo, a classe *Console* reside no namespace *System*. Isso significa que seu

nome completo é, na verdade, *System.Console*.

**Namespaces e assemblies**

Uma diretiva *using* coloca em escopo os itens de um namespace, e você não precisa

qualificar completamente os nomes das classes no seu código. As classes são

compiladas em *assemblies*. Um assembly é um arquivo que tem, em geral, a extensão

de nome de arquivo .dll, embora programas executáveis com a extensão de

nome de arquivo .exe também sejam assemblies.

Um assembly pode conter muitas classes. As classes de biblioteca abrangidas

pela biblioteca de classes do .NET Framework, como *System.Console*, são fornecidas

nos assemblies instalados no seu computador junto com o Visual Studio. Você descobrirá

que a bibl ioteca de classes do .NET Framework contém milhares de classes.

Se todas fossem armazenadas nos mesmos assemblies, estes seriam enormes e difíceis

de manter. (Se a Microsoft atualizasse um único método em uma única classe,

ela teria de distribuir toda a biblioteca de classes a todos os desenvolvedores!)

Por essa razão, a biblioteca de classes do .NET Framework é dividida em alguns

assemblies, agrupados de acordo com a área funcional a que as classes estão

relacionadas. Por exemplo, um assembly “básico” (na verdade, chamado *mscorlib.*

*dll*) contém todas as classes comuns, como *System.Console*, e outros assemblies

contêm classes para manipular bancos de dados, acessar web services, compilar

GUIs e assim por diante. Se quiser utilizar uma classe em um assembly, você deve

adicionar ao seu projeto uma referência a ele. Então, pode adicionar instruções

*using* ao seu código, colocando em escopo os itens do namespace nesse assembly.

Observe que não há necessariamente uma equivalência 1:1 entre um assembly

e um namespace. Um único assembly pode conter classes definidas para

muitos namespaces e um único namespace pode abranger vários assemblies. Por

exemplo, as classes e itens do namespace *System* são, na verdade, implementados

por vários assemblies, incluindo *mscorlib.dll*, *System.dll* e *System.Core.dll*, dentre

outros. Isso parece muito confuso agora, mas você logo irá se acostumar.

Ao utilizar o Visual Studio para criar um aplicativo, o template que você seleciona

inclui automaticamente referências aos assemblies adequados. Por exemplo,

no Solution Explorer do projeto TestHello, expanda a pasta References. Você

verá que um aplicativo de console contém automaticamente referências a assemblies

chamados *Microsoft.CSharp*, *System*, *System.Core*, *System.Data*, *System.Data.*

*DataExtensions*, *System.Xml* e *System.Xml.Linq*. Talvez você fique surpreso ao ver

que *mscorlib.dll* não está nessa lista. Isso acontece porque todos os aplicativos do

.NET Framework devem usar esse assembly, pois ele contém a funcionalidade de

tempo de execução fundamental. A pasta References lista somente os assemblies

opcionais; é possível adicionar ou remover assemblies dessa pasta, conforme for

necessário.

Para acrescentar referências para assemblies adicionais em um projeto, clique

com o botão direito do mouse na pasta References e então, no menu de atalho

que aparece, clique em Add Reference – você fará isso nos próximos exercícios.

Você também pode remover um assembly, clicando nele com o botão direito do

mouse na pasta References e, então, clicando em Remove.

Importante No Windows 7 e no Windows 8, O Visual Studio 2013 fornece

dois templates para compilar aplicativos gráficos: o template Windows Forms

Application e o template WPF Application. Windows Forms é uma tecnologia

que surgiu no .NET Framework versão 1.0. O WPF, ou Windows Presentation

Foundation, é uma tecnologia aprimorada que apareceu na versão 3.0 do .NET

Framework. O WPF oferece muitos recursos adicionais em relação ao Windows

Forms, e você deve considerar o seu uso no lugar do Windows Forms para todos

os novos desenvolvimentos para Windows 7.

**CAPÍTULO 2**

Variáveis, operadores

e expressões

**Instruções**

*Instrução* é um comando que executa uma ação, como calcular um valor e armazenar

o resultado, ou exibir uma mensagem para o usuário. Você combina instruções para

criar métodos. Para aprender mais sobre métodos, consulte o Capítulo 3, “Como escrever

métodos e aplicar o escopo”, mas, por enquanto, considere um método como uma

sequência nomeada de instruções. *Main*, que foi apresentado no capítulo anterior, é

um exemplo de método.

As instruções em C# seguem um conjunto bem definido de regras que descrevem

seu formato e sua construção. Estas são conhecidas coletivamente como *sintaxe*.

(Por outro lado, a especificação *do que* as instruções fazem é conhecida coletivamente

como *semântica*.) Uma das regras de sintaxe mais simples e mais importantes do C#

diz que você deve terminar todas as instruções com um ponto e vírgula.

**Identificadores**

*Identificadores* são os nomes utilizados para distinguir os elementos nos seus programas,

como namespaces, classes, métodos e variáveis. (Discutiremos as variáveis em breve.)

No C#, você deve seguir as regras de sintaxe abaixo ao escolher os identificadores:

j Você pode utilizar apenas letras (maiúsculas ou minúsculas), dígitos e o caractere

de sublinhado.

j Um identificador deve iniciar com uma letra (ou um sublinhado).

Por exemplo, *resultado*, *\_placar*, *timeDeFutebol* e *plano9* são identificadores válidos,

enquanto *resultado%*, *timeDeFutebol$* e *9plano* não são.

**Variáveis**

*Variável* é um local de armazenamento que contém um valor. Você pode considerar

uma variável como uma caixa na memória do computador que contém informações

temporárias. Você deve atribuir a cada variável em um programa um nome não ambíguo

que a identifique de forma única no contexto em que é utilizada. Um nome de

variável é utilizado para referenciar o valor que ela armazena.

Nomeie variáveis

Adote uma convenção de nomes que torne claras as variáveis definidas. Isso é especialmente

importante se você faz parte de uma equipe de projeto com vários desenvolvedores

trabalhando em diferentes partes de um aplicativo; uma convenção de

nomes consistente ajuda a evitar confusão e pode reduzir a extensão de erros. A lista a

seguir contém algumas recomendações gerais:

Não inicie um identificador com um sublinhado. Embora isso seja válido em C#,

pode limitar a interoperabilidade de seu código com aplicativos compilados em

outras linguagens, como Microsoft Visual Basic.

j Não crie identificadores cuja única diferença seja entre maiúsculas e minúsculas.

Por exemplo, não crie uma variável chamada *minhaVariavel* e outra chamada

*MinhaVariavel* para serem utilizadas ao mesmo tempo, porque será muito fácil

confundi-las. Além disso, a definição de identificadores cuja única diferença seja

a distinção entre maiúsculas e minúsculas pode limitar a reutilização das classes

nos aplicativos desenvolvidos com outras linguagens que não diferem maiúsculas

e minúsculas, como o Visual Basic.

j Comece o nome com uma letra minúscula.

j Em um identificador com várias palavras, comece a segunda palavra e as palavras

subsequentes com uma letra maiúscula. Isso é chamado de notação *camelo*

ou *camelCase*.

j Não utilize notação húngara. (Se você for desenvolvedor de Microsoft Visual

C++, provavelmente já conhece a notação húngara. Se não souber o que é isso,

não se preocupe!)

Declare variáveis

Variáveis locais não atribuídas

Quando você declara uma variável, ela contém um valor aleatório até que lhe seja

atribuído um valor. Esse comportamento era uma grande fonte de erros nos programas

C e C++ que criavam uma variável e a utilizavam acidentalmente como fonte de

informações antes de ela receber um valor. O C# não permite utilizar uma variável não

atribuída. É necessário atribuir um valor a uma variável antes de usá-la; caso contrário,

o programa não compilará. Essa exigência é chamada *regra de atribuição definitiva*.

Por exemplo, as instruções a seguir geram a mensagem de erro de tempo de compilação

“Use of unassigned local variable ‘age’” porque a instrução *Console.WriteLine* tenta

exibir o valor de uma variável não inicializada:

int age;

Console.WriteLine(age); // compile-time error

Importante O *F* é um tipo de sufixo especificando que 0.42 deve ser tratado

como um valor *float*. Se você esquecer o *F*, o valor 0.42 será tratado como um

*double* e seu programa não compilará, porque um valor de um tipo não pode

ser atribuído a uma variável de outro tipo sem se escrever código adicional – C#

é muito rígido nesse aspecto.