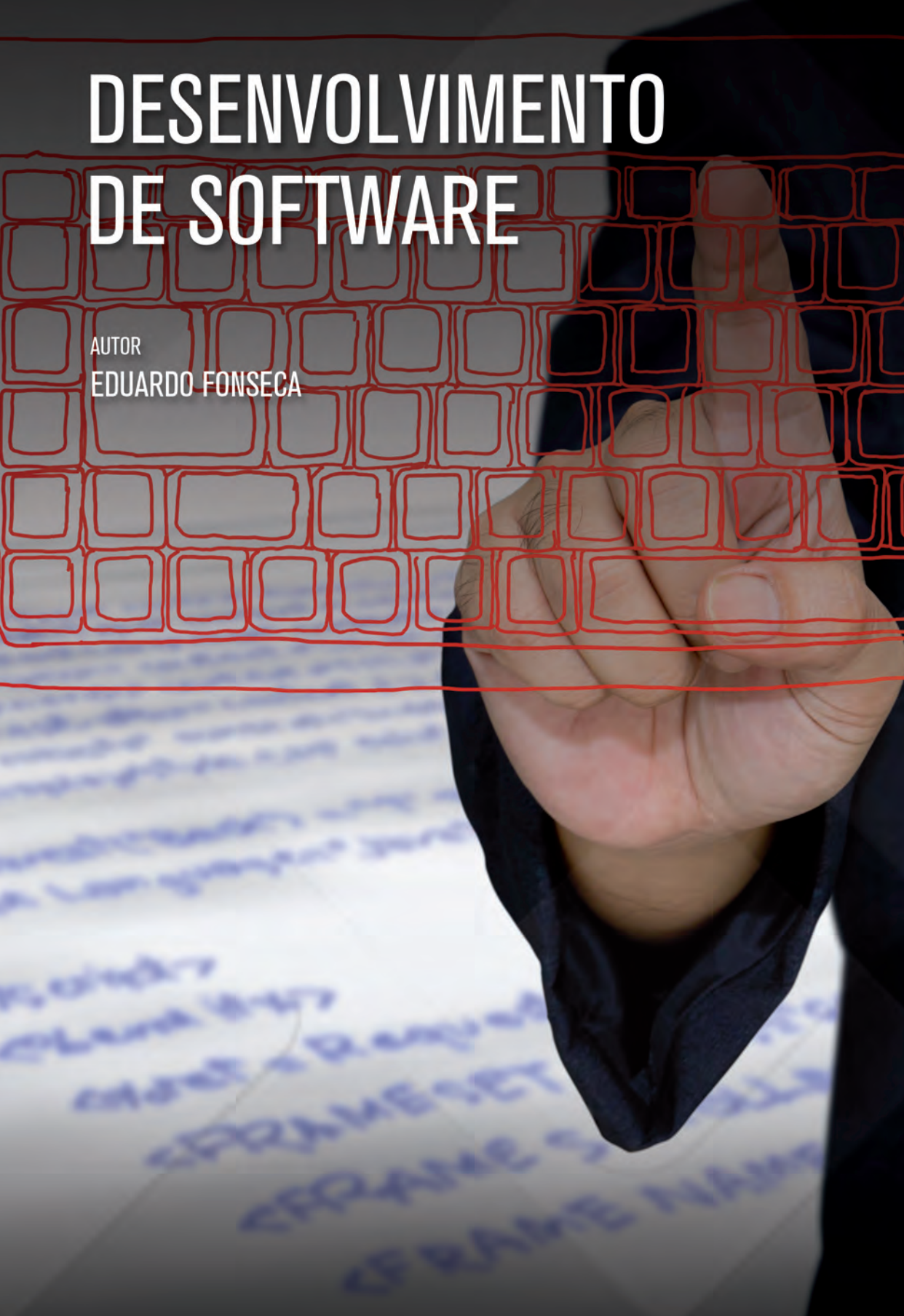


DESENVOLVIMENTO DE SOFTWARE

AUTOR

EDUARDO FONSECA



DESENVOLVIMENTO DE SOFTWARE

AUTOR
EDUARDO FONSECA

1ª EDIÇÃO
SESES
RIO DE JANEIRO 2015



Estácio

Conselho editorial REGIANE BURGER; ROBERTO PAES; GLADIS LINHARES; KAREN BORTOLOTI;
HELICIMARA AFFONSO DE SOUZA

Autor do original EDUARDO FONSECA DE ALMEIDA

Projeto editorial ROBERTO PAES

Coordenação de produção GLADIS LINHARES

Coordenação de produção EaD KAREN FERNANDA BORTOLOTI

Projeto gráfico PAULO VITOR BASTOS

Diagramação BFS MEDIA

Revisão linguística ROSELI CANTALOGO COUTO

Imagem de capa WEERAPAT WATTANAPICHAYAKUL | DREAMSTIME.COM

Todos os direitos reservados. Nenhuma parte desta obra pode ser reproduzida ou transmitida por quaisquer meios (eletrônico ou mecânico, incluindo fotocópia e gravação) ou arquivada em qualquer sistema ou banco de dados sem permissão escrita da Editora. Copyright SESES, 2015.

Dados Internacionais de Catalogação na Publicação (CIP)

F676d Fonseca, Eduardo

Desenvolvimento de software / Eduardo Fonseca.

Rio de Janeiro : SESES, 2015

192 p. : IL.

ISBN: 978-85-5548-003-4

1. Desenvolvimento de software 2. .NET Framework. 3. Visual Basic .NET.
4. Interface gráfica do usuário. I. SESES. II. Estácio.

CDD 005.1

Diretoria de Ensino — Fábrica de Conhecimento
Rua do Bispo, 83, bloco F, Campus João Uchôa
Rio Comprido — Rio de Janeiro — RJ — CEP 20261-063

Sumário

Prefácio	7
1. Introdução ao .NET Framework e Programação Básica com o VB.NET	9
Objetivos	10
1.1 Introdução: .NET Framework e Visual Studio	11
1.1.1 Mercado de Software	11
1.1.2 Características do .NET Framework	12
1.1.3 Edições do Visual Studio	17
1.1.4 Instalando o Visual Studio	19
1.1.5 Abordagem Inicial ao Visual Studio	21
1.1.5.1 Iniciando um Projeto	21
1.1.5.2 Primeiro Programa em Visual Basic: Olá, Mundo!	24
1.1.5.3 Sintaxe básica do Visual Basic	26
1.1.5.4 A Interface do Visual Studio	28
1.1.5.5 O Visual Studio no Modo de Depuração	30
1.1.5.6 Compilação e o Arquivo Executável Resultante	30
1.2 Programação Básica com VB.NET	31
1.2.1 Histórico do Visual Basic	32
1.2.2 Comentários, variáveis, constantes e tipos de dados	33
1.2.3 Declaração e atribuição de variáveis	36
1.2.4 Palavras-chave do VB.NET	37
1.2.5 Operadores	39
Reflexão	44
Atividades	44
Referências bibliográficas	45

2. Programação Modular e Programação Estruturada 47

OBJETIVOS	48
2.1 Programação Modular	49
2.1.1 Módulos	49
2.1.2 Métodos	52
2.1.3 Sub-rotinas e Funções	55
2.1.4 Escopo	60
2.2 Programação Estruturada	63
2.2.1 Estruturas de Controle de Decisão	64
2.2.1.1 Estrutura de controle de decisão simples (se... então)	64
2.2.1.2 Estrutura de controle de decisão estendida (se... então... senão)	65
2.2.1.3 Estrutura de controle de decisão aninhada	66
2.2.1.4 Estrutura de controle de decisão em sequência na mesma linha.	68
2.2.1.5 Estrutura de controle de múltiplas decisões (escolha)	69
2.2.2 Estruturas de Controle de Repetição	71
2.2.2.1 Estrutura de controle de repetição controlada por contador (para)	72
2.2.2.2 Estrutura de controle de repetição While (enquanto)	73
2.2.2.3 Estrutura de controle de repetição Do (faça)	74
Reflexão	76
Atividades	77
Referências bibliográficas	78

3. Tratamento de Exceções e Programação Orientada a Objetos 79

Objetivos	80
3.1 Tratamento de Exceções	81
3.1.1 Exceções	81
3.1.2 Tipos de Exceções	82
3.1.3 Estrutura de um Bloco de Tratamento de Exceções	83

3.2 Programação Orientada a Objetos	90
3.2.1 Conceitos de Programação Orientada a Objetos	91
3.2.2 Orientação a Objetos no Visual Basic	93
3.2.2.1 Análise do arquivo Imovel.vb	97
3.2.2.2 Análise do arquivo Module1.vb	100
3.2.2.3 Eventos	102
3.2.2.4 Classes instanciadas e classes estáticas (shared classes)	103
3.2.2.5 Membros compartilhados (shared members)	103
3.2.2.6 Herança	105
3.2.2.7 Modificadores de Acesso	106
Reflexão	107
Atividades	107
Referências bibliográficas	108

4. Banco de Dados na Plataforma Microsoft .NET 109

Objetivos	110
4.1 Sistemas de Banco de Dados	111
4.1.1 Sistema Gerenciador de Banco de Dados (SGBD)	112
4.1.2 Elementos de um Banco de Dados Relacional	115
4.1.3 SQL Server	118
4.1.4 Tipos de dados do SQL Server	119
4.1.5 Desenvolvendo um BD no SQL Server	123
4.2 Banco de Dados no .NET Framework	125
4.2.1 Script de Criação de BD	125
4.2.2 Script de criação de tabelas	132
4.2.3 Views	136
4.2.4 Criação de Views	138
4.2.5 Biblioteca ADO.NET	139
4.2.6 Componentes ADO.NET	140
Reflexão	143
Atividades	143
Referências bibliográficas	144

5. Interface Gráfica para o Usuário e os Elementos de Desenvolvimento de Software 145

Objetivos	146
5.1 Interface Gráfica para Usuário	147
5.1.1 Janelas, Forms e componentes	147
5.1.1.1 Tipos de janelas	148
5.1.1.2 Criando um Aplicativo com Janelas com Visual Basic	148
5.1.1.3 Label	151
5.1.1.4 TextBox	152
5.1.1.5 Button	153
5.1.1.6 GroupBox	154
5.1.1.7 RadioButton	155
5.1.1.8 CheckBox	156
5.1.1.9 ListBox	157
5.1.1.10 ComboBox	159
5.1.1.11 MenuStrip	161
5.1.2 Programação Orientada a Eventos	163
5.2 Desenvolvimento de Software	164
5.2.1 Criando uma Aplicação Windows Forms	164
5.2.2 Adicionando uma Fonte de Dados	167
5.2.3 Formulários para as entidade de dados	171
5.2.4 Resposta a um Evento: abrindo uma janela	174
Reflexão	182
Atividades	182
Referências bibliográficas	183

Gabarito 183

Prefácio

Prezados(as) alunos(as),

A disciplina Desenvolvimento de Software tem como objetivo abordar diversos elementos e ferramentas utilizados para o desenvolvimento e a implementação efetiva de código de um programa. Aqui serão feitas abordagens gerais de diversos assuntos que são tratados especificamente em um curso de desenvolvimento de sistemas: caracterização bibliotecas de uma linguagem de programação, ambiente de desenvolvimento integrado (IDE - *Integrated Development Environment*), elementos básicos de programação (variáveis, modularização, estruturas de controle, orientação a objetos e orientação a eventos) e conexão a um banco de dados.

A abordagem da disciplina é integrada e prática, pois aqui serão vistos esses diversos elementos dentro do contexto do ambiente de desenvolvimento do Framework .NET da Microsoft, o qual fornece uma biblioteca e ambiente de execução para as linguagens de programação Visual Basic .NET, C# e ASP.NET. A linguagem utilizada nessa disciplina será o Visual Basic .NET, também denominado VB.NET, que é uma versão moderna do Visual Basic.

Essa disciplina não trata dos processos de desenvolvimento de software, que é assunto de outra que se preocupa com a engenharia e o ciclo de desenvolvimento de software, através do gerenciamento de projetos, equipes, levantamento de requisitos de software e desenho de sua arquitetura. O foco aqui são os elementos diversos para a programação em si, observados de uma forma integrada.

Esse livro é composto de cinco capítulos, os quais procuram dividir entre si a apresentação dos diversos recursos para o desenvolvimento de softwares:

Capítulo 1 – descreve algumas características do Framework .NET, da ferramenta de desenvolvimento Visual Studio e da linguagem de programação Visual Basic .NET. Também apresentará os tipos de dados da linguagem e a sintaxe na declaração de variáveis;

Capítulo 2 – explica como o código fonte de um programa pode ser compartimentado em módulos, métodos, funções e sub-procedimentos. Além disso, será abordada a programação estruturada, com a sintaxe de estruturas de controle no VB.NET;

Capítulo 3 – aborda o tratamento de exceções, seu tipo e sua sintaxe. Na segunda parte do capítulo, o assunto é a programação orientada a objetos no VB.NET;

Capítulo 4 – tem como objetivo abordar o uso de bancos de dados em geral e, especificamente, apresentar o SQL Server, que é o Sistema Gerenciador de Banco de Dados (SGBD) da Microsoft, que pode ser acessado através do Framework .NET com a biblioteca ADO.NET;

Capítulo 5 – introduz os conceitos de interfaces gráficas para o usuário (GUI - Graphical User Interface), programação orientada a eventos e fecha a disciplina com um exemplo de aplicativo que se conecta a um banco de dados.

Bons estudos!

1

Introdução ao .NET Framework e Programação Básica com o VB.NET

Nesse primeiro capítulo, serão apresentadas as características básicas das ferramentas da Microsoft para desenvolvimento de software: sua biblioteca e recursos do .NET Framework, a linguagem Visual Basic e o ambiente integrado de desenvolvimento Visual Studio. Serão abordadas as ferramentas disponíveis, como obtê-las e o processo de instalação.

Aqui também serão vistos os componentes básicos e a sintaxe da linguagem Visual Basic em sua versão moderna, o VB.NET, linguagem que apresenta recursos avançados semelhantes a outras linguagens de programação, como o C#. Quem conhece a linguagem em versões mais antigas, como Visual Basic 6, deve considerar que ela muito mudou desde então.



OBJETIVOS

Após o estudo desse capítulo, você deverá ter um conhecimento geral do Framework .NET e do Visual Studio, sabendo o que são e como instalá-los em seu computador. Você saberá as características gerais da linguagem Visual Basic .NET e alguns de seus elementos básicos de programação, como: palavras reservadas, declaração de variáveis, tipos de dados e operadores.

1.1 Introdução: .NET Framework e Visual Studio

Este livro abordará diversos aspectos do desenvolvimento de softwares, com ênfase nos seus principais elementos, vistos de forma integrada no .NET Framework e utilizando a ferramenta de desenvolvimento da Microsoft, o Visual Studio. Com o objetivo de fornecer um contexto à atividade de desenvolvimento de software, antes de apresentar a plataforma, é interessante observar algumas estatísticas relacionadas à economia em torno do setor de softwares no Brasil.

1.1.1 Mercado de Software

No Brasil, o mercado de Tecnologias da Informação, em geral, é um dos que mais cresce. Esse mercado inclui atividades de fabricação de hardware, desenvolvimento e distribuição de software e prestação de serviços nessa área. A seguir, serão apresentados diversos dados divulgados pelo relatório anual do setor desenvolvido pela ABAS, a Associação Brasileira das Empresas de Software (ABAS, 2014).

Considerando todo esse mercado de TI, o crescimento de investimentos no país foi de 15,4% entre 2012 e 2013, colocando o Brasil na sétima posição mundial nesse quesito. Com isso, o país absorveu 3% dos investimentos em TI no mundo. Além disso, ficou entre os 10 primeiros no mundo em taxas de crescimento no setor nesse período, com 4,8%. A TI movimentou, em 2013, 61,6 bilhões de dólares, representando 2,74% do PIB brasileiro.

Mais especificamente, o setor de software, que abrange o desenvolvimento de sistemas, cresceu 13,5% em 2013. O setor de serviços, que inclui a distribuição e o suporte em TI, cresceu 7,7% no mesmo ano. Em conjunto, software e serviços cresceram 10,1% em 2013.

O crescimento de uso de software desenvolvimento no país foi de 15,3% em 2013, enquanto que o crescimento de mercado para softwares desenvolvidos no exterior foi de 12,9%, quer dizer, o software nacional ganhou espaço de mercado. O mercado nacional é explorado por 11.230 empresas de diversas nacionalidades voltadas ao desenvolvimento, produção, distribuição e prestação de serviços.

Os setores de Finanças, Serviços e Telecomunicações foram os clientes mais importantes para as empresas desenvolvedoras de software e prestadoras de serviços de TI no país em 2013, representando cerca de 51% do mercado usuário. Em seguida, os mais representativos foram Indústria, Governo e Comércio. O setor Comércio foi o que aumentou mais seu investimento, os quais foram 27% mais altos em relação a 2012. Ainda em 2013, no mercado mundial, os setores de software e serviços atingiram o valor de US\$ 1,039 trilhões, sendo que o Brasil assumiu a 8ª posição com um mercado interno que totaliza de US\$ 25,1 bilhões vendidos.

Novamente enfatizando apenas dados relacionados ao setor de software, o mercado interno foi de US\$ 10,7 bilhões em 2013, representando 2,8% do mercado mundial. As exportações de software desenvolvido no país foi de US\$ 209 milhões. Ainda segundo a ABAS, o setor, domesticamente, conta com .302 empresas dedicadas ao desenvolvimento e à comercialização de software.

É bastante importante notar que o investimento no setor de TI é considerado essencial para o desenvolvimento dos países, pois suas atividades trazem produtividade à economias nacionais. Isso não só quando as economias estão em crescimento, mas também quando elas se encontram em ciclos de crise, mesmo que o capital para investimentos seja mais escasso, já que a implementação adequada de sistemas nas empresas traz eficiências que permitem produzir mais com menos recursos.

1.1.2 Características do .NET Framework

Um *Framework* de software é um conjunto de ferramentas e funcionalidades que pode ser utilizado pelo programador para desenvolver seu próprio software. A palavra *framework* pode significar "arcabouço" ou "estrutura" e é aplicada, no contexto de desenvolvimento de software, a um conjunto integrado que pode incluir classes, programas auxiliares, compiladores, ambiente de execução e padrões de comunicação entre softwares, sendo que tudo isso pode ser utilizado pelo programador da forma que lhe convier. Outro termo aproximado para a palavra é plataforma de desenvolvimento. O objetivo de um framework é fornecer ferramentas que tornem o trabalho dos desenvolvedores de software muito mais eficiente, com recursos que os façam produzir softwares funcionais e seguros em menos tempo.

Segundo a própria Microsoft, "o .NET Framework é uma plataforma de desenvolvimento popular para a criação de aplicativos para Windows, Windows Store, Windows Phone, Windows Server e Microsoft Azure" (MSDN, 2015a). Isso significa que o .NET é a base para criar programas que executem nos diversos dispositivos que rodem o Windows em suas várias versões. Não apenas isso, mas há maneiras de utilizar essas ferramentas para programar para dispositivos que usam sistemas operacionais OSX ou com kernel Linux, o que tende a se tornar cada vez mais fácil. No início de 2015, a versão mais recente da plataforma .NET Framework é a 4.5.

Entre seus componentes mais importantes está o compilador, que suporta as linguagens Visual Basic e C#. O Framework também suporta muitas outras linguagens de programação, mas elas não são compiladas diretamente por ele, e sim por compiladores alternativos que se conformam com seu padrão e geram código intermediário que executa em seu ambiente, o Common Language Runtime (CLR). Os compiladores de Visual Basic e C# da plataforma tiveram o seu código aberto, sob uma licença livre, durante o ano de 2014, sendo incorporado à versão 2015 do .NET Framework - a tendência dessa versão, que tem o codinome Roslyn, é receber muitas inovações no médio prazo.

As plataformas de desenvolvimento, como o .NET, costuma ser formadas por partes com funções específicas, o que é uma estratégia de desenvolvimento modular, como veremos no capítulo 2. Para compreender a interação das partes de um framework, é comum visualizá-lo em camadas, como na figura a seguir.

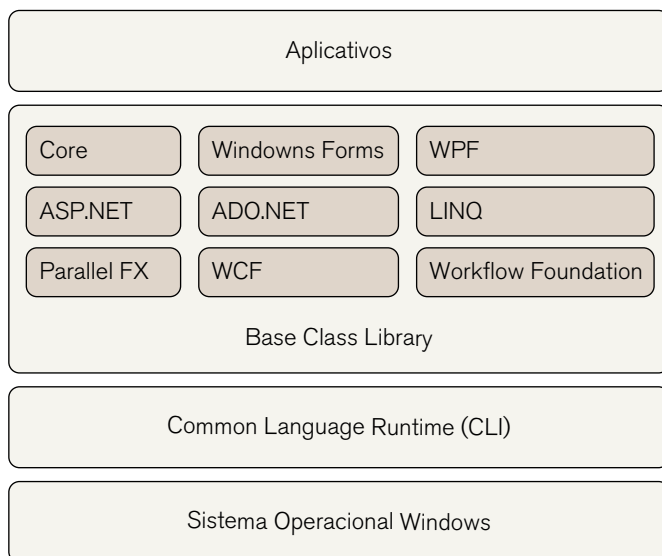


Figura 1.1 – Componentes do .NET Framework divididos em camadas.

Até 2014, o .NET foi concebido visando apenas o sistema operacional Windows, mas, desde então, a Microsoft tomou novos rumos estratégicos e vem buscando criar um contexto em que o Framework passe a ser multiplataforma, isto é, que execute em vários sistemas operacionais. A Figura 1 ainda mostra o Windows como a base, sobre a qual está o ambiente de execução, o Common Language Runtime (CLR), que é como uma máquina virtual que executa códigos intermediários, em *bytecode*, de maneira geral se assemelhando ao que ocorre com o Java. Acima do CLI, está a biblioteca de componentes que podem ser utilizados pelo aplicativos, como Windows Forms e WPF para construir interfaces gráficas, ADO.NET para se conectar a fontes de dados e ASP.NET para construir *scripts* para servidores Web.

A partir da versão 4.0 do .NET, não é necessário ter as versões anteriores para executar programas que utilizam novas funcionalidades. Entretanto, se no momento que o usuário for executar um programa que utiliza funcionalidades exclusivas do .NET versão 4.0 ou 4.5, ele apenas tiver versões mais antigas instaladas, será necessário que ele faça uma atualização, instalando a versão exigida da biblioteca. A seguir, alguns dos aspectos da execução de código no .NET serão descritos com mais detalhes.

Common Language Runtime (CLR)

Se um aplicativo .NET, tanto programado em Visual Basic como em C# ou outras linguagens, é compilado para *bytecode*, ele apenas executa em uma máquina em que o .NET Framework está instalado, pois ele necessita do CLR e das bibliotecas para funcionar. A Microsoft está trabalhando em um compilador, chamado .NET Native, que transforma o mesmo software em código de máquina, o que o torna mais rápido e independente do framework para executar, mas, no início de 2015, ele ainda estava em desenvolvimento e apenas suportava apenas C#.

O CLR é então uma camada que permite que uma aplicação execute em um ambiente gerenciado, que se chama assim porque intermedeia a comunicação entre esse aplicativo e o sistema operacional, gerenciando memória, acesso aos recursos do sistema e serviços de segurança. O código escrito para executar sobre esse ambiente é chamado *managed code*, ou código gerenciado (DEL SOLE, 2010). O CLR é uma implementação de uma especificação da Microsoft chamada Common Language Infrastructure (CLI), que é aberta e pode ser utilizada por

outros desenvolvedores. Se um software for desenvolvido por uma linguagem e essa compilada para um *bytecode* compatível, também pode ser executado sobre o CLR, e é isso que ocorre no caso do IronPython, IronRuby, J# e muitas outras.

.NET Assemblies

Em português, a palavra Assembly significa "montagem". No .NET, um Assembly é uma compilação de um software para um *bytecode* intermediário, nesse caso denominado Microsoft Intermediate Language (MSIL), juntamente com seus metadados. Um Assembly é formado por uma linguagem *assembly* de alto nível, a qual é capaz de executar em diversos ambientes independentemente do tipo de CPU, já que o CLR cria uma camada comum de execução. Um programa que seja elaborado em qualquer linguagem e seja compilado para um Assembly, de acordo com a Common Language Infrastructure (CLI), poderá ser executado pelo CLR.

Os metadados que acompanham o Assembly possuem informações sobre o código implementado, como tipos de dados, funções, rotinas, assinaturas etc. (DEL SOLE, 2010). Entre outras coisas, dessa forma, por exemplo, o .NET consegue saber se a versão instalada em um computador consegue executar as funcionalidades implementadas no aplicativo, ou se será necessário para o usuário instalar uma nova versão.

Execução do Código

Quando o Assembly é invocado, um compilador do tipo Just-In-Time (JIT) o transforma em um código de máquina compatível com o computador onde está instalado. Assim, após um primeiro início mais lento do programa, ele passa a utilizar essa versão compilada e executar mais rápido. Como o executável nativo é gerado por um compilador JIT, ele é chamado *jitted code*.

Além da compilação no momento da execução, é possível pré-compilar um Assembly para distribuí-lo utilizando a ferramenta ngen.exe. Essa ferramenta é diferente do .NET Native já comentado, pois o código pré-compilado com o ngen.exe ainda necessita do .NET instalado no computador do usuário. A biblioteca de classes do .NET (BCL - Base Class Library), instalada nos computadores dos programadores ou dos usuários, é formada por código já pré-compilado.

Base Class Library (BCL)

A Base Class Library (BCL) inclui muitas classes com seus métodos, tipos de dados e interfaces com outros componentes de sistema, os quais podem ser utilizados pelo desenvolvedor para construir seus softwares de maneira muito mais prática. Como veremos no Capítulo 2, as classes estão contidas em módulos que podem ser importados e chamados seguindo um *namespace* (espaço de nome), como System.Windows.Forms, o qual para criar controles de janelas gráficas. O conhecimento de uma ampla biblioteca de uma linguagem de programação e seus usos é uma das coisas que, na verdade, toma mais tempo quando se aprende uma nova linguagem de programação, e não sua sintaxe e semântica.

Linguagens de programação para .NET

Existem linguagens que são suportadas diretamente pelas ferramentas da Microsoft e outras indiretamente. Normalmente, sem nenhuma adição, o .NET Framework e o Visual Studio suportam as linguagens Visual Basic, Visual C#, Visual F#, Visual C++ e JScript. O uso de "Visual" diante do nome dessas linguagens significa que elas podem ser utilizadas para se programar interfaces gráficas.

Outras linguagens, como IronRuby e IronPython, também podem ser utilizadas. Com a instalação de extensões para o Visual Studio, ele suportará o uso de Python, por exemplo. Existem outras linguagens, como J#, que permite gerar Assemblies a partir do Java, que não são suportadas diretamente pelas ferramentas da Microsoft, mas geram código compatível com o CLR.



CONEXÃO

Instalação do .NET

Quanto mais nova a versão de seu Windows, maior a chance de ter o .NET Framework mais recente. Se precisar instalar uma versão do .NET, acesse:

<<https://msdn.microsoft.com/pt-br/library/5a4x27ek%28v=vs.110%29.aspx>>

1.1.3 Edições do Visual Studio

O Visual Studio é um Ambiente de Desenvolvimento Integrado (IDE - Integrated Development Studio) da Microsoft, o qual pode ser utilizado com várias linguagens de programação, acessando facilmente os recursos do .NET Framework. É uma ferramenta comercial, paga para ser utilizada em grandes empresas. Há alguns anos, a Microsoft vem oferecendo para desenvolvedores individuais uma versão gratuita chamada Visual Studio Express, a qual possui alguns recursos limitados e separa, em diferentes arquivos de instalação, suas versões para desenvolvimento para desktop tradicional, desktop no novo modelo do Windows 8 ou smartphone e para desenvolvimento web.

As versões pagas do Visual Studio são destinadas a empresas, as quais devem escolher uma versão apropriada para sua equipe de desenvolvimento:

- **Visual Studio Professional:** destinado a equipes que possuem necessidades mais básicas de desenvolvimento, permitem o uso de ferramentas para desenvolvimento, apuração e implantação de código;
- **Visual Studio Premium:** adiciona às ferramentas da versão Professional outras destinadas a testes automatizados de software, análise desses testes e conexão facilitada a banco de dados;
- **Visual Studio Test Professional:** oferece ferramentas de controle de qualidade de software, execução de testes manuais e integração com outras versões do Visual Studio em um fluxo de trabalho da equipe;
- **Visual Studio Ultimate:** possui todas as ferramentas criadas pela empresa, inclusive para a criação de diagramas UML para engenharia de software e gerenciamento de processos de desenvolvimento.



CONEXÃO

Uma tabela muito prática comparando versões do Visual Studio pode ser acessada no link:

<<http://www.visualstudio.com/pt-br/products/compare-visual-studio-products-vs.aspx>>

A partir de 2015, a Microsoft passou a ofertar uma versão denominada Visual Studio Community, a qual também é gratuita, mas engloba em apenas uma instalação as bibliotecas para desenvolvimento voltado a desktop, dispositivos móveis e web. Como há a possibilidade que esse se torne o modelo de distribuição gratuito do Visual Studio nos próximos anos, essa será a versão utilizada neste livro. No entanto, o Visual Studio Express 2013 permanecerá disponível para download durante tempo não determinado, podendo até não vir a ser extinto.

Para consultar as opções disponíveis do Visual Studio Express, acesse o endereço <http://microsoft.com/express>, o qual o direcionará para a página atual com as diferentes versões dos produtos. Se optar pelo Visual Studio Express 2013, há quatro alternativas, sendo que você deve utilizar o **Visual Studio Express para Desktop**. As outras opções, que não serão abordadas nesse livro, são: Visual Studio Express 2013 para Web, Visual Studio Express 2013 para Windows (esse serve para desenvolvimento para a interface do tipo Windows 8, aquela em que são apresentados em quadrados coloridos em sua tela inicial) e o Visual Studio Team Foundation.

Optando entre versões Community ou Express

Há um balanço envolvido nessa escolha: a versão Community agora é a versão mais completa entre as gratuitas, a qual pode ser utilizada para explorar e experimentar vários recursos, mas, logicamente, essa instalação ocupará bastante espaço no seu disco rígido. Caso você tenha limitação de espaço em disco, talvez seja melhor instalar a versão Visual Studio Express para Desktop por enquanto.

Apesar de o Visual Studio Community oferecer muitas funcionalidades do programa completo, como desenvolvimento para diversas plataformas, ele é gratuito apenas para desenvolvedores individuais, empresas com projetos de código aberto ou organizações acadêmicas. O link direto para essa versão é:

<http://www.visualstudio.com/pt-br/products/visual-studio-community-vs>

As versões gratuitas disponíveis do Visual Studio e seus links podem mudar de tempos em tempos, portanto, sempre que precisar, pesquise pela mais adequada para sua

necessidade. É bastante provável que sempre haja uma versão gratuita. Mais do que isso, desde 2014 a Microsoft está passando por uma grande transformação estratégica e pode vir a oferecer sua plataforma de desenvolvimento em outros sistemas operacionais, com parte das bibliotecas tendo seu código aberto.

1.1.4 Instalando o Visual Studio

Optando por instalar o Visual Studio Community do link indicado anteriormente, você baixará do site um pequeno arquivo executável **vs_community.exe** (caso instale a versão Express, o executável será o **wdexpress_full.exe**) Esse é um instalador que, caso executado, perguntará onde quer instalar em seu computador e quais componentes adicionais deseja instalar. Você pode instalar com todas as opções padrão, mas os componentes adicionais não são necessários no decorrer deste livro - ainda assim você pode deixar essas opções marcadas caso queria experimentá-las, considerando que download e instalação serão mais longos.

Lembre-se que é necessário ter as bibliotecas do Framework .NET em seu computador para que não haja problemas. Também garanta que o Windows esteja atualizado com todas as correções de segurança. Isso aumentará a probabilidade que a instalação ocorra sem problemas. A instalação pode ser demorada, portanto reserve um tempo para isso.

Você ainda usa Windows XP?

Se o seu computador usa o Windows XP, pode ter dificuldades em instalar e utilizar o Visual Studio Express 2013. Nesse caso, tente utilizar o Visual Basic 2010 Express, se ele ainda estiver disponível para download no rodapé da mesma página. Entretanto, a versão 2010 da IDE pode deixar de existir em algum momento.

É importante notar que o Windows XP deixou de ter suporte da Microsoft em 8 de abril de 2014 e não receberá mais atualizações. Se você continuar utilizando essa versão do sistema operacional, cada vez mais softwares deixarão de ser compatíveis com seu computador e, além disso, estará bastante vulnerável a vírus e outros malwares.

Para que a interface do Visual Studio seja configurada para usar a língua portuguesa, é necessário baixar e instalar separadamente o pacote da língua. Esse pacote está disponível no seguinte link:

<http://www.visualstudio.com/pt-br/downloads/download-visual-studio-vs.aspx>

Entre as diversas opções, é necessário clicar no menu do item correspondente ao Visual Basic Community (ou Express Desktop, caso o tenha escolhido), que mostrará o quadro da imagem a seguir. Tenha certeza que você escolheu o item correto, ou o procedimento pode falhar.

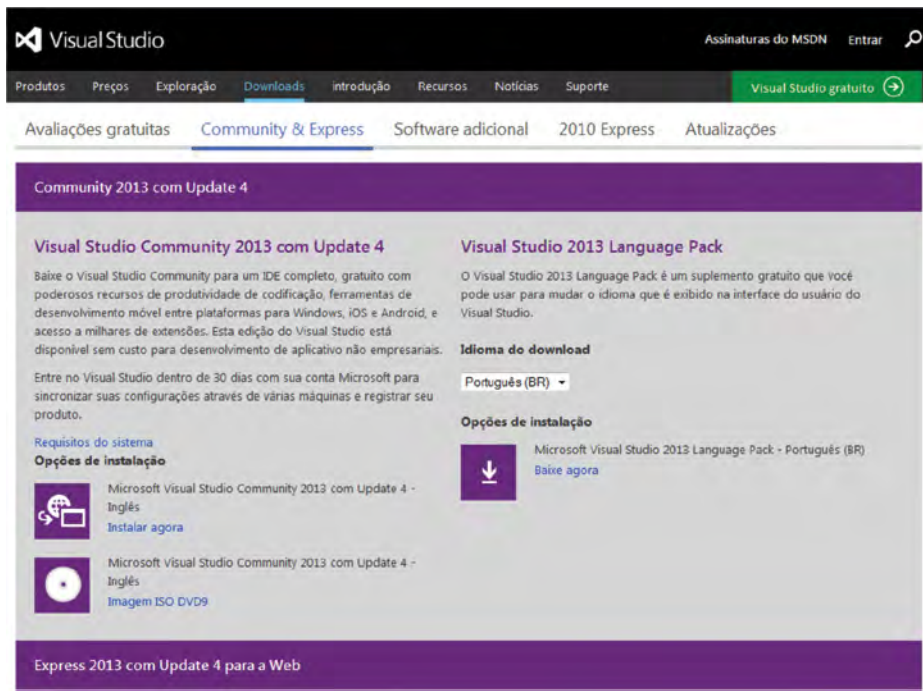


Figura 1.2 – Página de download do Visual Studio.

No lado direito, há uma caixa de seleção que oferece todas as línguas disponíveis para a interface. Escolha o português brasileiro e em seguida clique em "Baixar agora". Como resultado, um grande arquivo chamado `vs_langpack.exe` será baixado. Se o seu Visual Studio estiver aberto, feche o programa e execute o pacote de língua portuguesa para instalação.

Após a instalação do pacote de língua portuguesa, abra novamente o Visual Studio, vá ao item de menu "TOOLS > Options". Após a janela de opções abrir, entre os itens de "Environment" que está aberto, clique em "International Settings". Na caixa de opções de línguas haverá o português. Após aplicar essa mudança, feche o Visual Studio e o abra de novo. Agora ele estará com a interface traduzida.

1.1.5 Abordagem Inicial ao Visual Studio

Como um contato inicial com o VB.NET e com o Visual Studio, vamos fazer nosso primeiro pequeno e simples programa, um "Olá, Mundo!". Com ele, percorreremos os passos de criar um projeto, escrever um programa e executá-lo. Depois disso, usaremos esse programa bastante simples para salientar algumas características da sintaxe do VB.NET e aproveitaremos para descrever de forma geral alguns elementos da IDE Visual Studio.

O programa "Olá, Mundo!" será executado no console, e não em uma janela de interface gráfica. Apenas faremos um desenvolvimento de interface no capítulo 5 e, até lá, usaremos programas de linhas de comando no console para ilustrar exemplos de código. O console, também conhecido como terminal, é a forma de comunicação escrita mais básica com o sistema operacional, no qual o computador e usuário trocam dados entre si, um por vez, em uma tela de texto puro.

1.1.5.1 Iniciando um Projeto

Quando o Visual Studio é aberto, uma página de início é apresentada. Os projetos podem ser criados ou abertos a partir dos links nessa página, ou a partir dos itens no menu "ARQUIVO", acima e à esquerda da tela.

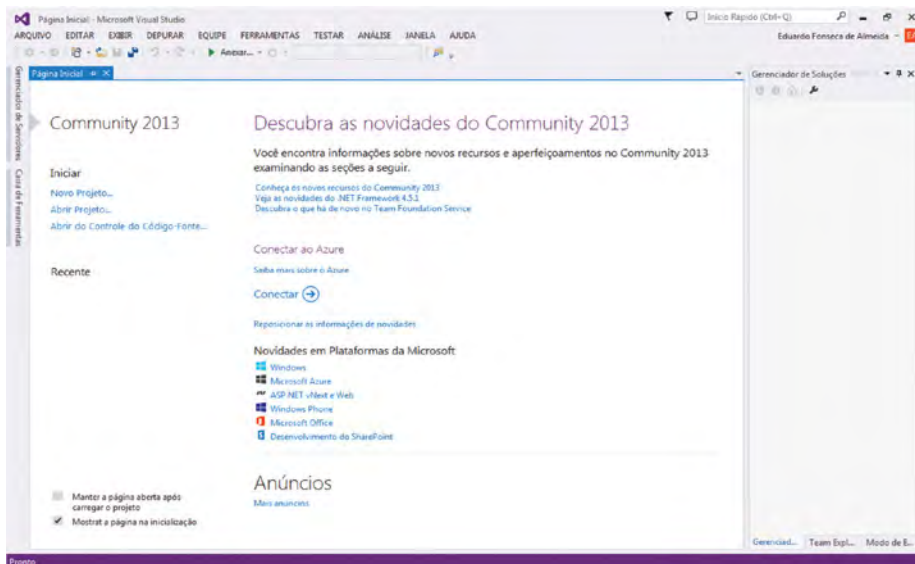


Figura 1.3 – Primeira tela ao abrir o Visual Studio, com a interface em português.

Através da tela inicial ou do item de menu "ARQUIVO", escolha a opção "Novo Projeto...". Uma janela de opções será aberta, como na figura a seguir.

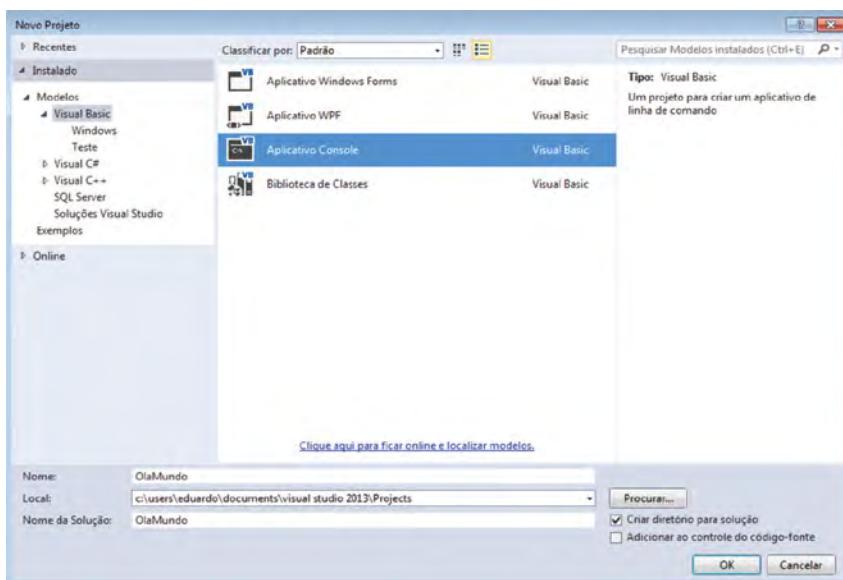


Figura 1.4 – Janela de opções para criação de um novo projeto.

A imagem mostra algumas opções e configurações para um novo projeto. Escolha as seguintes opções nessa tela:

1. Na parte esquerda, escolha "Visual Basic", pois faremos nosso programa nessa linguagem;
2. Na parte central da tela, escolha "Aplicativo Console", pois nosso programa será voltado à tela de texto do sistema operacional;
3. Na parte inferior da tela, no campo de "Nome", escreva "OlaMundo", onde originalmente deverá estar escrito algo como "ConsoleApplication1".

Veja que há uma opção marcada para criar um diretório para a solução. Deixe essa opção marcada, pois todos os arquivos do projeto serão colocados nesse diretório. A denominação "Solução" utilizada pelo Visual Basic é uma forma de organização do código. A maioria das linguagens e IDEs possuem essas formas de organizar códigos em arquivos e diretórios, mas os nomes e hierarquias podem variar para cada uma. O Visual Studio utiliza a estrutura Solução > Projeto > Módulo, sendo que uma solução pode ser um conjunto de projetos e os projetos conjuntos de módulos.

Após clicar em "OK", o Visual Studio criará as configurações de seu projeto. Isso pode levar um tempo, dependendo da capacidade de processamento de seu computador. Será criado e aberto um arquivo chamado "Module1.vb" com um modelo básico de código já escrito, como na figura abaixo. Nessa figura, o zoom está em 200% para que o código fique mais visível.

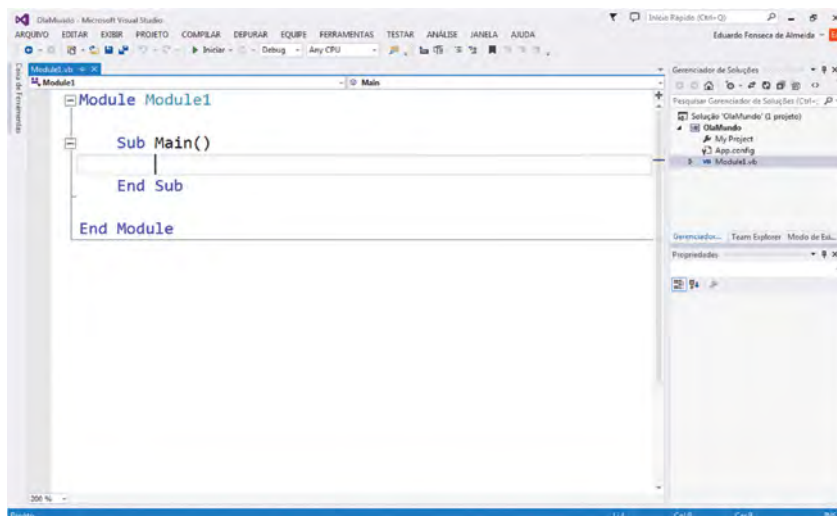


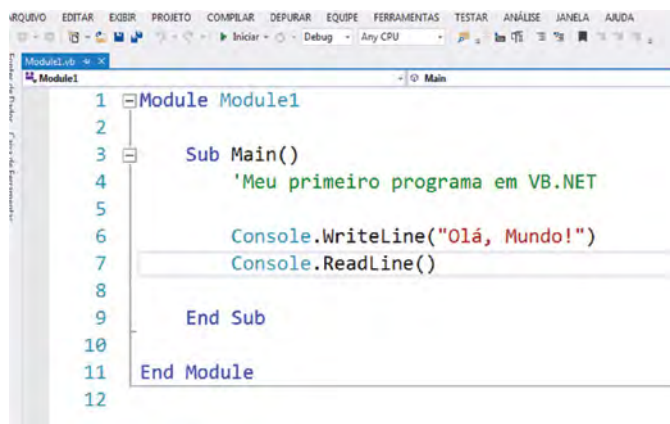
Figura 1.5 – Arquivo de código Module1.vb após a criação do projeto.

1.1.5.2 Primeiro Programa em Visual Basic: Olá, Mundo!

No modelo de código criado automaticamente, vemos a criação de um Módulo e um Sub-procedimento (ou Sub-rotina). Essas também são formas de organizar o código em elementos, as quais podem ser encontrados em outras linguagens de programação com nomes diferentes - mais adiante isso será explicado em maiores detalhes. Por enquanto, nós vamos criar o programa OlaMundo fazendo com que o Sub-procedimento "Main" imprima a frase "Olá, Mundo!" na tela do console. É comum os programadores utilizarem o termo em inglês: "Sub-procedure".

Para que fique mais fácil fazer a referência direta ao código nesse texto, mude uma configuração do Visual Studio que vai exibir o número das linhas de código. No menu superior, vá até "FERRAMENTAS" e então, clique no último item "Opções". Na janela que se abre, vá até o item "Editor de Texto" da lista à esquerda, então clique no item "Basic> Geral", aninhado a ele - haverá uma caixa de seleção "Números de Linha" que deve ser marcada.

Junto ao código criado automaticamente, no bloco de código da Sub-procedure Main, digite o código adicional como na próxima figura.



```
1 Module Module1
2
3     Sub Main()
4         'Meu primeiro programa em VB.NET
5
6         Console.WriteLine("Olá, Mundo!")
7         Console.ReadLine()
8
9     End Sub
10
11 End Module
12
```

Figura 1.6 – Código do Programa OlaMundo.

Quando alguma linha é adicionada, o Visual Studio automaticamente acrescenta uma indentação (um afastamento da linha para a direita, ou seja, uma tabulação). Essa indentação não é obrigatória na maioria das linguagens de programação, mas é uma boa prática de programação para que o código fique

mais bem organizado e legível. Em cada bloco de código, seu "miolo" fica uma tabulação adicional à direita. Assim, fica visível que os blocos de código agem como um contêiner para seu trecho.

Da mesma forma que a indentação dos blocos de código, as cores utilizadas para o texto do código servem para facilitar a leitura e compreensão. Essas cores não fazem parte do código em si, mas são uma interpretação que a IDE faz das instruções, palavras-chave e outros elementos. Isto é, se esse arquivo de código for aberto com um editor de textos comum, ele não estará colorido. Além disso, geralmente é possível mudar a configuração da IDE para alterar as cores utilizadas por cada elemento para uma palheta que seja mais adequada para o programador.

Você também deve ter percebido que, enquanto digitava algumas palavras, o Visual Studio exibiu uma caixa com uma listagem de palavras abaixo do local da digitação. Mais adiante voltaremos a isso, mas essa lista é um recurso de auto-completar, em que a IDE faz sugestões de comandos, métodos, classes e variáveis à medida em que se escreve - no Visual Studio, esse recurso é chamado IntelliSense. A IDE percebe o contexto da digitação e apenas dá sugestões adequadas à situação. Por exemplo, no momento em que se digita um método de uma classe, ela não fará sugestões que contenham variáveis declaradas no código fora dessa classe.

Há mais uma característica do Visual Basic e do Visual Studio que você pode vir a perceber: a linguagem é menos restritiva que outras quanto à sua sintaxe, e alguns erros são ignorados. Por exemplo, em alguns casos a troca de uma letra maiúscula por uma minúscula não acarretará em erros. Mas, mesmo assim, o Visual Studio pode corrigir o problema automaticamente para seguir o estilo da linguagem. Essa e outras características fazem com que ela seja mais amigável ao programador iniciante. No entanto, há erros que não podem ser contornados - se houver um sublinhado vermelho sob algo que você digitou, é porque a IDE detectou erros de sintaxe que devem ser corrigidos antes de compilar.

Após digitar o código e verificar que não há erros, clique no botão de salvamento de projeto . Esse botão salvará todo o seu projeto, o que é necessário para se fazer uma compilação final do código. No entanto, é possível executar o programa através da IDE durante o desenvolvimento sem fazer salvamento do novo código digitado.

1.1.5.3 Sintaxe básica do Visual Basic

A partir de agora, quando não for necessário utilizar imagens da tela do Visual Studio e o foco for o código em si, usaremos quadros com a transcrição do código, como a seguir.

```
OlaMundo.vb
1  Module Module1
2
3      Sub Main()
4          'Meu primeiro programa em VB.NET
5
6          Console.WriteLine("Olá, Mundo!")
7          Console.ReadLine()
8
9      EndSub
10
11 End Module
```


Código 1.1 – Código do programa OlaMundo.

Podemos fazer várias observações importantes sobre o código exibido, das quais algumas voltarão a ser abordadas com mais detalhes em outras partes do livro:

1. Em várias outras linguagens de programação, os blocos de código ficam dentro de chaves { }, mas elas não são utilizadas no Visual Basic. No VB, o bloco se inicia com sua declaração e na maioria das vezes finaliza com um comando "End", seguido do tipo de bloco que é terminado, como "End Sub" no caso de uma Sub procedure;
2. Na linha 4 do código de OlaMundo há um comentário, o qual é ignorado pelo compilador. Ele serve apenas para a leitura humana, como forma de documentar e explicar o código para si e para outras pessoas. Para inserir um comentário no VB, basta adicionar aspas simples antes do texto. Não feche essas aspas de comentário, nem as confunda com o acento agudo;
3. As instruções não terminam com ponto-e-vírgula, como em muitas outras linguagens. Só é possível ter uma instrução por linha. **Instruções** são comandos completos, uma tradução literal da palavra inglesa normalmente utilizada: "*statement*". Um exemplo de instrução é "Console.WriteLine("Olá, Mundo!")". Nas linguagens que usam o ponto-e-vírgula, é possível colocar várias instruções na mesma linha, desde que separadas por esse sinal;

4. A instrução `Console.WriteLine()` é uma **chamada de método** dentro da classe `Console` do .NET. Portanto "Console" é uma classe e "WriteLine()" é um método. O texto "Olá, Mundo!" dentro dos parênteses é uma passagem de argumento para esse método. Nesse caso, diz-se que o texto é um literal, pois ele é utilizado diretamente; em contraste, uma variável pode ser **passada como argumento**. Muitas linguagens utilizam essa forma com pontos entre os nomes de classes e seus métodos ou propriedades, e a isso se denomina notação de ponto.

5. Chamamos de **bloco de código** o trecho que está embutido em uma estrutura de programação. Por exemplo, uma Sub-rotina possui seu próprio trecho de código embutido e, por sua vez, dentro dela pode haver outros trechos de código referentes a uma estrutura de repetição `If...Then`, por exemplo. Cada uma delas é um bloco de código e, para clareza de leitura, seu miolo costuma ser indentado. Há até mesmo linguagens de programação em que essa indentação é obrigatória, mas não é o caso do Visual Basic e da maioria das linguagens.

Para expressar os resultados impressos no console, também usaremos o recurso gráfico de quadros para evitar as imagens de tela. Na barra superior de botões do Visual Studio, clique no botão . Após a execução do código, o texto é impresso na tela que se abre e fica aguardando uma digitação no teclado.

CONSOLE

Olá, Mundo!

Para fechar essa janela do console, basta pressionar a tecla `Enter`, pois o programa aguarda a execução da instrução `Console.ReadLine()` para fechar. Se no código for usada a instrução `Console.ReadKey()`, qualquer tecla pode ser pressionada para fechar o console. Ele também encerra caso clique no X de fechamento da janela. A janela do console também será fechada se, na IDE, for pressionado o botão "Parar", representado por um quadrado vermelho.

Em caso de erro!

Se no momento da compilação e execução do programa o Visual Studio notificou um erro, repasse pelo seu código com bastante atenção. Se você digitou de forma correta, ele tem que funcionar. Procure verificar se digitou as aspas quando deveria, e se pôs o código dentro do bloco `Sub Main()`, e não fora dele.

Quando estiver praticando, evite copiar e colar o código de onde estiver lendo, pois, além de o ato de digitar auxiliar no processo de aprendizado, pode ser que os caracteres, como as aspas, sejam modificados nesse processo de cópia.

Lembre-se que os números de linhas não devem ser digitados no arquivo do código. Eles fazem parte da barra lateral da IDE e servem apenas para facilitar a referência a linhas específicas do arquivo.

1.1.5.4 A Interface do Visual Studio

Ainda com o projeto OlaMundo aberto, vamos aproveitar para descrever rapidamente alguns dos principais elementos da IDE Visual Studio. A imagem a seguir possui uma descrição dos componentes mais importantes quando o Visual Studio se encontra em modo de edição, que é o modo padrão quando se está escrevendo código.

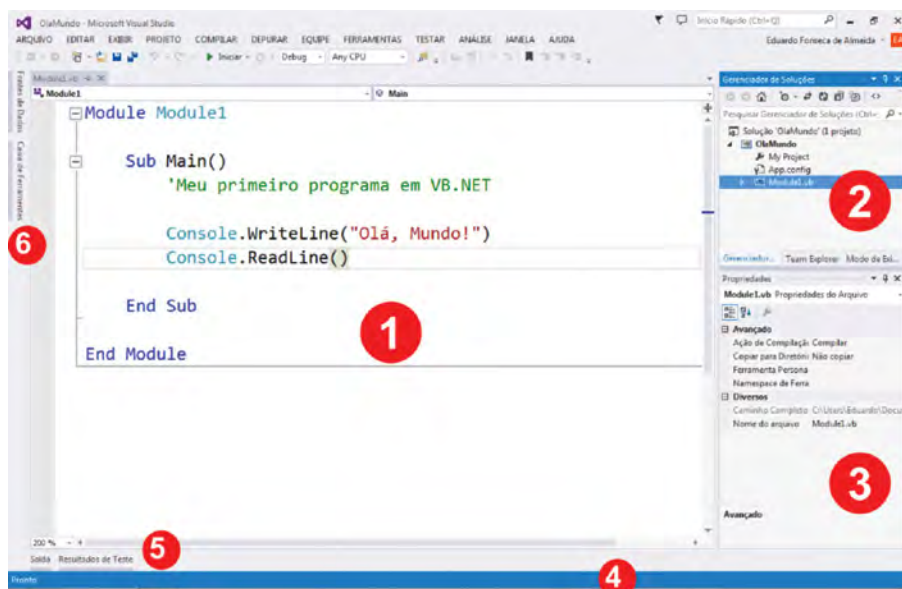


Figura 1.7 – Principais componentes do Visual Studio no modo de edição.

1. **Área Principal:** esse é o local onde os arquivos de código são editados. Acima dessa área e abaixo do menu de opções do Visual Studio há uma aba com o nome do arquivo aberto. Se houver mais arquivos abertos ao mesmo tempo, cada uma de suas abas estará ali. Abaixo do nome do arquivo e acima da área de escrita do código há listas de navegação para os módulos e outras declarações feitas nesse mesmo arquivo.

2. **Gerenciador de Soluções (Solution Explorer):** nesse local você pode ver a estrutura de sua "Solução", com módulos e arquivos listados. Quando abrir um arquivo de Solução que foi salva e fechada anteriormente, você deverá clicar nesse local, no arquivo que pretende editar, para ver seu código. O que é "Solução"? É a forma que o .NET impõe para a organização de código: **módulos** estão dentro de **arquivos**, que por sua vez estão dentro de soluções, as quais são, portanto, conjuntos de arquivos e módulos. Outras linguagens de programação costumam organizar o código com outros nomes, seguindo uma certa filosofia de programação.

3. **Propriedades:** essa janela exibe propriedades de um elemento do código. Cada vez que se clica em uma parte do código, o conteúdo dessa janela muda para o contexto adequado. Quando estiver desenhando um formulário e, por exemplo, selecionar um campo de texto, essa janela mostrará suas propriedades.

4. **Barra de status:** também é sensível ao contexto, fornecendo informações sobre um elemento selecionado. Quando se clica em uma linha de código, ela indicará a linha e a coluna em que se encontra o cursor.

5. **Menu inferior:** essa barra permite que se abra janelas auxiliares de acordo com seu contexto de trabalho. No modo de edição, como no caso da figura anterior, ele permite abrir a janela de saída de resultados e de erros encontrados durante a compilação e execução. Durante a execução e depuração (*debugging*) do código, outras caixas de resultado são abertas.

6. **Caixa de Ferramentas (Toolbox):** no caso da figura, ela está oculta. Se for aberta, nesse momento estará vazia, pois nosso programa é executado no Console. É nessa caixa que são exibidas as ferramentas para confecção de formulários, como botões e caixas de textos, os quais podem ser arrastados com o mouse. Ela apenas mostrará itens quando se estiver desenhando um programa gráfico com formulários.

1.1.5.5 O Visual Studio no Modo de Depuração

Quando o botão "Iniciar" na barra superior do Visual Studio é pressionado, a IDE entra no modo de depuração ou de compilação. Se, na pequena caixa de opções ao lado desse botão a opção selecionada for "Debug", o Visual Studio não fará a compilação final do programa, mas gerará um arquivo executável de depuração. Como o Visual Basic é uma linguagem compilada, não é possível executar um programa sem algum tipo de compilação.

Você pode encontrar esse arquivo executável de depuração através do Windows Explorer (gerenciador de Arquivos do Windows) na pasta Documentos > Visual Studio 2013 > Projects > OlaMundo > OlaMundo > bin > Debug. Lá haverá um arquivo "OlaMundo.exe", sendo que os nomes das pastas e dos arquivos vão variar de acordo com o nome do projeto e com a versão do seu Visual Studio. Se não estiver enxergando a terminação .exe do arquivo, é porque você deve configurar o Windows Explorer para exibir as terminações de arquivo - para isso, no Windows 7, na janela do Windows Explorer, clique no menu "Organizar > Opções de pasta e pesquisa"; na janela que se abre, vá na aba "Modo de Exibição" e desmarque a opção "Ocultar as extensões dos tipos de arquivo conhecidos".

Atenção! Esse arquivo executável vai funcionar se você clicar duas vezes sobre ele, mas não é o arquivo executável final, que você distribuiria para outras pessoas, já que ele contém muitas informações úteis para a depuração da IDE. Esse arquivo é um pouco maior e pode ser mais ineficiente que o resultante de uma compilação final.

1.1.5.6 Compilação e o Arquivo Executável Resultante

O arquivo executável final para uso e distribuição, chamado em inglês de versão de "Release" (lançamento) ainda precisa ser gerado através da compilação final. Se você ainda não fez uma compilação final de seu programa, a pasta com esse arquivo ainda não existirá.

Para fazer uma compilação final, basta seguir os mesmos passos da execução de depuração, com algumas pequenas diferenças: na caixa de opções ao lado do botão "Iniciar", escolha a opção "Release" - é possível fazer algumas opções mais avançadas, como o tipo de CPU alvo da compilação, mas isso apenas serve para casos mais específicos. Após a compilação, será criada uma pasta que contém o arquivo executável final, que pode ser encontrado em Documentos > Visual Studio 2013 > Projects > OlaMundo > OlaMundo > bin > Release.

Perceba que esse arquivo é menor que o gerado pela depuração. A diferença é pequena para nosso exemplo, pois o programa que desenvolvemos também é pequeno e simples.

1.2 Programação Básica com VB.NET

Quem está iniciando em programação pode se sentir sobrecarregado de conceitos a aprender. Para quem já está acostumado, através de uma visão geral é possível perceber que as diversas linguagens de programação possuem alguns elementos básicos comuns. É como aprender uma nova língua, como o inglês ou o francês: todas elas possuem substantivos, verbos, adjetivos, advérbios, e as frases possuem sujeito e predicado - o que faz ser difícil aprendê-las é o conjunto total de palavras, o vocabulário. Em programação, a diversidade de palavras é bem menor. O que faz com que se leve tempo para aprender uma nova linguagem de programação e se tornar fluente nela é se acostumar com suas ferramentas, as quais são oferecidas por bibliotecas, funções e métodos embutidos.

Podemos dividir os elementos básicos de programação nos seguintes grupos:

- **Tipos de dados e declaração de variáveis:** maneiras de declarar variáveis, alocando espaço em memória para valores, e quais tipos de dados podem ser utilizados para isso (números inteiros, números decimais, caracteres etc.);
- **Estruturas de decisão e repetição:** como o programa decide seguir por diversos caminhos ou repetir tarefas com base em certas condições;
- **Métodos, Sub-rotinas e Funções:** como compartimentar certas partes do código que representam ferramentas ou ações, para que elas sejam reaproveitadas de forma eficiente pelo programador;
- **Classes e objetos:** também é uma forma de compartimentação e reaproveitamento do código, mas que representa modelos de entidades e tipos de dados que podem ser criados pelo programador.

Logicamente, o último grupo apenas se aplica a linguagens orientadas a objetos, e há muitas que não são, ou seja, não adotam esse paradigma de programação em sua estrutura. Também há linguagens muito simples que não suportam o uso de funções, e essas apenas podem ser utilizadas para programação não estruturada - há linguagens que não suportavam funções, mas que com o tempo passaram a utilizar esse conceito.

1.2.1 Histórico do Visual Basic

O Visual Basic é uma linguagem criada pela Microsoft seguindo a estrutura da linguagem BASIC. Seu primeiro lançamento ocorreu em 1991 e, desde então, já sofreu modificações profundas. Na verdade, programadores mais experientes ainda podem ter uma percepção relativa às primeiras versões, quando essa linguagem era considerada muito simples, sendo indicada para iniciantes ou para a programação de macros (pequenos programas de automatização) nos aplicativos MS Office, mas ela foi inovativa em facilitar o desenvolvimento fácil de interfaces gráficas.

O suporte à versão antiga do Visual Basic se encerrou em 2005, mas ela deu origem a variações utilizadas de diversas formas em produtos da Microsoft e de terceiros. Por exemplo, o Visual Basic for Application (VBA) é utilizado para programar macros para o pacote Office ou outros programas, o VBScript pode ser utilizado para programação voltada para a web e o Visual Basic .NET é uma linguagem moderna e completa, baseada na sintaxe do Visual Basic original, mas com muitas novas características, como a orientação a objetos e o uso do .NET Framework, tornando-as semelhante ao C#, outra linguagem criada pela Microsoft, em termos de funcionalidades. As mudanças são tão grandes que pode ser questionado se essa continua sendo a mesma linguagem, apesar das semelhanças em termos de sintaxe.

Uma das principais características dessa linguagem é que ela é utilizada, desde sua origem com o Visual Basic antigo, em um contexto de desenvolvimento rápido de aplicativos (RAD - Rapid Application Development), que caracteriza o paradigma de desenvolvimento orientado a eventos. A parte “Visual” do nome da linguagem se refere à forma de desenvolver criando formulários, campos e botões, aos quais são associados trechos de código que são executados quando seus estados são modificados. Atualmente, a principal ferramenta que oferece um ambiente de desenvolvimento integrado (IDE - *Integrated Development Environment*) para o Visual Basic .NET é o Visual Studio, o qual também é utilizado para desenvolver em C#, além de linguagens diversas, como C, C++, Ruby, Python, Javascript etc.

O Índice TIOBE é uma estatística apresentada mensalmente pela empresa holandesa de desenvolvimento de software com o mesmo nome, a qual apresenta um ranking de linguagens de programação mais utilizadas e sua variação em relação a períodos anteriores. Esse ranking utiliza várias fontes de dados para calcular seu índice. Apesar de a popularidade de uma linguagem ser bastante difícil de medir, os sites de notícia usam esse ranking como um indicador. Em 2015, o Visual Basic .NET é a 10ª linguagem mais utilizada, com tendência de crescimento e subida no ranking.

A partir desse ponto, vamos entrar em detalhes de diversos aspectos dos elementos de programação no Visual Basic .NET.

1.2.2 Comentários, variáveis, constantes e tipos de dados

Os comentários são pequenos trechos de texto que são colocados no meio do código apenas para leitura humana. O compilador não os leva em conta quando geram os Assemblies, por isso não interferem no funcionamento ou desempenho dos programas. Eles servem apenas para documentação do código, seja para outros programadores saberem mais facilmente do que se trata certo trecho ou o para o próprio criador se lembrar porque fez aquilo em determinado momento. Comentar código é considerado uma prática de programação muito importante. Os comentários, no Visual Basic, são determinados por aspas simples antes de uma linha de texto, a qual o Visual Studio colore em verde, por exemplo:

```
'Esse é um comentário em Visual Basic
```

As **variáveis** são espaços em memória que armazenam dados temporariamente para uso na execução de um programa. A declaração de uma variável atribui a ela um nome identificador e um tipo de dado e, durante sua execução, ela recebe um endereço de memória e um espaço de alocação que depende do tipo de dados declarado. No Visual Basic a declaração de uma variável é feita da seguinte maneira:

```
Dim quantidade As Integer = 10
```

A declaração de variáveis será detalhada mais adiante, mas nesse trecho é importante apontar que "quantidade" é o nome identificador da variável, "Integer" é o seu tipo de dados, um número inteiro e 10 é o valor que a variável recebe inicialmente e pode mudar durante a execução do programa.

Uma **constante** é muito semelhante a uma variável em todos seus aspectos, exceto em que o valor que ela recebe no início do programa não pode mudar em toda sua execução, e essa atribuição de valor só pode ser feita quando ela é declarada. Um exemplo de constante seria seu uso para um software de simulações físicas em que é necessário utilizar o número Pi com precisão de dez casas decimais:

```
Const pi As Double = 3,14159265359
```

Os **tipos de dados** são aqueles atribuídos as variáveis e constantes em sua declaração, os quais definirão se elas podem receber números, textos ou objetos. Nos últimos exemplo, "quantidade" é uma variável que apenas pode receber números inteiros, enquanto "pi" pode receber números reais com um certo número de casas decimais. As linguagens de programação tipificam variáveis e constantes com o objetivo de permitir certas operações para cada tipo, otimizar essas operações, reservar espaço necessário em memória e evitar a introdução de *bugs* no código. Os tipos de dados podem ser divididos em duas categorias relacionadas à forma como utilizam o armazenamento em memória: Tipos de Valor e Tipos de Referência.

Tipos de Valor

Um tipo de valor é aquele tipo que armazena seu próprio dado (MSDN, 2015b). Os tipos de dados que se encaixam nessa categoria são os dados numéricos, os booleanos, os caracteres (Char), datas, estrutura (Structure, que são definidos pelo próprio programador) e os tipos de enumeração (Enum, que contém um conjunto definido pelo programador). Os Tipos de Valor recebem um endereço de memória e ficam armazenado em um local da memória chamado pilha (*stack*).

Tipos de Referência

Os tipos de referência são mais complexos, sendo, na verdade, conjuntos de tipos de valor ou de outros tipos de referência. Nessa categoria se encaixam cadeias de caracteres (*String*), vetores (*arrays*), matrizes e classes. Os tipos de referência não armazenam dados diretamente; eles são apontadores, os quais são armazenados na pilha da memória e apontam para os endereços de dados que o constituem e passam a ficar armazenados na área de memória chamada *heap*.

Os tipos de dados do Visual Basic são comuns para todas as linguagens que utilizam o .NET Framework, já que são definidas por ele em seu Common Type System. Alguns deles são utilizados muito mais frequentemente que outros. A adequação de cada um desses tipos se dá pela necessidade de valores específicos a serem manipulados e pela quantidade de memória que utilizam. Na tabela a seguir, são apresentados os tipos de dados utilizados na linguagem.

TIPO VISUAL BASIC	DESCRIÇÃO / VALORES PERMITIDOS
Boolean	True ou False
Byte	0 a 255 (não sinalizado), ocupa 1 byte de memória.
Char	0 a 65535 (não sinalizado), ocupa 2 bytes.
Date	Data, pode ir da meia-noite de 1/1/0001 até 23h59min59s de 31/12/9999. Ocupa 8 bytes.
Decimal	0 até +/-79.228.162.514.264.337.593.543.950.335 se não for usado valor decimal. Se for um valor decimal, pode ter até 28 casas depois da vírgula (+/-7,9228162514264337593543950335), utiliza 16 bytes.
Double	-1,79769313486231570E+308 até -4,94065645841246544E-324 para valores negativos; 4,94065645841246544E-324 até 1,79769313486231570E+308 para valores positivos. (valores em notação científica), ocupa 8 bytes.
Integer	-2.147.483.648 até 2.147.483.647 (sinalizado), usa 4 bytes.
Long	-9.223.372.036.854.775.808 até 9.223.372.036.854.775.807 (sinalizado), ocupa 8 bytes.
Object	Ele se referencia a qualquer outro objeto, podendo ser um tipo de referência ou um tipo de valor.
SByte	-128 a 127 (sinalizado), utiliza 1 byte.
Short	-32.768 a 32.767 (sinalizado), utiliza 2 bytes.
Single	-3,4028235E+38 até -1,401298E-45 para valores negativos; 1,401298E-45 até 3,4028235E+38 para valores positivos. (valores em notação científica), usa 4 bytes.
String	0 até aproximadamente 2 bilhões (2 ³¹) de caracteres Unicode
UInteger	0 até 4.294.967.295 (não sinalizado), utiliza 4 bytes.
ULong	0 até 18.446.744.073.709.551.615 (não sinalizado), ocupa 8 bytes.
UShort	0 até 65.535 (não sinalizado), ocupa 2 bytes.

Tabela 1.1 – Tipos de dados do .NET Framework.

A decisão de uso dos tipos numérico depende da necessidade do software em que a variável é utilizada. A ideia por trás de tantos tipos é utilizar aqueles que ocupam a menor quantidade de memória, a menos que haja necessidade de números maiores ou mais precisos. O programador deve determinar os valores máximos possíveis para uma variável em seu contexto. Por exemplo, não

faz sentido utilizar uma variável do tipo Long para enumerar os funcionários de uma empresa. Por outro lado, o tipo Double pode ser apropriado para softwares científicos ou de engenharia.



CONEXÃO

Uma tabela com links para especificações e informações sobre o uso de memória para cada tipo pode ser consultada em:

<http://msdn.microsoft.com/pt-br/library/47zceaw7.aspx>

O tipo *Char* serve para armazenar apenas um caractere alfanumérico, isto é, ele suporta letras, símbolos e números, mas esses últimos são armazenados na forma de texto, não podendo sofrer operações matemáticas. Para poder armazenar caracteres no padrão Unicode, que suporta símbolos de diversas línguas, o tipo *Char* ocupa 2 *bytes* de memória.

O tipo *String* também serve para armazenar texto, mas ele é um Tipo de Referência, isto é, um ponteiro que tem o propósito de receber uma cadeia de caracteres que forma palavras e frases. Logicamente, a quantidade de memória utilizada por ele depende da quantidade de caracteres que o compõe.

Além desses tipos citados, há o tipo *Structure*, que é um conjunto de membros pode ser declarado pelo programador e pode conter diversos elementos de outros tipos, como uma lista. Por exemplo, com esse tipo é possível criar uma estrutura que guarde nome, cargo e ramal de funcionários de uma empresa, e esses dados ficarão juntos na mesma variável. Como será visto mais adiante, isso pode ser feito também com uma classe na orientação a objetos, mas uso de memória de uma *Structure* é diferente.

1.2.3 Declaração e atribuição de variáveis

O comando para declaração de variáveis no Visual Basic é o Dim. O equivalente a esse comando em muitas outras linguagens, como Java, javascript, C e C++ é o "var". Esse é um exemplo com uma instrução de declaração de variável no VB:

```

Exemplo.vb
1  Module Module1
2
3      Sub Main()
4
5          Dim x As Integer
6
7          x = 5
8          Console.WriteLine(x)
9          Console.ReadLine()
10
11     EndSub
12
13 End Module

```

Código 1.2 – Exemplo com declaração de variável.

Se executar esse código, o resultado será o valor 5 impresso na tela do console:

CONSOLE

5

Dim é uma abreviação de *Dimension*, ou Dimensionar. A declaração "Dim x As Integer", em uma tradução para o português, quer dizer algo como "Dimensione a variável x como um número inteiro". Isto é, após a declaração, a variável x apenas pode assumir valores inteiros, sejam negativos, zero ou positivos. Na linha 7 do código, foi atribuído o valor 5 à variável x. Na linha seguinte, ao método WriteLine() é passado x como argumento, o que imprime o seu valor atribuído na tela do Console.

1.2.4 Palavras-chave do VB.NET

Em geral, as linguagens de programação possuem um conjunto de palavras reservadas que não podem ser utilizadas para nomear variáveis, métodos e classes. Essa restrição ocorre porque essas palavras possuem um uso interno, isto é, são utilizadas como nomes de comandos ou de propriedades internas, e por isso não podem receber outras atribuições. Cada linguagem de programação possui um conjunto diferente de palavras reservadas. No Visual Basic, o seguinte código implicará em erro:

Dim Long as Integer

O nome "*Long*" se refere a um tipo de dado e, portanto, é uma palavra reservada que não pode ser utilizada para nomear uma variável. O exemplo de declaração de Sub-rotina a seguir também não é permitido:

```
Sub Module()  
    Console.WriteLine("A palavra Module é reservada.")  
End Sub
```

O uso dessas palavras implicará em um aviso do seguinte erro: "A palavra-chave não é válida como identificador". Na lista de erros, acompanhando essa frase, será indicada a linha em que foi feita essa declaração inválida. Uma lista com as palavras reservadas do Visual Basic é apresentada na tabela a seguir.

AddHandler	Declare	Integer	Partial	ULong
AddressOf	Default	Interface	Private	UShort
Alias	Delegate	Is	Property	Using
And	Dim	IsNot	Protected	When
AndAlso	DirectCast	Let	Public	While
As	Do	Lib	RaiseEvent	Widening
Boolean	Double	Like	ReadOnly	With
ByRef	Each	Long	ReDim	WithEvents
Byte	Else	Loop	REM	WriteOnly
ByVal	Elseif	Me	RemoveHandler	Xor
Call	End	Mod	Resume	Const
Case	EndIf	Module	Return	Else
Catch	Enum	MustInherit	SByte	Elseif
CBool	Erase	MustOverride	Select	End
CByte	Error	MyBase	Set	If
CChar	Event	MyClass	Shadows	=
CDate	Exit	Namespace	Shared	&
CDec	False	Narrowing	Short	&=
CDBl	Finally	New	Single	*
Char	For	Next	Static	*=
CInt	Friend	Not	Step	/
Class	Function	Nothing	Stop	/=
CLng	Get	NotInheritable	String	\
CObj	GetType	NotOverridable	Structure	\=
Const	GetXMLNames- pace	Object	Sub	^
Continue	Global	Of	SyncLock	^=
CByte	GoSub	On	Then	+

CShort	GoTo	Operator	Throw	+=
CSng	Handles	Option	To	-
CStr	If	Optional	True	-=
CType	If()	Or	Try	>>
CUInt	Implements	OrElse	TryCast	>>=
CULng	Imports	Overloads	TypeOf	<<
CUShort	In	Overridable	Variant	
Date	Inherits	Overrides	Wend	
Decimal		ParamArray	UInteger	

Tabela 1.2 – Lista de palavras reservadas do Visual Basic .NET. Fonte: MSDN (2015).

À medida que uma linguagem de programação possui novas versões, às quais podem ser adicionadas novas funcionalidades e instruções, novas palavras reservadas podem ser adicionadas a essa lista. Os programadores devem se inteirar a respeito dessas atualizações.

CONEXÃO

A lista de palavras reservadas no site da Microsoft está no link:

[<http://msdn.microsoft.com/en-us/library/ksh7h19t\(v=vs.90\).aspx>](http://msdn.microsoft.com/en-us/library/ksh7h19t(v=vs.90).aspx)

1.2.5 Operadores

Os operadores são palavras ou sinais utilizados para realizar operações de soma, comparação ou testes lógicos, entre muitas outras, atuando diretamente com os dados ou com as variáveis que os representam. Todas as linguagens de programação possuem operadores, mas alguns deles se diferenciam entre elas. A seguir são apresentados os operadores disponíveis para as linguagens do .NET.

Operadores aritméticos

São os operadores que realizam operações matemáticas básicas. Obviamente, eles apenas podem ser utilizados com tipos numéricos.

OPERADOR	DESCRIÇÃO
+	Adição
-	Subtração
*	Multiplicação
/	Divisão
\	Parte inteira do resultado da divisão Exemplo: 1.2 \ 1 = 1
Mod	Resto do resultado da divisão Exemplo 1: 1.2 Mod 1 = 0.2 Exemplo 2: 5 Mod 2 = 1
^	Eleva à potência de um valor Exemplo: 3 ^ 2 = 9

Tabela 1.3 – operadores aritméticos das linguagens .NET.

Operadores de concatenação

Há apenas dois operadores de concatenação que executam exatamente a mesma operação, que é juntar duas cadeias de caracteres em uma só.

OPERADOR	DESCRIÇÃO
&	Concatena duas Strings
+	Concatena duas Strings (semelhante ao & quando aplicado a Strings)

Tabela 1.4 – operadores de concatenação das linguagens .NET.

O trecho de código a seguir exemplifica a concatenação de *Strings* armazenadas em duas variáveis.

```
Dim a As String = "Uma frase "  
Dim b As String = "inteira."  
Console.WriteLine(a & b)
```

O resultado desse trecho de código será a impressão, na tela do console, do texto "Uma frase inteira.". Quando se lida com Strings, essa operação é bastante comum.

Operadores de deslocamento de *bits*

Essas operações são usadas em casos específicos em que é necessário fazer diretamente a manipulação de *bits*. É de mais difícil compreensão, caso não tenha que utilizá-las.

OPERADOR	DESCRIÇÃO
<<	Desloca uma quantidade de bits à esquerda de um valor em bits, adicionando zeros à direita.
>>	Desloca uma quantidade de bits à direita de um valor em bits, eliminando bits à direita.

Tabela 1.5 – operadores de deslocamento de bits das linguagens .NET.

Por exemplo, o número decimal 10 equivale ao binário 1010. Se for realizada a operação:

`10 << 1`

O resultado será 20, já que seu binário terá um zero acrescentado à direita e se tornará 10100. A operação de deslocamento de *bits* à esquerda equivale a multiplicar o decimal por potências de 2, por características do sistema binário. Outro exemplo: `10 << 2 = 40`, portanto igual a 10×2^2 .

O deslocamento para a direita equivale a dividir um decimal por potência de 2 e subtrair sua sobra. Por exemplo, `10 >> 1 = 5`, já que resultará no binário 101 e do cálculo equivalente a $10/2$. Outro exemplo: `35 >> 1 = 17`, pois 35 equivale ao binário 100011 e 17 equivale a 10001.

Operadores de atribuição

Os operadores de atribuição são aqueles relacionados às instruções de atribuição de valores às variáveis. O de uso mais comum é o de sinal de igualdade (=), mas outros também são amplamente utilizados.

OPERADOR	DESCRIÇÃO	EXEMPLO
=	Atribui um valor a uma variável ou constante	variável = 10
+=	Adiciona e atribui o valor	variável += 10 <i>equivale a</i> variável = variável + 10
-=	Subtrai e atribui o valor	variável -= 10 <i>equivale a</i> variável = variável - 10
*=	Multiplica e atribui o valor	variável *= 10 <i>equivale a</i> variável = variável * 10
/=	Divide e atribui o valor	variável /= 10 <i>equivale a</i> variável = variável / 10
\=	Divide e atribui a parte inteira da divisão	variável \= 10 <i>equivale a</i> variável = variável \ 10

OPERADOR	DESCRIÇÃO	EXEMPLO
<code>^=</code>	Eleva à potência de um valor e atribui o resultado	<code>variável ^= 10</code> equivale a <code>variável = variável ^ 10</code>
<code><<=</code>	Desloca a quantidade de bits à esquerda de um valor em bits, acrescenta zeros à esquerda e atribui o valor	<code>variável <<= 10</code> equivale a <code>variável = variável << 10</code>
<code>>>=</code>	Desloca a quantidade de bits à direita de um valor em bits, acrescenta zeros à direita e atribui o valor	<code>variável >>= 10</code> equivale a <code>variável = variável >> 10</code>
<code>&=</code>	Concatena e atribui o resultado	<code>variável &= 10</code> equivale a <code>variável = variável & 10</code>

Tabela 1.6 – operadores de atribuição das linguagens .NET.

Os operadores de adição e atribuição (`+=`) ou de subtração e atribuição (`-=`) são normalmente usados dentro de estruturas de repetição, quando é necessário incrementar um valor a cada laço. Um exemplo de uso simples do operador de adição e atribuição é usado no trecho:

```
Dim a As Integer = 50
a += 10
Console.WriteLine(a)
```

Executando esse código, o valor impresso na tela do console será 60, já que a segunda linha equivale a "`a = a + 10`", que por sua vez é igual a "`a = 50 + 10`".

Operadores de comparação

Os operadores de comparação servem para comparar grandezas numéricas, conteúdo de cadeias de caracteres, valores lógicos, atribuições de objetos. Se o teste de comparação de dois valores ou variáveis for realizado em um teste lógico, o resultado será sempre `True` ou `False`.

OPERADOR	DESCRIÇÃO
<code><</code>	Menor que
<code><=</code>	Menor ou igual
<code>></code>	Maior que
<code>>=</code>	Maior ou igual
<code>=</code>	Igual
<code><></code>	Diferente
<code>Is</code>	Retorna <code>True</code> caso os dois nomes de objetos (ponteiros) apontem para o mesmo objeto. Retorna <code>False</code> caso contrário.

OPERADOR	DESCRIÇÃO
IsNot	Retorna True caso os dois nomes de objetos (ponteiros) não apontem para o mesmo objeto. Retorna False caso contrário.
Like	Retorna True caso um padrão de string esteja contido em uma string. Retorna False caso contrário. Isto é, localiza uma cadeia de caracteres em uma String.

Tabela 1.7 – operadores de comparação das linguagens .NET.

Os casos menos claros para um programador iniciante podem ser os dos operadores Is e IsNot, pois eles comparam valores referenciais. O exemplo a seguir assume que existe uma classe Cliente (se você não criá-la, o programa não funcionará, então por enquanto apenas entenda que Cliente é um objeto - Classes e Objetos serão vistos no Capítulo 3).

```

Sub Main()
    Dim a As New Cliente
    Dim b As Object = a
    Console.WriteLine(a Is b)
    Console.ReadKey()
End Sub

```

O trecho de código está testando se as variáveis "a" e "b" apontam para o mesmo objeto. Como "b" recebeu "a" por atribuição, o resultado do teste será True.

Operadores lógicos

Os operadores lógicos realizam álgebra booleana entre dados de tipo lógico. O resultado desse tipo de operação também será lógico, sempre resultando em True ou False.

OPERADOR	DESCRIÇÃO
And	Executa a operação AND (E) em dois valores booleanos, binários ou numéricos.
Not	Executa a operação NOT (NÃO) em dois valores booleanos, binários ou numéricos.
Or	Executa a operação OR (OU) em dois valores booleanos, binários ou numéricos.
Xor	Executa a operação XOR (Exclusive-OR) em dois valores booleanos, binários ou numéricos. Retorna True se os dois valores forem diferentes e False se forem iguais.
AndAlso	Executa a operação AND (E) em dois valores booleanos, binários ou numéricos, ignorando a avaliação do segundo valor caso o primeiro seja False.
OrElse	Executa a operação OR (OU) em dois valores booleanos, binários ou numéricos, ignorando a avaliação do segundo valor caso o primeiro seja True.

Tabela 1.8 – operadores lógicos das linguagens .NET.

Com isso, foram apresentados de forma geral os operadores das linguagens .NET, além de seus tipos de dados e como declarar variáveis. No próximo capítulo, ao abordar a programação estruturada, veremos como aplicá-los em funções e sub-rotinas.



REFLEXÃO

A escolha de uma linguagem de programação para aprendizado depende de muitos fatores. Se um profissional se insere em uma empresa que adota certo conjunto de ferramentas ou *frameworks*, é imperativo que ele se adapte a esses elementos, os quais podem ter sido escolhidos de forma estratégica. Por outro lado, se o aprendizado não está condicionado às circunstâncias profissionais, ele se dará pelo interesse pessoal por certas características de uma linguagem e suas ferramentas.

O Visual Basic .NET é uma linguagem muito interessante para o aprendizado, pois introduz o estudante ao ecossistema da Microsoft ao mesmo tempo que o faz de forma mais didática que o uso do C#, e esse é um dos interesses da empresa em continuar desenvolvendo fortemente a linguagem. Se houver interesse, após se habituar ao VB.NET, a transição para o C# se torna bem mais fácil.



ATIVIDADES

01. Descreva com suas palavras o que são os *frameworks* de software.
02. O que são as bibliotecas de um *framework* ou de uma linguagem de programação?
03. Explique o que são .NET Framework, Visual Studio e Visual Basic .NET, diferenciando-os entre si.
04. Faça um pequeno programa para o console em Visual Basic, o qual converte 150 centímetros em polegadas e imprime o valor na tela. Uma informação importante é que um centímetro equivale, aproximadamente, a 0,393700787 polegada. Para o fator de conversão, você deve declarar uma constante.



LEITURA

Uma boa fonte de informações para as tecnologias da Microsoft é o seu site da MSDN, a Microsoft Developer Network. Lá há muitos guias e especificações do .NET e de suas linguagens. A maioria das páginas possui tradução para o português, mesmo que tenham sido feitas automaticamente. Dois guias interessantes são os seguintes:

Introdução ao .NET Framework:

<<https://msdn.microsoft.com/pt-br/library/hh425099%28v=vs.110%29.aspx>>

Introdução ao Visual Basic:

<<https://msdn.microsoft.com/pt-br/library/8hb2a397.aspx>>



REFERÊNCIAS BIBLIOGRÁFICAS

ABES - Associação Brasileira das Empresas de Software. **Mercado Brasileiro de Software:**

panorama e tendências, 2014. 1a. ed.. São Paulo: ABES, 2014

DEL SOLE, Alessandro. **Visual Basic 2010 Unleashed**. Sams Publishing: Indianapolis, 2010.

MSDN, Microsoft Developer Network. .NET Framework. Disponível em: <<https://msdn.microsoft.com/pt-br/vstudio/aa496123.aspx>>. Acesso em: 10 fev. 2015.

_____. *Tipos de valor e referência*. Disponível em: <<https://msdn.microsoft.com/pt-br/library/t63sy5hs.aspx>>. Acesso em: 15 fev. 2015.

2

Programação Modular e Programação Estruturada

Este capítulo continuará descrevendo os elementos básicos de programação aplicados à linguagem Visual Basic .NET. Após conhecer como se inicia um projeto, os tipos de dados e como declarar variáveis nessa linguagem, veremos como estruturar programas. Primeiro, serão vistos como dividir o código em módulos e métodos e como as variáveis e os métodos ficam sujeitos a uma delimitação de escopo. Em seguida, veremos como funcionam os controles de fluxo de programação estruturada, os quais se dividem essencialmente em estruturas de decisão e estruturas de controle.

Aqui serão vistos, portanto, alguns dos principais meios de se arquitetar um programa, ainda se restringindo ao paradigma de programação estruturada. Por enquanto, ainda serão utilizados como exemplos apenas programas que interagem com o usuário através de texto na tela do console.



OBJETIVOS

Após a leitura deste capítulo, você será capaz de:

- Separar trechos de código em módulos e métodos
 - Compreender a diferença entre Sub-rotinas e Funções no Visual Basic
 - Programar estruturas de decisão e repetição nessa linguagem
-

2.1 Programação Modular

A Programação Modular é uma técnica de arquitetura de software que separa o código e as funcionalidades dos programas em partes, de forma que cada módulo contenha o que for necessário para desempenhar tais funcionalidades. Um módulo, suas propriedades e seus métodos podem ser acessados por outros módulos. Seu conceito é próximo ao da Programação Estruturada e da Programação Orientada a Objetos, já que esses paradigmas têm como base a compartimentação do código, mas todos esses conceitos se diferenciam em alguma maneira. Os módulos são relacionados à compartimentação em um nível mais elevado do código, já que ele pode conter classes e funções.

2.1.1 Módulos

Os módulos são uma forma de compartimentar partes do código fonte de um programa, dividindo-o em trechos que estão ligados a funções e objetivos comuns (MSDN, 2015b). Quando se divide o código fonte em Módulos, fica muito mais fácil desenvolver suas partes separadamente e acessá-las através de chamadas de métodos. Os módulos têm alguns comportamentos parecidos com Classes (cujo conceito será visto no Capítulo 3) e podem ser invocadas a qualquer momento quando se está escrevendo outros módulos em um programa.

No Visual Basic .NET, a criação de um Módulo se dá pelo uso da palavra-chave `Module`. A sua delimitação final é estabelecida por `End Module`. Além disso, todo módulo recebe um nome, da mesma forma que uma variável em sua declaração. Veja o trecho de código a seguir.

```
arquivo.vb
Module MóduloExemplo

    Sub Main()
        'Aqui entra o código a ser executado
    EndSub

End Module
```

Código 2.1 – Exemplo de declaração de um módulo.

Várias coisas importantes podem ser observadas nesse exemplo de estrutura de um módulo. Uma delas é que o módulo está dentro de um arquivo do Visual Basic (arquivo.vb) e, dentro desse arquivo, há apenas esse módulo o qual recebe um nome `ModuloExemplo`. Quando se cria um novo projeto no Visual Studio, por padrão ele cria um arquivo chamado `Module1.vb` e um módulo chamado `Module1`. Logicamente, esses nomes podem ser alterados, mas é necessário resolver alertas de erros que o Visual Studio possa levantar caso isso seja feito. Além disso, é uma boa prática denominar o Módulo com o mesmo nome que o arquivo em que está contido. É possível colocar dois módulos dentro de um arquivo só, mas o objetivo de se fazer a separação é que outros programadores possam entender facilmente como o código está organizado, então é importante levar isso em conta.

Se você está construindo um programa executável, isto é, que executa por conta própria e geralmente é compilado em um arquivo com extensão `.exe`, dentro de seu módulo deve haver pelo menos um método principal, o `Sub Main()`. Tudo que estiver dentro desse módulo será executado automaticamente quando o módulo for invocado (MSDN, 2015b). Como veremos adiante, um módulo pode conter outros métodos além de `Sub Main()`. No entanto, se você programando uma biblioteca auxiliar, que não executa por conta própria mas é invocada por outros programas executáveis, não há necessidade de uma rotina `Sub Main()`.

Além de um programador poder criar seus módulos, o Visual Basic possui um conjunto de módulos básicos auxiliares cujos métodos podem ser invocados a qualquer momento em um código fonte. Essas bibliotecas possuem não só métodos, mas também atributos e constantes. Tais módulos são descritos de forma resumida no quadro a seguir.

MÓDULO	DESCRIÇÃO
Constants	Contém diversas constantes que são utilizadas para compatibilidade com código Visual Basic antigo.
ControlChars	Contém constantes que representam caracteres de controle para impressão e exibição de texto.
Conversion	Contém métodos para converter números decimais em outras bases, dígitos de números para sequências de caracteres, sequências de números e dados de um tipo para outro.

MÓDULO	DESCRIÇÃO
DateAnd-Time	Contém membros que obtém a data ou hora atual, realizam cálculos de data, retornam uma data ou hora, configuram uma data ou hora e medem a duração de um processo.
ErrObject	Contém informações sobre erros de tempo de execução e métodos para alertar ou eliminar uma mensagem de erro.
FileSystem	Contém métodos que executam operações de arquivo, diretório ou pasta do sistema.
Financial	Contém procedimentos que são usados para cálculos financeiros.
Globals	Contém informações sobre a versão atual do mecanismo de execução de script.
Information	Contém os membros que retornam, testam ou verificam informações como, por exemplo, o tamanho de uma matriz, os nomes de tipos de dados e assim por diante.
Interaction	Contém membros que interagem com objetos, aplicativos e sistemas.
Strings	Contém membros que executam operações em cadeia de caracteres, como pesquisar conteúdo em uma sequência de caracteres, obter o comprimento de uma sequência de caracteres, reformatação e assim por diante.
VBMath	Contém os métodos que realizam operações matemáticas.

Tabela 2.1 – Módulos básicos oferecidos pelo .NET Framework. Fonte: Adaptado de MSDN (2015b).

CONEXÃO

Para saber mais sobre os módulos do .NET, consulte sua página de documentação:
[<https://msdn.microsoft.com/pt-br/library/y76404kz.aspx>](https://msdn.microsoft.com/pt-br/library/y76404kz.aspx)

O pequeno código a seguir mostra um exemplo de uso da propriedade `Now()` do módulo `DateAndTime`. Essa propriedade fornece a data e a hora do sistema operacional no momento em que é executado.

```

datahora.vb
Module Module1

    Sub Main()
        Dim data_hora As String

        data_hora = Now

        Console.WriteLine(data_hora)
        Console.ReadKey()
    EndSub

End Module

```

Código 2.2 – Exemplo de uso do módulo DateAndTime do .NET Framework

Nesse exemplo, seu valor foi passado para uma variável do tipo String, para depois ser impresso na tela do console. Em vez disso, o valor poderia ser passado para uma variável do tipo Date para ser manipulado, por exemplo, calculando a diferença de dias entre duas datas.

CONSOLE

20/02/2015 18:21:43

Para compreender melhor o uso de módulos, é importante também detalhar o que são métodos.

2.1.2 Métodos

Os termos Sub-rotina, Função e Método são praticamente sinônimos no contexto geral das linguagens de programação. Em primeiro lugar, é importante salientar que nesse livro estamos utilizando o termo "Sub-rotina" porque é comum, mas o equivalente em inglês seria "Sub procedure", mesmo que sua tradução literal seja sub-procedimento.

No Visual Basic há diferenças no uso dos três termos. Os Métodos podem ser Funções ou Sub-rotinas, cujas diferenças serão vistas mais adiante. Por sua vez, os Métodos são utilizados para compartimentar código e devem estar contidos em Módulos e Classes. O objetivo dessa modularização é organizar o código fonte, tornar seu desenvolvimento mais eficiente e evitar repetições.

Como exemplo, consideremos o caso de um programador que está desenvolvendo um sistema complexo de mapas e previsão do tempo. Como é um sistema internacional, ele terá que, em diversas de suas interações com os usuários, converter a temperatura entre graus Celsius e graus Fahrenheit (que é utilizado nos EUA e Inglaterra). Se a fórmula de conversão terá que ser utilizada com frequência, o ideal é que ela seja programada de tal maneira que possa sempre ser facilmente acessada, sem ser programada repetidamente. Veja o exemplo de código a seguir.

```
exemplo.vb
1  Module Temperatura
2
3      Function CParaF(ByVal temp_celsius As Decimal)
4          Dim temp_fahrenheit As Decimal
5          temp_fahrenheit = temp_celsius * 1.8 + 32
6          Return temp_fahrenheit
7      EndFunction
8
9      Function FParaC(ByRef temp_fahrenheit AsDecimal)
10         Dim temp_celsius AsDecimal
11         temp_celsius = (temp_fahrenheit - 32) / 1.8
12         Return temp_celsius
13     End Function
14
15 End Module
16
17 Module Module1
18     Sub Main()
19         Dim temp, temp_convertida As Decimal
20         Console.WriteLine("Digite uma temperatura em graus Celsius:")
21         temp = Console.ReadLine()
22         temp = Convert.ToDecimal(temp)
23         temp_convertida = CParaF(temp)
24         Console.WriteLine("A temperatura digitada equivale a {0} Fahrenheit.",
temp_convertida)
25         Console.ReadKey()
26     End Sub
27 End Module
```

Código 2.3 – Exemplo de uso de métodos para acesso em outro módulo.

Apenas para facilitar a visualização do exemplo, dois módulos foram colocados no mesmo arquivo.

O módulo Temperatura não possui uma sub-rotina Sub Main(), mas possui duas funções: CParaF(), que converte uma temperatura de Celsius para Fahrenheit, e FParaC(), que faz a conversão contrária. O Module1, por sua vez, possui a sub-rotina principal Sub Main(), isto é, esse é um programa executável que pede um valor de temperatura em Celsius do usuário e, na linha 23, faz uma chamada do método CParaF() do módulo Temperatura para converter esse valor em Fahrenheit e imprimi-lo na tela.

É importante salientar que normalmente o módulo Temperatura estaria em outro arquivo, e seus métodos poderiam ser invocados durante a programação de qualquer outro módulo, mas o programa de qualquer maneira funciona com os dois módulos juntos. Assim, esses métodos funcionariam como aqueles dos módulos padrões do Visual Basic que foram apresentados no tópico anterior, como os de manipulação de data e hora ou os de operações financeiras. Nesse exemplo, ainda podem ser observadas quatro chamadas de métodos de módulos padrões do Visual Basic: Console.WriteLine(), Console.ReadLine(), Console.ReadKey() e Convert.ToDecimal(). Esse último método converte o valor recebido do usuário como String em um tipo Decimal.

O uso da programação modular e a compartimentação de métodos pode trazer várias vantagens no desenvolvimento de software, principalmente quando se trata de um código fonte extenso e complexo:

- Eliminação de redundância de códigos que precisam ser executados várias vezes em um programa, já que eles podem ser definidos em métodos e ser chamados quando necessários;
- Reaproveitamento dos métodos em outros aplicativos. Para que isso possa ocorrer, é importante que os métodos sejam coesos e básicos, executando apenas uma tarefa simples que possa ser aproveitada de muitas maneiras;
- A modularização dos módulos de um software permite que ele seja mais organizado e legível, de mais fácil correção;
- Com a modularização, é possível que cada programador trabalhe em partes específicas do código, desde que fique especificado como cada método deve ser acessado, seguindo uma forma consistente.

No exemplo do conversor de temperaturas, CParaF() e FParaC() são funções. Então, qual seria a diferença de uma Função e uma Sub-rotina no Visual Basic?

2.1.3 Sub-rotinas e Funções

Funções e Sub-rotinas podem ser definidos, resumidamente, como um bloco de código que pode ser invocado dentro de outros blocos, servindo como meio de organizar e facilitar a compreensão e manutenção desse código, aproveitando trechos que podem ser reutilizados frequentemente.

Sobre os termos, quando se considera as linguagens de programação em geral, "Função" pode ser considerado o termo mais amplo, inclusive sendo esse o motivo de haver um paradigma de programação chamado Funcional. Geralmente, Métodos são Funções que estão dentro de objetos (encapsulados, como veremos adiante) e módulos, mas não é errado utilizá-lo em outros contextos. Sub-rotinas são equivalentes a funções, mas esse nome não é frequentemente utilizado em muitas outras linguagens de programação - no entanto, *procedure* é um nome bastante usado em alguns gerenciadores de bancos de dados, e nada mais é do que uma função que fica embutida em um banco de dados para automatizar suas tarefas.

No Visual Basic, de forma estrita, há uma pequena diferença técnica entre Sub-rotinas e Funções: o primeiro não retorna um valor, enquanto o segundo retorna. Para ilustrar isso, veja o exemplo a seguir. Você pode criar um novo projeto do tipo Console para testá-lo, ou usar um já existente.

```
exemploMetodos.vb
1  Module Module1
2
3      Sub Main()
4          criaMensagem()
5          Console.WriteLine(calculo())
6          Console.ReadKey()
7      End Sub
8
9      Sub criaMensagem()
10         Console.WriteLine("Mensagem da Sub procedure")
11     End Sub
12
13     Function calculo() As Integer
14         Return 5 + 5
15     End Function
16
17 End Module
```

Código 2.4 – Exemplos de Função e Sub-rotina.

Esse código resultará na seguinte impressão na tela do Console.

```
CONSOLE
Mensagem da Sub procedure
10
```

É importante analisar cada trecho do código. Leia atentamente as observações abaixo e estude o código anterior para compreender suas características:

1. Todo o trecho está dentro do módulo denominado **Module1**, mas há uma Sub-rotina e uma Função fora da Sub-rotina **Main()**. Portanto, temos duas Sub-rotinas e uma Função no mesmo nível de código (nenhuma aninhada, dentro de outra);

2. A Sub-rotina **Main()** é a que é chamada quando o programa inicia. As outras apenas são executadas porque são chamadas dentro da **Main()**, uma na linha 4 e outra na linha 5. Se não houvesse, por exemplo, o código da linha 4, a Sub-rotina **criaMensagem()** não seria executada, e na tela do Console seria impresso apenas "10". Para testar isso, transforme a linha 4 em um comentário colocando uma aspa simples antes do texto e então execute o programa novamente;

3. **criaMensagem()** é uma Sub-rotina porque não retorna um valor. Ela executa uma tarefa de impressão na tela com o **Console.WriteLine()**, mas não retorna um valor como resultado da execução. Em outras linguagens de programação, como C, C++ e Java, isso geralmente é feito através da declaração de uma função do tipo "*void*";

4. **calculo()** é uma Função, pois retorna um valor. Ela não imprime diretamente um texto ou valor na tela, então na **Sub Main()** temos que usar o comando **WriteLine()** e passar a própria função **calculo()** como parâmetro. Se a Função **calculo()** for usada da mesma forma que **criaMensagem()**, chamada diretamente e sem o uso de **WriteLine()**, seu valor não será impresso na tela do Console. Faça testes modificando o código para compreender melhor;

5. Em **Function calculo() As Integer**, a função recebe uma atribuição de tipo porque ela vai retornar um valor, que deve ser compatível com esse tipo declarado. Por exemplo, a função **calculo()** não pode retornar uma String.

Logicamente, as funções utilizadas nesse exemplo são extremamente simples, servindo apenas para a compreensão de seu funcionamento. Em geral, as

funções são usadas para tarefas mais complexas de cálculos, transformações de textos, meta-programação (código que modifica código) etc. Por exemplo, as funções **WriteLine()** e **ReadLine()** fazem parte da classe **Console** que pertence ao Framework .NET, e **WriteLine()** pode lidar com qualquer tipo de dado que se peça para imprimir na tela do console, o que requer variações nesse tratamento de dados.

Para explicar mais algumas características das funções, vamos utilizar outro exemplo a seguir.

```
Module1.vb
1  Module Module1
2
3      Sub Main()
4          Console.WriteLine(somador())
5          Console.WriteLine(somador(5))
6          Console.WriteLine(somador(5, 7))
7          Console.ReadLine()
8      End Sub
9
10     Function somador() As String
11         Return "A soma 0 + 0 é igual a 0"
12     End Function
13
14     Function somador(ByVal val1 As Integer) As String
15         Dim soma As Integer = val1 + val1
16         Return String.Format("A soma {0} + {1} é igual a {2}", val1, val1, soma)
17     End Function
18
19     Function somador(ByVal val1 As Integer, ByVal val2 As Integer) As String
20         Dim soma As Integer = val1 + val2
21         Return String.Format("A soma {0} + {1} é igual a {2}", val1, val2, soma)
22     End Function
23
24 End Module
```

Código 2.5 – Exemplo de sobrecarga de função.

O resultado impresso na tela do Console é o seguinte:

CONSOLE

```
A soma 0 + 0 é igual a 0  
A soma 5 + 5 é igual a 10  
A soma 5 + 7 é igual a 12
```

Os pontos que podemos discutir a respeito desse exemplo são:

1. Nas linhas 5 e 6, a função **somador()** está recebendo o que se chama "argumentos". Argumentos são valores que são passados a uma Função ou Sub-rotina que servem como um input. Esses valores, de alguma forma, serão utilizados pela Função ou Sub-rotina para produzir seu resultado. Os argumentos não são obrigatórios, a menos que a Função ou Sub-rotina os exija;

2. Existem três declarações de funções com o mesmo nome **somador()**. Achou isso estranho? Em algumas outras linguagens de programação a mesma coisa pode ser feita, enquanto em outras o compilador pode parar e reclamar do erro. No Visual Basic isso é possível, e o que acontece nesse caso é uma sobrecarga (*overload*) da função. Como a primeira declaração não possui argumentos, na hora de fazer uma chamada para a função na rotina **Main()**, se ela for feita sem argumentos, é a primeira declaração que será chamada, enquanto a chamada com dois argumentos invocará a segunda declaração e assim por diante. Se você quisesse que a sua função tratasse dados do tipo Float de forma diferente, poderia fazer uma outra declaração além dessas. Como exemplo prático dessa sobrecarga, você pode perceber, quando está digitando o comando **Console.WriteLine()**, o Visual Studio oferece no IntelliSense 18 forma de utilização de argumentos para essa função da classe Console do .NET.

3. A declaração de uma função exige a especificação do tipo retornado pela função e a especificação das variáveis para seus argumentos. Veja o exemplo:

```
Function somador(ByVal val1 As Integer) As String
```

A parte fora dos parênteses está declarando o tipo de valor a ser retornado pela função com o comando **Return**. No exemplo acima, uma frase é retornada, ou seja, um tipo String. Por outro lado, o que está dentro dos parênteses declara o tipo da variável "**val1**", que é um número inteiro, mas que na execução da função é convertido em uma String e inserido na frase retornada pela função **somador()**.

4. Na declaração do argumento (**ByVal val1 As Integer**) o modificador **ByVal** faz com que a variável **val1** não possa ser modificada dentro do escopo

da função. Isso significa que, dentro da função, por exemplo, não se pode fazer uma atribuição como `val1 = 10`, garantindo que será utilizado apenas o valor passado como argumento na chama da função em **Main()**.

5. O método **String.Format()** é uma forma de converter variáveis e inseri-las em uma String formatada. Veja o seguinte trecho do código:

```
Return String.Format("A soma {0} + {1} é igual a {2}", val1, val2, soma)
```

As chaves indicam onde as variáveis devem ser inseridas na String. Os números entre chaves correspondem à posição de cada variável passada como argumento. A String é o primeiro argumento, e os argumentos adicionais são as variáveis a serem inseridas. Lembre-se que, normalmente, na computação, a contagem de posições inicia em zero. Então, **val1** tem a posição 0, **val2** tem a posição 1 e **soma** tem a posição 2. Logicamente, isso também poderia ser feito:

```
Return String.Format("A soma {1} + {2} é igual a {0}", soma, val1, val2)
```

Encadeamento de Funções e Sub-rotinas

Um outro aspecto interessante de Funções e Sub-rotinas é que elas podem ser encadeados, isto é, uma função pode ser chamada dentro de outra função e, por sua vez, essa também pode ser chamada em outra função e assim sucessivamente. O código a seguir apresenta um exemplo de encadeamento.

```
Module1.vb
1  Module Module1
2
3      Function AreaQuadrado(ByVal Comprimento As Double)
4          'Calcula a área do quadrado
5          Return Comprimento * Comprimento
6      End Function
7
8      Function VolumeCubo(ByVal Comprimento As Double)
9          'Chama a função AreaQuadrado e com ela calcula o volume do cubo
10         Return AreaQuadrado(Comprimento) * Comprimento
11     End Function
12
13     Sub Main()
14         Console.WriteLine(VolumeCubo(3.0))
15         Console.ReadKey()
16     End Sub
17
18 End Module
```

Código 2.6 – Exemplo de encadeamento de funções.

Nesse exemplo, em vez de a função que calcula o volume do cubo, `VolumeCubo()`, usar uma operação de potência ou multiplicar o `Comprimento` duas vezes por si próprio, ela chama a função `AreaQuadrado()` e lhe passa o argumento `Comprimento`, o qual também lhe foi passado como argumento. Assim, o resultado da função `AreaQuadrado()` é multiplicado pelo `Comprimento` mais uma vez e o resultado é retornado para a sub-rotina `Main()`. É importante analisar e entender o que ocorreu: o argumento passado para `VolumeCubo(3.0)` também foi passado como argumento para `AreaQuadrado(Comprimento)`.

2.1.4 Escopo

A palavra "escopo", no contexto das linguagens de programação, significa limite, abrangência ou alcance. O escopo é uma delimitação de acesso a uma variável ou um método ao seu espaço que é definido em sua declaração. Uma variável definida no módulo principal do programa tem um escopo mais amplo, podendo ser acessada pelos seus métodos internos. Por outro lado, uma variável definida dentro de um método não pode ser acessada diretamente por outros métodos.

Escopo de Variáveis

No exemplo a seguir, `Numero` é uma variável definida para todo o módulo, a qual pode ser acessada pelas sub-rotinas `Duplicar()` e `Triplicar()`. Perceba que são sub-rotinas, e não funções, portanto não retornam valores. O que elas fazem é simplesmente alterar o valor da variável `Numero`.

```
Module1.vb
1  Module Module1
2      Private Numero As Double
3
4      Sub Duplicar()
5          'Acessa número declarado foram da sub-rotina e o duplica
6          Numero = Numero * 2
7      End Sub
8
9      Sub Triplicar()
10         'Acessa número declarado foram da sub-rotina e o triplica
11         Numero = Numero * 3
12     End Sub
13
```

```

14 Sub Main()
15     Numero = 1
16     Duplicar()
17     Triplicar()
18     Console.WriteLine(Numero)
19     Console.ReadKey()
20 End Sub
21
22 End Module

```

Código 2.7 – Exemplo de alteração de variável de escopo no módulo.

O resultado do código anterior é a simples impressão do número 6 no console:

```

CONSOLE
6

```

A variável `Numero` é declarada como `Private`, o que significa que ela pode ser acessada apenas pelos métodos internos a esse `Module1`. No caso desse exemplo, declarar a variável `Numero` com as palavras-chave `Dim` ou `Private` tem o mesmo efeito, permitindo o acesso por todos os métodos no escopo do módulo, mas impedindo o acesso dessa variável por outros módulos que o aplicativo possa conter. Uma variável com escopo restrito a seu módulo ou classe é chamada **Variável Membro**.

Para permitir esse acesso externo, é necessário declarar com a palavra-chave `Public`, tornando-a o que se chama de variável **Global**, que pode ser acessada por todo o aplicativo, fora de seu módulo:

```
Public Numero As Double
```

As palavras-chave `Public` e `Private` não podem ser utilizadas dentro de métodos, nesse escopo sendo possível apenas utilizar `Dim`. Nesse caso, essa será uma variável com escopo **Local**.

No exemplo anterior, as duas funções, `Duplicar()` e `Triplicar()`, estão alterando o valor de uma variável declarada fora de seus escopos, a qual não precisa ser passada como argumento. O resultado é que há uma persistência nas operações realizadas sobre essa variável.

Escopo de Métodos

O escopo de métodos funciona de forma parecida. No entanto, se for feita uma declaração de método sem explicitar se ela é Private ou Public, por padrão ela será Public, podendo ser acessada por qualquer outro módulo de seu programa. Para entender, voltemos ao exemplo do conversor de temperaturas fazendo pequenas mudanças.

```
exemplo.vb
1  Module Temperatura
2
3      Public Function CParaF(ByVal temp_celsius As Decimal)
4          Dim temp_fahrenheit As Decimal
5          temp_fahrenheit = temp_celsius * 1.8 + 32
6          Return temp_fahrenheit
7      End Function
8
9      Public Function FParaC(ByRef temp_fahrenheit As Decimal)
10         Dim temp_celsius As Decimal
11         temp_celsius = (temp_fahrenheit - 32) / 1.8
12         Return temp_celsius
13     End Function
14
15 End Module
16
17 Module Module1
18     Sub Main()
19         Dim temp, temp_convertida As Decimal
20         Console.WriteLine("Digite uma temperatura em graus Celsius:")
21         temp = Console.ReadLine()
22         temp = Console.ToDecimal(temp)
23         temp_convertida = Temperatura.CParaF(temp)
24         Console.WriteLine("A temperatura digitada equivale a {0} Fahrenheit.",
temp_convertida)
25         Console.ReadKey()
26     End Sub
27 EndModule
```

Código 2.8 – Exemplo de escopo global de método.

Nesse exemplo, temos dois módulos no mesmo arquivo. A chamada ao método `CParaF()` na linha 23 só é possível porque a função é declarada como sendo `Public`. Se fosse declarada `Private`, o Visual Studio avisaria da impossibilidade de fazer a chamada de método da linha 23.

Outra mudança feita nesse código foi o uso de espaço de nome na chamada do método da linha 23, sendo que agora foi utilizada a forma **Temperatura.CParaF(temp)**, onde **Temperatura** é o espaço de nome, termo mais amplamente utilizado em inglês: **namespace**. Esse namespace indica que a função utilizada é interna ao módulo **Temperatura**. Como os dois módulos desse exemplo estão no mesmo projeto, isso não seria necessário, mas o uso do namespace é interessante para evitar conflito de métodos e variáveis globais que tenham o mesmo nome.

Muito ainda poderia ser visto a respeito de Sub-rotinas e Funções, mas o interesse aqui é o conhecimento básico de sua operação e a filosofia por trás de sua programação. A partir daqui, nosso interesse é compreendê-las no contexto de classes e objetos, onde são utilizadas como Métodos de manipulação desses elementos, o que será visto no próximo capítulo.

2.2 Programação Estruturada

Antes do surgimento da programação estruturada na década de 1960, era comum as linguagens de programação serem compostas de testes lógicos e saltos. Se um teste lógico tinha um resultado ou outro, a execução de um programa levava a um comando `GOTO` (do inglês "*go to*", que significa "vá para"), que indicava uma linha do código fonte de onde ele devia continuar. A linguagem BASIC original era baseada nesse tipo de programação, o qual levava a muitos problemas de desenvolvimento: se um programador altera o código em alguma parte, tinha que repassar por todo o programa modificando a referência de linhas do comando `GOTO`. Em um código fonte extenso, isso podia torná-lo muito difícil de manter.

Com isso, o paradigma de programação estruturada ganhou evidência e muitas linguagens de programação a adotaram. O Visual Basic, apesar do nome indicar sua origem no BASIC, suporta completamente a programação estruturada (MSDN, 2015c). Mesmo que essa linguagem ainda mantenha o comando `GoTo`, ele não é utilizado da mesma maneira que antigamente e não é preferível na maioria das situações.

A programação estruturada se baseia, principalmente, em três estruturas de controle:

- **Sequência:** as instruções são executadas uma após a outra, sequencialmente;
- **Decisão ou Seleção:** as instruções de decisão levam a execução de um caminho ou outro do algoritmo, com base em um teste lógico e oferecendo uma sequência diferente a ser seguida;
- **Repetição ou Iteração:** essas instruções criam uma repetição de uma sequência, até que certa condição ocorra ou deixe de ocorrer.

2.2.1 Estruturas de Controle de Decisão

As estruturas de controle de decisão ou seleção são aquelas em que se cria uma condição lógica a ser testada e, dependendo de seu resultado, o programa pode seguir um caminho sequencial ou outros. Essas estruturas estabelecem as possibilidades de ações diferentes através da execução de blocos específicos de código. A seguir, serão descritas brevemente as principais formas de estruturas de decisão.

2.2.1.1 Estrutura de controle de decisão simples (se... então)

Essa é a forma mais básica de uma estrutura de decisão. Como o programa funciona seguindo uma sequência de instruções, caso encontre esse bloco simples, ele vai testar uma condição. Se ela for verdadeira, o bloco intrínseco será executado; se for falsa, o bloco não será executado e o programa continuará com o código seguinte. A forma básica de uma estrutura de decisão simples é:

```
If (condição) Then  
    Bloco de Código  
End If
```

Um exemplo dessa estrutura em Visual Basic é apresentado no código a seguir. Como a variável valor1 é maior que valor2, o código interior ao bloco de decisão será executado: valor2 terá seu valor acrescido em 10 e esse novo valor será impresso na tela do console.

```

exemplo.vb
1  Module Module1
2      Sub Main()
3          Dim valor1, valor2 As Integer
4          valor1 = 10
5          valor2 = 5
6
7          If (valor1 > valor2) Then
8              valor2 = valor2 + 10
9              Console.WriteLine("Novo valor2 = {0}", valor2)
10         End If
11
12         Console.ReadKey()
13     End Sub
14 End Module

```

Código 2.9 – Exemplo de estrutura de decisão simples.

Se o sinal de comparação na condição de teste fosse invertido (<), o teste lógico resultaria em False e o bloco não seria executado. Com isso nada seria impresso na tela do console.

2.2.1.2 Estrutura de controle de decisão estendida (se... então... senão)

A estrutura de decisão estendida permite que se execute um bloco de código caso o teste lógico seja verdadeiro e outro bloco caso seja falso. Dessa forma, de qualquer maneira algum dos blocos será executado e o programa não sairá da estrutura de decisão sem passar por algum deles. A forma básica dessa estrutura é:

```

If (condição) Then
    Bloco de Código 1
Else
    Bloco de Código 2
End If

```

Um exemplo prático dessa estrutura em uso no Visual Basic é apresentado no quadro a seguir. Ele é parecido com o exemplo da estrutura de decisão simples, mas o sinal da condição de teste está invertido, o que resultará em Falso. Assim, o bloco alternativo de Else será executado apenas exibindo uma mensagem na tela do console.

```

exemplo.vb
1  Module Module1
2      Sub Main()
3          Dim valor1, valor2 As Integer
4          valor1 = 10
5          valor2 = 5
6
7          If (valor1 < valor2) Then
8              valor2 = valor2 + 10
9              Console.WriteLine("Novo valor2 = {0}", valor2)
10         Else
11             Console.WriteLine("O valor1 é maior.")
12         EndIf
13
14         Console.ReadKey()
15     End Sub
16 End Module

```

Código 2.10 – Exemplo de estrutura de controle de decisão estendida.

Esse exemplo é bastante simples, servindo apenas para mostrar a sintaxe dessa estrutura no Visual Basic. Os blocos de código dentro de cada um dos caminhos de execução podem ser bem mais complexos, inclusive fazendo chamadas de métodos.

2.2.1.3 Estrutura de controle de decisão aninhada

Uma estrutura de controle de decisão aninhada permite que se estabeleça mais do que duas condições lógicas a serem testadas. Em sua forma mais intuitiva, utiliza-se várias estruturas de decisão estendidas, uma dentro da outra: caso a condição seja verdadeira ou falsa, cria-se outro bloco If que testa outra condição, como na estrutura básica a seguir.

```

If (condição) Then
    Bloco de Código 1
Else
    If (condição 2) Then
        Bloco de Código 2
    End If
End If

```

No entanto, essa forma de aninhamento pode parecer confusa e dificultar a leitura e compreensão do código, caso haja muitos níveis de aninhamento com blocos complexos. Como a necessidade dessa estrutura é comum, nas linguagens de programação costumam existir instruções do tipo Senão Se (ElseIf), que permitem o estabelecimento de várias condições de testes de uma forma mais clara, mas que tem efeito semelhante à estrutura anteriormente apresentada.

```
If (condição 1) Then  
    Bloco de Código 1  
ElseIf (condição 2) Then  
    Bloco de Código 2  
ElseIf (condição 3) Then  
    Bloco de Código 3  
...  
End If
```

O exemplo a seguir, que mostra o uso dessa estrutura no Visual Basic, exibe uma forma básica de colocar três valores inteiros em ordem crescente. Essa não é a forma mais eficiente de se ordenar valores, mas apenas serve para mostrar a sintaxe das instruções de decisão ElseIf.

```
exemplo.vb  
1  Module Module1  
2      Sub Main()  
3          Dim valor1, valor2, valor3 As Integer  
4          valor1 = 5  
5          valor2 = 10  
6          valor3 = 20  
7  
8          If (valor1 < valor2 And valor2 < valor3) Then  
9              Console.WriteLine("Ordem crescente: {0} {1} {2}", valor1, valor2,  
valor3)  
10             ElseIf (valor1 < valor3 And valor3 < valor2) Then  
11                 Console.WriteLine("Ordem crescente: {0} {1} {2}", valor1, valor3,  
valor2)  
12                 ElseIf (valor2 < valor1 And valor1 < valor3) Then  
13                     Console.WriteLine("Ordem crescente: {0} {1} {2}", valor2, valor1,  
valor3)  
14                     ElseIf (valor2 < valor3 And valor3 < valor1) Then  
15                         Console.WriteLine("Ordem crescente: {0} {1} {2}", valor2, valor3,  
valor1)
```

```

16     ElseIf (valor3 < valor1 And valor1 < valor2) Then
17         Console.WriteLine("Ordem crescente: {0} {1} {2}", valor3, valor1,
valor2)
18     ElseIf (valor3 < valor2 And valor2 < valor1) Then
19         Console.WriteLine("Ordem crescente: {0} {1} {2}", valor3, valor2,
valor1)
20     End If
21
22     Console.ReadKey()
23     End Sub
24 End Module

```

Código 2.11 – Exemplo de estrutura de controle de decisão aninhada.

Se não fosse utilizado o `ElseIf`, o aninhamento prejudicaria a compreensão do algoritmo. Note que para cada bloco há um testes de condições diferentes, considerando todas as alternativas de ordem entre as três variáveis.

2.2.1.4 Estrutura de controle de decisão em sequência na mesma linha.

A estrutura de decisão na mesma linha é apenas uma forma diferente de escrever a estrutura estendida. Ela expressa todo o bloco de decisão em um menor espaço vertical, como mostra a estrutura básica:

If (condição) Then instrução 1 Else instrução 2

Com essa forma também é possível expressar mais de uma instrução para cada caminho do bloco de decisão, separando cada instrução com dois pontos:

If (condição) Then instrução 1 : instrução 2 Else instrução 3 : instrução 4

O próximo exemplo demonstra o uso dessa forma. Perceba que o primeiro bloco é composto de duas instruções, enquanto o segundo possui apenas uma.

```

exemplo.vb
1  Module Module1
2      Sub Main()
3          Dim valor1, valor2 As Integer
4          valor1 = 10

```

```

5         valor2 = 5
6
7  If (valor1 < valor2) Then valor2 = 15 : valor1 = 0 ElseConsole.WriteLine("0
valor1 é maior.")
8
9         Console.ReadKey()
10     End Sub
11 End Module

```

Código 2.12 – Exemplo de estrutura de controle de decisão em sequência na mesma linha.

É evidente que essa forma é bem menos legível e deve ser usada apenas em casos específicos. A quebra de linhas e a indentação (afastamento das linhas) são recursos que facilitam a leitura do código fonte e são boas práticas de programação.

2.2.1.5 Estrutura de controle de múltiplas decisões (escolha)

Uma estrutura de decisão para múltiplas escolhas permite otimizar o código caso haja muitas opções de resultado para o teste de condição. O código resultante é bem mais claro do que o aninhamento de condições.

```

Select Case (condição)
    Case valor ou intervalo de valor ou lista de valores 1
        Bloco de código 1
    Case valor ou intervalo de valor ou lista de valores 2
        Bloco de código 2
    ...
    Case Else
        Bloco de código N
End Select

```

O exemplo a seguir pede que o usuário digite o número de um mês na sequência de um ano (1 a 12), imprimindo o nome do mês correspondente a esse número. Há 13 opções, sendo 12 relativas a cada mês e uma outra para o caso de ser digitado um número que não corresponda a algum mês.

```

exemplo.vb
1  Module Module1
2      Sub Main()
3          Dim dia As Integer
4          Console.WriteLine("Escolha um mês do ano (1 a 12):")
5          dia = Console.ReadLine()
6
7          SelectCase (dia)
8              Case 1
9                  Console.WriteLine("Você escolheu janeiro.")
10             Case 2
11                 Console.WriteLine("Você escolheu fevereiro.")
12             Case 3
13                 Console.WriteLine("Você escolheu março.")
14             Case 4
15                 Console.WriteLine("Você escolheu abril.")
16             Case 5
17                 Console.WriteLine("Você escolheu maio.")
18             Case 6
19                 Console.WriteLine("Você escolheu junho.")
20             Case 7
21                 Console.WriteLine("Você escolheu julho.")
22             Case 8
23                 Console.WriteLine("Você escolheu agosto.")
24             Case 9
25                 Console.WriteLine("Você escolheu setembro.")
26             Case 10
27                 Console.WriteLine("Você escolheu outubro.")
28             Case 11
29                 Console.WriteLine("Você escolheu novembro.")
30             Case 12
31                 Console.WriteLine("Você escolheu dezembro.")
32             Case Else
33                 Console.WriteLine("Esse não é um número válido!")
34         End Select
35
36         Console.ReadKey()
37     End Sub
38 End Module

```

Código 2.13 – Exemplo de estrutura de controle de múltiplas decisões.

A instrução `Case` é seguida do valor que se quer comparar com a variável. Esse valor poderia não ser um número, mas uma `String`. Além disso, em vez de apenas comparar um valor, o Visual Basic permite que se compare um intervalo de valores, sendo que nesse caso o valor é dado como em `Case 1 To 6`, para indicar que ele pode estar no intervalo entre esses dois número. Veja um exemplo desse uso no próximo código.

```
exemplo.vb
1  Module Module1
2      Sub Main()
3          Dim dia As Integer
4          Console.WriteLine("Escolha um mês do ano (1 a 12):")
5          dia = Console.ReadLine()
6
7          SelectCase (dia)
8              Case 1 To 6
9                  Console.WriteLine("Mês do primeiro semestre.")
10             Case 7 To 12
11                 Console.WriteLine("Mês do segundo semestre.")
12             Case Else
13                 Console.WriteLine("Esse não é um número válido!")
14             End Select
15
16             Console.ReadKey()
17         End Sub
18 End Module
```

Código 2.14 – Exemplo de estrutura de controle de múltiplas decisões com intervalos de valores.

Com isso, foram vistas as principais formas das estruturas de decisão. Apesar de parecerem simples, são, junto com as estruturas de repetição, um dos elementos básicos para a programação de softwares.

2.2.2 Estruturas de Controle de Repetição

As estruturas de repetição são as formas básicas de as linguagens de programação executarem iteração com os dados. Por iterações entende-se repetições que transformam esses dados de alguma maneira. Os ciclos de repetição dessas estruturas são amplamente conhecidas pelo termo em inglês *loop* e por seu equivalente em português "laço".

2.2.2.1 Estrutura de controle de repetição controlada por contador (para)

A estrutura de repetição do tipo Para é aquela que utiliza um contador, sendo que o programador consegue estabelecer de antemão quantas vezes o laço será repetido. Isso não quer dizer que se sabe especificamente quantas vezes a repetição será feita, mas que não há uma condição incerta a ser testada. Por exemplo, o usuário pode fornecer um valor para repetir uma tarefa um X número de vezes, portanto, nesse caso, essa variável estabelece previamente o número de laços a serem executados.

A forma básica dessa estrutura é a seguinte:

```
For inicializa uma variável de contagem  
    Bloco de código  
Next
```

É importante notar que a instrução de finalização do laço não segue o padrão comum do Visual Basic. Intuitivamente alguém poderia tentar adivinhar que a instrução seria "End For", mas a instrução de finalização é Next, que significa "Próximo", indicando que o programa continua a execução de outras instruções após o fim dos laços desse bloco.

O exemplo de código a seguir mostra a declaração de uma variável que serve como contador e seu uso na estrutura For. O trecho 1 To 5 cria um intervalo de números inteiros que gera uma iteração de cinco repetições. Após as cinco iterações, o programa passa para a próxima instrução.

```
exemplo.vb  
1  Module Module1  
2      Sub Main()  
3          Dim contador As Integer  
4          For contador = 1 To 5  
5              Console.WriteLine(contador)  
6          Next  
7  
8          Console.ReadKey()  
9      End Sub  
10 End Module
```

Código 2.15 – Exemplo de estrutura de controle de repetição Para.

Com esse código, o console imprimirá os valores de 1 a 5, um por linha. Uma outra forma de escrever o código anterior é declarando a variável contadora no próprio início da estrutura de repetição, como no trecho a seguir.

```
exemplo.vb
1  Module Module1
2      Sub Main()
3
4          For contador As Integer = 1 To 5
5              Console.WriteLine(contador)
6          Next
7
8          Console.ReadKey()
9      End Sub
10 End Module
```

Código 2.16 – Exemplo de estrutura de controle de repetição Para, com inicialização de variável.

É possível perceber que a estrutura For não depende de nenhuma condição a ser testada. Ela apenas repete o laço pelo número de vezes que lhe for passado. Mas e se for necessário repetir um laço até que certa condição a ser testada ocorra? Para isso existem as estruturas Enquanto e Faça Enquanto.

2.2.2.2 Estrutura de controle de repetição While (enquanto)

A estrutura do tipo Enquanto cria um laço de repetição enquanto alguma condição é satisfeita, isto é, enquanto o teste da condição resultar em Verdadeiro. Essa estrutura é implementada pela palavra-chave While:

```
inicialização da variável de condição
While (condição)
    Bloco de código
    Alteração da variável de condição
End While
```

Como a estrutura While necessita testar uma condição, a variável que é testada deve ser inicializada antes. Além disso, o teste da condição é feito no início do laço, fazendo com que ele nunca seja executado se a condição resultar em Falso na primeira tentativa. O código a seguir usa a estrutura While para calcular o fatorial de cinco, que é $5! = 5 \times 4 \times 3 \times 2 \times 1 = 120$.

```
exemplo.vb
1  Module Module1
2      Sub Main()
3          Dim contador As Integer = 5
4          Dim fatorial As Integer = 1
5
6          While (contador > 0)
7              fatorial *= contador
8              contador = contador - 1
9          End While
10
11         Console.WriteLine("Fatorial de 5 é {0}", fatorial)
12         Console.ReadKey()
13     End Sub
14 End Module
```

Código 2.17 – Estrutura de controle de repetição Enquanto.

Na forma como esse exemplo foi construído, é utilizado um contador, o que faz o While se comportar de forma parecida com o laço For. O valor do contador no início do programa determinará o número para o qual ele calcula seu fatorial. Para cada iteração é necessário alterar o valor do contador dentro do laço, o que poderia ser feito com o operador de subtração e atribuição:

```
contador -= 1
```

2.2.2.3 Estrutura de controle de repetição Do (faça)

A instrução Do tem um funcionamento parecido ao do While, sendo que normalmente eles podem ser usados da mesma maneira. Uma diferença entre as duas instruções é que Do pode ser usado com a realização do teste de condição depois do laço, ou seja, a cada iteração, primeiro executa o bloco de instruções e depois checa se determinada condição é satisfeita. Esse uso de Do pode ser feito de acordo com a seguinte forma:

Do

Bloco de código

Alteração da variável de condição

Loop Until (condição)

O exemplo a seguir usa o laço Do para que o usuário tente repetidamente adivinhar um número. Se ele digitar o número 7, que é a resposta certa, o laço é interrompido e o programa continua.

```
exemplo.vb
1  Module Module1
2      Sub Main()
3          Dim chute, resposta As Integer
4          resposta = 7
5
6          Console.WriteLine("Tente adivinhar um número de 1 até 10:")
7
8          Do
9              chute = Console.ReadLine()
10             If (chute <> resposta) Then Console.WriteLine("Errou! Tente de novo...")
11             Loop Until (chute = resposta)
12
13             Console.WriteLine("Acertou! O número é {0}.", resposta)
14             Console.ReadKey()
15         End Sub
16 End Module
```

Código 2.18 – Exemplo de estrutura de controle de repetição Faça com teste no fim.

A instrução Do pode ser utilizado da mesma forma que While como exemplo a seguir. Nesse caso, eles se tornam redundante e não há diferença entre usar uma instrução ou outra. Literalmente, Do While significa "faça enquanto".

```
exemplo.vb
1  Module Module1
2      Sub Main()
3          Dim Contador As Integer = 1
4
5          DoWhile (Contador <= 10)
6              Console.WriteLine(Contador)
7              Contador = Contador + 1
```

```

8      Loop
9
10     Console.ReadKey()
11 End Sub
12 End Module

```

Código 2.19 – Exemplo de estrutura de controle de repetição Faça com teste no início.

Há ainda outra forma de se utilizar a instrução Do: nesse caso, o laço será feito até o momento em que uma condição for atendida. Portanto, a forma de controle é um pouco diferente das anteriores. Em vez de se executar o bloco enquanto uma condição é satisfeita, é estabelecida a condição em que o laço deve parar. Isso é feito através do Do Until, que significa "faça até", cujo exemplo pode ser visto no código a seguir.

```

exemplo.vb
1  Module Module1
2      Sub Main()
3          Dim Contador As Integer = 1
4
5          DoUntil (Contador = 10)
6              Console.WriteLine(Contador)
7              Contador = Contador + 1
8          Loop
9
10         Console.ReadKey()
11     End Sub
12 End Module

```

Código 2.20 – Exemplo de estrutura de controle de repetição Faça com condição de finalização.

Com isso, foram vistas as principais estruturas de controle do Visual Basic. Esse elementos são essenciais para a programação estruturada, através da qual são constituídas Funções e Sub-rotinas.



REFLEXÃO

A modularização do código fonte de programas é uma excelente prática de programação, pois através dela é possível reparti-los em pedaços especializados. Quando bem realizada, fica muito mais fácil gerenciar o desenvolvimento do código, principalmente em equipe. Através do uso de métodos, os módulos

apresentam um padrão de comunicação com o resto programa, e isso permite que o código pertencente a um módulo seja modificado sem alterar o resto do programa, desde que o método de comunicação com nomes de métodos e seus argumentos sejam mantidos.



ATIVIDADES

01. No Visual Basic, o que é um módulo? Também explique o que é um método e qual a diferença entre uma função e uma sub-rotina.
 02. Faça um pequeno programa que continuamente peça um número inteiro para o usuário até que ele digite o número zero, com o qual ele deve encerrar. A cada vez que pedir um número, ele deve somar ao saldo anterior e exibir o resultado na tela. Use uma sub-rotina "soma" e uma variável declarada no escopo do módulo para acumular o valor em seu programa.
 03. Explique o que é uma estrutura de decisão e faça um pequeno código exemplificando, o qual deve solicitar um número inteiro para o usuário e responder dizendo se esse número é par ou ímpar.
 04. Explique o que é uma estrutura de repetição, diferenciando uma que é baseada em contador de outra que é baseada em condição.
-



LEITURA

O portal Microsoft Developer Network (MSDN) apresenta artigos e links explicando melhor a estrutura de um programa no Visual Basic e aspectos técnicos de um módulo:

Estrutura de um programa Visual Basic

<<https://msdn.microsoft.com/pt-br/library/022td33t.aspx>>

Instrução Module

<<https://msdn.microsoft.com/pt-br/library/aaxss7da.aspx>>

O mesmo site também apresenta uma visão sobre a programação estruturada na linguagem:

Fluxo de controle no Visual Basic

<<https://msdn.microsoft.com/pt-br/library/ca8tdhcs.aspx>>



REFERÊNCIAS BIBLIOGRÁFICAS

MSDN, Microsoft Developer Network. Estrutura de um programa Visual Basic. Disponível em: <<https://msdn.microsoft.com/pt-br/library/022td33t.aspx>>. Acesso em: 28 jan. 2015.

_____. Módulos (Visual Basic). Disponível em: <<https://msdn.microsoft.com/pt-br/library/y76404kz.aspx>>. Acesso em: 28 jan. 2015.

_____. Fluxo de controle no Visual Basic. Disponível em: <<https://msdn.microsoft.com/pt-br/library/ca8tdhcs.aspx>>. Acesso em: 10 fev. 2015.

3

Tratamento de Exceções e Programação Orientada a Objetos

Neste capítulo veremos dois assuntos distintos, mas que complementam o conjunto de recursos essenciais da linguagem Visual Basic .NET. Primeiro, será abordado o Tratamento de Exceções, que é a forma de uma linguagem de programação em lidar com comportamentos indesejados ou inesperados em um programa. Em muitas situações, o programador apenas considera que o programa terá sucesso em executar uma ação que depende do usuário ou de outras partes do programa. Mas e se essa operação não obtiver sucesso? O risco é que o programa gere uma mensagem de erro incompreensível, trave ou feche inesperadamente. Para isso serve o tratamento de exceções.

Na segunda parte do capítulo, veremos conceitos e exemplos da Programação Orientada a Objetos, paradigma que pode ser plenamente utilizado no Visual Basic .NET. O objetivo dessa parte do capítulo será o reforço do conceito de objetos e a demonstração da sintaxe do VB.NET para utilização desse recurso.



OBJETIVOS

Com a leitura desse capítulo, você terá aprendido:

- O que é Tratamento de Exceções
 - Quais os tipos e como utilizar o tratamento de erros no Visual Basic .NET
 - O que é Programação Orientada a Objetos (POO)
 - Como utilizar a Programação Orientada a Objetos no VB.NET
-

3.1 Tratamento de Exceções

O tratamento de exceções é uma forma de o programador garantir que erros ou comportamentos indesejados de seu programa não causem problemas para o usuário, além de ser uma forma de emitir uma mensagem útil que explique para o usuário o que deu errado. Alguma vez, usando algum software, já se deparou com uma mensagem do tipo "Erro cod. 472648"? Isso não explica muita coisa se você não tiver um catálogo de erros desse software em mãos, não acha? O tratamento de erros procura lidar com essas situações.

3.1.1 Exceções

As exceções são um dos meios de se emitir uma interrupção para o processador do computador. No momento que uma exceção ocorre, uma instrução de interrupção é emitida para o processador e ele para a execução desse programa para tratar essa ocorrência. Algumas exceções permitem que seja emitida uma mensagem e o programa continue executando, enquanto outras exigirão que o programa seja fechado.

Existem duas origens principais de exceções (MACKENZIE e SHARKEY, 2003):

- **Hardware:** quando a exceção é causada por algum dispositivo de hardware e a ocorrência é gerada pelo sistema operacional, *drivers* controladores ou software que gerencia o dispositivo. Por exemplo, se um programa tem uma rotina para abrir um arquivo armazenado no disco e esse arquivo é muito grande, é possível que o sistema operacional avise que não há espaço em memória para essa operação;
- **Software:** nesse caso a exceção é causada diretamente pelo software devido a algum algoritmo em seu código. Por exemplo, o usuário pode entrar com uma String em um campo relacionado a uma variável declarada para números inteiros, ou inserir o valor zero em uma variável que pode causar uma divisão por zero.

Para lidar com os diversos tipos de exceções, as bibliotecas de *frameworks* de linguagens de programação costumam oferecer formas unificadas de tratar erros. Apesar de haver diferenças entre as linguagens na forma com que fazem

isso, sempre há essa opção, que na verdade é uma boa prática de programação. O tratamento estruturado de exceções (Structured Exception Handling - SEH) se baseia nessa unificação de vários tipos de exceções, tanto de hardware como de software, através da biblioteca padrão de uma linguagem de programação.

3.1.2 Tipos de Exceções

No Framework .NET, as exceções são implementadas através de classes em suas bibliotecas. Elas podem fazer parte tanto dos escopos (de módulos) System e System.IO como outros, mas são derivadas da classe System.Exception. Existe uma quantidade muito grande de tipos de exceções, sendo que o quadro a seguir resume apenas algumas das mais utilizadas.

CLASSE	DESCRIÇÃO
ArgumentException	A exceção que é acionada quando um dos argumentos fornecidos para um método não é válido.
ArgumentNullException	A exceção que é lançada quando uma referência nula (Nothing no Visual Basic) é passada para um método que não aceita nulo como um argumento válido.
ArgumentOutOfRangeException	A exceção é lançada quando o valor de um argumento estiver fora do intervalo de valores permitido conforme definido pelo método chamado.
IndexOutOfRangeException	A exceção que é lançada quando é feita uma tentativa de acessar um elemento da matriz com um índice está fora dos limites da matriz.
NotImplementedException	A exceção que é jogada quando um método ou operação solicitada não é implementada.
OutOfMemoryException	A exceção que é lançada quando não há memória suficiente para continuar a execução de um programa.
OverflowException	A exceção é lançada quando uma operação aritmética, transmissão ou conversão em um contexto marcada resultados em um estouro.
FileNotFoundException	A exceção que é lançada quando uma tentativa de acessar um arquivo que não existe no disco falha. Faz parte do escopo System.IO.

Tabela 3.1 – Principais tipos de Exceções do .NET Framework. Fonte: Adaptado de MSDN (2015a) e MSDN (2015b).

Uma forma de se descobrir a classe de uma exceção para determinado erro é averiguar as mensagens de erro que o Visual Studio emite quando uma exceção ocorre. Um programador experiente saberá imediatamente do que se trata, mas, com o propósito de aprender, é importante observar o tipo de exceção e pesquisar a respeito na documentação do Visual Basic.

3.1.3 Estrutura de um Bloco de Tratamento de Exceções

O comando Try delimita o início do bloco de código, o qual é finalizado por End Try. Dentro desse bloco, procura-se capturar os erros de execução e manipulá-los, salvando-os em variáveis e exibindo uma mensagem de exceção. Um bloco completo de tratamento de exceção tem a seguinte estrutura:

Try

'tenta executar um ou mais comandos

Catch variavel As Exception

'captura uma possível exceção originado da

'tentativa de execução do comando anterior

Finally

'executa um ou mais comandos independente de

'uma exceção ocorrer ou não

End Try

O elemento Catch pode ocorrer mais de uma vez, enquanto o uso de um bloco Finally não é obrigatório.

Try... End Try

A palavra "*Try*" significa "tentar" ou "tente". Se ela não for utilizada, isto é, se não se utiliza um bloco de tratamento de erro, um comando que falha ao ser executado irá causar um erro de execução da mesma maneira, mas ele não será capturado pelo programa.

É interessante utilizar a estrutura Try... End Try sempre que se esteja lidando com situações em que se pode prever uma exceção, como na abertura de arquivos que podem não existir, ou quando forem feitas operações aritméticas em que os possíveis operadores podem incorrer em erros. O exemplo de código a seguir leva a um erro de divisão por zero.

```

execcao1.vb
1  Module Module1
2
3      Sub Main()
4          Dim i As Decimal = 10
5          Dim j As Decimal = 0
6          Dim Resultado As Decimal
7
8          Resultado = i / j
9          Console.ReadKey()
10     End Sub
11
12 End Module

```

Código 3.1 – Código que causa erro de divisão por zero.

Executando esse programa no ambiente de desenvolvimento do Visual Studio, a IDE chamará a atenção para o erro de operação de divisão.

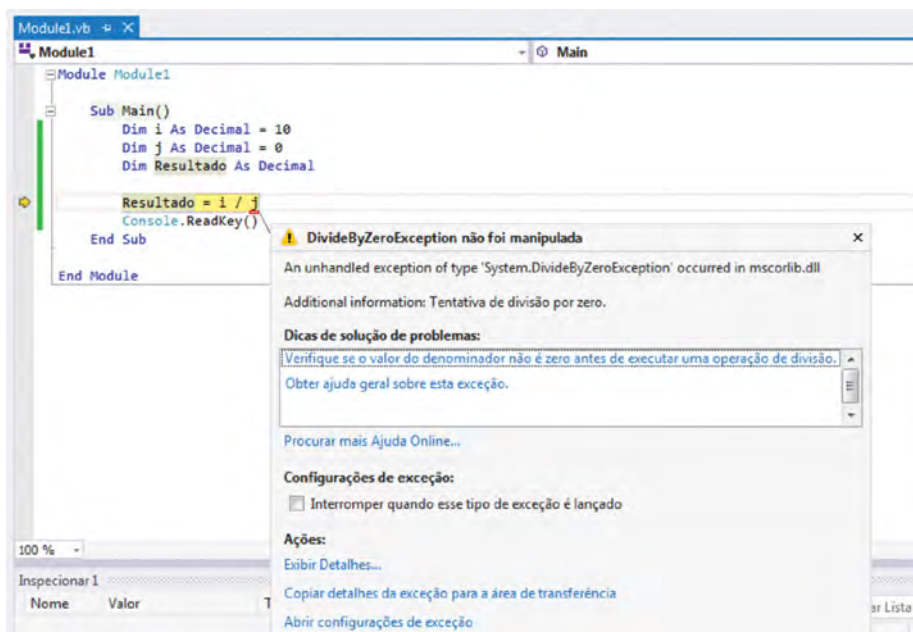


Figura 3.1 – Alerta do Visual Studio para erro de divisão por zero.

Logicamente, é difícil que um programador cometa esse erro simples do exemplo na elaboração código, mas é possível ocorrer em casos mais complexos. Programas que solicitam entradas de valores por parte do usuário são mais

propensos a esse tipo de erro de divisão por zero, ou então equações que recebem valores finais de outras equações.

Assim, se usa Try... End Try quando se deseja que o programa tente uma operação e, caso não consiga, faça alguma coisa. Para utilizar essa estrutura, é necessário compreender também a palavra-chave Catch.

Catch

"Catch" significa apanhar, agarrar ou capturar. Quando uma exceção de hardware ou software ocorre, diz-se que ela é atirada, lançada ou jogada. Então o comando Catch apanha essa exceção e faz alguma coisa com ela em seu bloco de código. O mais comum é emitir uma mensagem de erro, mas também é possível executar uma rotina mais complexa.

No trecho que código a seguir, o bloco Catch é utilizado para exceções genéricas, isto é, para qualquer exceção que ocorra, a mesma rotina será seguida. Perceba que a mensagem de depuração do erro é guardada na variável "ex", que é uma abreviação da palavra exceção, mas no lugar dela qualquer nome de variável pode ser utilizada.

```
excecao2.vb
1  Module Module1
2
3      Sub Main()
4          Dim i As Decimal = 10
5          Dim j As Decimal = 0
6          Dim Resultado As Decimal
7
8          Try
9              Resultado = i / j
10         Catch ex As Exception
11             'Captura uma exceção genérica, não específica
12             Console.WriteLine("Houve um erro devido a uma divisão por zero!")
13         End Try
14         Console.ReadKey()
15
16     End Sub
17
18 End Module
```

Código 3.2 – Uso da instrução Catch de forma genérica para capturar o erro de divisão por zero.

Com o uso da estrutura de exceção, o programa não termina com um erro que termina seu funcionamento abruptamente. Ele emite uma mensagem de erro que pode ser amigável ao usuário e continuará funcionando, caso seja possível. No exemplo anterior, o programa termina porque não há nada mais a fazer, executando o comando que espera o pressionamento de tecla do usuário para então fechar.

No exemplo anterior, há um tratamento genérico para qualquer exceção, mas é possível utilizar cláusulas Catch para diversos erros específicos. Na operação de divisão realizada, é pouco provável que ocorra algum outro tipo de erro que não seja a divisão por zero, mas há diversas situações em que mais problemas podem ocorrer. O exemplo a seguir mostra um tratamento de erro específico para divisão por zero, seguido de um tratamento de exceção genérico.

```
excecao3.vb
1  ModuleModule1
2
3      Sub Main()
4          Dim i As Decimal = 10
5          Dim j As Decimal = 0
6          Dim Resultado As Decimal
7
8          Try
9              Resultado = i / j
10         Catch exDivideByZero As DivideByZeroException
11             'Captura uma exceção específica de divisão por zero
12             Console.WriteLine("Houve um erro devido a uma divisão por zero!")
13         Catch ex As Exception
14             'Captura genérica, não específica
15             Console.WriteLine("Houve um erro qualquer!")
16         End Try
17         Console.ReadKey()
18
19     End Sub
20
21 End Module
```

Código 3.3 – Erro da instrução Catch para capturar erro específico de divisão por zero.

É importante notar que o tratamento genérico não pode vir antes, pois a divisão por zero encontraria primeiro o Catch genérico e seria tratado sem especificidade. Na forma do código do exemplo, caso haja uma exceção, primeiro

é checado se é uma divisão por zero e, caso não haja esse problema, será tratado de forma genérica.

Além de ser possível utilizar o Catch diversas vezes, também há possibilidade de aninhamento do bloco Try... End Try, da mesma forma que se faz com estruturas de decisão e de repetição. Com o aninhamento, pode-se capturar a exceção em caso de fracasso da primeira ação e, caso haja sucesso, tentar executar outra ação que pode ter suas respectivas exceções, como no exemplo a seguir.

```
excecao4.vb
1 Imports System.IO
2
3 Module Module1
4
5     Sub Main()
6         Dim FileNameAsString = "TestFile.data"
7
8         'Blocoexterno
9         Try
10            Dim fs As New FileStream(FileName, FileMode.Open, FileAccess.Read)
11            'Blocointerno
12            Try
13                Dim reader As New BinaryReader(fs)
14                reader.ReadInt32()
15            Catch ex As Exception
16                Console.WriteLine("Erro na leitura do arquivo.")
17            End Try
18        Catch ex As Exception
19            Console.WriteLine("Erro na abertura de arquivo.")
20        End Try
21        Console.ReadKey()
22    End Sub
23 End Module
```

Código 3.4 – Uso de tratamento de exceção para capturar erro de abertura e leitura de arquivo.

Nesse caso apresentado, o programa tentar abrir um arquivo no mesmo diretório em que o programa se encontra; se ele não conseguir abri-lo ou o arquivo não existe, incorre em uma exceção de abertura de arquivo. Caso tenha sucesso, tentará ler o conteúdo e, caso não consiga, capturará uma exceção de leitura de arquivo.

O caso desse exemplo pode ser construído sem aninhamento, com o uso de exceções mais específicas. Uma forma de fazê-lo seria substituindo o bloco pelo seguinte:

```
Try
    Dim fs As New FileStream(fileName, FileMode.Open, FileAccess.Read)
    Dim reader As New BinaryReader(fs)
    reader.ReadInt32()
Catch ex As FileLoadException
    Console.WriteLine("Erro na leitura do arquivo.")
Catch ex As FileNotFoundException
    Console.WriteLine("Erro na abertura de arquivo.")
```

Mas perceba que, no exemplo original, são utilizados tratamentos de exceções genéricos, que podem lidar com problemas inesperados diferentes para cada uma das ações de abertura e leitura de arquivo. Se um outro tipo de erro ocorrer, ele estará associado com a ação de abertura ou com a ação de leitura do arquivo, os quais poderão ser distinguidos.

Finally

A palavra-chave Finally significa "finalmente" e não é obrigatória em um bloco Try... End Try. Ela é usada para criar um bloco de instruções que devem ser executadas independentemente de uma exceção ocorrer ou não em seu bloco Try. No próximo exemplo há um bloco Finally que irá causar um sinal sonoro no computador e imprimir na tela "O programa finalizou.", havendo ou não uma exceção de divisão por zero.

```
excecao5.vb
1  Module Module1
2
3      Sub Main()
4          Dim i As Decimal = 10
5          Dim j As Decimal = 0
6          Dim Resultado As Decimal
7
8          Try
9              Resultado = i / j
10         Catch exDivideByZero As DivideByZeroException
```

```

11         'Captura uma exceção específica de divisão por zero
12         Console.WriteLine("Houve um erro devido a uma divisão por zero!")
13     Catch ex As Exception
14         'Captura genérica, não específica
15         Console.WriteLine("Houve um erro qualquer!")
16     Finally
17         Beep()
18         Console.WriteLine("O programa finalizou.")
19     End Try
20     Console.ReadKey()
21
22 End Sub
23
24 End Module

```

Código 3.5 – Exemplo de uso da instrução Finally.

O uso do Finally é importante porque, caso as instruções que a seguem fiquem fora do bloco Try, se houver uma exceção, elas não serão executadas. Também é importante saber que, caso no bloco Try não haja nenhuma instrução Catch, é necessário haver um bloco Finally.

Lançamento de Exceções (Throw)

Finalizando o assunto de tratamento de exceções, ainda há a palavra-chave Throw, que significa "lançar" ou "arremessar". O lançamento de uma exceção é a emissão de um sinal que será capturado pela instrução Catch. Isso significa que é possível utilizar a instrução Throw para provocar uma exceção que não exista na biblioteca padrão do .NET, mas que seja de interesse do programador. Observe o uso dessa instrução na linha 10 do próximo exemplo.

```

excecao6.vb
1  Module Module1
2
3      Sub Main()
4          Dim i As Decimal = 10
5          Dim j As Decimal = 0
6          Dim Resultado As Decimal
7
8          Try
9              If (j = 0) Then

```

```

10         Throw New System.Exception("Divisor j igual a zero.")
11     End If
12     Resultado = i / j
13     Catch ex As Exception
14         'Captura genérica, não específica
15         Console.WriteLine("Ocorreu uma exceção: {0}", ex.Message)
16     Finally
17         Beep()
18         Console.WriteLine("O programa finalizou.")
19     End Try
20     Console.ReadKey()
21 End Sub
22 End Module

```

Código 3.6 – Exemplo de uso da instrução Throw, para lançamento de erros.

Antes que uma exceção padrão de divisão por zero ocorresse, o algoritmo verifica se o divisor da operação, a variável *j*, tem valor zero. Se tiver, utiliza uma exceção personalizada que é capturada pelo tratamento genérico a seguir. No caso desse exemplo, a exceção de divisão por zero podia ser usada sem problemas, mas em outros casos o programador pode ter interesse em utilizar outros tipos, como um aplicativo que não consegue se conectar ao seu servidor de dados.

3.2 Programação Orientada a Objetos

A orientação a objetos começou a ser idealizada entre as décadas de 60 e 70, pois percebeu-se que a complexidade dos software aumentaria continuamente, e que o paradigma estruturado não seria suficiente para lidar com esse problema.

Classes e objetos são conceitos centrais do paradigma de programação orientada a objetos. Nem todas as linguagens de programação suportam esse paradigma. Apesar de seu uso ser bastante difundido atualmente, pode não ser a melhor solução para certos tipos de problemas computacionais, como, por exemplo, atualmente ainda não é o melhor paradigma para alguns problemas de computação distribuída. No entanto, para muitos casos, possui vantagens interessantes:

- Reuso do código;
- Redução de código;
- Separação software em componentes;
- Acesso a biblioteca de classes produzidas independentemente;
- Facilidade de manutenção e atualização.

Em seu processo de aprendizado ou em sua carreira profissional, muitas vezes ouvirá ou lerá situações em que os termos "classe" e "objeto" são usados de forma indiscriminada, mas é essencial que compreenda a diferença entre eles. De forma bem geral e básica: um objeto é uma instância de uma classe.

3.2.1 Conceitos de Programação Orientada a Objetos

Para que a orientação a objetos seja compreendida mais facilmente, é comum que se explique o tema através de analogias do mundo real, e não faremos diferente aqui. Imagine uma fôrma para assar bolos - essa assadeira é um molde, o qual será utilizado para preparar vários bolos no mesmo formato. Apesar de os bolos terem o mesmo formato, eles podem ter diferentes sabores, como de chocolate, cenoura ou baunilha. Nessa analogia, a fôrma seria a nossa Classe, enquanto cada bolo é uma instância dessa classe, ou seja, nossos Objetos. Os sabores são os atributos ou propriedades da Classe que a cada bolo produzido pode receber um valor diferente.

As classes são moldes para produção de representações e tipos de dados. Você se lembra alguns dos tipos de dados mais importantes do Visual Basic, como Integer, String, Decimal etc.? Com classes podemos criar nossos próprios tipos de dados e estabelecer comportamentos para eles. Por exemplo, seria possível criar o tipo Coordenada (coordenada geográfica) para uso em mapas, o qual poderia ter atributos como latitude, longitude e altitude. Através de um método associado a esse tipo de dado, é possível também criar uma forma de calcular qual é a hora atual com base na faixa de fuso horário terrestre em que se encontra o ponto geográfico representado por essa coordenada.

A classe Coordenada não faz parte das bibliotecas padrão da maioria das linguagens de programação, mas podem estar em bibliotecas adicionais, criadas por terceiros, para manipulação de dados geo-espaciais. Ela também pode ser criada por um desenvolvedor de aplicativo que necessita de características

especiais para seu software. Podemos colocar a descrição da classe em um pequeno quadro que é semelhante ao utilizado em notações de desenho de sistema, como o UML (Unified Modeling Language), mas aqui não será usado seu rigor técnico por finalidade didática:

CLASSE COORDENADA
Atributos: <ul style="list-style-type: none"> - Latitude - Longitude - Altitude
Método: <ul style="list-style-type: none"> - Horário

O exemplo da classe Coordenada geográfica é um caso em que o tipo de dado é mais abstrato e podemos dizer até que é bem próximo da matemática. Mas as classes podem ser utilizadas para manipular representações de objetos reais de nosso mundo físico em softwares, como cadastro de pessoas, simuladores de linhas de produção, personagens de games etc. Vamos nos reter a um exemplo para discutir as diversas características do desenvolvimento orientado a objetos: um cadastro de imóveis de uma imobiliária.

Se uma empresa imobiliária está desenvolvendo um software para administrar seus negócios de imóveis, na verdade ele terá que utilizar representações de muitos objetos em seu sistema: funcionários (principalmente seus agentes), clientes, locais, imóveis etc. Mais especificamente, vamos utilizar algumas características dos imóveis para imaginar de forma inicial como desenhar a sua classe e seus atributos. Aqui serão utilizados apenas alguns exemplos de dados associados a imóveis, mas é possível pensar em muitos outros; o mais importante não é criar o maior número possível de atributos que esse objeto possui no mundo real, mas apenas aqueles que serão necessários para o software em desenvolvimento.

CLASSE IMÓVEL
Atributos: <ul style="list-style-type: none"> - Endereço - Metros quadrados - Número de quartos - Tipo de imóvel - Valor de oferta
Método: <ul style="list-style-type: none"> - Preço do Metro Quadrado

Os atributos são variáveis internas à classe, representando características dela. Quando se cria uma instância da classe, isso é, um objeto Imóvel, podemos "preencher" essas características, ou melhor, atribuir valores às variáveis no escopo da classe. Os métodos podem modificar os atributos ou obter valores deles. Nessa representação que adotamos até agora em nosso exemplo, os nomes de classe, atributos e método estão escritos em linguagem de comunicação humana, mas, mais adiante daremos nomes legais de linguagem de programação a todos os elementos.

Perceba que atributos são como variáveis (dados) na programação estruturada, e métodos são funções e sub-rotinas. Em várias linguagens de programação, seus usos são iguais. Algumas outras exigem formas de controle de acesso diferentes, tornando-as de acesso aberto ou não. Tanto atributos quanto métodos de uma classe são denominados seus Membros.

3.2.2 Orientação a Objetos no Visual Basic

Para implementar essa classe de Imóveis no Visual Basic, vamos criar um novo projeto do tipo Console seguindo os passos de exemplos anteriores. Nomeie o projeto como "cadastroImoveis". Após a criação do projeto, temos o modelo normal de Module1. Agora, para criar uma classe, o Visual Studio fornece uma maneira prática: na caixa "Gerenciador de Soluções", acima e à direita, clique com o botão direito do mouse no nome do projeto, cadastroImoveis; um menu de opções aparecerá, sendo uma delas "Adicionar"; colocando o mouse sobre essa opção, um outro submenu surgirá e a última opção da lista é "Classe". Veja na figura a seguir:

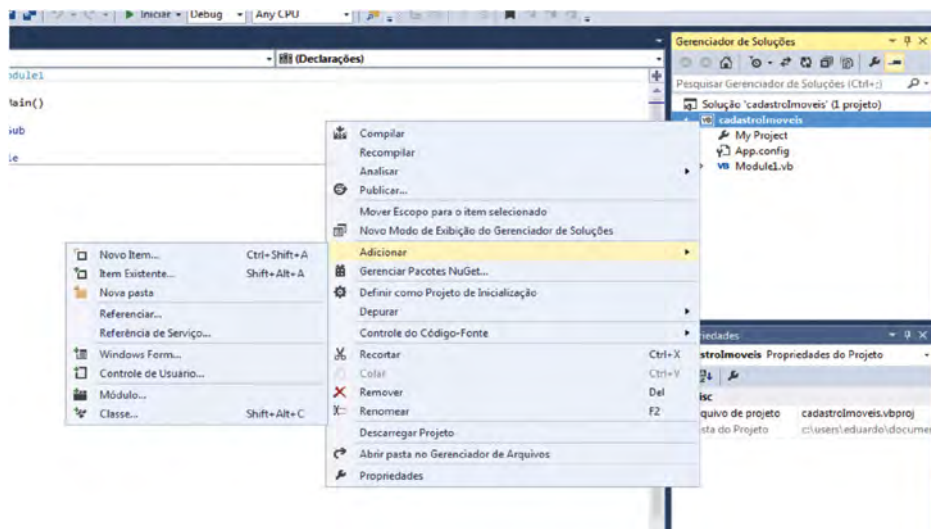


Figura 3.2 – Menu de acesso ao assistente de criação de itens ao projeto do Visual Studio.

Após clicar nessa opção "Classe", uma janela de auxílio aparecerá com diversas opções, mas "Classe" estará pré-selecionada. Dê um nome para essa classe, como "Imovel" (sem acento), e clique no botão "Adicionar". A próxima figura ilustra essa janela de criação:

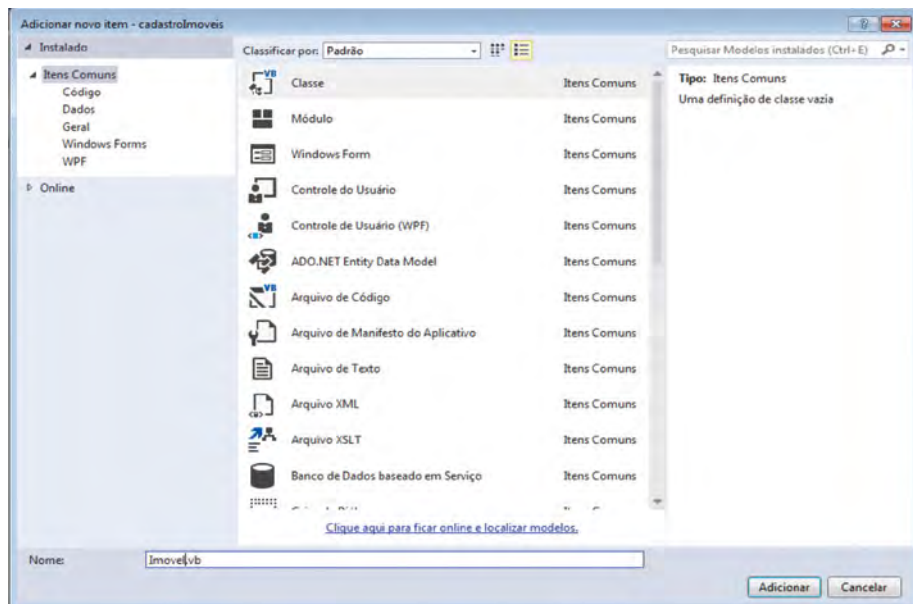


Figura 3.3 – Janela com a opção para a criação de uma nova Classe.

A adição da classe "Imovel" criará o arquivo Imovel.vb e, dentro dele, a classe Imovel. Acentos ou caracteres especiais não podem ser usados para nomear variáveis, funções ou classes em nenhuma linguagem de programação, por isso a restrição aqui também. Após a criação da classe Imovel, teremos dois arquivos abertos: Imovel.vb e Module1.vb, como na figura seguinte.

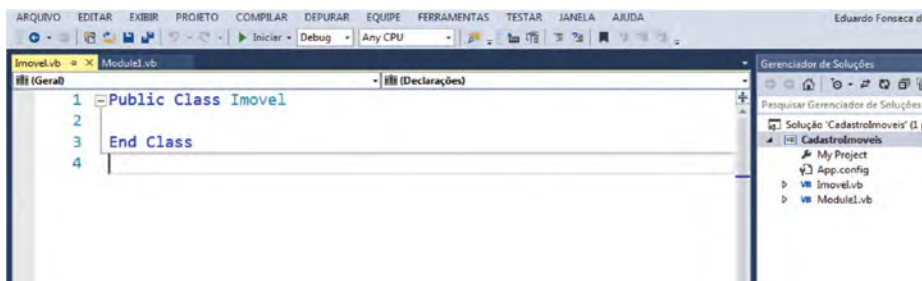


Figura 3.4 – Nova classe criada, ainda sem métodos e atributos.

Normalmente, é possível escrever o código de uma classe no mesmo arquivo que a rotina Main(), mas isso, em muitos casos, não é uma boa prática de programação - claro que isso depende da filosofia, contexto e da linguagem de programação. A divisão em arquivos é essencial como forma de organização de programas. A ideia é manter a organização do código para quem lê, mas, mais do que isso, isolar trechos de código com propósitos específicos. Em uma equipe de desenvolvimento, cada programador pode ser responsável pelo desenvolvimento de uma parte e, com a modularização, o que cada um precisa saber é a forma de comunicação com cada classe através de seus métodos e atributos.

Agora teremos, então, dois arquivos nos quais escreveremos código. Os arquivos e os respectivos códigos a serem digitados são representados nos quadros a seguir.

```
Imovel.vb
1 Public Class Imovel
2
3     Public Property Endereco As String
4     Public Property MetrosQuadrados As Decimal
5     Public Property NumeroQuartos As Integer
6     Public Property TipoImovel As String
7     Public Property ValorOferta As Decimal
8
9     Public Function PrecoMetroQuadrado() As Decimal
```



```

10     Dim Valor As Decimal
11     Valor = Me.ValorOferta / Me.MetrosQuadrados
12     Return Valor
13 End Function
14
15 End Class

```

Código 3.7 – Código de atributos e método da classe Imovel.

```

Module1.vb
1  ModuleModule1
2
3  Sub Main()
4
5      Dim novoImovel As New Imovel()
6
7      novoImovel.Endereco = "Rua do Sabão, 10"
8      novoImovel.MetrosQuadrados = 100
9      novoImovel.NumeroQuartos = 2
10     novoImovel.TipoImovel = "Apartamento"
11     novoImovel.ValorOferta = 250000
12
13     Console.WriteLine("{0} localizado em {1}",
14                         novoImovel.TipoImovel,
15                         novoImovel.Endereco)
16
17     Console.WriteLine("{0} quartos, {1} metros quadrados",
18                         novoImovel.NumeroQuartos,
19                         novoImovel.MetrosQuadrados)
20
21     Console.WriteLine("Preço: {0:C}", novoImovel.ValorOferta)
22
23     Console.WriteLine("Preço do Metro Quadrado: {0:C}",
24                         novoImovel.PrecoMetroQuadrado())
25
26     Console.ReadLine()
27
28 End Sub
29
30 End Module

```

Código 3.8 – Código do Module1, que cria uma instância da classe, um objeto, e o manipula.

Após a digitação desse código, salvamento do projeto e compilação/execução, o resultado na tela do Console será:

CONSOLE

```
Apartamento localizado em Rua do Sabão, 10  
2 quartos, 100 metros quadrados  
Preço: R$ 250.000,00  
Preço do Metro Quadrado: R$ 2.500,00
```

Temos aqui um pequeno programa que nos permitirá analisar muitos aspectos da programação orientada a objetos. Nas próximas páginas, os comentários serão relacionados aos dois arquivos de código do programa `CadastroImoveis`, portanto procure estudá-los e retornar a eles a cada comentário. Se possível, mantenha uma cópia impressa com o código ao seu lado e a rasbique com lápis para fortalecer seu aprendizado.

Antes de continuar a análise detalhada, é importante destacar que esse é um programa em que tipicamente se usa bancos de dados para guardar e resgatar os dados necessários, mas esse não é o escopo desse capítulo. Como os dados não são armazenados, lidamos apenas com sua manipulação momentânea. Se fosse utilizado um banco de dados, no momento de criação de um objeto, seus valores de atributos poderiam ser armazenados através de instruções apropriadas.

3.2.2.1 Análise do arquivo `Imovel.vb`

Esse arquivo possui apenas uma classe, que leva o mesmo nome "Imovel". Na linha 1, a classe é declarada como sendo pública, o que significa que ela pode ser manipulada diretamente:

```
Public Class Imovel
```

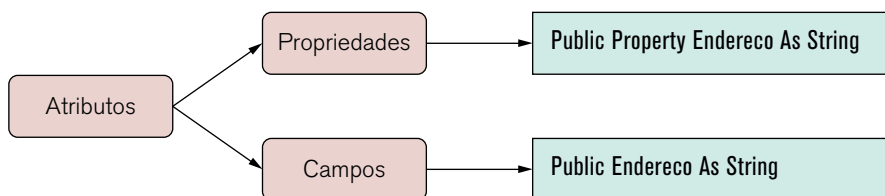
O mesmo ocorre com seus atributos entre as linhas 3 e 7, são todos declarados públicos, e por isso seus valores podem ser manipulados externamente. Se fossem privados, somente métodos dentro da classe poderiam manipulá-los, o que significa que eles estariam protegidos de acesso externo. Isso, de forma simplista, é o que se chama "encapsulamento":

```

Public Property Endereco As String
Public Property MetrosQuadrados As Decimal
Public Property NumeroQuartos As Integer
Public Property TipoImovel As String
Public Property ValorOferta As Decimal

```

Os atributos dentro de uma classe são como variáveis. Mas eles podem ser divididos em dois tipos: Propriedade e Campos. Eles são Propriedades quando usam a palavra-chave “Property”, por outro lado, um atributo é um Campo quando a palavra-chave Property não é usada. As propriedades recebem mais proteções de validação por parte do .NET, além de poderem realizar mais conexões de dados com a plataforma; apesar de poderem ser alteradas externamente de forma direta, é mais comum apenas acessá-las e alterá-las através de métodos específicos. Em nosso exemplo, utilizamos propriedades e atribuímos valores a elas sem o auxílio de métodos, mas poderíamos ter usado campos.



A função `PrecoMetroQuadrado()` também é declarada como pública na linha 9. Como essa função é interna a uma classe e não usa argumentos adicionais, não é necessário declarar argumentos entre parênteses. Como o valor a ser retornado pela função é monetário, com apenas duas casas depois da vírgula, usamos o tipo `Decimal` para declará-la:

```

Public Function PrecoMetroQuadrado() As Decimal

```

Dentro da função, há a declaração da variável `Valor`, que armazenará o resultado do cálculo do valor do metro quadrado. Essa variável apenas existe dentro do escopo desse método e deixa de existir se ele não estiver sendo executado, portanto não pode ser utilizada por outras partes do programa. Na verdade, ela só foi criada nesse exemplo para fins explicativos, pois o método funcionaria da mesma maneira se fosse escrito assim:

```
Public Function PrecoMetroQuadrado() As Decimal
Return Me.ValorOferta / Me.MetrosQuadrados
End Function
```

A linha 11 do arquivo Imovel.vb apresenta um detalhe que pode parecer estranho, que é o pequeno trecho "Me" antes das propriedades utilizadas no cálculo do preço do metro quadrado. A linha original do arquivo é:

```
Valor = Me.ValorOferta / Me.MetrosQuadrados
```

Utiliza-se "Me", no Visual Basic, quando se quer fazer referência a um atributo (propriedade) de uma classe dentro da própria classe. Essa característica é comum a diversas linguagens de programação, mas essa palavra-chave pode ser diferente para cada uma. Em Java, Javascript, C++ e C#, a palavra é "this", que analogamente, no nosso exemplo, seria algo como "this.ValorOferta". Em outras linguagens, como Python, Ruby ou Objective-C, a palavra é "self", o que resultaria em algo como "self.ValorOferta". Todas essas palavras (Me, this, self), em inglês, estão tentando expressar um pronome reflexivo (Me = "Mim", this = "isso", self = "si mesmo"). O objetivo é expressar que, por exemplo, ValorOferta é uma variável que pertence ao escopo da própria classe em que está sendo manipulada e não é uma variável que existe fora dela, a menos que se utilize o espaço de nome (namespace) do objeto. Isto é, fora do próprio objeto, o qual seja gerado a partir dessa classe, só é possível manipular um atributo utilizando a notação de ponto, como em "novoImovel.ValorOferta", e isso se ela for declarada pública.

No entanto, no Visual Basic, ao contrário de algumas outras linguagens, não é obrigatório utilizar o "Me" como em nosso exemplo. Mas, não utilizá-lo é uma má prática de programação, já que os programadores, lendo esse código que não utiliza "Me", podem não ter certeza de onde vem aquela variável. Isso pode ocorrer porque, se programadores diferentes utilizarem o nome "ValorOferta" dentro e, ao mesmo tempo, fora de uma classe, o compilador não chama atenção para isso e elas não entram em conflito devido à existência do conceito de namespace. Quer dizer, "ValorOferta" (declarado fora da classe) é diferente de "novoImovel.ValorOferta" (declarado dentro da classe), e se na programação da classe Imovel não for utilizado "Me.ValorOferta", isso pode induzir ao erro humano.

Enfim, o cálculo e retorno de Valor pela função será um tipo de dado Decimal, que pode ser usado para representar valor monetário e, no nosso caso, também pode ser utilizado para representar metros quadrados. O valor atribuído à variável Valor será retornado toda vez que for chamado o método `PrecoMetroQuadrado()` de um objeto `Imovel`. Perceba que, como nosso exemplo é simples, não há tratamento de erros e, caso não houvesse preço ou metragem atribuídos ao `novoImovel`, a função retornaria um erro da operação de divisão sem avisar que faltam valores de atributos.

3.2.2.2 Análise do arquivo `Module1.vb`

O código em `Module1` é o responsável pela execução principal do programa. Quando se utiliza o botão `Inciar` do Visual Studio ou se executa o arquivo compilado, a função `Main()` é a primeira a ser chamada e outras funções e métodos apenas serão executados se, por sua vez, chamados dentro dela. Se houver alguma função ou classe que estão declarados no código do programa, eles serão compilados junto com o resto, mas nunca serão executados se não forem chamados. No entanto, pode haver uma execução em cadeia, quer dizer, se uma função em um terceiro arquivo fosse chamado pelo método `Imovel.PrecoMetroQuadrado()`, e esse, por sua vez, chamado em `Main()`, essa terceira função hipotética também seria executada em nosso programa.

Em nosso programa, três atividades diferentes são executadas:

1. A classe `Imovel` é instanciada, gerando um objeto `novoImovel`;
2. Os atributos do objeto `novoImovel` recebem valores;
3. São impressos na tela do Console os atributos formatados e também o resultado do método `PrecoMetroQuadrado()`.

A instancição de uma classe gera um objeto. É comum o uso do termo em inglês, *instantiation*, que significa criar uma instância. Esse objeto recebe um nome, como qualquer outra variável, e passa a ocupar espaço e um endereço na memória do computador. Toda vez que quiser se referir a essa instância criada, basta se referir ao seu nome atribuído, isto é, à sua variável declarada. Essa declaração se dá por:

```
Dim novoImovel As New Imovel()
```

ou por:

```
Dim novoImovel As Imovel = New Imovel()
```

Essas duas formas são equivalentes. A instanciação de um objeto é aproximadamente como a declaração de uma variável do tipo Integer ou String.

Como no arquivo Imovel.vb, na criação da classe, declaramos suas propriedades como públicas, é possível fazer a atribuição direta de valores como em:

```
novoImovel.Endereco = "Rua do Sabão, 10"
```

Se as propriedades fossem declaradas privadas, isso não seria possível. Nesse caso, ou métodos internos à classe gerariam algum tipo de valor automático para os atributos, ou seria necessário haver um método declarado como público que fizesse essa atribuição através de passagem de argumentos, como em:

```
novoImovel.set_Endereco("Rua do Sabão, 10")
```

Essa forma não foi usada em nosso exemplo, mas faz parte da filosofia de programação voltada ao encapsulamento. Usando um método como esse é possível, por exemplo, estabelecer um algoritmo que se certifica que aquilo que foi digitado é realmente um endereço, que esse endereço ainda não está cadastrado, ou algo ainda mais sofisticado.

Depois de feitas todas as atribuições no momento de execução, os valores dos atributos são impressos na tela com formatação. Como exemplo, a primeira instrução foi digitada com quebra de linha entre cada argumento:

```
Console.WriteLine("{0} localizado em {1}",  
                    novoImovel.TipoImovel,  
                    novoImovel.Endereco)
```

Essa quebra de linha foi feita unicamente para economizar espaço horizontal. Ela é possível porque os argumentos estão entre parênteses e separados por vírgulas, então o Visual Studio compreende que as três linhas fazem parte de uma única instrução. Esse recurso é bastante utilizado por programadores para que o código não ocupe muito espaço horizontal e fique mais legível.

Quando se observa o último trecho de impressão no Console, nas linhas 23 e 24, podemos fazer algumas observações:

```
Console.WriteLine("Preço do Metro Quadrado: {0:C}",  
    novoImovel.PrecoMetroQuadrado())
```

Em primeiro lugar, chama a atenção o parâmetro de formatação {0:C}. A presença de :C faz com que o resultado na tela seja formatado como valor monetário, de acordo com a configuração local do sistema - tudo automaticamente identificado pelo Framework .NET.

Em seguida, percebe-se que a chamada do método de cálculo de preço é passado como argumento de Console.WriteLine(). Poderíamos ter declarado uma variável, como precoM2, atribuir o resultado do método a ela e depois passá-la como argumento de WriteLine(). Mas nada disso é necessário, pois quando passamos uma função como argumento, o seu valor retornado será utilizado diretamente na impressão no Console. Lembre-se que, nesse caso, o método retorna um número do tipo Decimal.

Ainda restam alguns conceitos de programação orientadas a objetos que não foram abordados no exemplo da classe imóvel, como: eventos, classes estáticas e herança.

3.2.2.3 Eventos

No próximo capítulo serão apresentados alguns conceitos de programação orientada a eventos. Esse é um paradigma de programação que se baseia na orientação a objetos. Os objetos podem receber ou transmitir sinais, chamados eventos, através de funções específicas para isso. Sempre que um evento relacionado a um objeto ocorre, essa função manipula o evento e emite um sinal.

Por exemplo, no uso de uma interface gráfica, quando se clica em um botão, esse objeto botão emite um sinal indicando que o evento de clique ocorreu. Isso desencadeará ações em outros objetos que venham a reagir em função do evento.

3.2.2.4 Classes instanciadas e classes estáticas (shared classes)

É importante fazer uma pequena observação que pode ter chamado sua atenção durante o capítulo: por que alguns objetos devem ser explicitamente declarados e instanciados e outros não? Lembre-se que foi dito que Console é uma classe pertencente ao Framework .NET, e que WriteLine() é um de seus métodos. Então por que não é necessário declará-la para utilizá-la? Exemplo:

```
Dim impressao As New Console()  
impressao.WriteLine("Olá, Mundo!")
```

Ora, essa foi a forma que utilizamos para criar um novo objeto do tipo Imovel, então há algo diferente. E, realmente, há: classes do framework são do tipo "estáticas" ou "compartilhadas" (shared), e são declaradas de uma forma especial para que não haja essa necessidade de instanciação. Quando rodamos o programa, ele gera uma instância única que é utilizada em todo o código. Em resumo: apesar do uso diferente, as classes do framework .NET ainda são classes como as que criamos.

3.2.2.5 Membros compartilhados (shared members)

Da mesma forma que há classes compartilhadas, também há membros compartilhados de uma classe. Esses membros não necessitam de instanciação para serem acessados e alterados, o que pode ser feito diretamente através do nome da classe. Para que isso ocorra, ele deve ser declarado como:

```
Shared nome_do_membro As Tipo
```

Para ilustrar esse conceito, observemos o seguinte exemplo. Observe principalmente a declaração do atributo TotalContas e suas alterações de valores nas linhas 6 e 11, onde foi feito "Conta.TotalContas = Conta.TotalContas + 1".


```

Module1.vb
1  Module Module1
2
3      Sub Main()
4          Dim conta1 As Conta
5          conta1 = New Conta
6          Conta.TotalContas = Conta.TotalContas + 1
7          conta1.Saldo = 1000
8          conta1.Exibir()
9          Dim conta2 As Conta
10         conta2 = New Conta
11         Conta.TotalContas = Conta.TotalContas + 1
12         conta2.Saldo = 2000
13         conta2.Exibir()
14
15         Console.Write("O total de contas é: ")
16         Console.WriteLine(Conta.TotalContas)
17         Console.ReadKey()
18     End Sub
19
20     Public Class Conta
21         Public Property Saldo As Decimal
22         Public Shared Total Contas As Integer = 0
23
24         Public Sub Exibir()
25             Console.Write("O saldo da sua conta é: ")
26             Console.WriteLine(Saldo)
27         End Sub
28     End Class
29
30 End Module

```

Código 3.9 – Exemplo com membro compartilhado.

Normalmente, alteraríamos um atributo relacionado a um objeto instanciado, e não da sua classe, como em `conta1.TotalContas`. No entanto, um membro compartilhado pode ter seu valor alterado diretamente em sua classe, funcionando como uma espécie de variável global.

3.2.2.6 Herança

A herança é a capacidade de uma classe em herdar características de outra em que é baseada, acrescentando atributos ou métodos que lhe forem necessários. Segundo a MSDN (2015c):

Visual Basic oferece suporte herança, que é a capacidade de definir classes que servem sistema autônomo base para classes derivadas. Classes derivadas herdam e podem estender, propriedades, métodos e eventos da classe base. Classes derivadas também podem substituir métodos herdados com novas implementações. Por padrão, todas as classes criadas com Visual Basic são herdáveis.

Isso é, para qualquer classe que um programador criar, ele pode criar classes derivadas. Por exemplo, no desenvolvimento de um sistema de controle acadêmico de uma faculdade, um programador pode criar a classe Pessoa. Dessa classe ele pode derivar Professor, Aluno e Funcionario, adicionando a cada um atributos e métodos necessários para controle no sistema, como notas e curso dos alunos.

Um outro exemplo está no código a seguir, que cria uma classe Conta, para uma conta bancária genérica. Dessa classe, outra herdada é criada: ContaRemunerada, que possui um atributo adicional (TaxaRemuneracao) e um método adicional (ExibirRemuneracao).

```
Module1.vb
1  Module Module1
2      Sub Main()
3          Dim conta1 As Conta
4          conta1 = New Conta
5          conta1.Saldo = 1000
6          Dim conta2 As ContaRemunerada
7          conta2 = New ContaRemunerada()
8          conta2.Saldo = 2000
9          conta2.TaxaRemuneracao = 1.0
10         conta2.Exibir()
11         conta2.ExibirRemuneracao()
12         Console.ReadKey()
13     End Sub
```

```

14
15     Public Class Conta
16         Public Property Saldo As Decimal
17         Public Sub Exibir()
18             Console.WriteLine("O saldo da sua conta é: ")
19             Console.WriteLine(Saldo)
20         End Sub
21     End Class
22
23     Public Class ContaRemunerada
24         Inherits Conta
25         Public Property TaxaRemuneracao As Decimal
26         Public Sub ExibirRemuneracao()
27             Console.WriteLine("A taxa de remuneração da sua conta é: ")
28             Console.WriteLine(TaxaRemuneracao)
29         End Sub
30     End Class
31 End Module

```

Código 3.10 – Exemplo de herança de classe.

A palavra-chave que indica a herança é `Inherits` (herda). Então, com `Inherits Conta`, é definida a relação de herança. Mesmo que dentro da classe `ContaRemunerada` não haja uma declaração da propriedade `Saldo`, ela possuirá esse atributo porque o herdou de sua chamada superclasse, `Conta`. Nessa relação, `ContaRemunerada` é uma subclasse e `Conta` é uma superclasse.

3.2.2.7 Modificadores de Acesso

Até agora vimos algumas palavras-chave que modificam as restrições de acesso a classes, funções, variáveis etc. Muitos elementos podem ser tornados acessíveis ou não no .NET. Agora que foram vistos os diversos usos desses modificadores em certos contextos, pode-se resumir os principais deles assim:

- `Public`: esse modificador de acesso torna o objeto, membro ou outros elementos públicos em qualquer parte do código da aplicação;
- `Protected`: não foram apresentados exemplos desse modificador, mas ele permite apenas que código dentro de uma classe ou de seus derivados (subclasses herdeiras) acessem seus elementos;

- **Private:** nesse caso, o elemento declarado privado apenas pode ser acessado pelo código em sua classe ou outro elemento a que pertença, como um módulo ou estrutura.
- **Shared:** o membro de uma classe declarado dessa maneira pode ser acessado por todos os objetos instanciados dessa classe.

Com isso foram, vistos os principais conceitos e práticas relacionados à Programação Orientada a Objetos no Visual Basic. A ideia por trás de POO pode parecer bastante abstrata para programadores iniciantes, mas releia esse trecho do capítulo, escreva códigos para praticar e pesquise bastante. Esses conceitos são bastante importantes para o desenvolvimento de software.



REFLEXÃO

O uso de tratamento de exceções é mais uma boa prática no desenvolvimento de software. Diz-se que, se é inevitável que eventualmente o software falhe para certas operações, que isso aconteça graciosamente. Mas a previsibilidade de ocorrência de exceções muitas vezes vem acompanhada com a experiência do programador associada ao conhecimento que tem das ferramentas que usa.

Quanto à orientação a objetos, é importante notar que o uso desse paradigma de programação não elimina o uso de outros. O uso de objetos em um programa não quer dizer que não se usa a programação estruturada em trechos do código. Como se verá também no Capítulo 5, a Programação Orientada a Eventos é baseada em recursos oferecidos pela orientação a objetos.



ATIVIDADES

01. Explique com suas próprias palavras o que é tratamento de exceções e por que é importante em programação.
02. Faça um pequeno programa que solicita dois números inteiros para o usuário e faz a divisão entre eles. Use tratamento de exceções para capturar o erro de divisão por zero especificamente e outra captura, genérica, para capturar outros erros. Forneça uma mensagem amigável ao usuário.

03. Explique o que são Classes, Objetos, Propriedades de Objetos e Métodos de Objetos.

04. Crie um pequeno programa que possui uma classe Retangulo, que possui as propriedades "base", "altura" e "cor". Crie um método baseado em uma sub-rotina que altera a propriedade "cor" e que se chama "definir_cor". A rotina Main deve instanciar esse objeto e definir a cor do retângulo para "azul".



LEITURA

O site da Microsoft Developer Network (MSDN) possui mais informações, e aprofunda no assunto de tratamento de exceções:

Tratamento de Exceção e Erro no Visual Basic

<<https://msdn.microsoft.com/pt-br/library/vstudio/s6da8809%28v=vs.100%29.aspx>>

O mesmo site também explica com bastante detalhes a Programação Orientada a Objetos com o Visual Basic .NET:

Objetos e classes no Visual Basic

<<https://msdn.microsoft.com/pt-br/library/527aztek.aspx>>



REFERÊNCIAS BIBLIOGRÁFICAS

MACKENZIE, Duncan; SHARKEY, Kent. **Aprenda Visual Basic.net em 21 Dias**. Makron: São Paulo, 2003.

MSDN, Microsoft Developer Network. Namespace System. Disponível em: <<https://msdn.microsoft.com/pt-br/library/System%28v=vs.90%29.aspx>>. Acesso em: 12 fev. 2015.

_____. Namespace System.IO. Disponível em: <<https://msdn.microsoft.com/pt-br/library/System.IO%28v=vs.110%29.aspx>>. Acesso em: 12 fev. 2015.

_____. Herança no Visual Basic. Disponível em: <<https://msdn.microsoft.com/pt-br/library/5x4yd9d5%28v=vs.90%29.aspx>>. Acesso em: 18 jan. 2015.

4

Banco de Dados na Plataforma Microsoft .NET

Neste capítulo, veremos vários aspectos do uso de bancos de dados: o que são, os seus benefícios para o desenvolvimento de softwares, quais seus elementos e como interagir com eles. Depois de uma apresentação geral, enfatizaremos o uso do SQL Server, que é o Sistema Gerenciador de Bancos de Dados (SGBD) da Microsoft. Será apresentado um pequeno guia de instalação do SQL Server, como utilizar sua interface gráfica, o SQL Server Management Studio, e como escrever pequenos *scripts* para criação de bancos de dados, tabelas e *views*.

Logicamente, disciplinas específicas de bancos de dados e de modelagem de dados detalham melhor o que será visto aqui, mas o nosso propósito é colocar tudo isso no contexto do desenvolvimento de um software com vários elementos, além do acesso aos dados armazenados através do .NET Framework

OBJETIVOS

Após a leitura da leitura desse capítulo e prática de suas atividades, você saberá:

- O que são bancos de dados e quais as vantagens de seu uso
 - Características do SQL Server da Microsoft
 - Como utilizar a interface gráfica SQL Server Management Studio
 - Como criar bancos de dados, tabelas e *views* no SQL Server
 - Que ADO.NET é a biblioteca do Framework .NET que se utiliza para acessar dados
-

4.1 Sistemas de Banco de Dados

Um banco de dados é uma coleção de dados relacionados. Os dados são fatos que podem ser gravados e que possuem um significado implícito (ELMASRI e NAVATH, 2005). De forma genérica, qualquer coleção de dados pode ser considerado um banco de dados, como o conjunto de pastas de prontuários de pacientes em um hospital, ou até a coleção de livro em uma biblioteca. Esses conjuntos de dados costumam ser organizados de acordo com um método, de forma que seja possível acessá-los facilmente. Apesar de a organização física de dados ser necessária em muitas situações, ela se torna ineficiente em outras, principalmente quando em grandes volumes, ou quando se necessita de acesso imediato, integração de dados, redução de redundâncias, segurança de acesso etc.

No contexto da informática, o armazenamento de dados em qualquer arquivo do tipo texto também pode ser considerado um banco de dados em um sentido amplo. Para manipular esses dados, programas específicos devem ser desenvolvidos para extrair os dados armazenados em determinado formato e processá-los. Esse tipo de armazenamento de dados ainda pode manter alguns dos problemas do armazenamento em papel. Resumidamente, alguns dos problemas apresentados por Silberschatz et al. (2006) são:

- **Redundância e inconsistência de dados:** como os programas que manipulam os dados precisam ser alterados de acordo com certas necessidades, com o tempo essas alterações podem causar repetições e conflitos de dados;
- **Dificuldade de acesso a dados:** não há métodos comuns de acesso a dados de acordo com todas as necessidades, e um programa deve ser desenvolvido para cada uma delas;
- **Isolamento de dados:** os dados podem estar dispersos em vários arquivos e até em vários computadores, não sendo isolados em um local centralizado, com um mesmo programa de acesso;
- **Problemas de integridade:** é muito mais difícil estabelecer regras de valores que podem ser preenchidos em certos campos de dados;
- **Problemas de atomicidade:** é mais difícil garantir que um conjunto de alterações relacionadas sejam feitas em um só movimento de dados, certificando que, caso haja uma falha, os dados voltem ao estado anterior a essa alteração conjunta;

- **Anomalias de acesso concorrente:** não garante que haja consistência de dados caso mais de um usuário acesse e altere dados em um intervalo de tempo muito curto;
- **Problemas de segurança:** é mais difícil controlar acessos aos dados.

Com esses problemas em vista e as necessidades que surgiram com o desenvolvimento de softwares que requisitavam mais acurácia, eficiência e controle no acesso aos dados, surgiram programas específicos para essas tarefas, os quais continuam a ganhar funcionalidades, além de manter alguns preceitos básicos de manipulação e acesso aos dados. Esses programas substituíram o armazenamento em sistemas de arquivos e atualmente são indispensáveis para softwares que lidam com muitos dados.

4.1.1 Sistema Gerenciador de Banco de Dados (SGBD)

Um Sistema Gerenciador de Banco de Dados (SGBD) é um software que intermedeia a relação entre um programa que usa os dados e os dados armazenados em si. É uma camada de abstração, a qual tira do programador do aplicativo a necessidade de otimizar de forma específica a administração e manipulação de dados.

Em geral, os SGBD fornecem uma linguagem declarativa para que o desenvolvedor do aplicativo apenas descreva quais dados quer e com quais relações entre si. O SGBD otimiza um algoritmo para fornecer esses dados, tornando-o bastante eficiente. O desenvolvedor do aplicativo usuário não necessita pensar qual o melhor algoritmo para obter essa otimização.

O tipo de banco de dados predominantemente utilizado nas últimas décadas é o relacional, apesar haver vários outros tipos. Os bancos de dados relacionais, gerenciados por um Sistema Gerenciador de Banco de Dados Relacionais (SGBDR), trabalha com tabelas que normalizam os dados, separando-os em tabelas auxiliares para que não haja repetição de dados em certos campos. Sua linguagem declarativa padrão é o SQL (Structured Query Language), que torna eficiente o uso de matemática relacional, a qual aplica métodos matemáticos para que o relacionamento entre campos de tabelas seja o mais rápido possível.

A arquitetura de implementação de um banco de dados costuma seguir um de três padrões:

- **Banco de dados embutido:** na qual uma biblioteca implementa um mecanismo de gerenciamento de dados, que utiliza poucos recursos do computador e pode ser embutido em um aplicativo. Ele não fica em funcionamento o tempo todo, como um serviço, mas apenas ocupa recursos quando solicitado pelo aplicativo. São exemplos desse tipo o SQLite, LocalDB e Firebird Versão Embedded. Exemplo de uso: navegadores de internet atuais usam para pesquisas em seu histórico de navegação quando o usuário digita palavras na caixa de endereço;

- **Banco de dados local:** são SGBDs que armazenam os dados em arquivos locais, os quais podem ser utilizados para desenvolver sistemas simples, não distribuídos. Diferentemente dos embutidos, esse SGBD não é incorporado ao aplicativo usuário, mas é utilizado de forma separada, com comunicação na mesma máquina. Um exemplo desse tipo é o MS Access.

- **Arquitetura cliente-servidor:** em que o SGBD funciona como um servidor, que pode rodar em uma máquina local, mas, geralmente, em um ambiente de produção, fica em um servidor dedicado à função. Esse software servidor terá um endereço na rede, ao qual um ou vários clientes podem se conectar. Exemplos de sistemas desse tipo são SQL Server, MySQL, Oracle e PostgreSQL. Os sistemas empresariais ou sites da internet que utilizam bancos de dados são exemplos de uso dessa forma.

Também é possível desenvolver aplicativos que possuem um banco de dados embutido que, de tempos em tempos, sincroniza com um banco de dados central, principalmente em dispositivos móveis, que não possuem garantia de conexão o tempo todo.

Seguindo uma arquitetura cliente-servidor, o SGBD fica disponível para conexão de diversos aplicativos, os quais podem ser iguais e instalados em vários computadores em rede, ou diferentes entre si, cada um acessando dados de forma específica. Como fica no meio, entre os aplicativos e o banco de dados, o SGBD costuma ser considerado um *middleware*. Junto com o Banco de Dados, forma um sistema de banco de dados, como mostra a seguinte figura.

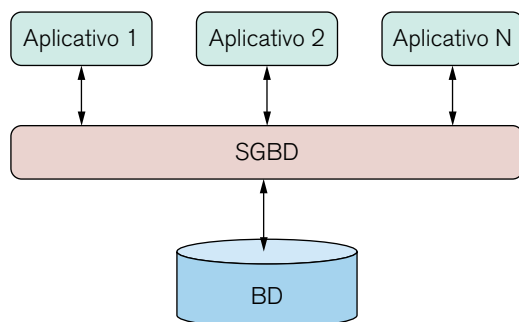


Figura 4.1 – Vários aplicativos conectados a um sistema de banco de dados.

Apesar de diversos sistemas de bancos de dados (SQL Server, Oracle, MySQL, PostgreSQL etc.) utilizarem arquitetura semelhante, eles possuem diferenças importantes. As formas de armazenar os dados em disco são diferentes, portanto não é simples abrir um conjunto de dados de um sistema em outro - se for necessário mudar o sistema, é necessário usar ferramentas de migração. Além disso, apesar de usarem o SQL para interação com programas, nenhum desses sistemas segue o padrão da linguagem com rigor, cada um com pequenas diferenças, o que algumas pessoas chamam de "dialeto" da linguagem. Apesar das variações de SQL, alguém que a aprende em um sistema não tem muita dificuldade em utilizar outro, bastando alguma prática.

Quando se utiliza um SGBD centralizado, na arquitetura cliente-servidor, é possível eliminar muitos problemas decorrentes de armazenamento de dados em arquivos de texto ou planilhas ou relacionados a bancos de dados não centralizados. Alguns deles são apresentados por Elmasri e Navath (2005) e por Silberschatz et al. (2006):

- Separação entre programas e dados: permite que o desenvolvimento seja separado e menos complexo;
- Armazenamento persistente de dados: não é necessário ler os dados de um arquivo e interpretá-los para uso em memória;
- Padronização de dados: os quais seguirão um esquema geral de acordo com um tipo atribuído a seus campos e pela convenção do desenho do sistema;
- Compartilhamento de dados: os mesmos dados ficam disponíveis em rede, sem diferenças para cada usuário;

- Controle de redundância e restrições de integridade: a normalização dos dados e as regras das tabelas impõem certos padrões;
- Restrições de acesso: podem ser estabelecidas regras de acesso a certos conjuntos de dados para cada usuário ou aplicativo;
- Eliminação de código redundante nas aplicações para gerenciamento de dados e flexibilidade: os aplicativos não precisam se preocupar com boa parte do gerenciamento dos dados;
- Disponibilidade: com os dados em um servidor, ele pode ser gerenciado de forma a funcionar o maior tempo possível sem problemas;
- Escalabilidade: o servidor de banco de dados pode ter sua capacidade aumentada à medida que a quantidade de dados e sua necessidade aumentam.

4.1.2 Elementos de um Banco de Dados Relacional

O **processo de Modelagem de Dados** é aquele em que se desenha o conjunto e relacionamento de dados em um BD. Para bancos de dados relacionais, isso inclui a especificação de suas tabelas e o relacionamento entre elas. A modelagem de dados exige o uso de ferramentas formais para documentar o modelo de dados, entre as quais a mais utilizada é o Modelo Entidade-Relacionamento (MER). Esse modelo especifica, de forma textual ou gráfica, as relações entre entidades, sendo que sua ferramenta gráfica é chamada Diagrama de Entidade-Relacionamento (DER).

No contexto da modelagem de dados, uma **Entidade** é um objeto do mundo físico, o qual pode ser um objeto concreto, abstrato ou conceitual. Um exemplo desses tipos de objetos poderia ser identificado no desenho de um sistema de gerenciamento de ideias de funcionários em uma empresa. Ele conteria, entre outros, os seguintes objetos:

- Pessoas: são objetos concretos, que podem ser os criadores de ideias ou os colegas que fazem uma avaliação da ideia;
- Ideias: que são objetos abstratos, podendo conter qualquer tipo de ideia que possa ajudar a empresa em suas atividades;
- Avaliações: que é um conceito, dependendo de muitas coisas para que se chegue a um consenso do que é uma avaliação de uma ideia.

As Entidades possuem um conjunto de **Atributos**, que são características que descrevem ou compõem as Entidades. Cada entidade pode ter um ou mais Atributos, e cada Atributo pode ter um conjunto de valores possíveis, o qual é chamado domínio ou conjunto de atributos. Como é possível perceber, os conceitos de Entidades em bancos de dados e de Objetos na programação Orientada a Objetos são muito parecidos.

Um banco de dados relacional é formado por **Tabelas**, sendo que as instâncias das entidades preencherão as linhas em uma tabela, e os seus atributos são representados em suas colunas. Cada instância de Entidade deve ser identificada univocamente, isto é, deve possuir uma identificação única em sua tabela, seja através de um ou mais atributos em conjunto. Essa identificação única é chamada **Super-chave**.

Como exemplo de Super-chave, podemos considerar uma tabela de cidadãos cadastrados no banco de dados da Polícia Federal. Sabe-se que nem todo cidadão possui CPF, que é um número exclusivo em todo país, mas não obrigatório para todos, sendo um identificador da Receita Federal. Por outro lado, a maioria das pessoas deveria ter um RG, que não é um número de identificação exclusivo no país, mas por Estado. Assim, uma Super-chave poderia ser formada pelo conjunto RG e Estado de registro.

Nesse exemplo, o conjunto RG-Estado também pode ser considerado uma **Chave Candidata**, porque nenhum dos dois campos individualmente pode ser considerado uma Super-chave. Se, realmente, a dupla RG-Estado não pode ser repetido, então o trio CPF-RG-Estado não pode ser considerado uma Chave Candidata, pois o subconjunto RG-Estado já identifica exclusivamente cada Entidade. Assim, o projetista do banco de Dados, confirmando que o código RG-Estado não pode ser repetido, pode utilizá-lo como uma **Chave Primária**.

Com isso, os conceitos de chaves utilizadas na modelagem de bancos de dados são definidos por Silberschatz et al. (2006) e citados ou adaptados a seguir:

- Super-chave: "é um conjunto de um ou mais atributos que nos permitem identificar unicamente uma entidade";
- Chaves Candidatas: são super-chaves para as quais nenhum subconjunto possa ser uma super-chave;
- Chaves Primárias: "são chaves candidatas escolhidas pelo projetista do banco de dados como de significado principal para a identificação de entidades".

Com a definição de chaves primárias, é possível estabelecer relações entre as entidades em banco de dados. Voltemos ao exemplo de um sistema de registro de ideias de funcionários em uma empresa: as pessoas, suas ideias e as avaliações que colegas fazem dessas ideias são entidades que se relacionam. Os funcionários geralmente possuem um código de identificação dado pelo sistema de gestão de recursos humanos - sua chave primária pode ser seu CPF, mas é mais comum o uso de um número de série do sistema de RH, para que os números sejam mais curtos e os CPF não sejam divulgados amplamente. O sistema de Gestão de Ideias também terá que criar números de identificação para as próprias ideias assim como para suas avaliações. Note, portanto, que é comum utilizar chaves primárias que não se baseiam em outros atributos das entidades, mas em números gerados pelo próprio sistema.

Nesse mesmo exemplo, uma pessoa pode gerar várias ideias, e cada ideia pode receber várias avaliações. As chaves primárias de cada entidade vão fazer a relação entre uma tabela e outra. Com isso podemos perceber que um **Relacionamento** é uma associação entre uma ou várias entidades (SILBERSCHATZ, 2006).

Quando as tabelas são desenhadas e os relacionamentos estabelecidos em uma modelagem de dados, são necessárias regras que definam sua implementação:

- Regras de Entidades: regras para manter a integridade das entidades nas tabelas, sendo que as chaves primárias não podem ser nulas, modificadas ou deixar de baterem;
- Regras de Relacionamentos: a maneira que os relacionamentos serão estabelecidos entre as tabelas, o que implicará na forma de normalização das tabelas;
- Regras de Atributos: são aquelas que determinam o domínio e o preenchimento dos campos das tabelas, por exemplo, estabelecendo tipos de dados, valores máximos ou mínimos e operação a serem realizadas nos valores.
- Regras de negócio: são aquelas que determinam o conteúdo dos atributos de forma que eles reflitam valores reais em conformidade com seu uso, evitando dados de má qualidade e valores irreais.

A estrutura geral de tabelas e seus relacionamentos fica definida em seu **Esquema**, o qual gera um projeto em linguagem formal que é suportada pelo SGBD. O Esquema é como se fosse uma planta, um projeto do banco de dados, que o SGBD guarda e utiliza para manter os relacionamentos e suas regras na geração de operações relacionais durante seu funcionamento.

4.1.3 SQL Server

O SQL Server é o sistema gerenciador de bancos de dados da Microsoft. É um software comercial e proprietário bastante utilizado em empresas. Há uma versão gratuita que usaremos no restante desse livro, o SQL Server Express, que pode ser usado gratuitamente por qualquer um, mas possui um limite de armazenamento de 10 GB, o qual não existe para a versão paga. O SQL Server Express pode ser usado inclusive para desenvolvimento de softwares com fins comerciais. Caso seja necessário armazenar mais dados do que permite sua versão gratuita, e possível migrar para uma versão paga. A partir daqui usaremos esse SGBD para mostrar sua integração com o .NET Framework, Visual Studio e Visual Basic.

CONEXÃO

Para baixar o SQL Server, acesse o link: <http://microsoft.com/sqlserver>

Tenha certeza de baixar a versão Express.

Assim como o Visual Studio, a empresa nomeia versões com o ano de lançamento. Nós usaremos nessa disciplina a versão **Express**. Para fazer o download, você deverá usar uma conta Microsoft, sendo que você já pode possuir uma caso utilize o Outlook online, Skype, ou usava o comunicar MSN.

Preencha o pequeno cadastro com seus dados e a versão que deseja baixar - nesse momento é necessário saber se seu computador tem o Windows de 32 ou 64 bits, pois a escolha da versão errada acarretará em erro na instalação. Selecione a alternativa que possui a interface gráfica do gerenciador de banco de dados, incluída entre outras ferramentas (*tools*), algo como "**SQL Server 2014 Express with Tools 32 Bit**" se seu Windows for de 32 bits. Escolha língua

e país e, caso não queira receber propaganda da empresa, certifique-se que não há nada marcado autorizando isso. Um grande arquivo executável será baixado e isso pode demorar um tempo, dependendo de sua conexão com a internet. Depois de baixar o arquivo de instalação, execute-o e verá a tela a seguir.

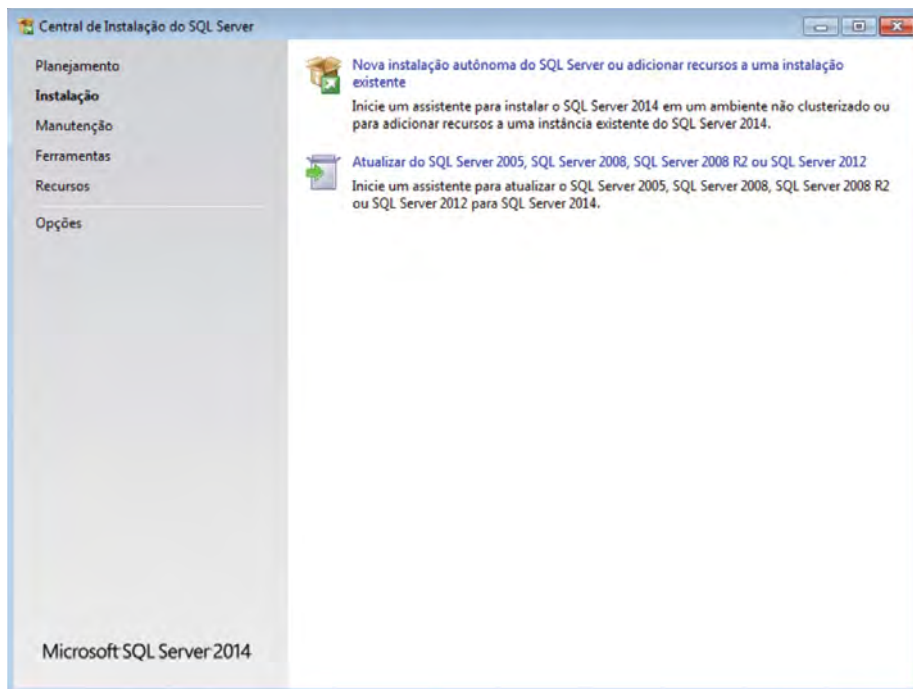


Figura 4.2 – Primeira tela da instalação do SQL Server.

Escolha a primeira opção, "Nova instalação autônoma..." e, em seguida, basta aceitar todas as opções padrões no passo a passo. Várias das configurações são de interesse de um administrador de bancos de dados, e seu interesse é apenas no desenvolvimento de um aplicativo. Tenha certeza de possuir espaço em disco, pois essa instalação pode ocupar alguns *gigabytes*.

4.1.4 Tipos de dados do SQL Server

Os bancos de dados utilizam tipos de dados para os atributos das entidades, da mesma forma que as linguagens de programação usam tipos de dados para suas variáveis. O objetivo da definição de dados é otimizar seu armazenamento

em disco, espaço em memória e operações específicas. Aqui são apresentados brevemente os tipos de dados disponíveis no SQL Server.

Os dados numéricos exatos são aqueles que representam números com precisão fidedigna. Podem ser números inteiros ou decimais com número definido de casas após a vírgula.

NUMÉRICOS EXATOS	
bigint	-2^63 (-9.223.372.036.854.775.808) a 2^63-1 (9.223.372.036.854.775.807) Ocupa 8 bytes de armazenamento.
bit	Tipo de dados inteiro que pode armazenar os valores 1, 0 ou NULL.
decimal	Tipo numérico que pode ter precisão e escala definidos. Exemplo: [Coluna_exemplo decimal(5,2)] Assim, essa coluna armazenará valores como 123,00. Precisão é o numero total de dígitos, e escala é a quantidade de dígitos após o ponto decimal. A quantidade de bytes ocupados depende de precisão e escala definidos.
int	-2^31 (-2.147.483.648) a 2^31-1 (2.147.483.647) Ocupa 4 bytes de armazenamento.
money	-922.337.203.685.477,5808 a 922.337.203.685.477,5807 Ocupa 8 bytes de armazenamento.
numeric	É equivalente ao tipo decimal.
smallint	-2^15 (-32.768) a 2^15-1 (32.767) Ocupa 2 bytes de armazenamento.
smallmoney	-214.748,3648 a 214.748,3647 Ocupa 4 bytes de armazenamento.
tinyint	0 a 255 Ocupa 1 byte de armazenamento.

Tabela 4.1 – Tipos de dados numéricos exatos do SQL Server.

Os números aproximados são aqueles que utilizam ponto flutuante para representar valores aproximados de dízimas ou redução de valores muito longos.

NUMÉRICOS APROXIMADOS	
float	Ocupada até 8 bytes, representando valores: [-1,79E+308 a -2,23E-308]; [0]; [2,23E-308 a 1,79E+308]
real	0 a 255 Ocupa 1 byte de armazenamento.

Tabela 4.2 – Tipos de dados numéricos aproximados do SQL Server.

Os dados de data e hora podem ser armazenados em diferentes formatos: apenas hora, apenas data, data e hora baseados em fuso horário e outros.

DATA E HORA	
Date	Define uma data no SQL Server. Ocupa 3 bytes. Formato da literal: AAAA-MM-DD Intervalo: 000-01-01 a 9999-12-31
datetime2	Define uma data combinada com uma hora do dia que se baseia em um período de 24 horas. datetime2 pode ser considerada uma extensão do tipo datetime existente, que tem um intervalo maior de datas, uma precisão fracionária padrão mais ampla e precisão opcional especificada pelo usuário. Ocupa 6 bytes Formato da literal: YYYY-MM-DD hh:mm:ss[segundos fracionários] Intervalo: 0001-01-01 a 9999-12-31
datetime	Define uma data combinada com uma hora do dia que inclui frações de segundos e se baseia em um período de 24 horas. Ocupa 8 bytes Intervalo de datas: Janeiro 1, 1753, a dezembro 31, 9999 Intervalo de horas: 00:00:00 a 23:59:59.997
datetimeoffset	Define a data combinada com uma hora de um dia que possui reconhecimento de fuso horário e é baseada em um relógio de 24 horas. Ocupa 10 bytes. Formato da literal: AAAA-MM-DD hh:mm:ss[. nnnnnnn] [{+ -}hh:mm] Intervalo de datas: 0001-01-01 a 9999-12-31 Intervalo de horas: 00:00:00 a 23:59:59.9999999
smalldatetime	Define uma data que é combinada com uma hora do dia. A hora se baseia em um dia de 24 horas, com segundos sempre zero (:00) e sem frações de segundo. Ocupa 4 bytes. Intervalo de datas: 1900-01-01 a 2079-06-06 Intervalo de horas: 00:00:00 a 23:59:59
time	Define uma hora de um dia. A hora se encontra sem reconhecimento de fuso horário e se baseia em um relógio de 24 horas. Formato da literal: hh:mm:ss[. nnnnnnn] Intervalo: 00:00:00.0000000 a 23:59:59.9999999 Ocupa 5 bytes

Tabela 4.3 – Tipos de dados de data e hora do SQL Server.

As cadeias de caracteres são usadas para armazenar textos e símbolos, inclusive podendo conter textos muito grandes. Esses tipos não suportam o padrão Unicode.

CADEIAS DE CARACTERES	
char	Dados de cadeia de caracteres não Unicode de comprimento fixo. Uso: char(n), onde n é o tamanho da cadeia e pode ser de 0 a 8.000. Número de bytes ocupados é igual a n.
text	Os não Unicode de comprimento variável na página de código do servidor e com um comprimento máximo de cadeia de caracteres de $2^{31}-1$ (2.147.483.647).

CADEIAS DE CARACTERES	
varchar	Dados de cadeia de caracteres não Unicode de comprimento variável. Uso: char(n), onde n é o tamanho máximo da cadeia e pode ser de 0 a 8.000. Número de bytes ocupados é igual a n.

Tabela 4.4 – Tipos de dados de caracteres do SQL Server.

CADEIAS DE CARACTERES UNICODE	
nchar	Dados de cadeia de caracteres Unicode de comprimento fixo. Uso: nchar(n), onde n é o tamanho da cadeia e pode ser de 0 a 4.000. Número de bytes ocupados é igual a 2×n.
ntext	Dados Unicode de comprimento variável com um comprimento máximo de cadeia de caracteres de 2 ³⁰ - 1 (1.073.741.823) bytes. O tamanho de armazenamento, em bytes, é duas vezes o comprimento da cadeia de caracteres inserido.
nvarchar	Dados de cadeia de caracteres Unicode de comprimento variável. Uso: nchar(n), onde n é o tamanho máximo da cadeia e pode ser de 0 a 4.000. Número de bytes ocupados é igual a 2×n.

Tabela 4.5 – Tipos de dados de caracteres Unicode do SQL Server.

CADEIAS DE CARACTERES BINÁRIAS	
binary	Dados binários de comprimento fixo com um comprimento de n bytes, em que n é um valor de 1 a 8.000. O tamanho de armazenamento é n bytes. Uso: binary(n)
image	Dados binários do comprimento variável de 0 a 2 ³¹ -1 (2.147.483.647) bytes.
varbinary	Dados binários de comprimento variável. n pode ser um valor de 1 a 8.000. Uso: varbinary(n)

Tabela 4.6 – Tipos de dados binários do SQL Server.

Outros tipos de dados são aqueles que não caem em categorias anteriores e são distintos entre si. Incluem arquivos XML, como os de Nota Fiscal Eletrônica, ou coordenadas geográficas, que podem ser usadas para sistemas de mapas.

OUTROS TIPOS DE DADOS	
cursor	Um tipo de dados para parâmetros OUTPUT de variáveis ou procedimento armazenado que contém uma referência a um cursor. Serve para manipulações do banco de dados.
hierarchyid	Hierarchyid é um tipo de dados de tamanho variável. Usa-se hierarchyid para representar posição em uma hierarquia, e assim é possível fazer uma relação hierárquica entre as entidades no BD.
sql_variant	Um tipo de dados que armazena valores de vários tipos de dados que o SQL Server. Ele é como um coringa para dados.
table	É um tipo de dados especial que pode ser usado para armazenar um conjunto de resultados para processamento posterior. table é utilizada principalmente para o armazenamento temporário de um conjunto de linhas retornadas como o conjunto de resultados de uma função com valor de tabela.

OUTROS TIPOS DE DADOS	
timestamp	É um tipo de dados que expõe números binários exclusivos, gerados automaticamente, em um banco de dados. Serve para controlar versões através de um tempo relativo.
uniqueidentifier	É um Identificador Único Global (GUID - Globally Unique Identifier) de 16 bytes. Um número único sequencialmente gerado.
xml	É um Identificador Único Global (GUID - Globally Unique Identifier) de 16 bytes. Um número único sequencialmente gerado.
Tipos espaciais	Serve para armazenar coordenadas geográficas e geometrias formas por elas.

Tabela 4.7 – Outros tipos diversos de dados do SQL Server.

Apesar de a maioria dos desenvolvedores utilizar poucos desses tipos de dados na maior parte do tempo, alguns deles são de grande utilidade para softwares específicos, em que problemas podem ser resolvidos com um tipo adequado.

CONEXÃO

Se precisar saber mais sobre os tipos de dados do SQL Server, consulte essa página da Microsoft:

[<http://msdn.microsoft.com/pt-br/library/ms187752.aspx>](http://msdn.microsoft.com/pt-br/library/ms187752.aspx)

4.1.5 Desenvolvendo um BD no SQL Server

SQL Server Management Studio e SQL Server são coisas diferentes. O SQL Server Management Studio é uma ferramenta, um software aplicativo, que se conecta ao servidor de bancos de dados e oferece uma forma gráfica de administrá-lo - ele se conecta ao banco de dados e emite comandos em formato texto ao servidor, os quais são gerados a partir de nossas ações na interface gráfica. Por isso, quando abrimos o SQL Server Management Studio, ele solicita dados de conexão ao gerenciador de banco de dados. Além disso, através do Management Studio, vários programadores e administradores de bancos de dados podem se conectar ao mesmo tempo ao servidor que pode estar instalado em um computador separado.

Utilizar o SQL Server Management Studio é uma das formas de criar, modificar e gerenciar bancos de dados e tabelas no SQL Server. Também é possível modificar tabelas e dados a partir de nosso próprio programa, como veremos

mais adiante. No entanto, aqui utilizaremos o SQL Server Management Studio para criar o banco de dados e tabelas necessárias para nosso projeto que servirá de exemplo, um sistema simples de controle de clientes para uma empresa.

O procedimento padrão do SQL Server deverá ter instalado também o SQL Server Management Studio em seu computador, colocando um ícone no menu de programas do botão Iniciar do Windows. Procure esse programa e o execute. A primeira tela a ser exibida é a de *login*.

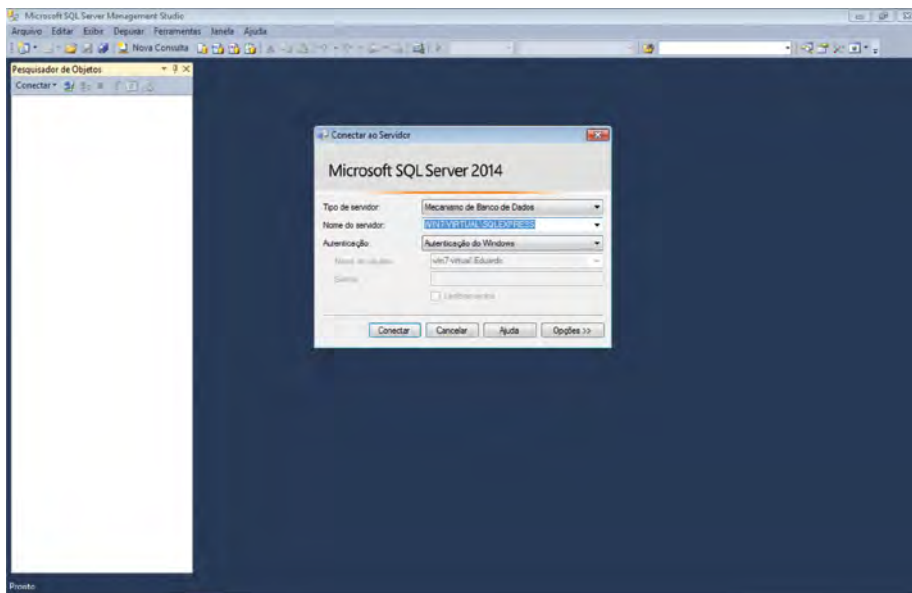


Figura 4.3 – Tela de login e conexão ao banco de dados.

Se você fez toda a instalação do SQL Server com as configurações padrões, basta clicar em "Conectar" com o caminho para seu computador selecionado. O Management Studio usará sua autenticação do Windows para se conectar ao SQL Server, sem necessitar de uma senha. A tela seguinte mostra que, navegando pelo Pesquisador de Objetos há apenas bancos de dados do sistema, mas nenhum criado por você.

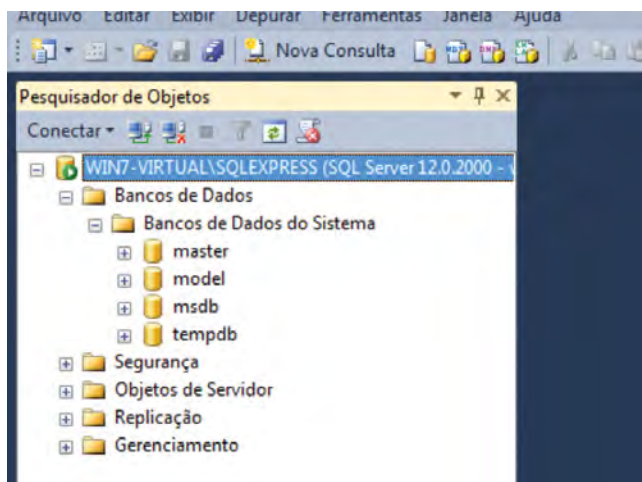


Figura 4.4 – Pesquisador de Objetos exibindo bancos de dados do sistema.

Nós criaremos bancos de dados e tabelas para nosso próprio uso a seguir.

4.2 Banco de Dados no .NET Framework

Daqui até o final do capítulo, vamos executar uma sequência de passos ilustrando o uso do SQL Server Management Studio para criar os bancos de dados e tabelas necessárias para o exemplo prático ilustrativo, um sistema simples de controle de vendas. Ao final do capítulo será apresentada a biblioteca ADO.NET, que permite a conexão das linguagens .NET com bancos de dados e a manipulação desses dados.

4.2.1 Script de Criação de BD

Quando se utiliza o SQL Server Management Studio, muitas das atividades para gerenciamento dos bancos de dados pode ser feito de forma gráfica ou através de comandos SQL. Faremos aqui das duas formas, primeiro criando um banco de dados "vendas" pela interface gráfica. Clique com o botão direito do mouse no item "Bancos de Dados" do Pesquisador de Objetos e escolha a opção "Novo Banco de Dados...".

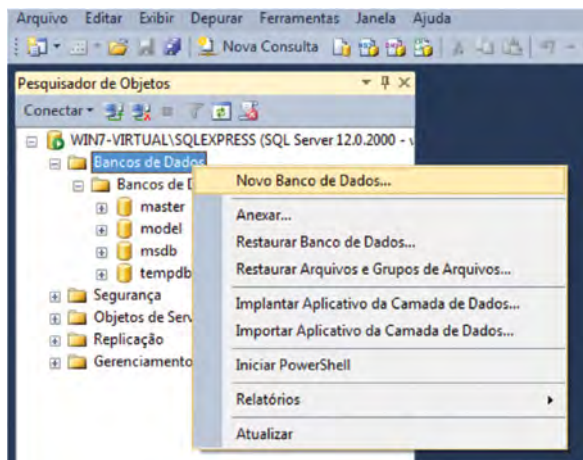


Figura 4.5 – Menu com opção de criação de banco de dados.

Uma janela com um campo para o nome do banco de dados se abrirá. É possível configurar muitas opções do banco de dados nessa janela, mas não nos preocuparemos com isso, criando um que possui configuração padrão. Veja, no quadro abaixo do nome, que serão criados dois bancos de dados: "vendas" e "vendas_log". O primeiro conterà nossos dados, enquanto o segundo serve para registrar alterações nos dados, servindo para fazer controles de segurança e recuperar dados caso necessário.

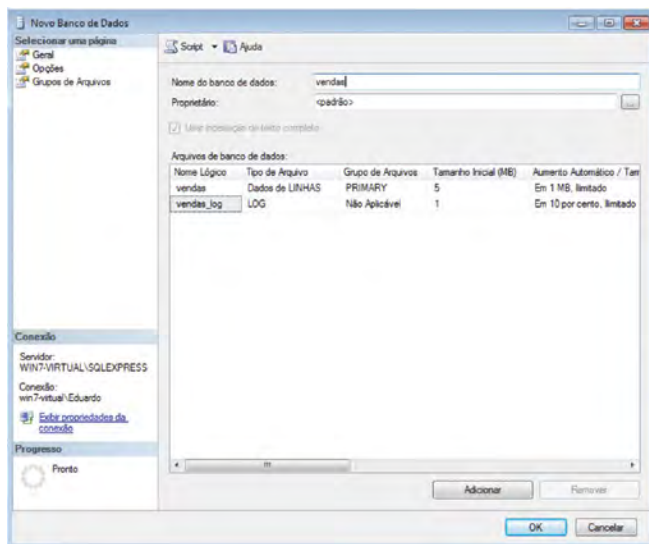


Figura 4.6 – Janela de configuração e criação de novo banco de dados.

Após pressionar o botão OK, o banco de dados será criado. Para ele aparecer na árvore de itens do Pesquisador de Objetos é necessário clicar no item "Bancos de Dados" com o botão direito do mouse e escolher "Atualizar". Com isso, o banco de dados "vendas" aparecerá, mas perceba que ele ainda não possui nossas tabelas, mas apenas tabelas do sistema.

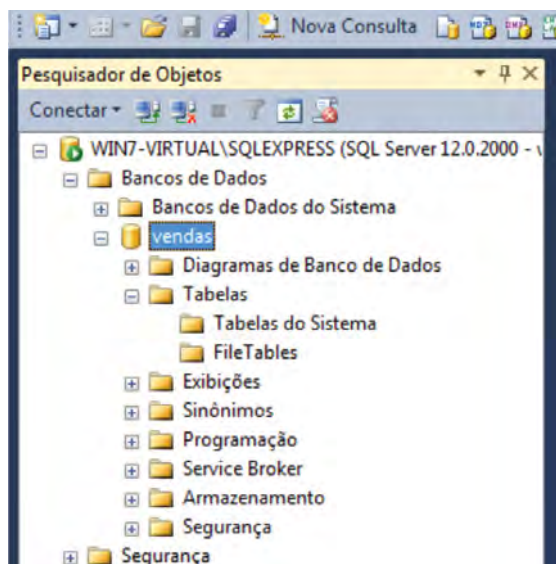


Figura 4.7.

Esses passos de criação de banco de dados pela interface gráfica podem ser substituídos por apenas um comando SQL:

```
CREATE DATABASE vendas
```

Para emitir um trecho de código SQL diretamente ao SGBD, basta abrir a janela de consultas, com o botão "Nova consulta" na barra superior de ferramentas. Na aba que se abre, qualquer código SQL digitado causará efeitos no servidor.

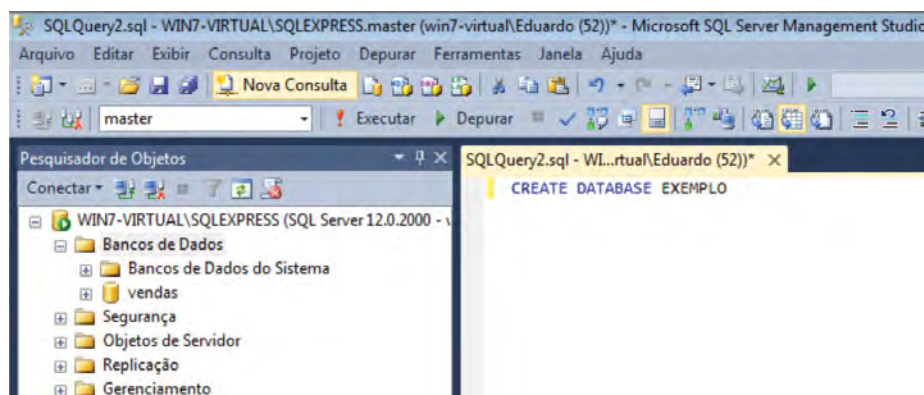


Figura 4.8 – Campo para composição de comandos SQL.

Se você digitar o código da figura e clicar no botão "Executar", um novo banco de dados chamado EXEMPLO será criado, com as mesmas configurações do nosso "vendas". Se quiser apagar um banco de dados, na interface gráfica é necessário clicar com o botão direito do mouse sobre o banco de dados desejado e escolher "Excluir". Para excluir o banco de dados através de SQL, basta usar a instrução DROP:

```
DROP DATABASE EXEMPLO
```

Quando utilizamos a linha de comando "CREATE DATABASE vendas", é importante saber que ela é a mais simples possível para a criação de um novo banco de dados. No entanto, sempre se pode utilizar um *script* com mais expressões que façam configurações do BD diferentes das padrões. Nesse caso também é possível fazer a mesma coisa pela interface gráfica ou através de uma *query*. Caso muitas configurações tenham sido feitas a um banco de dados em sua criação, é interessante guardar o código SQL que criaria um banco de dados semelhante. Para isso, clique com o botão direito do mouse sobre o ícone do banco de vendas, como na figura, e escolha as opções que permitam salvar um *script* do tipo CREATE em um arquivo no computador.

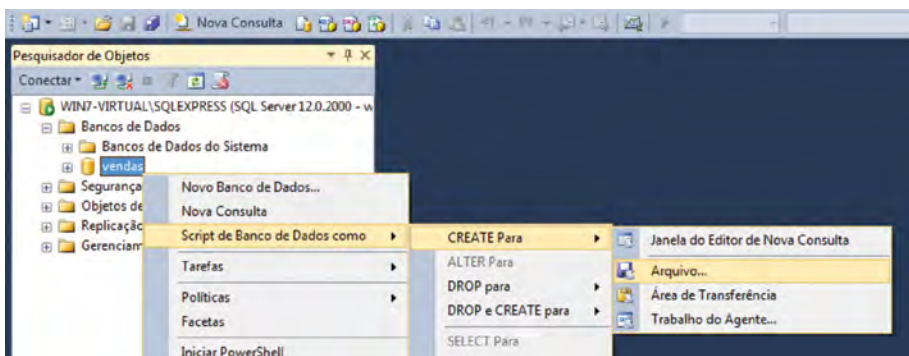


Figura 4.9

Salve esse arquivo onde desejar, com um nome parecido com "create_vendas". Automaticamente, esse arquivo receberá uma extensão .sql. Esse é um arquivo texto e é possível abri-lo com o programa Bloco de Notas do Windows. Ele terá um conteúdo parecido com o seguinte.

```
USE [master]
GO
/***** Object: Database [VENDAS] Script Date: 07/18/2011 22:25:04
*****/
CREATE DATABASE [VENDAS] ON PRIMARY
(NAME = N'VENDAS', FILENAME = N'c:\Program Files\Microsoft SQL
Server\MSSQL10_50.SQLEXPRESS\MSSQL\DATA\VENDAS.mdf' , SIZE = 3072KB ,
MAXSIZE = UNLIMITED, FILEGROWTH = 1024KB )
LOG ON
(NAME = N'VENDAS_log', FILENAME = N'c:\Program Files\Microsoft
SQL Server\MSSQL10_50.SQLEXPRESS\MSSQL\DATA\VENDAS_log.ldf' , SIZE =
1024KB , MAXSIZE = 2048GB , FILEGROWTH = 10%)
GO
ALTER DATABASE [VENDAS] SET COMPATIBILITY_LEVEL = 100
GO
IF (1 = FULLTEXTSERVICEPROPERTY('IsFullTextInstalled'))
begin
EXEC [VENDAS].[dbo].[sp_fulltext_database] @action = 'enable'
end
```

```
GO
ALTER DATABASE [VENDAS] SET ANSI_NULL_DEFAULT OFF
GO
ALTER DATABASE [VENDAS] SET ANSI_NULLS OFF
GO
ALTER DATABASE [VENDAS] SET ANSI_PADDING OFF
GO
ALTER DATABASE [VENDAS] SET ANSI_WARNINGS OFF
GO
ALTER DATABASE [VENDAS] SET ARITHABORT OFF
GO
ALTER DATABASE [VENDAS] SET AUTO_CLOSE OFF
GO
ALTER DATABASE [VENDAS] SET AUTO_CREATE_STATISTICS ON
GO
ALTER DATABASE [VENDAS] SET AUTO_SHRINK OFF
GO
ALTER DATABASE [VENDAS] SET AUTO_UPDATE_STATISTICS ON
GO
ALTER DATABASE [VENDAS] SET CURSOR_CLOSE_ON_COMMIT OFF
GO
ALTER DATABASE [VENDAS] SET CURSOR_DEFAULT GLOBAL
GO
ALTER DATABASE [VENDAS] SET CONCAT_NULL_YIELDS_NULL OFF
GO
ALTER DATABASE [VENDAS] SET NUMERIC_ROUNDABORT OFF
GO
ALTER DATABASE [VENDAS] SET QUOTED_IDENTIFIER OFF
GO
ALTER DATABASE [VENDAS] SET RECURSIVE_TRIGGERS OFF
GO
ALTER DATABASE [VENDAS] SET DISABLE_BROKER
GO
```

```

ALTER DATABASE [VENDAS] SET AUTO_UPDATE_STATISTICS_ASYNC OFF
GO
ALTER DATABASE [VENDAS] SET DATE_CORRELATION_OPTIMIZATION OFF
GO
ALTER DATABASE [VENDAS] SET TRUSTWORTHY OFF
GO
ALTER DATABASE [VENDAS] SET ALLOW_SNAPSHOT_ISOLATION OFF
GO
ALTER DATABASE [VENDAS] SET PARAMETERIZATION SIMPLE
GO
ALTER DATABASE [VENDAS] SET READ_COMMITTED_SNAPSHOT OFF
GO
ALTER DATABASE [VENDAS] SET HONOR_BROKER_PRIORITY OFF
GO
ALTER DATABASE [VENDAS] SET READ_WRITE
GO
ALTER DATABASE [VENDAS] SET RECOVERY SIMPLE
GO
ALTER DATABASE [VENDAS] SET MULTI_USER
GO
ALTER DATABASE [VENDAS] SET PAGE_VERIFY CHECKSUM
GO
ALTER DATABASE [VENDAS] SET DB_CHAINING OFF
GO

```

Código 4.1 – Script SQL de criação do banco de dados criado automaticamente.

Esse *script* possui diversos comandos SQL que determinam configurações do BD sendo criado. Veja que cada uma das linhas de configurações determina uma palavra-chave para um atributo do banco de dados. É importante salientar que muitos desses comandos e configurações são específicos do SQL Server e podem não funcionar para outros gerenciadores.

Qualquer *script* SQL pode ser salvo em um arquivo separado e executado quando necessário, inclusive durante a instalação de sistemas que necessitam se comunicar com um servidor de banco de dados. Para salvar qualquer *script* em um arquivo para uso futuro, clique com o botão direito do mouse em sua aba e escolha a opção de salvamento.

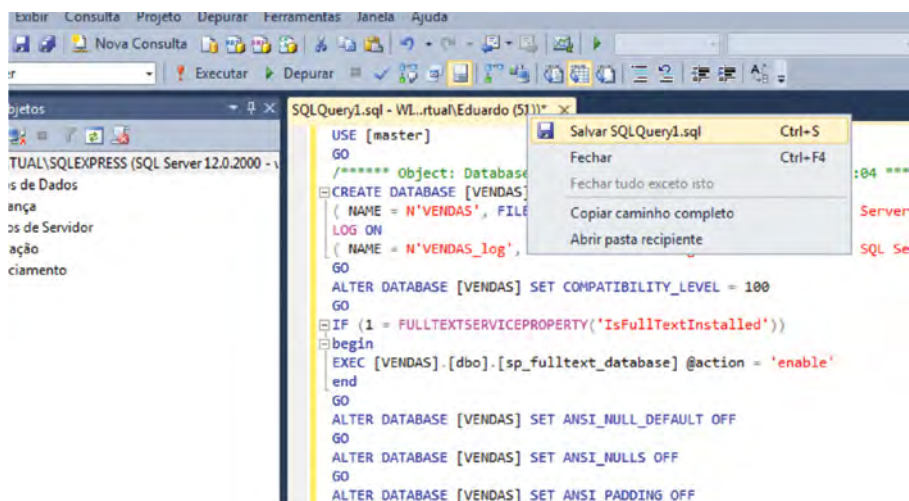


Figura 4.10 – Salvamento de um script em uma arquivo texto.

4.2.2 Script de criação de tabelas

Um banco de dados é formado por tabelas, as quais possuem entidades e fatos definidos pelo modelo de dados. Na criação de uma tabela de dados, é necessário definir os campos de atributos que são representados em suas colunas. Por exemplo, no momento de criação de uma tabela que armazenará nossos clientes no banco de dados de vendas, é necessário definir alguns de seus campos, como seu código, nome, endereço, telefone etc.

No exemplo de criação da tabela de clientes a seguir, foi utilizado o tipo de dado `nvarchar` para alguns campos de texto, o qual permite o armazenamento de caracteres da língua portuguesa, como acentos e cedilha.

```
CREATE TABLE Cliente
(
    ID bigint NOT NULL Primary Key,
    Nome nvarchar(200) NOT NULL,
    Telefonevarchar(50) NULL,
    Celularvarchar(50) NULL,
    Logradouronvarchar(200) NOT NULL,
    Numerovarchar(50) NOT NULL,
```

```

Complementovarchar(50) NULL,
Cidadenvarchar(50) NOT NULL,
UF varchar(2) NOT NULL,
)

```

Código 4.2 – Script SQL para a criação da tabela Cliente.

Atenção: quando for executar um código SQL relacionado a um banco de dados, lembre-se de escolhê-lo no menu de contexto, como indica a figura a seguir, onde o banco de dados "vendas" foi selecionado. Se não o fizer, o SQL Server poderá criar a sua tabela entre as de sistema - caso isso ocorra, apague a nova tabela e a crie de novo no contexto correto.

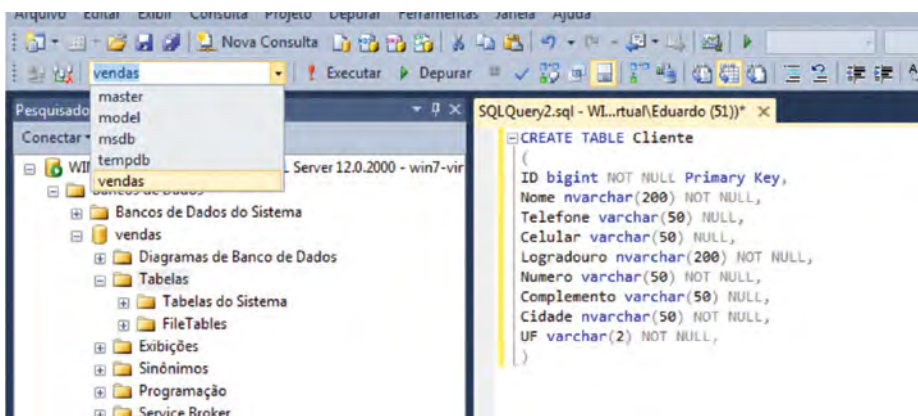


Figura 4.11 – Escolha do banco de dados de contexto para execução da consulta SQL.

Após executar o código SQL, atualize o banco de dados. A tabela Cliente estará então disponível e seus campos estarão visíveis no Pesquisador de Objetos.

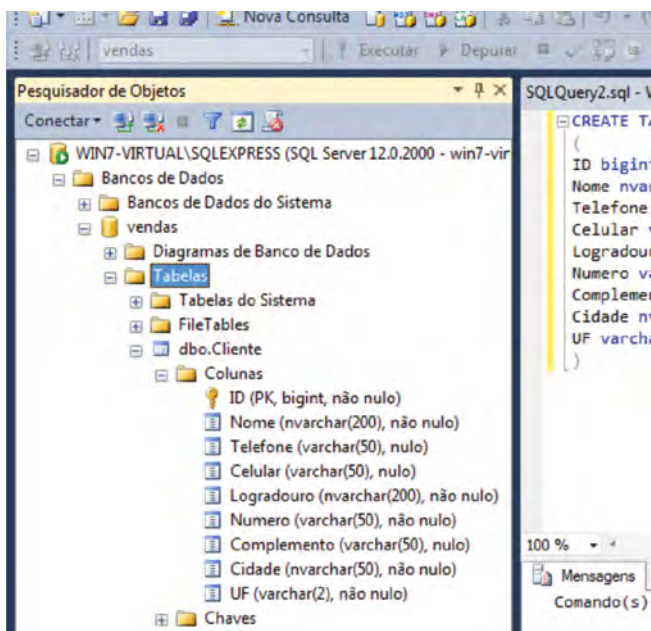


Figura 4.12 – Exibição da tabela Cliente e seus campos.

Também é possível criar uma tabela de forma gráfica no SQL Server Management Studio, bastando clicar com o botão direito do mouse no item "Tabelas" do banco de dados desejado e escolher a opção "Tabela...".

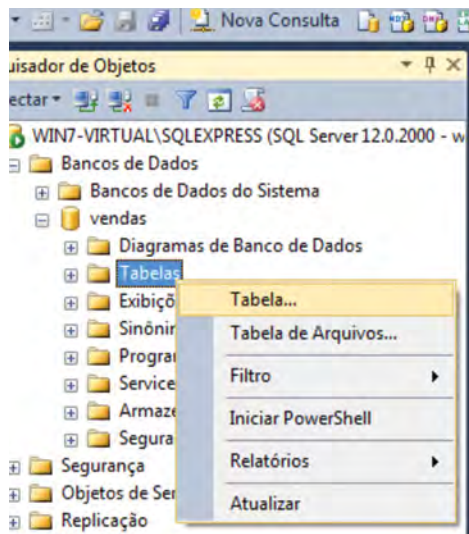


Figura 4.13 – Menu para criação gráfica de tabelas.

Uma nova tela se abrirá, a qual permite nomear cada campo, escolher seus tipos de dados e modificar outras configurações de campos, como a determinação de chaves primárias.

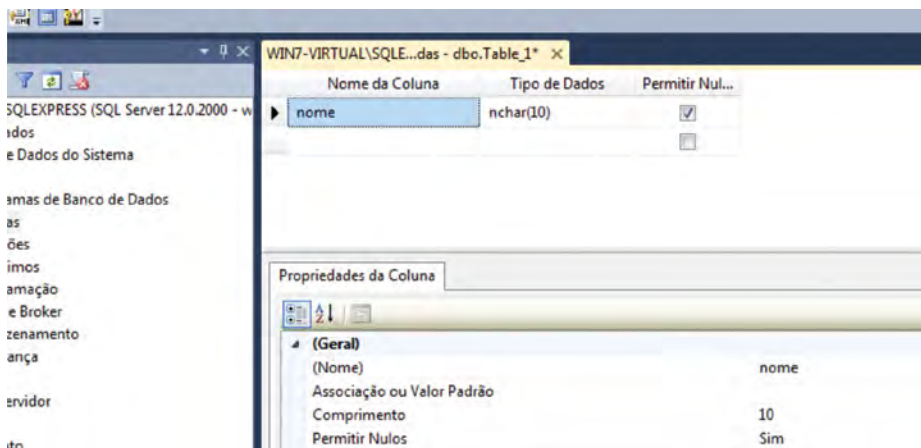


Figura 4.14 – Tela para edição gráfica de nova tabela.

Se preferir criar uma tabela dessa maneira, basta adicionar cada campo e seus tipos. Depois de definir todos eles, salve a tabela ou clique para fechar a aba. Um diálogo permitirá que escolha um nome para a tabela e a salve.

Para continuar com o exemplo do banco de dados de vendas, devemos criar mais duas tabelas: uma de Pedidos e outra de Itens de Pedidos. Perceba que essas são duas entidades que se relacionam, cada pedido para um cliente pode ter vários itens de pedido de produtos. Os códigos SQL são os seguintes.

```
CREATE TABLE Pedido
(
  ID bigint NOT NULL Primary Key,
  Numerobigint NOT NULL,
  IDClientebigint NOT NULL,
  Data datetime NOT NULL,
  PreçoPedido money NOT NULL,
)
```

Código 4.3 – Script SQL para a criação da tabela ItemPedido.


```

CREATE TABLE ItemPedido
(
  ID bigint NOT NULL Primary Key,
  IDPedidobigint NOT NULL,
  Numerosmallint NOT NULL,
  Descricaovarchar (200) NOT NULL,
  Quantidadeint NOT NULL,
  PrecoUnitario money NOT NULL,
  PrecoItemPedido money NOT NULL,
)

```

Código 4.4 – Script SQL para a criação da tabela Pedido.

Cada um desses trechos deve ser executado por vez. Após sua execução, teremos as tabelas necessárias, mas, logicamente, elas estão vazias de dados. Adicionaremos conteúdo a elas no próximo capítulo.

4.2.3 Views

Em uma tradução sem contexto para o português, a palavra "view" pode significar exibição, visão, ponto de vista, visualização, panorama, paisagem, compreensão ou ideia. Em bancos de dados, a palavra está ligada mais a uma forma de "enxergar" os dados, através de cruzamentos e filtros. A versão em português do SQL Server utiliza o termo "Exibição", mas como é comum o uso da palavra em inglês, faremos o mesmo aqui.

Uma View nada mais é do que uma Query, em que se faz filtragem e cruzamento de dados em uma ou mais tabelas. Essa Query é salva no conjunto de objetos do banco de dados, mas não faz parte de seu esquema. A ideia é que ela se torne uma espécie de tabela virtual que pode ser consultada a todo momento, como se fosse uma tabela real de dados, mas, na verdade, ela não existe como tabela e o gerenciador de banco de dados sempre atualiza seus resultados para refletir as atualizações nas tabelas a que essa View se refere.

Seguindo nosso exemplo de um banco de dados de controle de vendas, uma View "pedidos_view" poderia ser criada a partir de uma Query que exibe o nome dos clientes e as datas de seus pedidos das tabelas Cliente e Pedido. Com isso, se for necessário criar uma Query que filtre nome de clientes e datas, elas podem ser realizadas sobre a View criada (pedidos_view) ao invés de se criar um

código SQL que faça as duas coisas: cruzamento de dados entre tabelas e filtragem. Assim, é possível perceber que é criada uma camada de abstração que facilita a programação de buscas no banco de dados.

A figura a seguir mostra, à esquerda, o aplicativo se comunicando diretamente com as tabelas. Nesse caso, ele terá que fazer consultas mais complexas para obter os dados necessários. À direita está um esquema em que a *view* faz as consultas mais complexas utilizando JOINS, e o aplicativo apenas precisa fazer consultas simples para obter os dados.

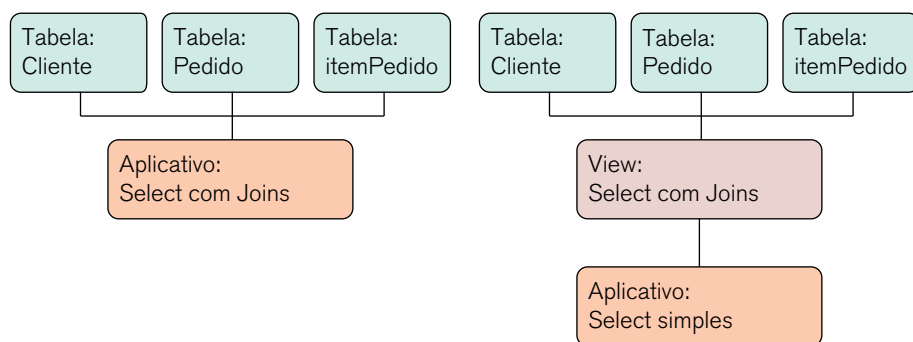


Figura 4.15 – Uma view funcionando como camada de abstração para simplificar as consultas do aplicativo.

Utilizar Views pode trazer diversas vantagens para eficiência, segurança e simplificação do software e seu código fonte. Algumas dessas vantagens são as seguintes:

- Uma View ocupa muito pouco espaço de armazenamento, não sendo necessário manter fisicamente sua tabela resultante;
- Com o SQL armazenado no gerenciador de banco de dados, ele pode fazer uma única otimização inicial que tornará sua execução mais rápida;
- É possível permitir a autorização de acesso apenas à View, restringindo o acesso direto aos dados nos quais ela se baseia;
- As Views permitem uma abstração das fontes de dados, tornando mais simples a programação da interface do software;
- Elas podem ser usadas para cálculos de agregados de dados, como somas e médias, de forma mais eficiente pelo SGDB.

4.2.4 Criação de Views

Assim como na criação de bancos de dados e tabelas, é possível criar *views* diretamente com código SQL ou através da interface gráfica do SQL Server.

```
CREATE VIEW Lista_Cliente AS
SELECT
    Cliente.ID AS Cliente_ID,
    Cliente.Nome AS Cliente_Nome,
    Cliente.Telefone AS Cliente_Telefone,
    Cliente.Celular AS Cliente_Celular,
    Cliente.Logradouro AS Cliente_Logradouro,
    Cliente.Numero AS Cliente_Numero,
    Cliente.Complemento AS Cliente_Complemento,
    Cliente.Cidade AS Cliente_Cidade,
    Cliente.UF AS Cliente_UF,
    Pedido.ID AS Pedido_ID,
    Pedido.Numero AS Pedido_Numero,
    Pedido.PrecoPedido AS Pedido_PrecoPedido,
    ItemPedido.ID AS ItemPedido_ID,
    ItemPedido.Numero AS ItemPedido_Numero,
    ItemPedido.Descricao AS ItemPedido_Descricao,
    ItemPedido.Quantidade AS ItemPedido_Quantidade,
    ItemPedido.PrecoUnitario AS ItemPedido_PrecoUnitario,
    ItemPedido.PrecoItemPedido AS ItemPedido_PrecoItemPedido
FROM Cliente
    INNER JOIN
        Pedido ON Pedido.IDCliente = Cliente.ID
    INNER JOIN
        ItemPedido ON ItemPedido.IDPedido = Pedido.ID
```

Código 4.5 – Script SQL para a criação da viewLista_Cliente.

Essa *view* criará uma tabela virtual que possui todos os campos das tabelas Cliente, Pedido e ItemPedido. Veja que o *script* utiliza "INNER JOIN" para unificar as linhas das tabelas que estão relacionadas pelas identificações do

clientes, dos pedidos e dos itens. A *view* resultante terá várias linhas para cada cliente, se houver mais de um pedido por cliente e mais de um item por pedido. Também é importante notar que são dados apelidos aos campos para ajudar a identificá-los, além de que as linhas são ordenadas por ordem alfabética de nomes de clientes.

4.2.5 Biblioteca ADO.NET

ADO.NET é um conjunto de classes e métodos que faz parte do Framework .NET que permite o acesso e manipulação de bases de dados a partir das linguagens de programação suportadas. Através dessa biblioteca o programador pode criar funções que conectam seu aplicativo a bases de dados específicas, enviam comandos SQL para esse banco de dados e recebem os conjuntos de dados resultantes desses comandos SQL.

Essa biblioteca foi criada no ano 2000 e sucedeu sua versão anterior, que se chamava apenas ADO (ActiveX Data Objects). No entanto, apesar dos nomes parecidos, elas são muito diferentes e nem podem ser consideradas como diferentes versões de uma mesma biblioteca. Através das instruções ADO.NET é possível tanto se conectar a um banco de dados que não possua um servidor, como um arquivo do MS Access ou arquivos de texto, de planilhas ou XML, como a um servidor de banco de dados, que pode ser o SQL Server diretamente ou outros (MySQL, PostgreSQL etc.) através de provedores de dados (*data providers*), que são bibliotecas que acrescentam conectividade a esses produtos.

Quando se utiliza ADO.NET para obter conjuntos de dados de um banco, é possível escolher duas formas de se manipular esses dados: uma forma conectada e outra desconectada. Na forma conectada, esses dados retornam no formato de datasets, que literalmente significa "conjuntos de dados". Esses datasets ficam armazenado na memória e por isso podem ser manipulados com mais rapidez, sem a necessidade de acessar o banco de dados com frequência, além de aliviar a carga no servidor e na rede.

Com os datasets na memória, é possível manipulá-los através de *views* (filtros e cruzamentos) de forma mais eficiente, sendo necessário consultar o servidor novamente apenas para atualização desses dados. Assim, é possível desenvolver aplicativos em camadas:

- **Camada de apresentação:** é aquela exibida para o usuário, que pode ser formatada com paginação e através de organização gráfica;
- **Camada de negócios:** quais usuários podem ver ou alterar quais dados, ou fazer filtros por região geográfica ou datas de pedidos de vendas, por exemplo;
- **Camada de dados:** na qual é feita a manipulação através de visões para obtenção dos dados do banco.

4.2.6 Componentes ADO.NET

Os principais componentes ADO.NET são as suas classes que são instanciadas em objetos para conexão, armazenamento e manipulação de dados. Há quatro tipos principais de componentes para interagir com as bases de dados:

- **Objeto de Conexão** - utilizado para estabelecer conexão com a fonte de dados;
- **Objeto de Comando** - serve para enviar comandos SQL para o provedor de dados conectado à fonte de dados;
- **Objeto de Resultado** - lê e armazena os dados que serão manipulados. O *DataReader* é um meio conectado, que obtém os dados diretamente do banco de dados, enquanto o *DataSet* é um meio desconectado, que armazena os dados a partir de um *DataAdapter*;
- **Adaptador de dados** - o *DataAdapter* recebe o resultado de um comando SQL e é usado para passar um conjunto de dados resultantes para um *DataSet*.

Como foi dito anteriormente, o ADO.NET pode ser utilizado para se conectar a diversos tipos de fontes de dados. O uso com outros tipos de servidores de bancos de dados (PostgreSQL, MySQL, Oracle etc.) varia de caso a caso, usando bibliotecas específicas. Para uso com o SQL Server ou outras fontes como MS Access e arquivos texto ou XML, isso pode ser feito diretamente, mas as classes são diferentes para cada um dos dois tipos de fontes de dados.

Para conexão a arquivos locais, como Ms Access, planilhas ou arquivos textos, é necessário se conectar à camada de abstração que o Windows oferece para transformá-las e fontes de dados, a OLE DB (*Object Linking and Embedding Database*), por isso as classes que correspondem a cada um dos componentes descritos anteriormente são:

- OleDbConnection: para conexão a fonte de dados;
- OleDbCommand: para emissão de comandos à fonte de dados;
- OleDbDataReader ou OleDbDataSet: para armazenamento dos dados para manipulação;
- OleDbDataAdapter: para passar dados ao DataSet.

Por outro lado, a conexão ao SQL Server é direta e não precisa usar essa camada intermediária. Nesse caso, as classes que representam os componentes do ADO.NET para conexão com o SQL Server são:

- SqlConnection
- SqlCommand
- SqlDataReader ou SqlDataSet
- SqlDataAdapter

O relacionamento entre todos esses elementos pode parecer confuso à primeira vista, sendo necessário observar cuidadosamente o exemplo a seguir para compreender a relação entre os componentes do ADO.NET. Este exemplo de código em VB.NET se conecta ao banco de dados de vendas, usa um comando SELECT do SQL para obter o conteúdo da tabela Cliente e, através do DataAdapter conta quantas linhas há nessa tabela. Como ainda não inserimos dados, o resultado deve ser 0 (zero) e será impresso na tela do console.

```

Conexao.vb
1 Imports System.Data.SqlClient
2
3 Module Module1
4
5     Sub Main()
6         conectar_db() 'Executa a sub-rotina criada abaixo
7         Console.ReadKey()
8     End Sub
9
10    Sub conectar_db()
11        Dim conexao As New SqlConnection 'Cria SqlConnection
12        Dim strConexao As String = "Data Source=WIN7-VIRTUAL\SQLEXPRESS;" & vb-
CrLf & _
13            "Initial Catalog=vendas;Integrated Security=SSPI;" &
vbCrLf & _
14            "Connection Timeout=10;" 'Dados de autenticação no
SQL Server

```

```

15
16     Dim strComando As String = "SELECT * FROM Cliente"
17     Dim comando As SqlCommand      'Cria a variável para o SqlCommand
18     Dim da As SqlDataAdapter        'Cria a variável para o SqlDataAdapter
19     Dim dt As New DataTable         'Cria o DataTable
20     Try
21         conexao.ConnectionString = strConexao 'Faz a conexão efetiva
22         comando = New SqlCommand(strComando, conexao) 'Instancia um objeto
de comando
23         comando.CommandTimeout = 3000        'Configura o envio do comando
24
25         da = New SqlDataAdapter(comando) 'Instancia um objeto de adaptador
de dados
26         da.Fill(dt)                        'Recebe os dados e preenche o
adaptador de dados
27         Console.WriteLine(dt.Rows.Count.ToString()) 'Imprime número de li-
nhas da tabela Cliente
28
29     Catch ex As Exception
30         Console.WriteLine(ex.Message, "Erro de conexão")
31     Finally
32         If conexao.State = ConnectionState.Open Then
33             conexao.Close()
34         End If
35     End Try
36 End Sub
37
38 End Module

```

Código 4.6 – Código VB.NET para conexão com o banco de dados “vendas” e contagem de linhas da tabela “Cliente”.

Perceba que o código que opera no banco de dados está em um bloco Try, pois, se ele não conseguir se conectar do BD, emitirá um aviso de erro. Como isso é apenas um exemplo, todo o código de conexão e execução está em uma mesma sub-rotina, mas em um software de uso real, ele poderia estar dividido em funções, para que cada operação no banco de dados não precisasse repetir todos os procedimentos novamente.

No próximo capítulo abordaremos como fazer uma conexão com o banco de dados utilizando o assistente gráfico do Visual Studio.



REFLEXÃO

Cada linguagem de programação possui uma forma de interação com bancos de dados através de suas bibliotecas. Geralmente, o uso dessas bibliotecas envolve uma camada de abstração que utiliza conectores de dados, os quais são especializados para diferentes bancos de dados. É importante para um programador que necessite utilizar armazenamento permanente de dados se familiarizar com um ou mais SGBDs e seu dialeto SQL, com as bibliotecas de conexão entre a linguagem e esses SGBDs e como otimizar consultas.

Os exemplos apresentados nesse capítulo são bastantes simples e não procuram detalhar essa otimização de consultas a banco de dados, o que é oferecido por uma disciplina de modelagem de dados, mas apenas apresentar os componentes envolvidos em no desenvolvimento de um sistema com ferramentas integradas.



ATIVIDADES

01. Qual é a linguagem que se usa para manipular bancos de dados relacionais e quais são algumas de suas características?
 02. Explique, com suas palavras, o que é um sistema gerenciador de banco de dados que utiliza uma arquitetura do tipo cliente-servidor.
 03. Programe dois pequenos scripts SQL: um para criar um banco de dados "concessionaria" e outro para criar nesse banco de dados uma tabela "automovel", com os campos, "ID", "marca", "modelo", "ano", "cor" e "chassi".
 04. O que é uma *view* e quais são as vantagens de seu uso?
-



LEITURA

A W3Schools é reconhecido por ter ótimos guias para aprender linguagens de programação, mas eles são em inglês. Se você precisa aprender ou lembrar SQL e não tem problemas com a língua inglesa, este é o guia da instituição: <<http://www.w3schools.com/sql/>>



REFERÊNCIAS BIBLIOGRÁFICAS

MSDN, Microsoft Developer Network. **Tipos de dados (Transact-SQL)**. Disponível em: <<https://msdn.microsoft.com/pt-br/library/ms187752.aspx>>. Acesso: 5 fev. 2015.

ELMASRI, Ramez; NAVATH, Shamkant B.. **Sistemas de banco de dados**. São Paulo: Pearson Addison Wesley, 2005.

SILBERSCHATZ, Abraham; KORTH, Henry F.; Sudarshan, S. **Sistema de Banco de Dados**. Elsevier: Rio de Janeiro, 2006.

5

Interface Gráfica para o Usuário e os Elementos de Desenvolvimento de Software

Neste último capítulo, serão apresentados os fundamentos da programação de interfaces gráficas para o usuário. O Visual Basic, em suas versões iniciais, tinha o propósito de facilitar a construção de programas visuais através da união da linguagem BASIC com ferramentas de desenvolvimento rápido de aplicativos. Hoje, essas ferramentas são utilizadas em muitas linguagens de programação, mas o Visual Basic continuou a ser aperfeiçoado para se equiparar a linguagens modernas de programação. Como resultado, ele continua sendo um meio muito interessante de se desenvolver programas com interfaces gráficas.

A primeira parte do capítulo abordará as características das interfaces gráficas e alguns elementos principais para desenvolvê-las. A segunda parte fará um passo a passo, mostrando como iniciar o desenvolvimento de um programa usando interface gráfica e conexão a um banco de dados. O exemplo descrito não é um programa completo, mas é um início para sua exploração do Framework .NET, do Visual Basic, do Visual Studio e do SQL Server.



OBJETIVOS

Após a leitura desse capítulo e desenvolvimento de suas atividades, você saberá:

- O que é uma interface gráfica e como ela é desenvolvida
 - Quais são os principais elementos de uma interface que encontramos no dia a dia
 - Como criar um programa com interface gráfica e conexão com banco de dados.
-

5.1 Interface Gráfica para Usuário

Interface Homem-Computador (IHC) é uma área de estudo e pesquisa que se desenvolveu com a popularização dos computadores. Ela se preocupa com a idealização, desenho e testes de formas de as pessoas passarem e receberem dados dos computadores. Originalmente, a interação com computadores incluíam a conexão de fios e sinais emitidos por lâmpadas, evoluindo para teclados e monitores que exibiam apenas textos. Experimentos e pesquisas trouxeram cada vez mais opções, incluindo interação por mouse, monitores que exibem desenhos, voz, telas sensíveis ao toque, joysticks, sensores de movimento e muitos outros.

Interfaces Gráficas para o Usuário (Graphical User Interface - GUI) são aquelas que possuem formas de interação que vão além do texto puro ou de sinais muito simples. Por gráficos devem ser entendidos desenhos e imagens que muitas vezes representam elementos abstratos ou metáforas de objetos do mundo físico, os quais facilitam o entendimento e uso por pessoas que interagem com computadores. Uma interface adequada deve satisfazer a necessidade do público usuário do software, seja ele leigo ou tecnicamente preparado.

5.1.1 Janelas, Forms e componentes

O paradigma de interação com o computador através de mouse e ícones foi criado na década 1970 pela empresa Xerox, que possuía um centro de pesquisa chamado Xerox PARC (Palo Alto Research Center Incorporated). Antes da criação dessa forma de interação, a interação com os computadores era feita predominantemente através de teclado e texto puro.

Devido aos elementos básicos que foram criados para esse novo tipo de interação, ele era conhecido como WIMP, do inglês "Window, Icon, Menu and Pointing device", que significa "Janela, Ícone, Menu e Dispositivo de apontamento", sendo que o último elemento, na época, deu origem ao mouse. Essa concepção foi tão relevante que até hoje ela é predominante na interação com os computadores. Mais tarde, tanto a Apple quanto a Microsoft passaram a utilizá-la e a popularizaram.

Mesmo sendo um paradigma relativamente antigo, ele ainda continua sendo aperfeiçoado, com várias empresas e desenvolvedores individuais modificando pequenos detalhes que colaboram com a interação entre usuários e máquinas. Tipos de janelas e controle, novas formas de entrada de dados e

apontamento na tela e menus mais fáceis de navegar. Alguns experimentos progridem e outros acabam sendo abandonados. No entanto, os mais comuns acabam sendo incorporados pelas ferramentas de desenvolvimento de software.

5.1.1.1 Tipos de janelas

Um exemplo de variação de elementos de interação são os tipos de janelas. Uma janela comum permite que o usuário navegue entre ela e outras. Por outro lado, de vez em quando, o programador necessita utilizar um tipo de janela que impede que o usuário continue usando o programa se ele não responder a uma solicitação de informação - esse tipo de janela é chamada modal. Recentemente, principalmente devido ao desenvolvimento de sistemas móveis, a Microsoft tentou incentivar o desenvolvimento de programas que ocupam toda a tela a um dado momento, o que levou à interface do Windows 8. Entretanto, a empresa percebeu que isso não era prático em todos os casos, e passou a colocar menos ênfase nesse tipo de software.

Do ponto de vista técnico, uma janela ou qualquer outro tipo de elemento de uma janela é um objeto derivado de uma classe. Essa classe é um modelo daquele objeto, com suas respectivas propriedades (MSDN, 2015a). Por exemplo, uma janela criada para um software é uma instância da classe janela, a qual pode ter uma propriedade alterada para que se torne modal ou comum. Na verdade, o nome de uma classe que dá origem a uma janela é um "formulário".

Um formulário pode criar qualquer tipo de janela: com apenas uma animação de introdução, com campos para preenchimento do usuário, apenas com um menu para navegação, etc. São os elementos a serem introduzidos no formulário que o tornam de uma maneira ou outra. Esses elementos normalmente são chamados "controles": botões, menus, caixas de textos e muitos outros. Eles também são objetos derivados de uma classe, os quais possuem propriedades e métodos. Como ilustração, vamos mostrar alguns deles.

5.1.1.2 Criando um Aplicativo com Janelas com Visual Basic

A primeira coisa a se fazer para desenvolver um aplicativo com janelas no Visual Studio é criar um novo projeto. Clique no menu ARQUIVO e escolha a opção Novo > Projeto..., o que fará abrir uma janela com várias opções de projetos. Ela poderá apresentar diferentes opções dependendo da versão de seu Visual

Studio; a versão Community terá mais tipos de projetos do que a versão Express. Aqui será abordado o desenvolvimento de um projeto utilizando o conjunto de ferramentas Windows Forms.

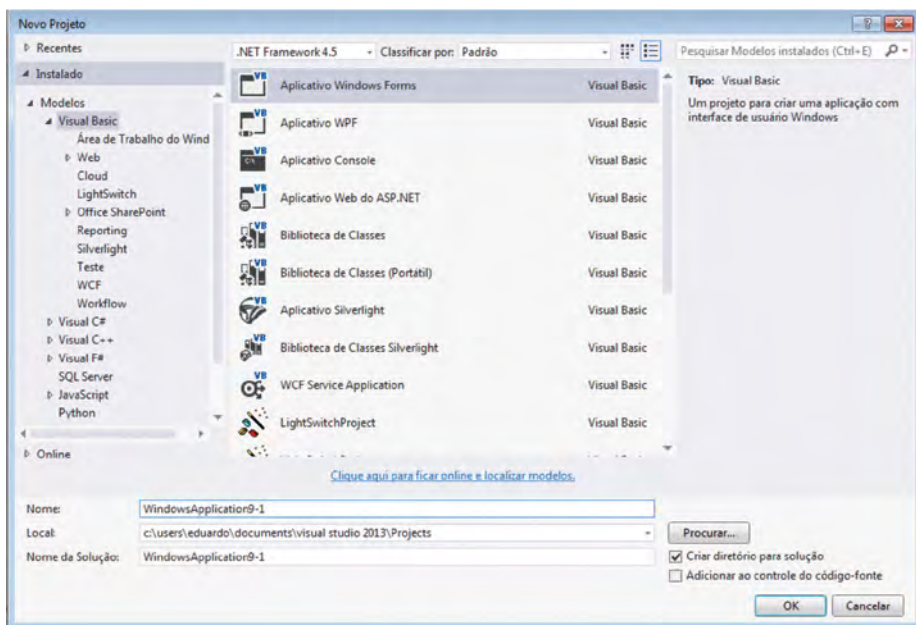


Figura 5.1 – Tela de opções para criação de novo projeto no Visual Studio.

Primeiro, selecione o modelo Visual Basic entre os filtros na parte esquerda da janela. Na parte central, escolha a opção Aplicativo Windows Forms. Abaixo nessa mesma janela, de um nome que desejar para o projeto e clique OK.

Além de Windows Forms, outro tipo de API (Application Programming Interface) disponível para criar programas visuais é o de Aplicativos WPF (Windows Presentation Foundation). Esse é um conjunto de ferramentas mais novo para a criação de janelas no Windows, o qual as desenha a partir de arquivos do tipo XML. No entanto, nos últimos anos, a Microsoft já deixou de dar foco para o desenvolvimento do WPF para se voltar à ferramenta de desenvolvimento de uma plataforma para aplicativos voltados ao Windows 8, denominados Windows Store Apps, que funcionam com um novo conjunto de APIs chamado Windows Runtime. Em resumo, há três conjuntos de APIs amplamente utilizados para o Windows (Windows Forms, WPF e Windows Runtime), cada uma delas com suas características. Nos retornaremos aqui a usar o Windows Forms, o qual ainda é muito utilizado.

Após a criação do projeto, será criada também, automaticamente, a primeira janela nomeada "Form1". Um programa visual tem pelo menos uma janela, sendo que, nesse caso, ela será a sua janela principal. Em inglês, o termo normal de uso é "form", que pode e uma abreviação de "formulário". Até o fim desse texto, usaremos os termos "form" ou "janela".

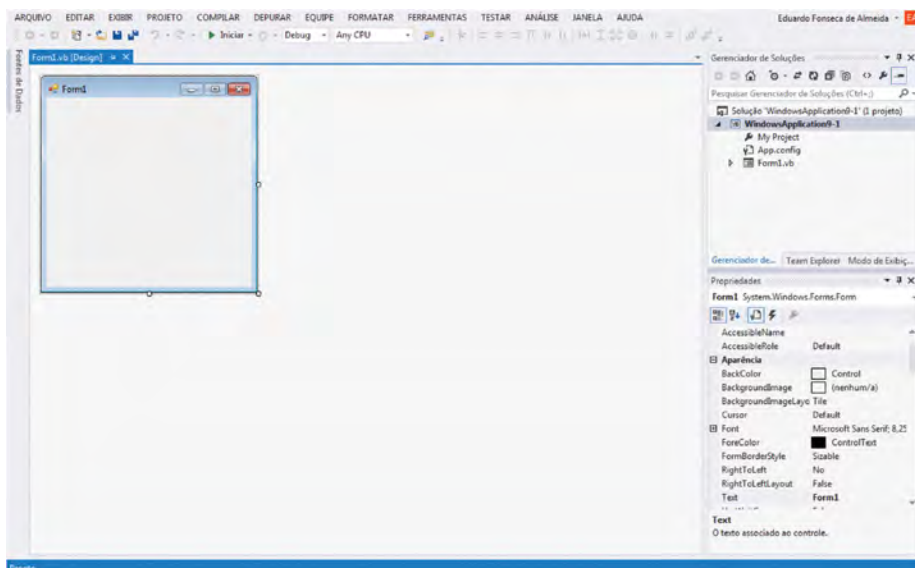


Figura 5.2 – Tela após criação de novo projeto, com um form já criado.

Vamos modificar algumas propriedades dessa janela principal. Para isso, veja a caixa de propriedades do objeto Form1 que se localiza abaixo e à direita. Lembre-se que essa caixa sempre mostra as propriedades dos objetos que estão selecionados na área de desenho. Nesse local é possível modificar propriedades de formatação e comportamento. Procure as propriedades do quadro a seguir e modifique o texto que irá no alto da janela e o seu tamanho (pode colocar dimensões que sejam adequadas a tela de seu monitor).

Propriedade	Valor
Text	Aplicação Gráfica 1
Size	600;480

Após o redimensionamento, o formulário terá um tamanho mais adequado.

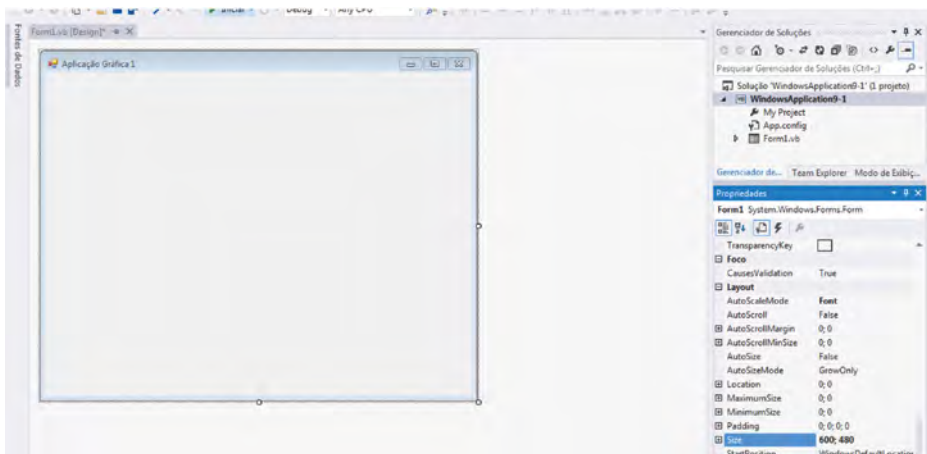


Figura 5.3 – Janela do Form1 redimensionada.

Após essas modificações, elas serão refletidas imediatamente no design da janela. Se você salvar o projeto e clicar no botão "Iniciar" da barra de ferramentas, o Visual Studio irá compilar seu programa e exibir uma janela vazia.

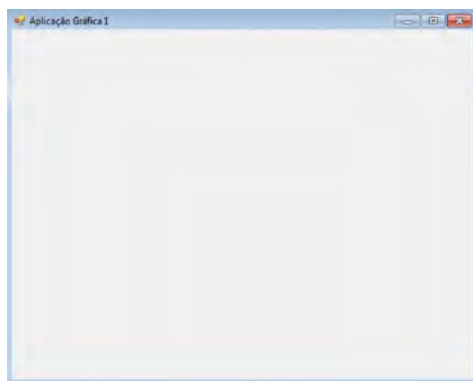


Figura 5.4 – A janela do Form1 sendo executada, sem nenhum controle adicional.

Isso já é seu programa funcionando, mas ele ainda não faz nada de especial, por isso vamos continuar adicionando elementos.

5.1.1.3 Label

A classe Label é um rótulo que pode ser utilizado para exibir textos indicando o significado de cada campo em um formulário ou colocar legendas na janela.

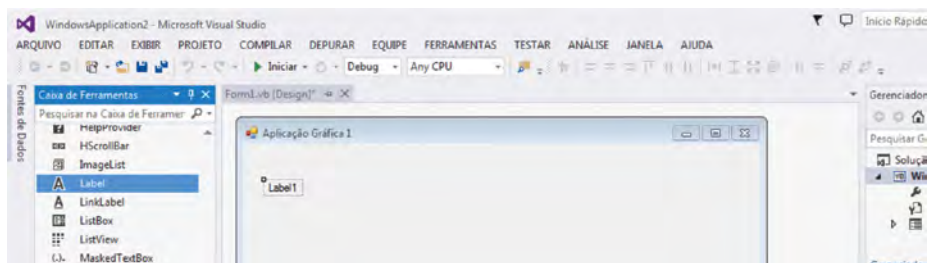


Figura 5.5 – Um controle Label, arrastado da Caixa de Ferramentas para a janela.

Para inserir um Label, procure esse item na Caixa de Ferramentas e o arraste até o local desejado da janela de formulário. Para alterar o texto do rótulo, você pode se sentir tentado a clicar duas vezes nele, mas não faça isso, pois o resultado é que o Visual Studio abrirá o arquivo de código do formulário e criará um método para o Label. Se isso ocorrer incidentalmente ocorrer, ignore o código por enquanto e volte para a aba de Design do formulário.

Para alterar o texto do Label, vá até a caixa de Propriedades e altere a propriedade Text, como na imagem a seguir, inserindo "Nome". Você também pode alterar a posição na caixa de Propriedades, através da propriedade Location, ou arrastando o controle com o mouse.

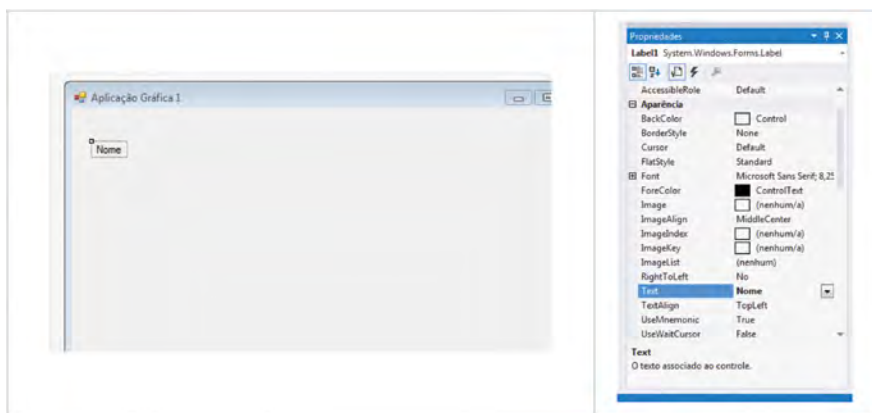


Figura 5.6 – Modificação da propriedade Text do Label.

5.1.1.4 TextBox

O TextBox é um campo em que o usuário pode digitar texto como entrada para o formulário. Por padrão ele tem apenas uma linha, mas a sua propriedade pode ser alterada para suportar múltiplas linhas de texto.

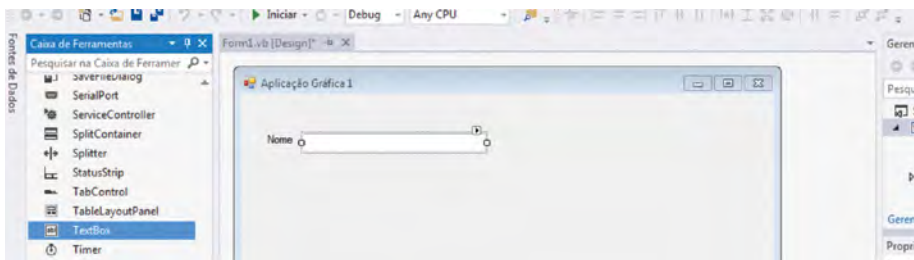


Figura 5.7 – Colocação de um campo de texto na janela.

Para inserir um TextBox, faça como no caso do Label e arraste esse item da Caixa de Ferramentas até o local desejado. Esse é o procedimento padrão para todos os controles. Seu tamanho e posição também podem ser alterados na caixa de Propriedades (Size e Location) ou arrastando com o mouse.

Também por padrão a caixa de texto tem a propriedade Text vazia, mas ela pode ser alterada para conter um texto inicial que o usuário pode substituir. Em nosso exemplo, ela deve ser deixada vazia.

5.1.1.5 Button

Button é um controle que implementa os botões, os quais podem ser os normalmente utilizados (OK, Cancelar etc.) assim como qualquer outro necessário para o software em desenvolvimento.

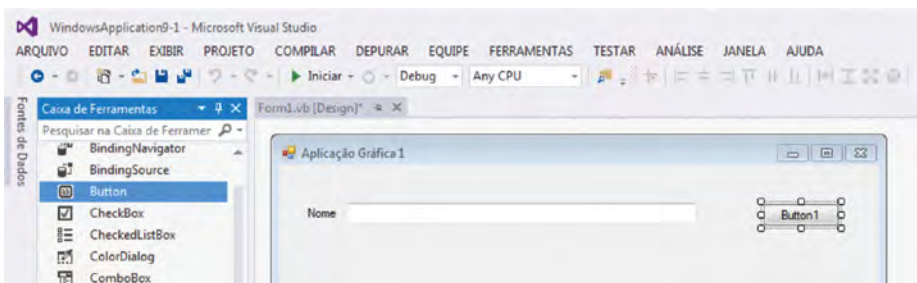


Figura 5.8 – Colocação de um botão na janela.

O texto a ser exibido pelo botão também é alterado pela propriedade Text, a qual deve ser modificada, nesse exemplo, para "OK".

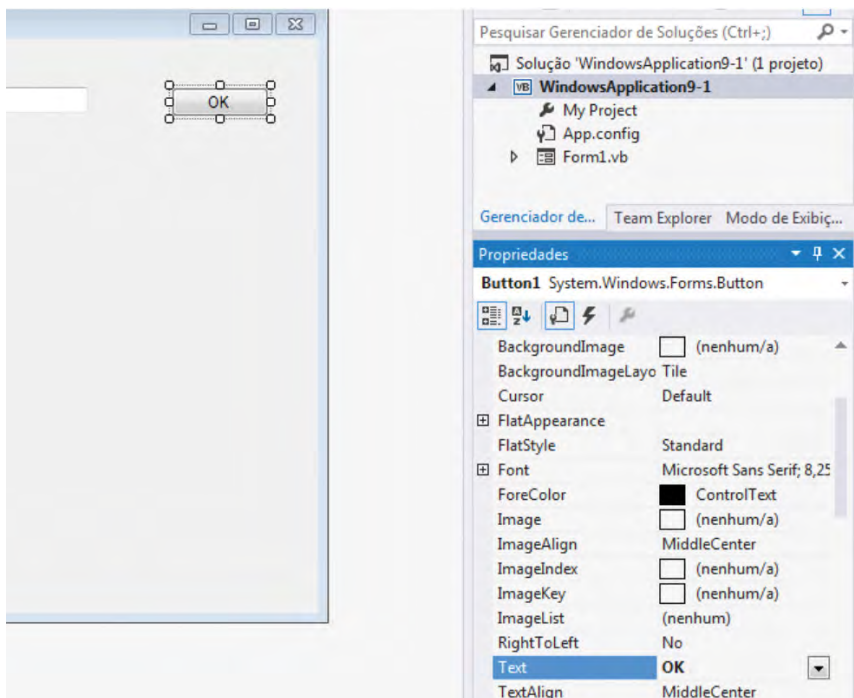


Figura 5.9 – Alteração da propriedade Text do botão.

A posição e tamanho do botão também podem ser alterados. Perceba que, movendo o botão com o mouse, o Visual Studio oferece linhas azuis e vermelhas para alinhar com os outros elementos do formulário - essas linhas são auxiliares para melhorar o *layout* do formulário, e os controles tendem a "grudar" nessas posições alinhadas, ajudando no alinhamento.

5.1.1.6 GroupBox

O controle GroupBox serve para agrupar outros controles em um conjunto. Em alguns casos ele tem apenas a função de melhorar o visual do formulário, mas em outros ele é necessário, como no caso dos RadioButtons, que serão explicados a seguir. Para inserir uma caixa de agrupamento, arraste-a para o formulário como qualquer outro controle. No exemplo que estamos construindo, a sua propriedade Text deve ser alterada para "Sexo", pois dentro desse grupo serão dadas opções de resposta ao usuário.

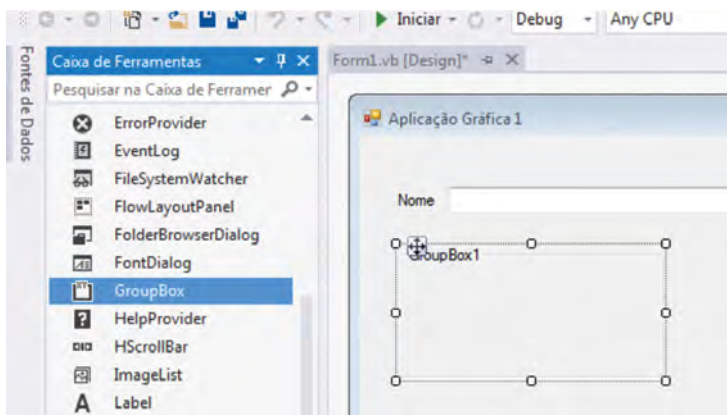


Figura 5.10 – Colocação de uma caixa de grupo na janela.

Seu tamanho pode ser alterado também com o mouse ou na caixa de Propriedades. O primeiro caso pode ser melhor para organizar os itens visualmente, mas o segundo método é interessante para se obter mais precisão: com ele é possível saber exatamente quantos pixels tem o tamanho do controle e se ele segue um padrão de desenho.

5.1.1.7 RadioButton

O **RadioButton**, ou botão de rádio, é um controle que se utiliza para dar ao usuário um conjunto de opções mutuamente exclusivas, isto é, dentro de grupo ele pode escolher apenas uma opção. Para que um conjunto seja considerado mutuamente exclusivo, é necessário colocá-los em um **GroupBox**. Se forem utilizados mais **RadioButtons** para outras partes do formulário, eles devem ser colocados em seus próprio grupos.

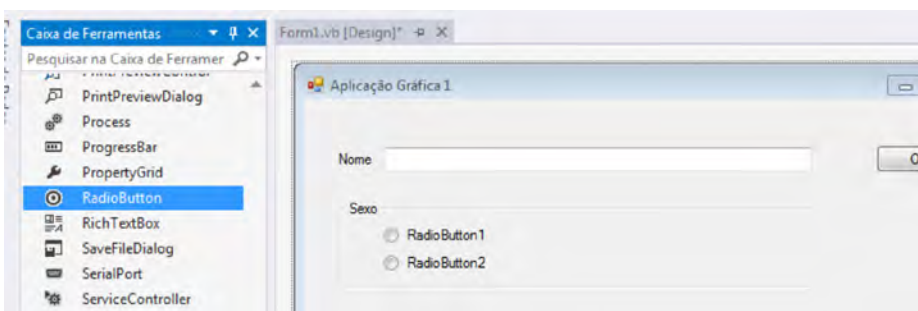


Figura 5.11 – Colocação de dois RadioButtons dentro da caixa de grupo.

Para alterar o texto de cada uma das opções, é necessário selecionar uma por vez e alterar suas propriedades Text na caixa de Propriedades. Altere cada uma para "Masculino" e "Feminino".



Figura 5.12 – RadioButtons com as suas propriedades Text alteradas.

5.1.1.8 CheckBox

Um CheckBox, ou caixa de seleção, se diferencia do RadioButton porque, em um grupo delas, o usuário pode escolher várias opções ao mesmo tempo. Isto é, as opções não são mutuamente exclusivas. Como os CheckBoxes são independentes, não é necessário colocá-los em um GroupBox, mas fazê-lo favorece a organização do formulário.

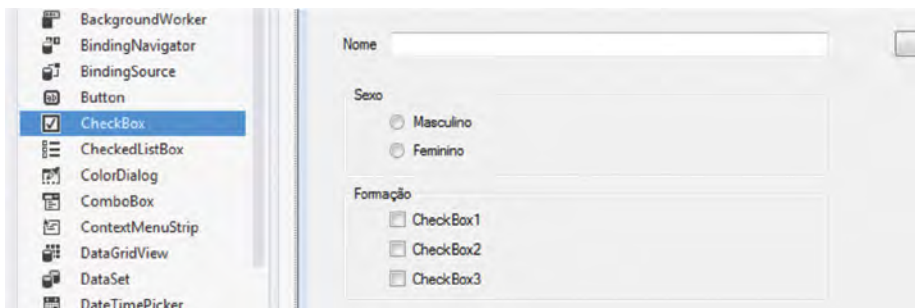


Figura 5.13 – Colocação de três caixas de seleção em uma caixa de grupo.

No nosso exemplo, cada caixa de seleção deve ter um nível de formação acadêmica: "Graduação", "Mestrado" e "Doutorado". Logicamente, se uma pessoa tem um doutorado, com certeza ela terá graduação e mestrado completos, mas nosso exemplo serve apenas para a compreensão de uso da ferramenta.



Figura 5.14 – Caixas de seleção com as propriedades Text alteradas.

5.1.1.9 ListBox

Um ListBox, uma caixa de lista, oferece uma lista de opções que o usuário pode escolher. Ela pode ter o mesmo comportamento de um grupo de RadioButton ou um grupo de CheckBox, isto é, pode oferecer várias opções mutuamente exclusivas ou mutuamente inclusivas.

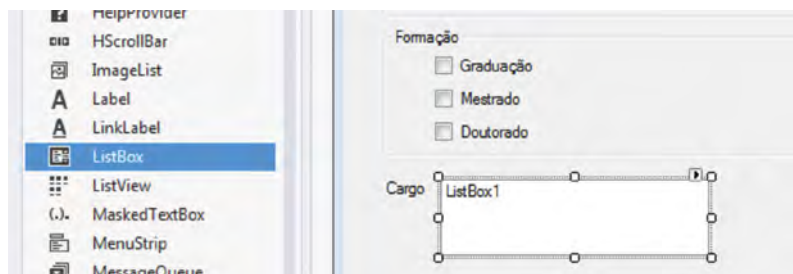


Figura 5.15 – Colocação de um ListBox.

Para inserir uma lista de opções, clique na pequena seta sobre a caixa do controle e então em "Editar itens...", como na próxima figura.

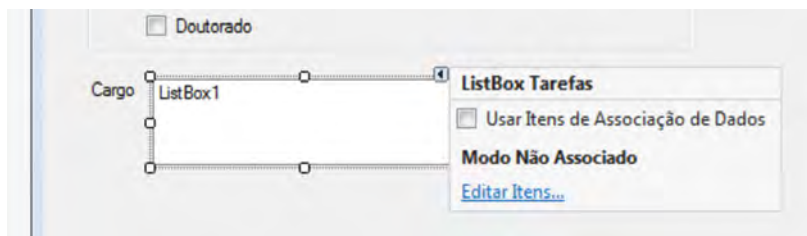


Figura 5.16 – Acesso à edição do listBox.

Uma janela se abre e nela os itens podem ser livremente digitados, cada um em uma linha.

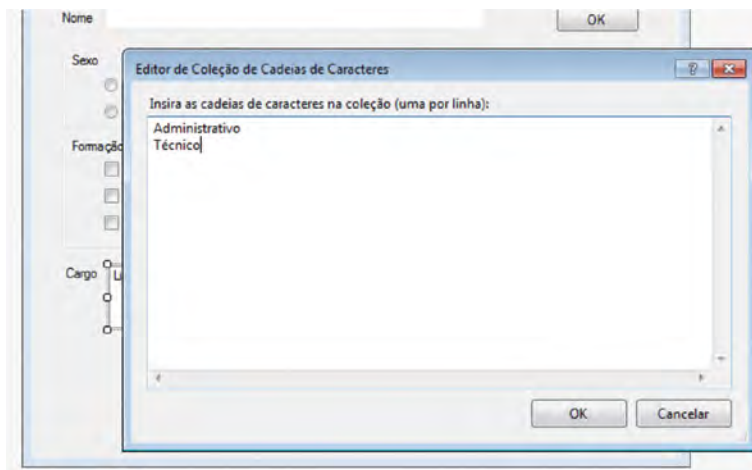


Figura 5.17 – Janela para edição do conteúdo do ListBox.

O resultado final se parece com uma caixa de texto, mas cada item em uma linha representa uma opção a ser selecionada.

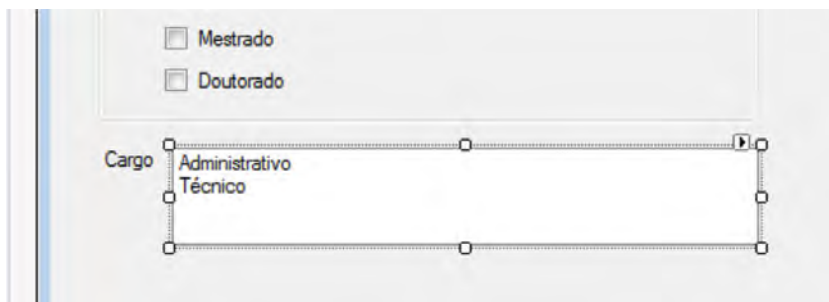


Figura 5.18 – ListBox com conteúdo alterado.

O comportamento padrão é de opções mutuamente exclusivas, sendo possível escolher apenas uma opção. Mas isso pode ser alterado através da propriedade `SelectionMode` (lembre-se de selecionar o controle para suas propriedades aparecerem).

5.1.1.10 ComboBox

O ComboBox também é uma caixa de opções mutuamente exclusivas, mas ela ocupa menos espaço que outros controles do tipo, pois as opções apenas são exibidas quando o usuário clica em uma caixa expansível.

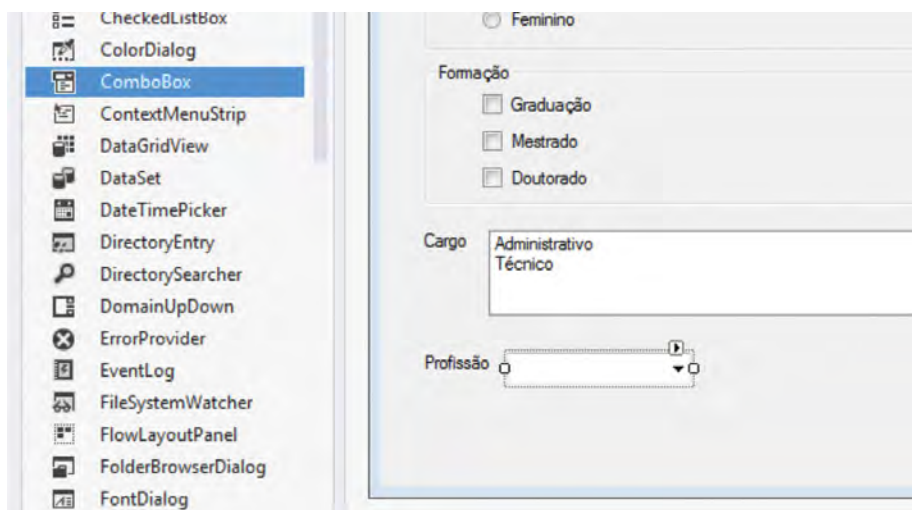


Figura 5.19 – Colocação de um ComboBox.

O processo para adicionar conteúdo ao ComboBox é semelhante ao do ListBox: clique nas opções do controle e então em "Editar itens...". Isso abrirá uma janela de edição.

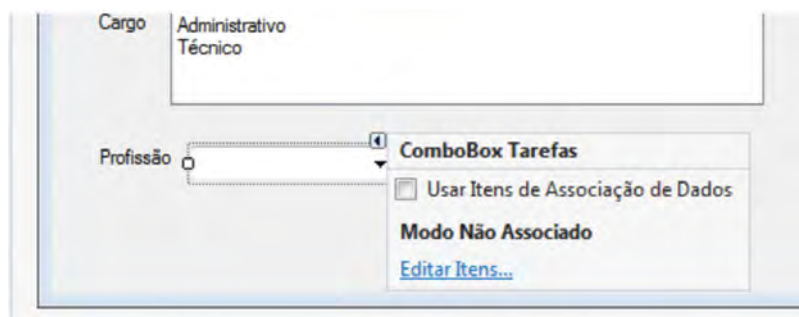


Figura 5.20 – Acesso à edição do ComboBox.

Em nosso exemplo, esse controle será utilizado para que o usuário escolha sua profissão entre três opções.

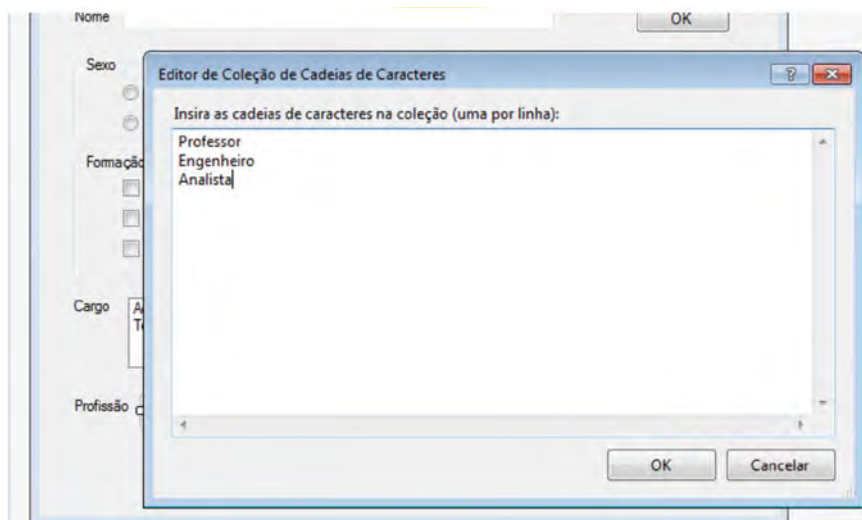


Figura 5.21 – Edição do conteúdo do ComboBox.

Apenas é possível checar como o ComboBox funciona se o programa for executado. Clique em "Iniciar" na barra superior do Visual Studio para experimentar o formulário.

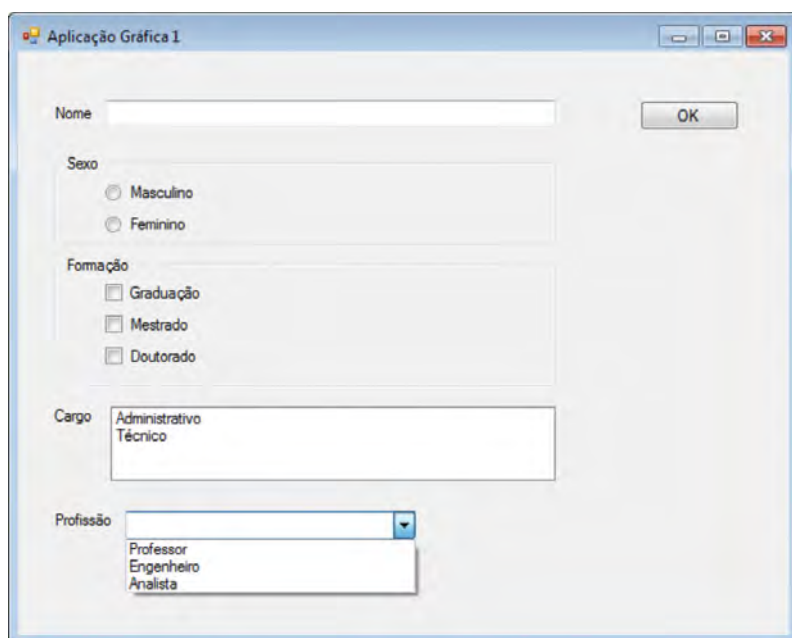


Figura 5.22 – O programa sendo executado, com o ComboBox funcionando.

5.1.1.11 MenuStrip

O menu MenuStrip é uma faixa de menu, a qual fica na parte superior da janela e geralmente é utilizada para que o usuário tenha acesso a outras janelas e formulários. Para incluí-la, basta arrastar o item da Caixa de Ferramentas e ela automaticamente será colocada em seu lugar.

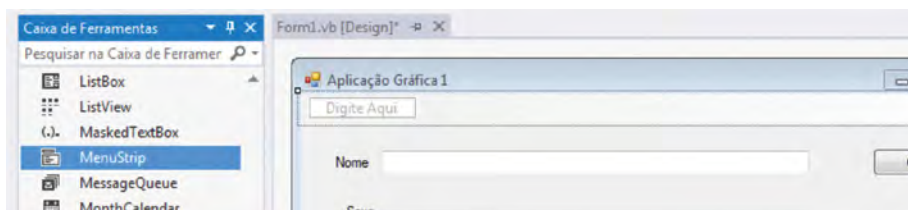


Figura 5.23 – Colocação de uma barra superior de menu.

Para adicionar itens ao MenuStrip, basta digitá-los diretamente. Com isso é possível criar menus e submenus.

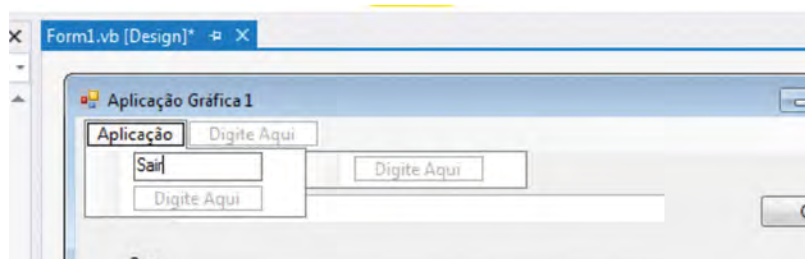


Figura 5.24 – Edição de itens de um MenuStrip.

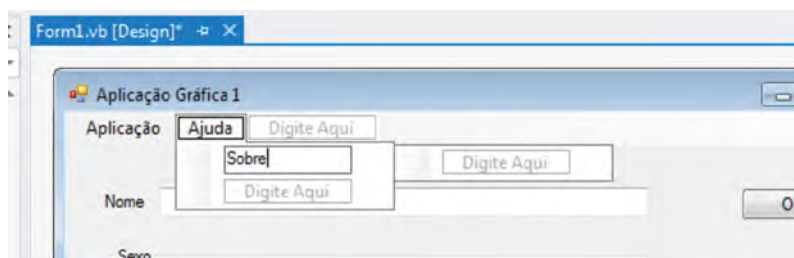


Figura 5.25 – Mais itens sendo adicionados ao MenuStrip.

Cada um dos itens de menu criados terá o comportamento parecido com um botão. Para experimentar seu resultado final, execute o programa do formulário.

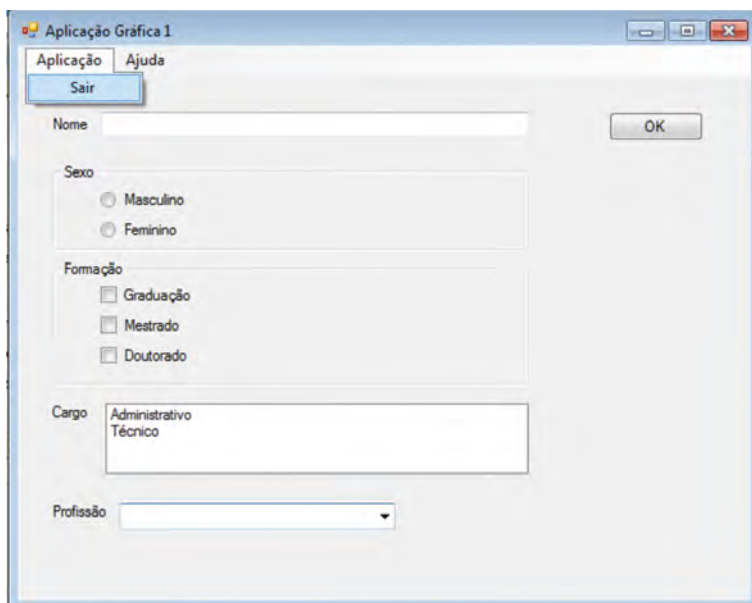


Figura 5.26 – O programa sendo executado novamente, com o MenuStrip funcionando.

Como era de se esperar, o formulário criado nesse exemplo não executa nenhuma tarefa especial. Podemos considerar que ele é uma "casca", cheia de recursos para acessarmos o código de um programa e executar seus métodos. Esse foi apenas um experimento para entender o funcionamento de alguns dos elementos de formulários mais comuns, mas logo criaremos um exemplo em que um formulário realmente faz alguma coisa significativa.

Cada um dos controles adicionados a um formulário é um objeto criado a partir de uma classe. Esses objetos possuem propriedades e métodos, sendo que alguns desses métodos reagem a ações do usuário, as quais são chamadas eventos.



CONEXÃO

Você pode encontrar uma lista dos controles Windows Forms divididos por função nesse link: <https://msdn.microsoft.com/pt-br/library/xfak08ea.aspx>

5.1.2 Programação Orientada a Eventos

Até o capítulo anterior vimos apenas como criar programas que se comunicam com o usuário utilizando o Console do sistema operacional. Quando utilizamos esse meio de interação com o usuário, há apenas um evento de sistema sendo utilizado, que é justamente essa interação de um canal: o computador imprime um texto na tela, e o usuário responde quando há uma instrução `ReadLine()`. Não há outra forma de interagir com o programa elaborado para o Console e, por isso, podemos dizer que o evento é a entrada de dados seguido do pressionamento da tecla "Enter".

Por outro lado, quando um programa possui uma ou mais janelas, campos de texto, botões etc., temos várias possibilidades de eventos: o clicar em um botão, a passagem do mouse por uma área da janela, a entrada de texto em um campo e até mesmo o fechamento de uma janela, entre várias outras possibilidades. Cada um dessas ações é um Evento e, à sua ocorrência, podemos atribuir uma ação que deve ser executada por nosso programa. Por exemplo, em linguagem humana, podemos programar algo como:

Se o evento "clique no botão nomeado MudarCor" ocorrer:
Execute a mudança de cor da janela

Nesse exemplo, utilizamos um botão e uma janela, os quais são normalmente chamados de Controles na programação gráfica de janelas ou de páginas web. Existe vários eventos passíveis de serem associados a cada um deles. O evento mais comumente usado para botões são cliques (pressionamentos), logicamente, enquanto para as janelas são sua abertura ou fechamento. Mas os *frameworks* de programação gráfica permitem que outros eventos sejam associados aos controles, como a passagem do mouse sobre eles, ou o clique do botão direito em vez do esquerdo. Na verdade, podemos dizer a nosso programa para executar várias ações quando ocorrer um evento.

Fazendo um detalhamento técnico mais profundo, o que o *framework* de programação gráfica faz é associar um elemento gráfico (desenho de um botão, por exemplo) a um objeto (objeto botão, que é uma instância da classe `Button`) e utilizar seus métodos de interação (clique no botão) para criar um sequência de ações a serem executadas quando o evento ocorre. Por enquanto parece confuso, mas após a montagem e análise de um exemplo ficará mais claro. Nas próximas páginas faremos passo a passo um programa de exibição de imagens com uma interface gráfica simples.

Antes disso, é importante perceber uma coisa importante: o fato de estarmos utilizando o paradigma de programação orientada a eventos não exclui o uso do paradigma orientado a objetos. Com as tecnologias atuais, a programação com eventos está sempre junta à programação com objetos, sendo os eventos associados a métodos dos objetos.

5.2 Desenvolvimento de Software

A partir daqui, será descrito um passo a passo com as etapas de construção de um aplicativo ligado a um banco de dados utilizando o Visual Studio. Muitas outras linguagens de programação e suas IDEs (Integrated Development Environment) para construção visual de aplicativos utilizam alguns elementos parecidos, então é interessante observar um caso prático de forma analítica. No entanto, para cada linguagem e suas IDEs disponíveis, há muito detalhes que devem ser aprendidos quando o programador inicia seu uso.

Para ilustrar um caso de programa criado com formulários dos Windows e com o VB.NET, vamos iniciar a construção um programa simples para controle de vendas. Ele faz bem mais do que imprimir "Hello, World!" em uma janela, mas ainda é bastante fácil de se programar, principalmente porque utiliza recursos prontos e disponíveis no Visual Studio e no Framework .NET. Como um sistema de controle de vendas pode necessitar de muitas janelas e cruzamentos de dados, apenas construiremos um início desse sistema, o qual você pode tentar terminar depois por conta própria.

5.2.1 Criando uma Aplicação Windows Forms

A primeira coisa a se fazer é criar um novo projeto do tipo Aplicativo Windows Forms. Para isso, no menu ARQUIVO, clique em "Novo Projeto..." e na janela que se abre escolha um modelo Visual Basic, Aplicativo Windows Forms e, abaixo, em "Nome", mude para algo como "WindowsApplication10-1". Clicando em OK, seu projeto será criado.

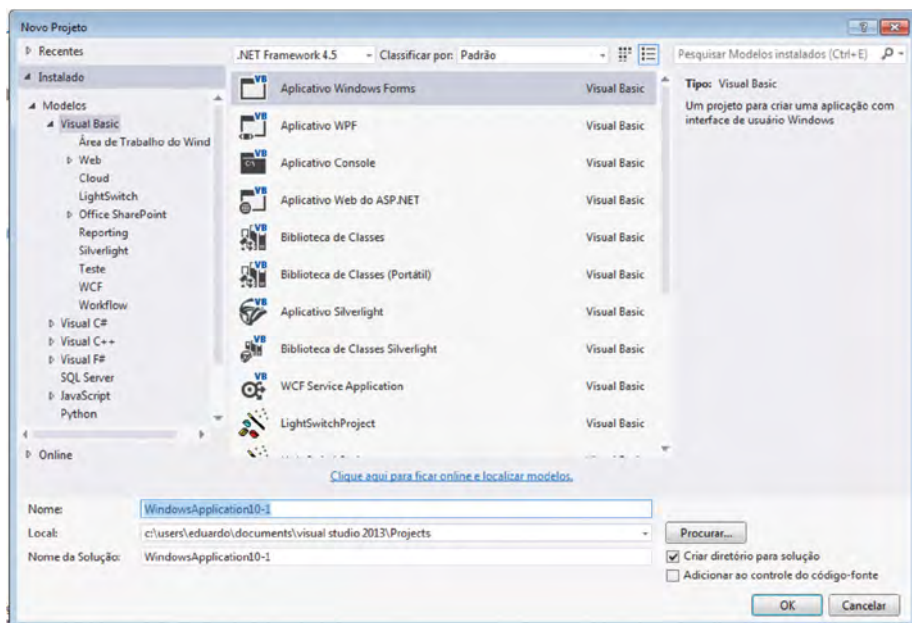


Figura 5.27 – Janela de opções para criação de um novo projeto.

Para acessar os dados do banco "vendas" contidos no SQL Server (criado no Capítulo 4), criaremos janelas com detalhes das tabelas. Nesse exemplo, criaremos apenas uma para acessar os dados de clientes. Para isso, é necessário adicionar controles para abrir as janelas, o que faremos através de um menu de opções no alto de nossa janela principal.

Para adicionar barra de menu, abra a Caixa de Ferramentas entre as opções que se encontram na borda esquerda do Visual Studio. Se a Caixa de Ferramentas não estiver visível, é possível exibi-la acessando o item de menu EXIBIR, do Visual Studio. Navegue nessa caixa e ache o controle ToolStrip (o nome pode variar dependendo da sua versão do Visual Studio). O caminho completo desse controle é:

Caixa de Ferramenta > Menus & Barras de Ferramentas > ToolStrip

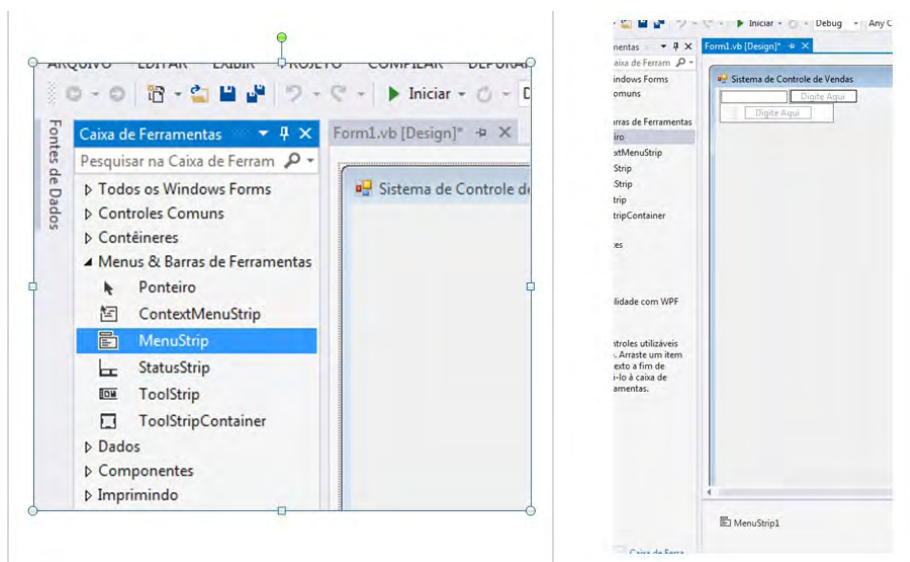


Figura 5.28 – Colocação de um MenuStrip do sistema de gestão de vendas.

Após localizar o controle MenuStrip, o arraste para o Form1. Automaticamente, ele será colocado no topo da janela e permitirá que os itens e subitens de menu sejam digitados diretamente. No rodapé do Visual Studio, abaixo do Form1, há uma área que também conterá um item MenuStrip. Essa área serve para indicar que alguns itens estão lá, associados à janela, mesmo que invisíveis.

Adicione a seguinte estrutura de itens ao menu, cujo resultado é exibido na próxima figura:

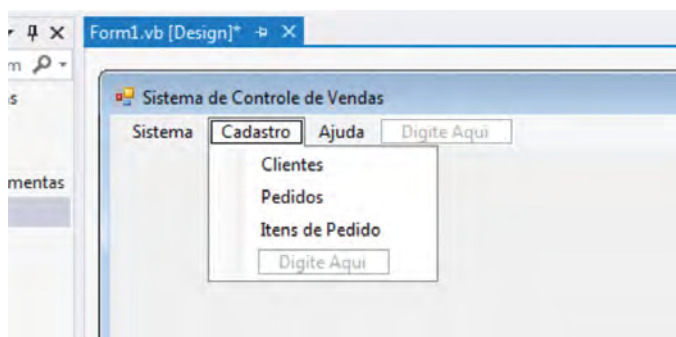


Figura 5.29 – Itens do MenuStrip.

Agora já temos um meio de acessar outras janelas que podemos criar para acessar os dados do SQL Server. Antes de criar essas janelas, é interessante fazer a ligação de nosso aplicativo ao banco de dados de vendas criado no Capítulo 4.

5.2.2 Adicionando uma Fonte de Dados

É possível fazer conexões com fontes de dados diretamente através do código de seu programa em Visual Basic, assim como através da interface gráfica. Aqui nos deteremos a mostrar como adicionar uma fonte de dados graficamente. O interessante é que essa forma de adição cria uma camada de código automaticamente, que deixa essa fonte de dados disponível para uso de forma geral em qualquer local do código fonte e, além disso, permite a adição de controle de dados ligados a ela.

Para adicionar como fonte (DataSource) o banco de dados de vendas criado no capítulo 4, é interessante utilizar o guia oferecido pelo Visual Studio. Uma das formas de iniciá-lo é através do menu superior: PROJETO > Adicionar Nova Fonte de Dados...

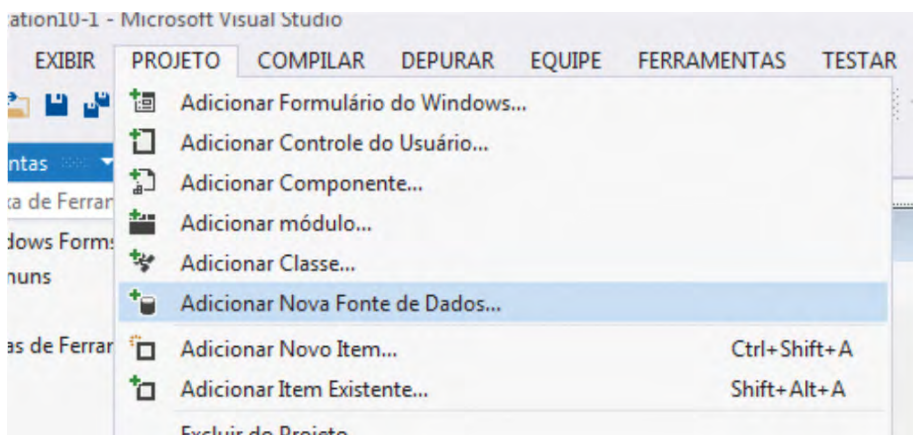


Figura 5.30 – Item de menu para iniciar o assistente de adição de fonte de dados.

Isso fará com que o assistente (wizard) se inicie. Ele permitirá que se crie uma conexão direta com a fonte de dados, a qual pode ser um arquivo do MS Access (que precisa estar instalado no computador), arquivos texto ou uma instância SQL Server. Se a conexão for feita com o SQL Server, ele pode estar instalado no próprio computador ou em outro na mesma rede. Na primeira tela, escolha "Banco de Dados" e avance. Na segunda tela, escolha "Conjunto de Dados".

É comum encontrarmos software que nos ajudam a instalar, realizar novas configurações ou criar coisas utilizando um passo a passo, como o Visual Studio faz. Em inglês, esses guias são chamados Wizards, ou “magos”, que tornam fácil para um novo usuário ou mesmo um experiente realizarem tarefas que podem ser complicadas. Isso é um estilo de desenho de interfaces que busca facilitar a experiência do usuário. Você pode incorporar essas técnicas em suas próprias interfaces. Já pensou nisso?

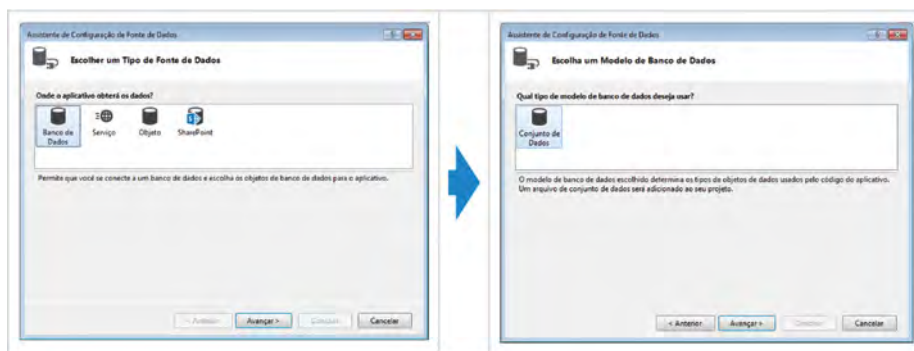


Figura 5.31 – Passos de escolha do tipo de fonte de dados.

Um Conjunto de Dados é um DataSet sobre o qual discutimos no capítulo 4. Ele permite que o conjunto de dados seja manipulado na memória, exigindo que um DataAdapter obtenha os dados na fonte. Quando se cria essa conexão através do guia do Visual Studio, as opções de configuração são um pouco limitadas - se quiser ter mais controle sobre o DataSet, é interessante que procure na documentação oficial e o faça diretamente através do código fonte.

Após escolher a opção "Conjunto de Dados" e avançar, a conexão com o banco de dados em si será criada. Na tela seguinte, será mostrado que ainda não há uma conexão. Clique em "Nova Conexão..." e escolha Microsoft SQL Server.

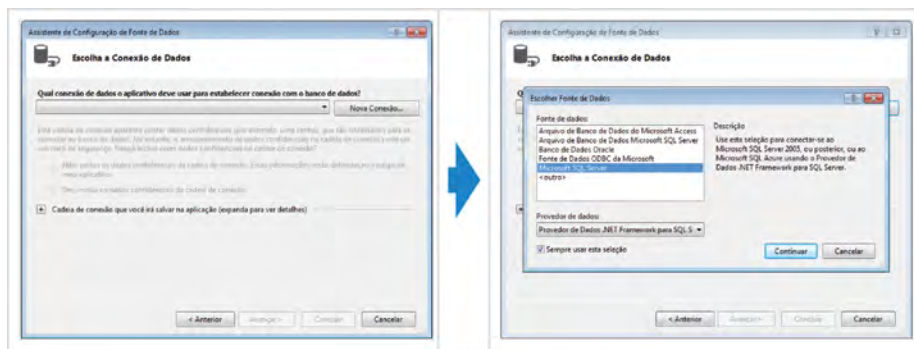


Figura 5.32 – Passos de criação de uma nova fonte de dados.

Após escolher o SQL Server como fonte de dados, basta continuar para abrir a janela de configuração da conexão. No campo do nome do servidor é necessário colocar o mesmo caminho que se utiliza para fazer login no SQL Server Manager. Se você fez a instalação do SQL Server com as configurações padrão, o nome do servidor será algo como "NOME_COMPUTADOR/SQLEXPRESS". No caso da máquina usada para criar esses exemplos, o nome é "WIN7-VIRTUAL/SQLEXPRESS", mas você deve se adaptar ao seu caso. Além disso, é necessário escolher a forma de autenticação - se fez a instalação padrão do SQL Server, deve deixar marcada a autenticação do Windows. Por fim, mais abaixo na janela é necessário escolher o banco de dados para conexão, que no nosso caso é o "vendas".

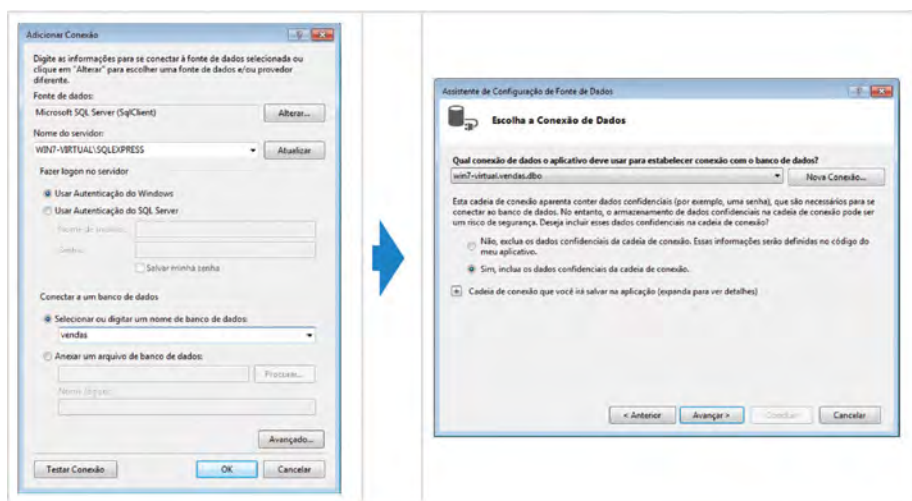


Figura 5.33 – Telas de conexão com o banco de dados existente, criado no Capítulo 4.

Agora há uma opção de conexão de dados com o banco de vendas. Veja que nesse passo há um aviso sobre a segurança de transmissão de senha entre o aplicativo e o servidor de banco de dados. Em nosso contexto, não nos preocuparemos com isso, pois estamos utilizando aplicativo e servidor de banco de dados no mesmo computador, mas evite fazer esse tipo de conexão a um servidor através da internet.

Nos próximos passos, aceite a cadeia de conexão padrão e, na escolha de objetos de bancos de dados, escolha "Tabelas" e "Modos de exibição" (que são as *views*). Desse modo você terá à disposição todas as tabelas do banco de dados de vendas para poder continuar o desenvolvimento do aplicativo.

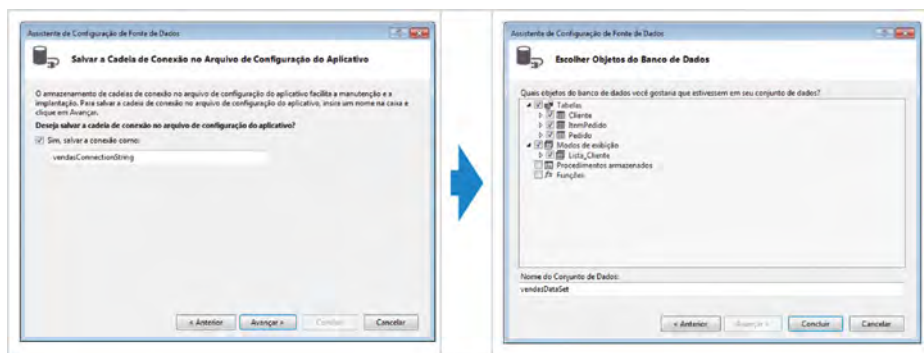


Figura 5.34 – Telas de escolha do nome da cadeia de conexão e tabelas a acessar.

Após a escolha dos objetos do banco de dados, basta concluir o processo. Depois disso, na caixa do Gerenciador de Soluções, acima e à direita, o DataSet com o banco de dados de vendas está disponível como um dos objetos do projeto.

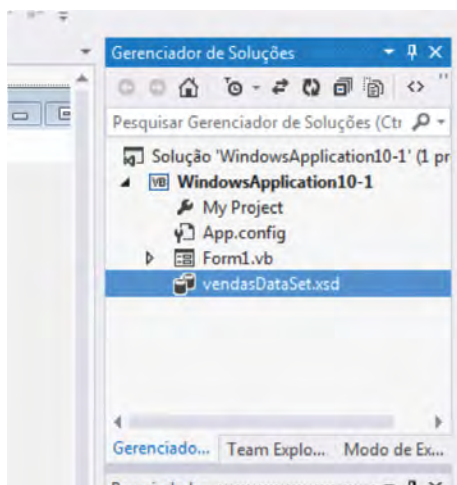


Figura 5.35 – A fonte de dados aparece agora no Gerenciador de Soluções.

Agora é possível utilizar controles de manipulação de dados, os quais se comunicam diretamente com o SQL Server.

5.2.3 Formulários para as entidades de dados

Quando se trata de desenhar interfaces para o usuário lidar com dados organizados em um banco de dados, existem algumas formas básicas de *design*, sendo que as duas principais são listas e detalhes. Mesmo que as pessoas em geral não percebam, são arquétipos bastante utilizados em interfaces:

- **Listas:** como o nome diz, são listas que mostram o conjunto dos dados, como os clientes, pedidos de vendas, álbuns de música, fotos armazenadas em uma pasta etc. Pode ser uma lista apenas em formato texto, com códigos e nomes, mas também, por exemplo, pode ser um conjunto de miniaturas de fotos de um álbum;
- **Detalhes:** é uma tela ou janela que mostra os dados relativos a um dos itens. Em geral, quando se clica em um item da lista, é aberta uma tela onde podem ser editados ou apenas vistos os dados detalhados desse item.

Há outras formas de organização e visualização, como através de gráficos, mas listas e detalhes são os mais comuns. Como exemplo para o nosso sistema de controle de vendas, faremos uma janela de detalhes dos clientes, a qual se abre quando esse item é clicado no menu superior da janela principal Form1.

Primeiro, é necessário criar uma nova janela vazia. No menu superior do Visual Studio, em PROJETO, escolha "Adicionar Formulário do Windows...".

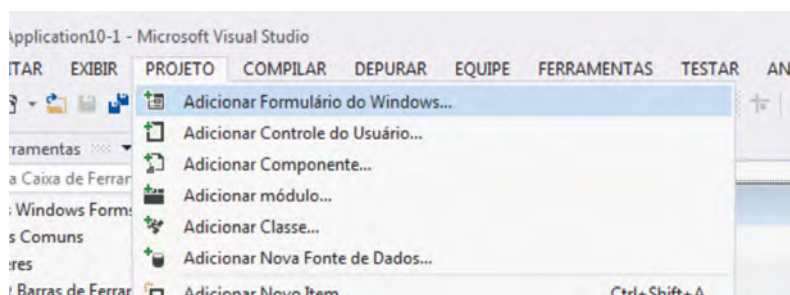


Figura 5.36 – Menu para adição de um novo formulário.

A caixa de opções de itens para adição se abrirá. Nele, deve ser escolhido o item Windows Form e, abaixo, o nome para a janela deve ser "Clientes.vb". Depois de mudar o nome, clique no botão Adicionar.

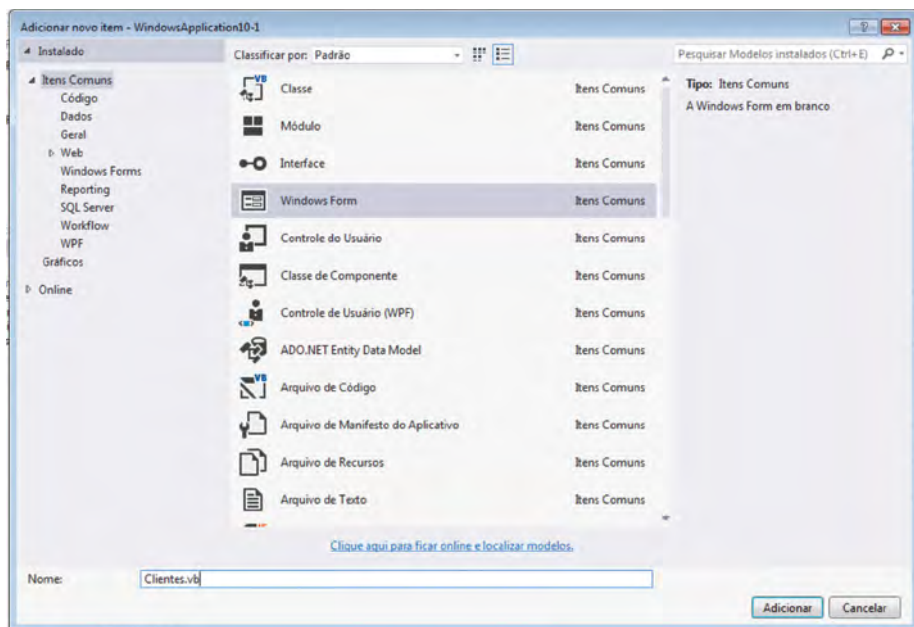


Figura 5.37 – Tela de opções para novo formulário. Escolha “Windows Forms”.

Um novo Form será criado, já configurado com a propriedade Text recebendo o valor Clientes. Não é necessário mudar o tamanho da janela, pois ela se ajustará à nossa necessidade no próximo passo. Agora deve ser adicionado um formulário que permita a adição de clientes e alteração de dados já existentes.

Um meio de interagir com os controles de fontes de dados é através da caixa “Fontes de Dados”. Ela pode estar minimizada na lateral esquerda do Visual Studio, do mesmo modo que a “Caixa de Ferramentas” também minimiza. Se não houver nada a respeito da caixa de Fontes de Dados visível, use o atalho de teclado SHIFT+ALT+D ou vá ao menu superior do Visual Studio e navegue pelo seguinte caminho:

EXIBIR > Outras Janelas > Fontes de Dados

Na caixa de Fontes de Dados é possível observar o DataSet, as tabelas, as views e os campos das tabelas. Tenha certeza que a janela Clientes está aberta na área de *design*, pois isso permitirá que apareçam botões de opções ao lado do nome das tabelas, como na figura a seguir (se na área principal estiver aberto um arquivo de código, esses botões de opções não aparecem).

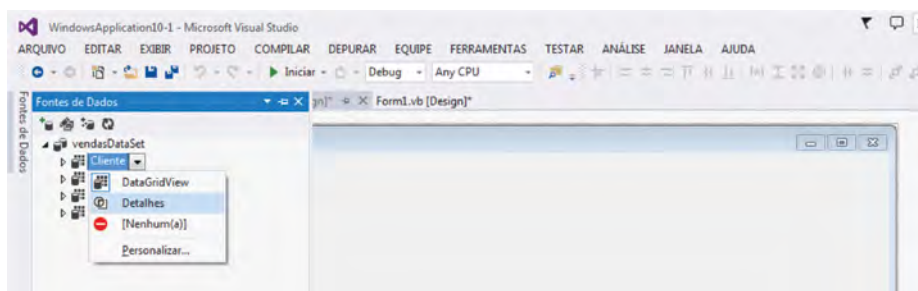


Figura 5.38 – Menu para escolha e arrastamento do controle de acesso a dados.

Entre as opções estão DataGridView e Detalhes. O DataGridView nada mais é do que uma lista dos dados da tabela, como discutido há alguns parágrafos. "Detalhes" também é um objeto que já foi apresentado: um formulário para adicionar dados e editar dados já existentes. Arraste a opção "Detalhes" da tabela Cliente para a janela Clientes que está na área de *design*, e o Visual Studio se encarrega de criar todas as conexões necessárias e montar um formulário com todos os campos da tabela Cliente.

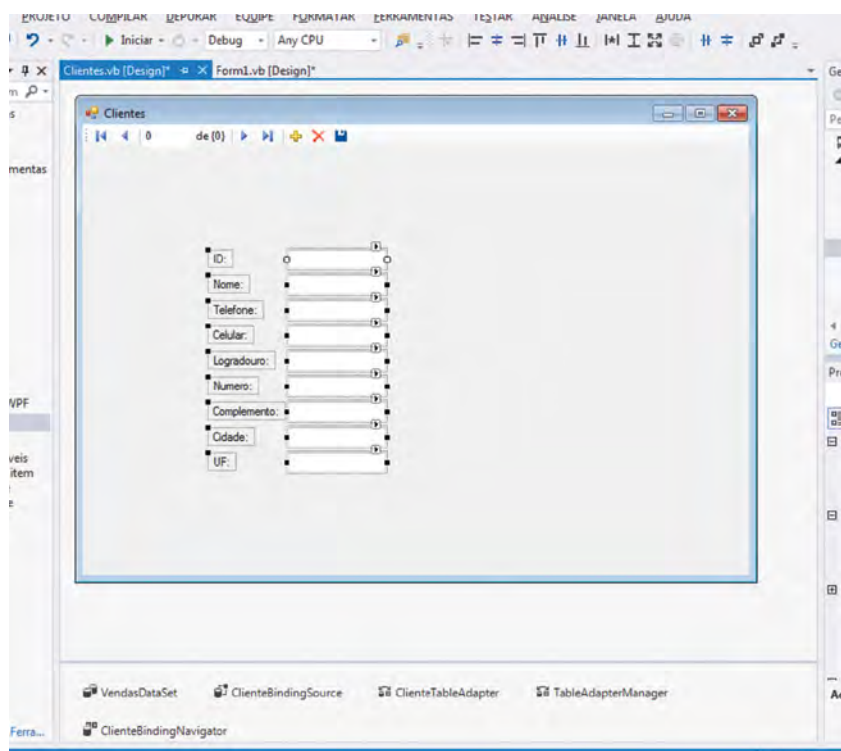


Figura 5.39 – Formulário recebe controles de acesso à tabela Cliente.

Observe que no rodapé da área de *design* aparecem os vários elementos de conexão com o DataSet, as quais são invisíveis na janela, mas estão presentes no código gerado automaticamente (eles foram comandados no final do Capítulo 4). Se a sua janela era pequena demais para todos os campos, ela deve ter se ajustado para contê-los. Se não gostou da aparência, arrume o *layout* experimentando arrastar os campos e usando as ferramentas de alinhamento na barra superior. O ideal é deixar os campos de preenchimento de acordo com o número de caracteres definidos na criação das tabelas de bancos de dados.

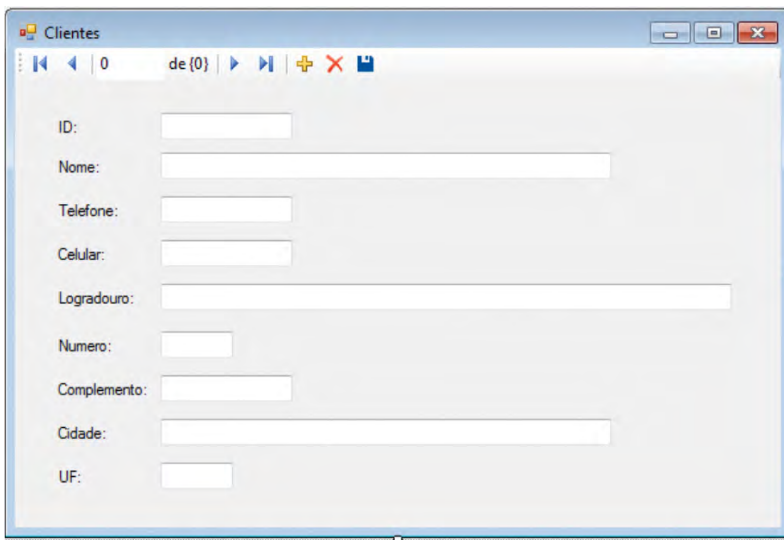


Figura 5.40 – Formulário de acesso a dados de Cliente, após organização do layout.

Lembre-se de salvar o projeto. Ainda é necessário criar uma forma de abrir essa janela. Isso deve ocorrer quando o usuário clica no item Clientes do menu da janela principal.

5.2.4 Resposta a um Evento: abrindo uma janela

Para abrir a janela de cadastro de clientes, é necessário utilizar um evento que crie uma instância desse form e faça uma chamada de seu método de exibição. A primeira coisa a se fazer, portanto, é criar um trecho de código para esse evento: acesse a aba do Form1 na área de design e, em seu menu superior, clique duas vezes no item "Clientes".

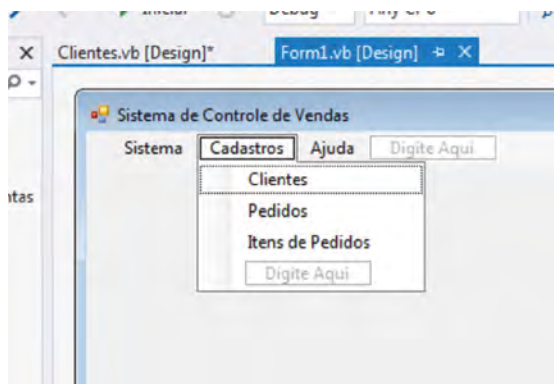


Figura 5.41 – Clique duas vezes no item de menu “Clientes” para editar seu código.

Quando se clica duas vezes em um controle, o Visual Studio abre o arquivo de código fonte daquela janela e cria um evento relacionado à ação principal de tal controle.

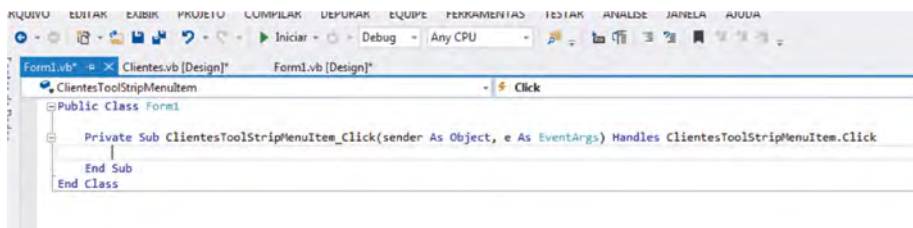


Figura 5.42 – Método que gerencia os eventos de clique no item de menu “Clientes”.

No caso de um item de menu, o evento é "Click". Para facilitar a leitura, o trecho de código correspondente ao evento "Click" do objeto `ClientesToolStripMenuItem` é reproduzido no quadro a seguir:

```
Form1.vb
Public Class Form1
    Private Sub ClientesToolStripMenuItem_Click(sender As Object, e As EventArgs) Handles ClientesToolStripMenuItem.Click
    End Sub
End Class
```

Código 5.1 – Reprodução da sub-rotina vazia do item de menu “Clientes”.

Perceba que essa subrotina ainda está vazia, isto é, se o programa for executado agora e o item Clientes no menu for pressionado, nada ocorre. Todo o código que for colocado dentro dessa subrotina será executado quando houver essa seleção no menu. Portanto, adicione as duas linhas como no trecho a seguir:

```
Form1.vb
Public Class Form1
    Private Sub ClientesToolStripMenuItem_Click(sender As Object, e As EventArgs)
Handles ClientesToolStripMenuItem.Click
        Dim ClientesForm As New Clientes
        ClientesForm.Show()
    End Sub
End Class
```

Código 5.2 – Método do item de menu “Clientes” invocando a abertura da tela criada.

É importante analisar o que ocorre nesse trecho de código. O Form1 é uma classe que é instanciada na execução do programa. Os controles também são objetos que são criados no início do programa, e eles possuem métodos que representam eventos (MSDN, 2015b). No exemplo, ClientesToolStripMenuItem é um objeto que possui um método Click que apresenta um evento. O Form1 possui uma subrotina ClientesToolStripMenuItem_Click que recebe um sinal do objeto ClientesToolStripMenuItem e executa seu conteúdo quando ele é clicado.

Então, quando o item de menu Cliente é clicado, o seguinte acontece:

1. Dim ClientesForm As New Clientes <<uma instância da janela Clientes é criada>>
2. ClientesForm.Show() <<a instância da janela criada é exibida para o usuário>>

O verbo *Handle* significa manipular, manusear ou controlar. Além disso, na declaração da subrotina (*procedure*) os argumentos são o objeto como emissor do sinal e seus eventos que causaram o sinal de ação. Isso significa que a *procedure* recebe o sinal e manipula com sua subrotina o que for necessário para o clique do objeto desencadear ações. A figura a seguir procura resumir tudo isso.

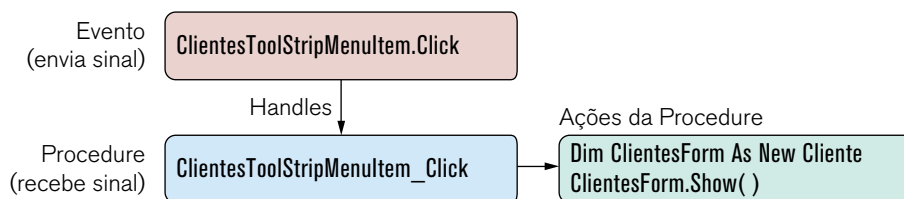


Figura 5.43 – Esquema da comunicação entre eventos e métodos que os manipulam.

Depois que o trecho de código foi adicionado, já é possível executar o programa que permite o cadastro de clientes. Veja que esse é um tipo de controle simples, que interaja diretamente com o banco de dados. É necessário preencher todos os campos que na criação da tabela foram declarados NOT NULL, pois eles não podem ficar vazios.

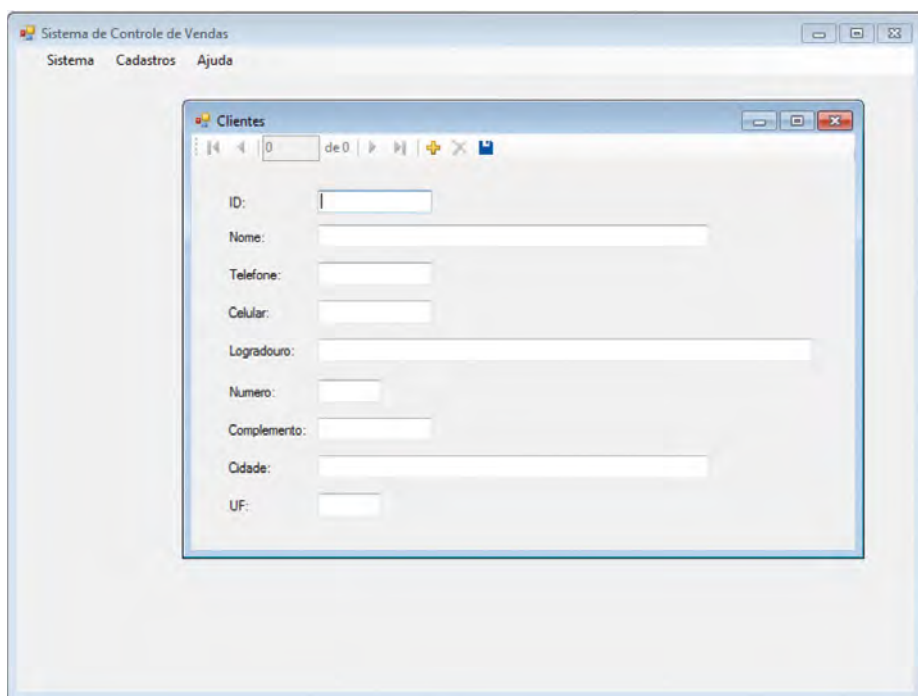


Figura 5.44 – Programa de gerenciamento de vendas executando, com a tela de clientes aberta.

Atenção: como estamos fazendo um exemplo simples, não serão criadas funções para tratamento de erros, então vários deles podem ocorrer no preenchimento do formulário. Por exemplo, se um campo obrigatório não for

preenchido, ou se forem colocadas letras em um campo numérico, o Visual Studio vai parar de executar e apresentar mensagens de erro. Nesse caso será necessário parar o programa (botão com quadrado vermelho na barra superior) e iniciar de novo.

Com o objetivo de verificar como fica o código do Form1 caso adicionemos mais elementos ao programa, vamos criar uma janela do tipo "Sobre", que fornece ao usuário informações sobre o nome do programa, sua descrição, seu desenvolvedor e versão atual. A partir das versões mais novas do Visual Studio, esse tipo de informação não é mais digitado diretamente em uma janela comum, mas em uma caixa de diálogo específica que traz os dados dos meta-dados do código fonte do programa.

Da mesma forma que já fez anteriormente, crie um novo Windows Form, mas dessa vez escolha uma opção específica, do tipo "Caixa Sobre", como na figura a seguir. O nome do form pode ser Sobre.vb.

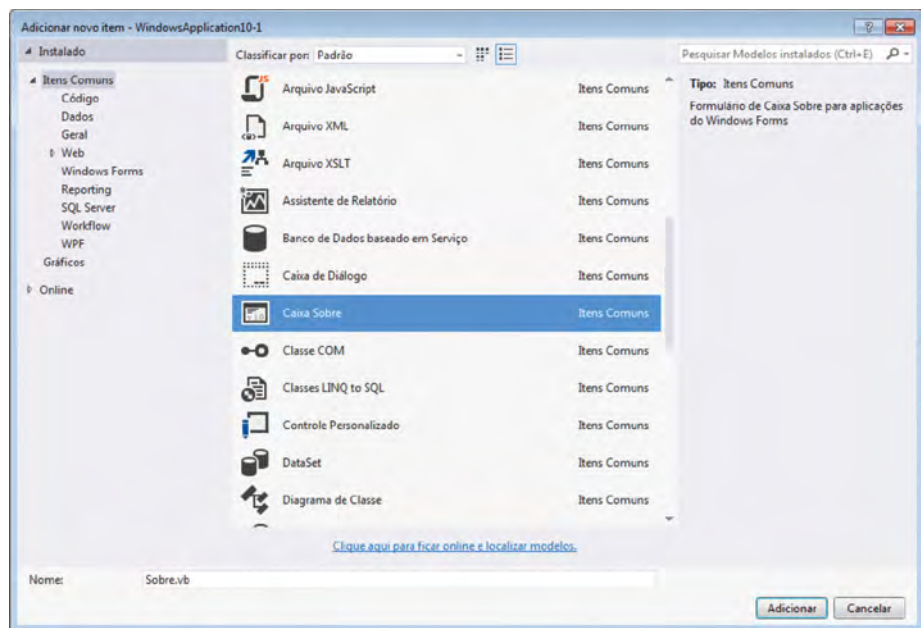


Figura 5.45 – Tela de opções para uma nova janela. Escolha "Caixa Sobre".

Essa caixa tem uma aparência padrão. A sua figura pode ser alterada para o logotipo de uma empresa, por exemplo. No entanto, o texto não deve ser alterado manualmente, como está explicado na própria janela.

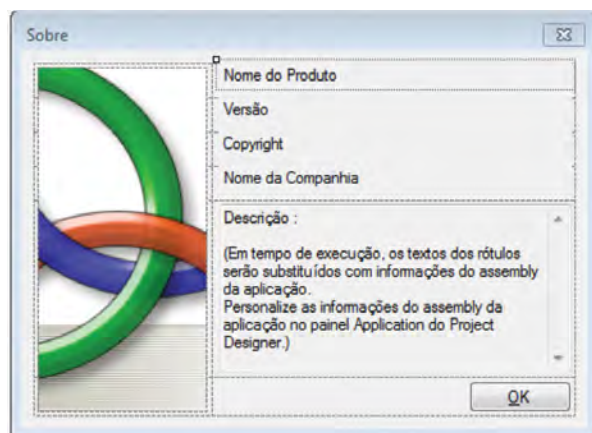


Figura 5.46 – Tela de desenho para a “Caixa Sobre”. Não edite o texto diretamente nela.

Para alterar os dados dessa caixa de diálogo, é necessário preencher as configurações da Assembly (que é uma versão compilada do programa, nos termos que o Visual Studio utiliza). Para isso, vá ao item de menu:

PROJETO > Propriedades de WindowsApplication10-1...

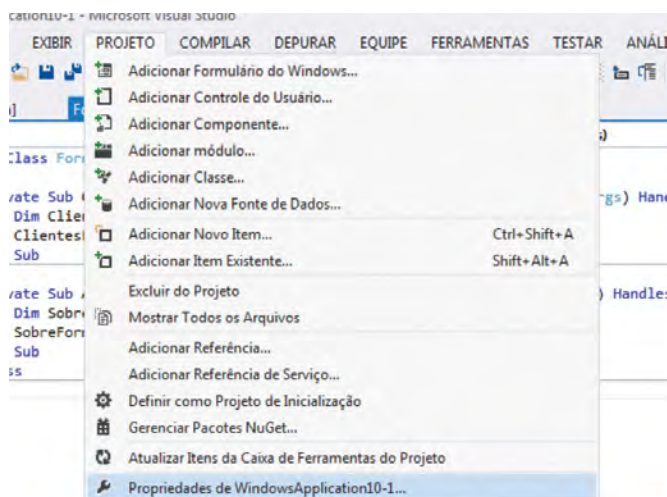


Figura 5.47 – Menu para acesso de informações sobre o projeto.

Na janela que se abre, é possível alterar várias configurações do aplicativo. Clicando no botão “Informações do Assembly...”, a pequena janela que se abre permite alterar todos os dados que serão transportados para a janela “Sobre” no momento da compilação.

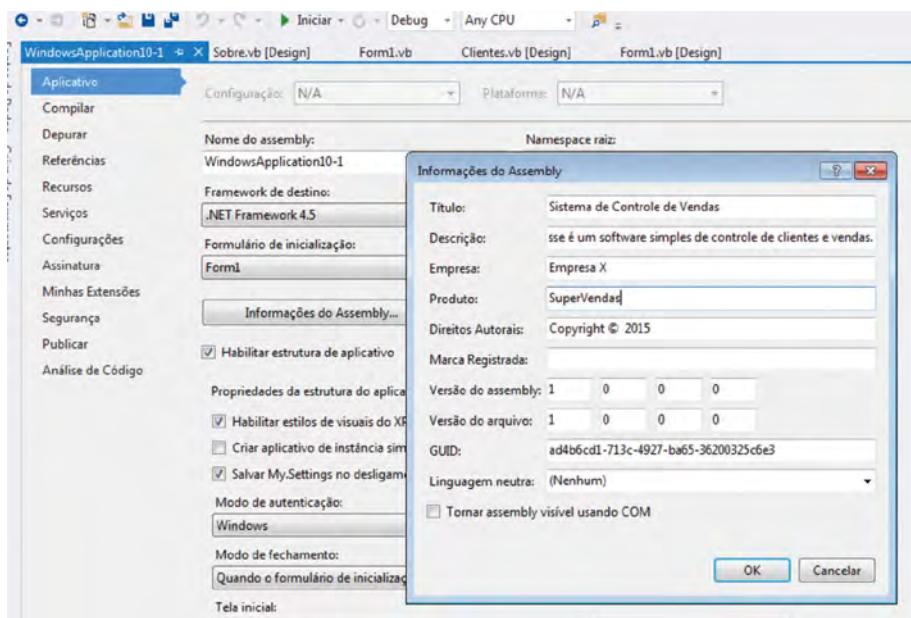


Figura 5.48 – Tela de edição de informações sobre o Assembly.

Ainda é necessário criar um item "Sobre" no menu "Ajuda" do Form1. Os passos iguais ao da janela de cadastro de clientes.

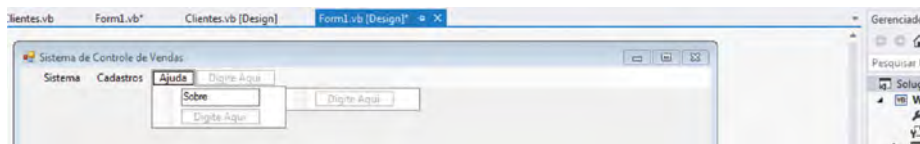


Figura 5.49 – Criação de item de menu para acessar a tela "Sobre".

Depois que o item de menu for criado, clique duas vezes nele. A janela com o código do Form1 será exibida novamente, com uma subrotina pronta para ser preenchida. O código completo está no quadro a seguir.

Form1.vb

Public Class Form1

```
Private Sub ClientesToolStripMenuItem_Click(sender As Object, e As EventArgs)
Handles ClientesToolStripMenuItem.Click
Dim ClientesForm AsNew Clientes
```

```

        ClientesForm.Show()
    End Sub

    Private Sub SobreToolStripMenuItem_Click(sender As Object, e As EventArgs)
Handles SobreToolStripMenuItem.Click
        Dim SobreForm As New Sobre
        SobreForm.Show()
    End Sub

End Class

```

Código 5.3 – Método do item de menu “Sobre” com código que inova a abertura da janela.

Perceba, portanto, que cada evento para cada controle adicionado ao Form1 terá uma subrotina dentro da classe Form1. Cada conjunto de ações é executado a medida que o seu evento é disparado por uma ação do usuário. Após digitar esse código, salve o projeto e o execute. Clicando no item de menu “Sobre”, o resultado será o da figura seguinte.

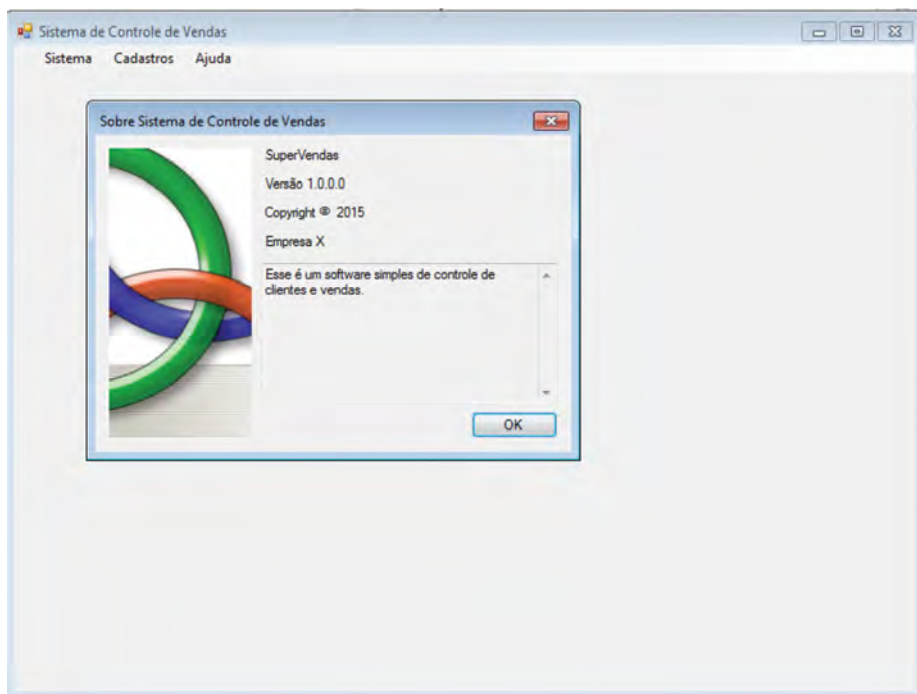


Figura 5.50 – Programa executando, com janela sobre aberta e informações do projeto atualizadas.

Agora você tem um protótipo de um programa e passamos por diversas etapas e conceitos de desenvolvimento de um software com interface gráfica e conexão ao um servidor de banco de dados. Sinta-se à vontade para explorar outras características do Visual Studio e tentar modificar o programa até que tenha algo interessante em mãos. É importante destacar que a profissão de desenvolvedor de software é baseada em muita pesquisa e aprendizado sobre as ferramentas disponíveis, e que cada uma delas possui muitas especificidades. Busque consultar guias e documentação da Microsoft e experimente bastante!



REFLEXÃO

O desenvolvimento de um software completo, que desempenha tarefas reais do dia a dia, possui muitas partes, tanto na interface, com diversas telas para interação com o usuário, como bibliotecas que desempenham tarefas de fundo (*background*), isto é, módulos e classes que realizam computações auxiliares para o programa.

O desenho ideal de janelas e instrumentos de interação com o usuário é aprendido tanto através de experiência, testes quanto com o estudo de disciplinas de engenharia de software, usabilidade e experiência do usuário. Os objetivos principais são que se atenda às necessidades do usuário (requisitos), ainda mantendo formas fáceis, agradáveis e eficientes de uso.



ATIVIDADES

01. Explique com suas palavras o que é Programação Orientada a Eventos e como ela se relaciona com o desenvolvimento de interfaces gráficas.
02. O que são os controles RadioButton, CheckBox e ListBox? Quais as diferenças e semelhanças entre eles?
03. É possível criar janelas e adicionar controles a elas apenas de forma gráfica, ou é possível também através de código? Justifique sua resposta.



LEITURA

Uma das melhores formas de se habituar à montagens de programas com interfaces gráficas é seguindo guias com casos práticos. Veja esse que está disponível no MSDN. O título é em inglês, mas a maior parte dele está traduzido:

Tutorial 1: Create a Picture Viewer

Disponível em: <<https://msdn.microsoft.com/pt-br/library/dd492135.aspx>>



REFERÊNCIAS BIBLIOGRÁFICAS

MSDN, Microsoft Developer Network. Visão geral dos Windows Forms. Disponível em: <<https://msdn.microsoft.com/pt-br/library/8bxy49h.aspx>>. Acesso em: 14 dez. 2014.

_____. Visão geral sobre eventos (Windows Forms). Disponível em: <<https://msdn.microsoft.com/pt-br/library/1h12f09z%28v=vs.110%29.aspx>>. Acesso em: 22 fev. 2015.



GABARITO

Capítulo 1

01. *Frameworks* de software são arcabouços, estruturas ou plataformas de desenvolvimento que fornecem ferramentas que facilitam a programação de software. Eles podem fornecer ferramentas na forma de classes, programas auxiliares, compiladores, ambiente de execução e padrões de comunicação entre softwares. Quando utiliza um *framework*, o programador pode usar essas ferramentas da forma que lhe convierem e seu propósito é agilizar a programação, deixá-la mais robusta e segura.

02. Bibliotecas de software são módulos ou classes que possuem tipos de dados, métodos e padrões de comunicação de software que podem ser invocados por qualquer outro módulo de software desenvolvido pelo programador que a utiliza. Um programador pode desenvolver sua própria biblioteca para facilitar o uso de ferramentas que necessita com frequência. Uma biblioteca normalmente se refere a programas não executáveis, mas que são embutidos em outros programas que, por sua vez, são executáveis.

03. O **.NET Framework** é o conjunto de ferramentas na forma de classes, programas auxiliares, compiladores, ambiente de execução e padrões de comunicação entre softwares ofe-

recidos pela Microsoft para apoiar a programação. O **Visual Studio** é apenas uma IDE (*Integrated Development Environment*), que é um programa com um ambiente que facilita a escrita de código, mas tem como uma de suas funcionalidade principais o acesso facilitado ao .NET Framework. O **Visual Basic** é apenas uma linguagem de programação, mas ela se apoia em toda a estrutura do .NET Framework, utilizando-a como biblioteca para desenvolvimento e como ferramenta para compilação e execução.

04. Para o cálculo da medida em polegadas, basta multiplicar o valor em centímetros pelo fator de conversão

```
Module Module1

    Sub Main()
        Dim medida_centimetro As Decimal = 150
        Dim medida_polegada As Decimal

        Const fator As Decimal = 0.393700787

        medida_polegada = medida_centimetro * fator

        Console.WriteLine(medida_polegada)
        Console.ReadKey()
    End Sub

End Module
```

Capítulo 2

01. Um módulo é uma forma de compartimentar o código de um programa em partes que possuem funcionalidades afins. Esse módulo pode ser composto de métodos, propriedades e constantes e ser invocadas a qualquer momento em outras partes do código. Os módulos podem conter classes e métodos.

Os métodos podem ser funções ou sub-rotinas que podem ser invocadas de outros módulos ou objetos com objetivo de manipular dados. As funções são trechos de código que executam uma ou mais tarefas que podem ser chamados com ou sem parâmetros; as funções, no Visual Basic, sempre retornam valores. Ainda no VB.NET, as sub-rotinas são como funções, mas não retornam valores.

02. Para criar um programa que acumule valores somados de uma função "soma", é necessário criar uma variável que tenha escopo no módulo e seja alterada a cada laço que chama essa função. É possível pensar em diferentes formas de fazer esse programa.

```

1  Module Module1
2      Dim total As Integer = 0
3
4      Sub Main()
5          soma()
6      End Sub
7
8      Sub soma()
9          Dim valor As Integer
10         Do
11             Console.WriteLine("Digite um valor:")
12             valor = Console.ReadLine()
13             total = total + valor
14             If (valor <> 0) Then
15                 Console.WriteLine("O total agora é: {0}", total)
16             End If
17         Loop Until (valor = 0)
18     End Sub
19
20 End Module

```

03. Uma estrutura de decisão é uma forma de controle de um algoritmo que pode alterar sua sequência de execução em função de um teste lógico. O exemplo a seguir pede um número para o usuário. se o número for par, exibe essa informação na tela do console, caso contrário, imprime "Ímpar". Para fazer esse teste, é necessário usar o operador Mod, para saber se a divisão do número por 2 é igual a 0 ou 1.

```

1  ModuleModule1
2
3      Sub Main()
4          Dim valor As Integer
5          Console.WriteLine("Digite um número:")
6          valor = Console.ReadLine()
7
8          If (valor Mod 2 = 0) Then
9              Console.WriteLine("Par")
10         Else
11             Console.WriteLine("Ímpar")
12         End If
13         Console.ReadKey()
14     End Sub
15
16 End Module

```

04. Uma estrutura de repetição é uma forma de controle que faz com que uma sequência de código seja repetida um certo número de vezes ou até que uma determinada condição ocorra. Uma estrutura de repetição baseada em contador permite que se determine quantas vezes esse laço vai se repetir, independentemente de qualquer condição. Uma estrutura baseada em condição não limita as repetições a uma certa quantidade, mas a que determinada condição ocorra alterando um teste lógico que o faz parar.

Capítulo 3

01. Tratamento de exceções é a utilização de recursos que permitem capturar erros ou comportamentos inesperados em um programa, criando um procedimento que manipula essa informação de erro e dê uma resposta útil para o programador e para o usuário do software. O tratamento de exceção é importante porque assume que certa rotina pode falhar, impedindo que o programa trave, feche ou não execute apropriadamente, fornecendo informações relevantes sobre a ocorrência.

02. As variáveis precisam ser todas decimais, ou haverá um outro erro de tipo que terá que ser tratado também.

```
1  Module Module1
2
3      Sub Main()
4          Dim valor1, valor2, resultado As Decimal
5          Console.WriteLine("Forneça o primeiro número:")
6          valor1 = Console.ReadLine()
7          Console.WriteLine("Forneça o segundo número:")
8          valor2 = Console.ReadLine()
9
10         Try
11             resultado = valor1 / valor2
12             Console.WriteLine("Divisão dos valores = {0}", resultado)
13         Catch ex As DivideByZeroException
14             Console.WriteLine("O segundo valor não pode ser zero!")
15         End Try
16         Console.ReadKey()
17     End Sub
18
19 End Module
20
```

03. Classes são trechos de código compartimentados que constituem construções ou modelos dos quais podem ser gerados objetos; as classes apresentam atributos e comportamentos para as entidades representadas pelos objetos. Por sua vez, os objetos são instâncias criadas pelas classes, os quais podem efetivamente ser manipulados pelo programa.

As propriedades de objetos são variáveis em seu escopo, as quais armazenam dados a seu respeito e que os caracterizam individualmente. Métodos são funções ou sub-rotinas que manipulam dados no escopo de objetos e são uma forma de alterar suas propriedades.

04. Para esse exercício, é necessário criar a classe retângulo com suas propriedades e seu método. Na sub-rotina Main, deve-se criar uma instância do retângulo e usar o texto digitado pelo usuário como argumento para o método `mudar_cor()`. Com isso feito, o objeto `retangulo1` passa a ter o atributo digitado.

```
1  ModuleModule1
2
3      Sub Main()
4          Dim retangulo1 As New Retangulo
5
6          Console.WriteLine("Qual deve ser a cor do retângulo?")
7          retangulo1.mudar_cor(Console.ReadLine())
8          Console.WriteLine("A cor do retângulo agora é {0}", retangulo1.cor)
9          Console.ReadKey()
10     End Sub
11
12     Public Class Retangulo
13
14         Public Property base As Decimal
15         Public Property altura As Decimal
16         Public Property cor As String
17
18         Public Sub mudar_cor(ByVal nova_cor As String)
19             Me.cor = nova_cor
20         End Sub
21
22     End Class
23
24 End Module
```

Capítulo 4

01. A linguagem padrão para manipulação de bancos de dados é o SQL (*Strucured Query Language*). Ela é uma linguagem declarativa, com a qual o programador não necessita estabelecer o algoritmo de filtragem e cruzamento de dados para obter o que precisa. Os mecanismos de bancos de dados usam a declaração feita com o SQL para realizar matemática relacional e tornam a busca e gravação de dados no BD bastante rápidas.

02. Um SGBD com a arquitetura cliente-servidor é aquele em não é embutido nos softwares que o utilizam. Eles são centralizados, geralmente em uma rede, e os diversos dispositivos com seus aplicativos se conectam a eles para obter dados. A principal característica é essa divisão entre aplicativos clientes e o servidor de dados, que costuma ficar em um computador separado. A recepção e transmissão de dados se dá por protocolos de rede.

03. Os comandos para criação de um banco de dados padrão é bastante simples:

```
CREATE DATABASE concessionaria
```

A criação de uma tabela necessita a definição dos tipos de dados de cada um de seus campos. O campo ID serve para identificação única, mas nesse caso a chave primária pode ser o número de chassi. Quanto ao "ano", não é necessário usar o tipo data. De forma geral, outros tipos e tamanhos podem ser utilizados ao invés dos que estão a seguir.

```
CREATE TABLE automovel  
(  
  ID bigint NOT NULL Primary Key,  
  marcanvarvhar(15) NOT NULL,  
  modelonvarchar(15) NOT NULL,  
  anoint NOT NULL,  
  cornvarchar(15) NOT NULL,  
  chassi char(17) NOT NULL,  
)
```

04. Uma view é uma espécie de tabela virtual, a qual pode resultar da consulta de uma ou mais tabelas conjuntas, com cruzamento de dados. A vantagem de uma view é que ela pode ser registrada no banco de dados, de forma que possa ser consultada como qualquer outra tabela, não utilizando espaço adicional. Ela é otimizada pelo SGBD, tornando-se mais rápida para consultas com JOIN que são frequentemente realizadas. As views também constituem uma camada de dados que simplifica as buscas realizadas pelo aplicativo final, facilitando seu desenvolvimento.

Capítulo 5

01. A Programação Orientada a Eventos permite que multiplas formas de interação com o software estejam à disposição do usuário. Na interação por interface de texto há apenas um evento, que é através da entrada de dados pelo teclado. Na interface gráfica, há vários elementos, como botões, janelas, caixas de textos e controles em geral. Cada interação com um elemento gráfico gera um evento, para o qual o computador pode ser programado para responder. O evento gera um sinal, o programa captura esse sinal e responde com uma ação apropriada.

02. Os controles `RadioButton` são aqueles que permitem o usuário escolher uma alternativa entre um grupo pré-determinado, isto é, são opções mutuamente exclusivas. Os controles `CheckBox` podem pertencer a uma mesmo grupo também, mas permitem que o usuário escolha e marque mais de uma opção de dado ao mesmo tempo. As `ListBox` são lista de opções que podem ter seu comportamento alterado, podendo ser mutuamente exclusivas ou não.

03. É possível também adicionar janelas e controles a uma interface gráficas apenas através de código, sem necessitar arrastar ícones a uma área de desenho. Apesar de a construção de interfaces ser facilitada pela forma gráfica, em algumas situação pode ser interessante fazê-lo através de código, principalmente para um programador experiente no *framework*. Na verdade, quando construímos interfaces de forma gráfica, o que a ferramenta de *design* faz, como o Visual Studio, é transformar aquele *layout* em um código, o qual poderia ser feito sem sua ajuda.



ANOTAÇÕES



ANOTAÇÕES



ANOTAÇÕES