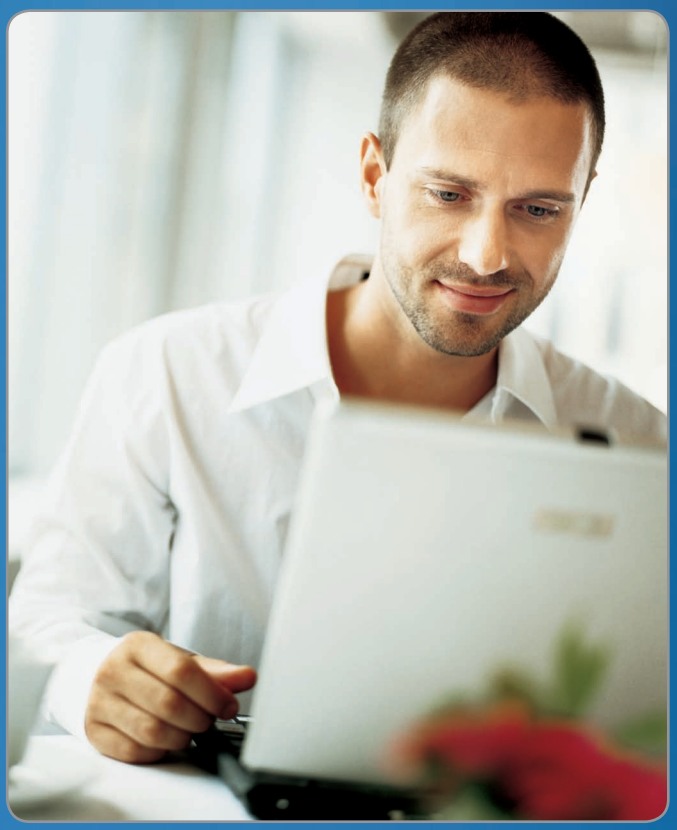
Microsoft®

Vaše možnosti. Naše inspirace.



Ján Hanák

VB 6.0 → VB 2005:

Přecházíme z jazyka Visual Basic 6.0 na jazyk Visual Basic 2005

Příručka pro programátory, vývojáře, softwarové odborníky a IT specialisty

Microsoft Visual Studio 2005

Microsoft Visual Studio Express Editions

VB 6.0 → VB 2005:
Přecházíme z jazyka Visual Basic 6.0 na jazyk Visual Basic 2005

Příručka pro programátory,
vývojáře, softwarové odborníky a IT specialisty

Obsah příručky

Úvod

Vývojářská příručka *VB 6.0 → VB 2005: Přecházíme z jazyka Visual Basic 6.0 na jazyk Visual Basic 2005* je zaměřena na vývojáře a programátory, kteří vytvářejí svá řešení v jazyce Visual Basic 6.0, ovšem rádi by svoji hodnotnou práci povýšili na novou úroveň a začali psát moderní aplikace .NET pomocí jazyka Visual Basic 2005 a vývojově-exekuční platformy Microsoft .NET Framework 2.0. Publikace tak plní roli startovního bodu, o který se mohou vývojáři opřít při poznávání nového stylu programování aplikací, jenž softwarový produkt Visual Basic 2005 přináší. Cílem příručky je poskytnout vám relevantní informace nejenom o samotném jazyku Visual Basic 2005, ale také o architektuře platformy .NET Framework 2.0. Podrobnější obsah jednotlivých kapitol je uveden níže.

Kapitola 1.:
Úvod do jazyka Visual Basic 2005

V této kapitole se podíváme na dosavadní historické milníky vývoje programovacího jazyka Visual Basic od společnosti Microsoft. Dozvíte se základní informace o nové vlně ve vývoji počítačového softwaru, kterou pozvedlo uvedení vize Microsoft .NET společně s uvedením vývojového rámce .NET Framework a kompletu vývojářských nástrojů sdružených v sadě Visual Studio .NET. Kromě přínosů verzí .NET 2002 a .NET 2003 jazyka Visual Basic se budeme soustředit také na stručný popis novinek, s nimiž mezi vývojáře, programátory a IT specialisty zavítá Visual Basic 2005.

Kapitola 2.:
Směr Visual Basic 2005 – Instalace, konfigurace a integrované vývojové prostředí (IDE)

Druhá kapitola se soustřeďuje na charakteristiku instalace a konfigurace softwarového produktu Visual Basic 2005. Poté, co je Visual Basic 2005 připraven na svoji práci, společně prozkoumáme všechny důležité komponenty integrovaného vývojového prostředí (IDE). Pochopení nových partií IDE je nepostradatelné zejména pro programátory používající Visual Basic 6.0, neboť získání správných návyků při práci s vývojovým prostředím je půlkou úspěchu při migraci do nového prostředí jazyka Visual Basic 2005. Kapitola obsahuje rovněž informace o změnách v projektovém managementu a představuje také spektrum aplikací .NET, které je možné pomocí Visual Basicu 2005 vyvinout.

Kapitola 3.:
Architektura vývojově-exekuční platformy Microsoft.NET Framework 2.0

Hlavní obsahovou náplní třetí kapitoly je ozřejmění stavby a funkcí platformy .NET Framework 2.0 v procesu vývoje moderních aplikací .NET. Výklad je zaměřen na objasnění všech hlavních součástí společné jazykové infrastruktury, ke kterým patří:

1. Bázová knihovna tříd platformy .NET Framework 2.0 (.NET Framework Class Library, FCL)
2. Společná jazyková specifikace (Common Language Specification, CLS)
3. Společný typový systém (Common Type System, CTS)
4. Společné běhové prostředí (Common Language Runtime)

Dále můžete v této části publikace najít statě pojednávající o základní funkční aplikační jednotce, kterou je v prostředí platformy .NET Framework 2.0 takzvané sestavení (assembly). Dozvíte se, jak se sestavení tvoří a které komponenty jej formují. Stranou nezůstane ani podrobný popis životního cyklu aplikací .NET s přihlédnutím na Just-In-Time kompilaci kódu jazyka Microsoft Intermediate Language (MSIL).

Kapitola 4.:
Syntaktické a sémantické inovace jazyka Visual Basic 2005

Programovací jazyk Visual Basic 2005 přináší velkolepé inovace, které zpříjemňují vývojářům jejich náročnou práci v daleko větším rozsahu, než tomu bylo v předcházejících verzích jazyka Visual Basic. Pro programátory ve Visual Basicu 6.0 bude přesun do prostředí jazyka Visual Basic 2005 zcela jistě krokem do neznáma. Abyste se ovšem v novém světě Visual Basicu 2005 neztratili, je pro vás připravena samostatní kapitola, která vám představí všechny podstatné změny, modifikace a úpravy, které se dotýkají programové syntaxe a sémantiky tohoto programovacího jazyka. Pro lepší pochopení nových principů bude vysvětlovaná tematika hojně překládána praktickými postupy a snadno pochopitelnými ukázkami fragmentů zdrojového kódu. Pokud to bude možné, budeme se snažit využívat analogii s Visual Basicem 6.0, a to tak, že nejprve představíme způsob, jak programátoři pracovali s daným programovým elementem v jazyce Visual Basic 6.0, a poté se podíváme na změny, které přinesl Visual Basic 2005. Domnívám se, že uvedený styl výkladu vám bude, jako „migrujícím" vývojářům, vyhovovat v nejlepší možné míře.

Kapitola 5.:
Objektově orientované programování v jazyce Visual Basic 2005

V minulosti byl Visual Basic 6.0 mnohdy kritizován zejména ze strany programátorů v C++, že není dostatečně objektově orientován, a že neposkytuje kompletní podporu pro implementaci koncepce objektově orientovaného programování (OOP). Budeme-li abstrahovat od „věčných válek" mezi příznivci Visual Basicu a C++, můžeme prohlásit, že Visual Basic 6.0 byl objektově orientován, ovšem implementace prvků koncepce OOP nebyla kompletní. Ve Visual Basicu 6.0 jste kupříkladu nemohli aplikovat dědičnost, abyste z jedné třídy odvodili podtřídu. Také zde chyběla syntaktická podpora pro překrývání, zastiňování a přetěžování programových elementů. Problematická byla rovněž práce s rozhraními: Programátoři sice mohli rozhraní implementovat do tříd, ovšem nebyli schopni je vytvářet. Vzhledem k tomu, že ve Visual Basicu 6.0 absentovala podpora dědičnosti, bylo možné polymorfizmus implementovat pouze prostřednictvím rozhraní. Zkrátka a dobře, po uvedených skutečnostech musíme přiznat, že v oblasti kompletního začlenění koncepce objektově orientovaného programování měl jazyk Visual Basic 6.0 nemalé rezervy. Situace se ovšem radikálně změnila s příchodem Visual Basicu .NET, který zavedl jednoduchou dědičnost, možnost vytvářet rozhraní, pracovat s konstruktory a destruktory, a také přetěžovat a překrývat programové prvky. Na základech OOP, jež položil jazyk Visual Basic .NET, staví i nově uvedený Visual Basic 2005, který třímá další esa v rukávu: jednoznačně nejpřínosnější je technika přetěžování operátorů a podpora generických typů.

O nejmarkantnějších změnách v oblasti objektově orientovaného programování mezi jazyky Visual Basic 6.0 a Visual Basic 2005 si budeme povídat v páté kapitole této příručky. Dozvíte se, jak od bázové třídy odvodit novou třídu a dovedností této třídy dále rozšířit, či jak volat metody bázové třídy. Rovněž pochopíte práci konstruktorů a destruktorů neboli finalizačních metod.

Kapitola 6.:
Visual Basic 2005 Express

Pojednání o softwarovém produktu Microsoft Visual Basic 2005 Express uzavírá různorodou směsici témat vývojářské příručky *VB 6.0 → VB 2005: Přecházíme z jazyka Visual Basic 6.0 na jazyk Visual Basic 2005*. Visual Basic 2005 Express je samostatnou edicí jazyka Visual Basic 2005, která se spíše než na profesionální vývojáře, programátory a IT odborníky zaměřuje na adepty programování, studenty informačních technologií či programátorské fanoušky. V kapitole naleznete základní informace o tomto novém produktu společnosti Microsoft a dojde také na porovnání „velkého" Visual Basicu 2005 a jeho mladšího brášku řady Express.

Pro koho je tato příručka určena

Vývojářská příručka *VB 6.0 → VB 2005: Přecházíme z jazyka Visual Basic 6.0 na jazyk Visual Basic 2005* je určena všem programátorům, vývojářům a tvůrcům softwarových aplikací, kteří mají několikaleté zkušenosti s programováním v jazyce Visual Basic 6.0 a kteří by rádi začali programovat působivé aplikace .NET v prozatím nejnovější verzi jazyka Visual Basic s číselným označením 2005. Obsahová náplň příručky vychází z odborného profilu, který u cílového spektra čtenářů předpokládá níže uvedené schopnosti a dovednosti:

Tab. 1: Odborní profil cílového publika uživatelů této příručky

	Znalostní předpoklady, které JSOU vyžadovány		Znalostní předpoklady, které NEJSOU vyžadovány
✓	Patříte mezi pokročilé programátory v jazyce Visual Basic 6.0 a dokážete samostatně vytvářet funkční softwarové aplikace.	✗	Není nutné, abyste znali architektonickou stavbu a funkci jednotlivých komponent vývojově-exekuční platformy .NET Framework.
✓	Syntaxi jazyka Visual Basic 6.0 ovládáte na takové úrovni, že vám nečiní potíže zapsat anebo pochopit i složitější partie programového kódu.	✗	Nemusíte mít zkušenosti s jazykem Visual Basic .NET, ani s jazykem Visual Basic .NET 2003.
✓	Disponujete základními znalostmi objektově orientované koncepce programování (OOP).	✗	Nemusíte znát proces modelování vztahů mezi komponentami aplikací v jazyce UML.
✓	Víte, jak pracuje integrované vývojové prostředí jazyka Visual Basic 6.0 a dokážete jej plně ovládat.		
✓	Znáte základní aspekty technologií COM (Component Object Model) a ActiveX.		
✓	Umíte přistupovat k Win32 API funkcím a používat je ve svých aplikacích.		

Hlavním cílem této vývojářské příručky je poskytnout programátorům v jazyce Visual Basic 6.0 základní informace jednak o architektuře a funkcích platformy .NET Framework 2.0, a také o jazyce Visual Basic 2005. Tento úkol ovšem není jednoduché splnit: je to proto, že Visual Basic urazil od své šesté verze již hodně dlouhou cestu a vývojáři, kteří budou chtít migrovat do prostředí jazyka Visual Basic 2005, se budou muset obeznámit nejenom s novými syntaktickými a sémantickými změnami, ale také s novým stylem programování, jenž přináší Microsoft .NET. Sám si dobře pamatuji doby, kdy jsem programoval své aplikace ve Visual Basicu 6.0. Vím, že Visual Basic 6.0 byl svého času opravdu mocným vývojářským nástrojem, s kterým byla radost pracovat. Když jsem však poprvé spatřil onu tajemnou zář Visual Basicu .NET a vývojově-exekuční platformy .NET Framework, uvědomil jsem si, že přichází nová etapa ve vývoji softwaru. To už nebyl pouze „další" Visual Basic, nýbrž zbrusu nový produkt pro vytváření moderního softwaru, který všem programátorům předvedl, že softwarové aplikace mohou být vytvářeny s daleko větší efektivností, než tomu bylo dříve. Visual Basic 2005 staví na prověřené koncepci Visual Basicu .NET. Kromě toho přináší opravdu velké množství nových prvků, které z Visual Basicu dělají robustní a sofistikovaný programovací nástroj. Kdybyste vedle sebe postavili Visual Basic 6.0 a Visual Basic 2005, nestačili byste se divit, kolik nových dovedností a schopností Visual Basic za sedm let nabyl.

Ano, je jistě pravda, že přechod z Visual Basicu 6.0 na Visual Basic 2005 se může na první pohled jevit jako klikatá cesta plná skrytých pastí a nástrah. I zkušení programátoři ve Visual Basicu 6.0 se po spuštění poslední verze tohoto vývojářského nástroje ocitnou v doposud neznámém prostředí. Provedené inovace se netýkají pouze modifikované syntaxe, nových příkazů, klíčových slov a modifikátorů. V notné míře bylo upraveno také integrované vývojové prostředí a souprava nástrojů s ovládacími prvky a komponentami. Změnil se styl projektového managementu, přibyla dynamická nápověda, asistent pro práci s chybovými výjimkami a další užitečné pomůcky. Při prvních krůčcích ve Visual Basicu 2005 budou programátoři, kteří doposud pracovali pouze v jazyce Visual Basic 6.0, nacházet neustále nové věci, jejichž cílem je zefektivnění programování a zvýšení produktivity při vývoji počítačového softwaru. Důležité je ovšem uvědomit si, že všechno úsilí, které věnujete studiu nového prostředí jazyka Visual Basic 2005, se vám vrátí i s pomyslnými úroky. Visual Basic 2005 je skutečně silný vývojářský nástroj, jehož pomocí můžete programovat nejenom standardní „okenní" aplikace pro Windows, ale také pokročilé webové aplikace a XML webové služby, služby systému Windows, vyspělé databázové aplikace spolupracující s technologií ADO.NET, a v neposlední řadě také aplikace běžící na platformách Windows Mobile for Pocket PC, Windows Mobile for Smartphone a Windows CE .NET. Prostřednictvím mohutné podpory bázové knihovny tříd platformy .NET Framework můžete vyvíjet rovněž síťové aplikace, knihovny ovládacích prvků či aplikace pracující v konzolovém režimu. Visual Basic 2005 vás nenechá na holičkách ani tehdy, pokud byste rádi vytvářeli působivé multimediální aplikace nebo počítačové hry s podporou aplikačního programového rozhraní Microsoft DirectX 9. To ovšem stále není vše: můžete vytvářet doplňky (add-ins), které rozšíří stávající funkcionalitu integrovaného vývojového prostředí, automatizovat aplikace sady Microsoft Office System 2003, sestavovat vlastní kolekce inteligentních značek anebo vykouzlit impresivní grafické efekty prostřednictvím grafického subsystému GDI+. Jak jistě sami uznáte, Visual Basic 2005 vám může dát hodně. A pokud čtěte tuto příručku, jste na nejlepší cestě k ovládnutí nejpopulárnějšího programovacího jazyka na světě, jímž Visual Basic bezesporu je.

Typografické konvence

Abychom vám čtení této publikace zpříjemnili v co možná největší míře, byl přijat kodex typografických konvencí, jehož pomocí došlo ke standardizaci a unifikaci použitých textových a grafických symbolů a stylů. Věříme, že přijaté konvence napomohou zvýšení přehlednosti a uživatelské přívětivosti výkladu. Přehled použitých typografických konvencí je uveden v tab. 2.

Tab. 2: Přehled použitých typografických konvencí

Typografická konvence	Ukázka použití typografické konvence
Veškerý text, který neoznačuje zdrojový kód, jména identifikátorů a klíčových slov jazyka Visual Basic 2005, ani názvy jiných programových elementů a entit, je psán standardním písmem.	Vývojově-exekuční platforma Microsoft .NET Framework 2.0 vytváří společně s jazykem Visual Basic 2005 jednotné zázemí pro vytváření moderních aplikací .NET pro Windows, web a inteligentní mobilní zařízení.

Názvy nabídek, položek nabídek, ovládacích prvků, komponent, dialogových oken, podpůrných softwarových nástrojů, typů projektů, jakožto i názvy dalších součástí grafického uživatelského rozhraní, jsou formátovány **tučným písmem**.	Pro založení nového projektu standardní aplikace pro Windows (**Windows Application**) v prostředí jazyka Visual Basic 2005 postupujte následovně: 1. Otevřete nabídku **File** a klepněte na položku **New Project**. 2. V dialogovém okně **New Project** klepněte ve stromové struktuře **Project Types** na položku **Visual Basic**. 3. Ze sady projektových šablon (**Templates**) vyberte ikonu šablony **Windows Application**. 4. Do textového pole **Name** zapište název pro novou aplikaci a stiskněte tlačítko **OK**.
Klávesové zkratky a jejich kombinace jsou uváděny KAPITÁLKAMI.	Jestliže chcete otevřít již existující projekt jazyka Visual Basic 2005, použijte klávesovou zkratku CTRL+O.

Fragmenty zdrojových kódů jazyků Visual Basic 6.0 a Visual Basic 2005, případně také jiných programovacích jazyků, jsou formátovány neproporcionálním písmem `Courier New`. Kromě toho jsou ve všech výpisech barevně rozlišeny následující programové elementy:
 	
1. Klíčová slova programovacího jazyka jsou formátována modrou barvou.
2. Komentáře, které blíže popisují charakter činnosti programového příkazu nebo bloku programových příkazů, jsou zobrazeny pomocí zelené barvy.
3. Textové řetězce jazyka Visual Basic 2005 snadno rozeznáte podle hnědé barvy.
4. Veškerý ostatní kód, jenž neoznačuje ani klíčová slova, ani komentáře, je formátován standardní černou barvou.
	`'Deklarace proměnné typu String.`
	`Dim Zpráva As String`
	`'Inicializace deklarované proměnné.`
	`Zpráva = "Vítejte v jazyce " & _`
	`"Visual Basic 2005!"`
	`'Zobrazení okna se zprávou pomocí`
	`'metody Show třídy MessageBox.`
	`MessageBox.Show(Zpráva)`

Neproporcionální písmo `Courier New` je kromě výpisů zdrojových kódů použito také při uvádění názvů programových entit v základním výkladovém textu. Tímto stylem písma jsou formátovány například názvy tříd a jejich instancí, názvy proměnných, názvy procedur Sub a funkcí a rovněž názvy jiných programových identifikátorů.	Sdílená metoda `Sleep` třídy `Thread` z jmenného prostoru `System.Threading` je schopná uspat aktivní programové vlákno na specifikovaný počet milisekund (ms). Když metodě `Sleep` předáte celočíselní hodnotu udávající počet milisekund pro uspání programového vlákna, metoda zabezpečí bezpečné uspání vlákna po určenou dobu.

Vyjma typografických konvencí uvedených v tab. 2 se můžete v textu publikace setkat také s informačními ikonami, které vám poskytují hodnotné informace související s právě probíranou problematikou. Výčet informačních ikon můžete vidět v tab. 3.

Tab. 3: Přehled použitých informačních ikon		
Informační ikona	**Název informační ikony**	**Charakteristika**
	**Upozornění**	Upozorňuje čtenáře na důležité skutečnosti, které by měl mít v každém případě na paměti, neboť od nich může záviset pochopení dalších souvislostí nebo úspěšné provedení postupu či pracovního algoritmu.
	**Poznámka**	Sděluje čtenáři další a podrobnější informace, které se pojí s vykládanou tematikou. Ačkoliv je míra důležitosti této informační ikony nižší než výše uvedené ikony, ve všeobecnosti se doporučuje, aby čtenář věnoval doplňujícím informačním sdělením svoji pozornost. Může se tak dozvědět nová fakta, nebo najít skryté souvislosti mezi již známými poznatky.
	**Tip**	Poukazuje na lepší, efektivnější nebo rychlejší splnění programovacího úkolu či postupu. Uvidí-li čtenář v textu publikace tuto informační ikonu, může si být jist, že nalezne jedinečný a prověřený způsob, jak produktivněji dosáhnout kýženého cíle.

Informační ikony vystupují jako samostatné ostrůvky, které vám nabízejí relevantní informace z oblasti programování v jazycích Visual Basic 6.0 a Visual Basic 2005. Při jejich tvorbě byly brány v potaz následující aspekty:

1. Informační ikony musejí byt schopny upoutat pozornost čtenáře, a to zejména v okamžiku, kdy je nutné podat vysvětlení obzvlášť důležitého pojmu, termínu nebo technologie. Kromě toho je úkolem informačních ikon přinášet dodatečné poznatky a poukazovat na možnosti efektivnějšího vyřešení programátorského problému či postupu.

2. Informační ikony musejí dodržovat standardní linii výkladu. Jedině tak je zabezpečeno, že čtenář bude moci předstírané skutečnosti okamžitě využít ve svůj prospěch.

3. Informační ikony musejí být hezky graficky vyvedeny, aby byly oku lahodící a dokázaly tak přispět k zvýšení uživatelského komfortu publikace.

Poděkování

Na tomto místě bych chtěl vyjádřit své srdečné díky všem lidem, kteří se na přípravě této vývojářské příručky podíleli. Osobitně bych velice rád poděkoval panu Jiřímu Burianovi ze společnosti Microsoft za vstřícný přístup a skvělou spolupráci během celé doby, kdy jsem tuto publikaci psal.

Kapitola 1.: Úvod do jazyka Visual Basic 2005

Od dob, kdy společnost Microsoft uvedla na softwarový trh produkt Visual Basic 6.0, už uplynula spousta času. Když si totiž uvědomíme, s jakou rychlostí se vyvíjejí stále nové a nové technologie a na trh proudí doslova obrovské množství sofistikovaných a inovačních řešení, člověku se ani nechce věřit, že to byl rok 1998, kdy světlo světa spatřil jazyk Visual Basic ve své šesté verzi.

Visual Basic vždycky sehrával roli korunního prince společnosti Microsoft pro vývoj širokého spektra počítačových aplikací. Největší konkurenční výhodou oproti jiným programovacím jazykům a prostředím byla bezesporu schopnost Visual Basicu nabídnout vývojářům a programátorům důmyslné nástroje pro rychlou a produktivní práci. Velkou zásluhu na úspěchu Visual Basicu měla zcela jistě přepracovaná forma vizuálního vývoje aplikací. Tento styl programování, jenž se může pyšnit přívlastkem vizuální, umožňoval tvůrcům aplikací snadnou cestou navrhnout celé grafické uživatelské prostředí, což bylo něco, s čím se programátoři rychle sžili.

Zatímco první verze jazyka byly původně určeny ještě pro systém MS-DOS, vpravdě přelomovým se stalo čtvrté vydání, které bylo uvedeno v roce 1995. Visual Basic 4.0 nabízel programátorům lepší podporu vizuálního programování a možnost psát aplikace pro 16 i 32bitové prostředí. Kromě toho Visual Basic v této verzi také více nakoukl do světa objektově orientovaného programování, což byla v té době vlastnost vysoce oceňována nejenom odbornou softwarovou komunitou. Vývojářům se ovšem líbili také jiné věci, jmenujme třeba svižnější práci s ovládacími prvky, možnost volat funkce nativního aplikačního programového rozhraní (API), přistupovat k datovým zdrojům pomocí různých technologií a automatizovat aplikace sady Microsoft Office.

V progresivním trendu, jenž nastartoval jazyk Visual Basic 4.0, pokračoval také jeho nástupce. Visual Basic 5.0 byl uveden v roce 1997 a kromě syntaktických jazykových inovací přinesl vylepšení v mnoha důležitých oblastech, integrovaným vývojovým prostředím počínaje a tvorbou ActiveX ovládacích prvků konče. Pokud bychom situaci poněkud zjednodušili, mohli bychom říci, že pečlivá implementace technologie ActiveX představovala největší eso v rukávu Visual Basicu verze 5.0. S pomocí technologie ActiveX se možnosti působnosti programátorů posunuly zase o krok dál. Vývojáři mohli vytvářet vlastní ovládací prvky ActiveX, které bylo možné využít v mnoha aplikacích, například v Internet Exploreru nebo sadě Microsoft Office.

O rok později společnost Microsoft připravila pro komunitu vývojářů novou verzi jazyka Visual Basic s pořadovým číslem 6.0. Ve své šesté reinkarnaci se Visual Basic stal plně profesionálním a respektovaným vývojářským nástrojem, nad jehož dovednostmi a schopnostmi uznale kývali hlavou také vývojáři používající při své práci jiné mocné nástroje, zejména však Visual C++. Bohatá podpora pro syntaktické konstrukce, sofistikovaný debugger, možnost tvořit DHTML a IIS aplikace, sestavovat multifunkční ovládací prvky a komponenty, hluboká podpora průvodců či pečlivá optimalizace generovaného nativního kódu – to vše byly prvky, jejichž pomocí se Visual Basic 6.0 zapsal do paměti několika milionům tvůrců softwarových aplikací po celém světě. Jednoduše řečeno, vývojář se ve společnosti Visual Basicu 6.0 mohl cítit jako zkušený mariňák ve svém oblíbeném obrněném vozidle: Vždy věděl, že na dosah ruky má všechny nezbytné nástroje, které ke své práci skutečně potřebuje.

Je nepopiratelné, že softwarový produkt Visual Basic 6.0 vskutku předběhl svoji dobu. Důkazem tohoto tvrzení je skutečnost, že i v dnešních časech existuje mnohočetná skupina vývojářů, programátorů a IT specialistů, kteří svá důvtipná softwarová řešení vyvíjejí v prostředí tohoto jazyka.

Čas ovšem nejde zastavit. A když nahlédneme do počítačové historie dále, zjistíme, že záhy po uvedení Visual Studia 6.0 začali specialisté z amerického Redmondu ve svých tajných laboratořích připravovat

produkt, jehož cíl byl velice ambiciózní: odstartovat novou revoluční epochu v způsobu, jakým jsou tvořeny a vyvíjeny moderní softwarové aplikace jednadvacátého století. Když pak na začátku roku 2002 představil Microsoft plody své několikaleté práce, tisícům vývojářů se zatajil dech. .NET – pouhá kombinace čtyř znaků stačila na pojmenování nové etapy ve vývoji softwaru. Cílem strategie .NET bylo uvedení moderního, snadno spravovatelného a rozšířitelného vývojově-exekučního prostředí, na jehož základech by mohly bez jakýchkoliv potíží koexistovat a vzájemně spolupracovat různé typy softwarových aplikací, které byly připraveny v rozličných vývojových nástrojích. A tak se zrodila platforma .NET Framework, reprezentující srdce strategie Microsoft .NET. Zajímavostí této platformy je právě její vývojově-exekuční povaha, což znamená, že .NET Framework je nejenom základním pilířem, na němž operují .NET vývojářské nástroje, ale jde také o prostředí, nabízející všechny požadované nízkoúrovňové softwarové služby, které se pojí s exekucí programového kódu aplikací .NET. Stručně řečeno, .NET Framework nabízí všechny potřebné elementy, které souvisejí s vývojem, laděním a během aplikací .NET. Termínem aplikace .NET se začínají označovat ty softwarové programové jednotky, které využívají dovedností platformy .NET Framework. Ačkoliv podrobněji si architekturu platformy .NET Framework představíme ve třetí kapitole této publikace, již nyní můžeme říci, že jedinečné společné běhové prostředí (Common Language Runtime, CLR) se stará o správný chod aplikací .NET. Prostředí CLR kupříkladu zabezpečuje verifikaci programového kódu aplikací .NET a načtení ověřeného kódu do operační paměti počítače, dále aplikační a platformovou interoperabilitu či automatickou správu alokovaných paměťových segmentů. Jelikož běhové prostředí CLR spravuje životní cyklus aplikací .NET, říkáme, že aplikace .NET jsou řízeny běhovým prostředím CLR. Rovněž rozlišujeme řízený programový kód, který je vykonáván pomocí běhového prostředí CLR, od neřízeného neboli nativního kódu, který ke své aktivaci služby běhového prostředí CLR nepotřebuje. CLR má na starosti také spoustu dalších věcí: dohlíží totiž na typovou bezpečnost, jazykovou interoperabilitu a správu verzí.

Jedním z kamínků v mozaice Microsoft .NET je, samozřejmě kromě platformy .NET Framework, také sada vývojářských nástrojů společnosti Microsoft pro tvorbu aplikací .NET s názvem Visual Studio .NET. První verze této vývojářské edice byla uvedena společně s revoluční strategií .NET v roce 2002. Nedílnou součástí kolekce nástrojů sdružených v sadě Visual Studio .NET byl také Visual Basic .NET – zcela přepracovaná verze jazyka Visual Basic 6.0, která s sebou přinesla takřka nespočetné množství změn a modifikací. Tyto změny se přitom netýkaly pouze syntaktické struktury jazyka, ale zasahovaly do všech oblastí programování v jazyce Visual Basic. Visual Studio .NET přišlo s jednotným integrovaným vývojovým prostředím, které bylo sdíleno napříč všemi .NET programovacími nástroji sdruženými ve Visual Studiu .NET. V praxi to tedy vypadalo tak, že vývojář používající jazyk Visual Basic .NET pracoval ve stejném vývojovém prostředí jako jeho kolegové píšící své aplikace v jazycích C# a C++ s Managed Extensions.

POZNÁMKA: Programovací jazyky C# a C++ s Managed Extensions

Jazyk C# byl navržen a kompletně vyvinut společností Microsoft jako primární programovací nástroj pro vývoj aplikací .NET. C# je ve své podstatě „programátorský hybrid", který kombinuje všechny pozitivní vlastnosti jazyků Visual Basic a C++. Zatímco svou syntaktickou stavbou se jazyk C# ponáší na C++, elegantnost a vysokou produktivitu práce zdědil po Visual Basicu. Jazyk C# byl představen v podobě softwarového produktu Visual C# .NET.

Aby vývojáři mohli pro psaní aplikací .NET používat také jazyk C++, Microsoft uvedl produkt Visual C++ .NET, který v sobě integroval syntaktická rozšíření standardního C++. Tato rozšíření byla pojmenována jako Managed Extensions for C++. Pomocí C++ s Managed Extensions bylo možné tvořit řízené aplikace .NET i z prostředí Visual C++ .NET. Navíc, Visual C++ .NET dovoloval vytvářet i standardní, tedy neřízené aplikace, které nebyly pod správou společného běhového prostředí CLR platformy .NET Framework. Programátoři tak mohli vytvářet softwarová řešení založená na knihovně MFC či „čistém" Win32 API.

Programovací jazyk Visual Basic .NET nebyl pouze přímým nástupcem jazyka Visual Basic 6.0, jak by se na první pohled mohlo zdát. Ve skutečnosti byl Visual Basic .NET téměř naprosto jiným jazykem, a to nejenom ze syntaktického hlediska. Vzhledem k tomu, že úroveň jazyka ve smyslu začlenění nové funkcionality byla výrazně pozdvižena, mnoha programátorům ve Visual Basicu 6.0 se Visual Basic .NET jevil jako zcela jiný programovací jazyk. Ačkoliv tento „katastrofický scénář" jistě není daleko od pravdy, je nutno připomenout, že všichni programátoři, kteří se vydali vstříc nové verzi jazyka Visual Basic, byli po „přeškolení" tak nadšeni inovačními změnami a důmyslnými programovými konstrukcemi, že k „starému" Visual Basicu by se již nikdy nevrátili.

Pakliže se podíváme na jazyk Visual Basic .NET blíže, zjistíme, že tvůrci tohoto nástroje do něj vložili opravdu přehršel změn, inovací a vylepšení. Za všechny vzpomeňme alespoň některé:

1. **Plná podpora koncepce objektově orientovaného programování (OOP).** Ačkoliv Visual Basic již ve své šesté verzi skýtal podporu základních principů koncepce OOP, teprve .NET verze tohoto programovacího nástroje přišla s kompletní sadou syntaktických a sémantických konstrukcí pro tvorbu objektově orientovaných aplikací. Ve Visual Basicu .NET mohli vývojáři uplatnit všechny zásadní principy OOP, mezi něž patří abstrakce, zapouzdření a skrývání dat, dědičnost, polymorfizmus a opětovní použitelnost programového kódu. Stěžejním bylo zejména zařazení plné podpory pro dědičnost, což programátorům umožnilo odvozovat od jednou vytvořených tříd třídy nové, které dědily všechny charakteristiky svých materských tříd. Přestože platforma .NET Framework dovolovala aplikaci pouze jednoduché dědičnosti (každá třída mohla mít právě jednoho přímého předka), jednotlivé třídy mohly implementovat i několik rozhraní. Visual Basic .NET ovšem nabídl mnohem více: vývojáři začali poznávat a pracovat s konstruktory a destruktory, resp. finalizačními metodami, dokázali vytvářet více variant jednoho programového elementu (což je technika známá jako přetěžování elementů), a také mohli překrývat a zastiňovat datové členy bázových tříd v odvozených třídách. Jistě, takový razantní zásah do funkčnosti jazyka se musel odrazit na nových programových konstrukcích, s nimiž byli vývojáři nuceni se nejprve důkladně obeznámit. Vynaložené úsilí se však všech záhy vrátilo i s pomyslnými úroky, neboť použití zásad objektově orientovaného programování dělá vývoj softwarových aplikací daleko přirozenější a intuitivnější, než tomu bylo kdykoliv dříve.

2. **Strukturovaná správa chyb.** Jazyk Visual Basic 6.0 byl charakteristický začleněním nestrukturované správy chyb prostřednictvím příkazu `On Error`. Pomocí tohoto příkazu mohli programátoři specifikovat způsob zpracování programové chyby vzniklé za běhu programu. V každé proceduře Sub nebo funkci bylo možné vytvořit speciální blok kódu s návěstím, jenž vystupoval jako zpracovatel chyb: Vždy, když došlo v těle kódu dané procedury ke generování

chybové výjimky, příkaz `On Error` tento stav rozeznal a posléze aktivoval příslušného zpracovatele chyb. Vývojáři mohli použít také příkaz `On Error Resume Next`, který sice generovanou chybu zachytil, ovšem nijak ji neošetřoval – běh programu proto pokračoval exekucí další instrukce, která byla umístěna za řádkem, v němž došlo k chybě. Abychom byli zcela přesní, musíme připomenout, že jazyk Visual Basic 6.0 dovoloval softwarovým tvůrcům správu chyb zcela vypnout. Na tento účel sloužil příkaz `On Error 0`, ovšem jeho použití bylo v mnoha případech přinejmenším diskutabilní. V prostředí Visual Basicu .NET je správa chyb daleko propracovanější, neboť zde mohou programátoři aplikovat takzvanou strukturovanou správu chyb, která je založena na příkazu `Try-Catch-Finally`. Strukturovaná správa chyb pracuje podle následujícího algoritmu:

a) Veškerý potenciálně nebezpečný programový kód, jehož zpracování by mohlo vyústit do generování chybové výjimky, je umístěn do bloku `Try`.

b) Dojde-li během exekuce kódu v bloku `Try` ke generování chybové výjimky, tato je zachycena a náležitě ošetřena v bloku `Catch`. Blok `Catch` může být přitom naprogramován tak, aby mohl analyzovat a zpracovávat všechny generované chybové výjimky, nebo jenom specifické chybové výjimky.

c) Poté, co blok `Catch` dokončí svoji činnost, se strukturovaná správa chyb přesouvá do bloku `Finally`. V tomto bloku je umístěn kód, který bude vykonán vždy, tedy bez ohledu na to, zdali bude chybová výjimka generována či nikoliv.

Strukturovaná správa chyb jazyka Visual Basic .NET je efektivnějším řešením, neboť nabízí přehlednější syntaxi a dovoluje vám produktivněji přistupovat k zachycování a následnému ošetřování chybových výjimek. Softwarová aplikace se tudíž dokáže lépe zotavit z následků, které vznikly při neočekávaných situacích. Výhodou strukturované správy chyb je její interoperabilita napříč jinými .NET-kompatibilními programovací jazyky, jako je C# a C++ s Managed Extensions.

3. **Jazyková a platformová interoperabilita.** Vývojově-exekuční platforma Microsoft .NET Framework je položena na pevných základech, které avizují bohatou podporu pro jazykovou i platformovou softwarovou interoperabilitu. Jazyková interoperabilita znamená, že aplikace .NET napsané v různých programovacích jazycích mohou spolupracovat bez jakýchkoliv potíží, které by jim bránily ve vzájemné komunikaci. Když si uvědomíte praktické dosahy jazykové interoperability, velice rychle přijdete na to, že jde o vskutku nevídanou a báječnou vlastnost platformy .NET Framework. Představte si následující imaginární situaci: V jazyce Visual Basic .NET vytvoříte vlastní třídu, jejíž instance budou působit jako grafická tlačítka. Když vytvoříte instance své třídy ve Visual Basicu .NET a tuto pak přidáte na formulář aplikace, získáte tlačítko s bohatou grafickou výbavou, jehož vizuální ztvárnění bude oku lahodící. Nyní si představte, že vaše třída bude tak skvělá, že ji bude chtít použít také váš kolega, který programuje v jazyce C++ s Managed Extensions. V dávných dobách byla opětovní použitelnost programového kódu napsaného v jedné aplikaci omezena právě na tuto aplikaci, nebo jiné aplikace připravené pomocí stejného vývojářského nástroje. Jazyková interoperabilita na platformě .NET Framework však bourá všechna zažitá pravidla, mýty a pověry. Zde platí heslo „napiš jednou a použij všude". V praxi to znamená, že váš kolega pracující v C++ s Managed Extensions může do svého projektu nejenom okamžitě začlenit vaši třídu s grafickým tlačítkem, nýbrž je schopen od vaší třídy odvodit novou třídu a tu rozšířit o nové dovednosti a funkcionální prvky. Podobný scénář, jaký jsme si uvedli mezi jazyky Visual Basic .NET a C++ s Managed Extensions, může být uplatněn mezi libovolnými programovacími nástroji, které vyhovují standardům platformy .NET Framework. Pomocí jazykové interoperability mohou vývojáři sestavovat pokročilá

komponentová řešení, která budou pozůstávat z komponent připravených v různých vývojových prostředích (Visual Basic .NET, C#, J#, C++ s Managed Extensions).

Vedle pečlivé podpory jazykové interoperability si platforma .NET Framework rozumí také s platformovou interoperabilitou. Je tedy možné vzájemně propojovat softwarové programové jednotky, které používají pro svůj běh rozdílné technologie. Vezměme si kupříkladu technologie COM a COM+ nebo sadu nativních funkcí aplikačního programového rozhraní operačních systémů řady Windows (Win32 API). Na světě existují tisíce a tisíce aplikací, které byly zkonstruovány před uvedením platformy .NET Framework. Je přitom zcela nepodstatné, zdali mluvíme o ryzích Win32 aplikacích napsaných v C/C++, nebo o COM komponentách připravených ve Visual Basicu 6.0. Tyto softwarové jednotky jsou plně funkční, a proto je důležité, aby s nimi mohly spolupracovat také nové aplikace .NET, které jsou poháněny platformou .NET Framework. Naštěstí, vývojáři v Microsoftu tuto skutečnost při navrhování architektury .NET Frameworku brali v potaz. Moderní aplikace, které byly napsány v jazyce Visual Basic .NET, ale obecně řečeno v jakémkoliv .NET-kompatibilním vývojovém produktu, mohou úzce komunikovat jednak s COM komponenty, Win32 dynamicky linkovanými knihovnami, a také samotnými nativními funkcemi aplikačního programového rozhraní. Požadavek na bezporuchovou platformovou interoperabilitu byl zcela pochopitelný: Jelikož bylo prostřednictvím starších softwarových technologií připraveno množství užitečného programového kódu, byla by velká škoda jej opětovně nevyužít. Platforma .NET Framework si byla tohoto požadavku vědoma, a proto nabídla vývojářům technologie jako COM Interop nebo Platforme Invoke (P/Invoke), jejichž pomocí mohou i nadále využívat pozitivních vlastností jednou napsaného programového kódu také v prostředí moderních aplikací .NET.

Vývojově-exekuční platforma .NET Framework společně se sadou nástrojů Visual Studio .NET nabídla vývojářům hodnotné pomůcky pro vývoj širokého spektra aplikací, přičemž velký důraz se kladl především na webové aplikace a webové služby pracující na bázi standardu XML. Příklon k vývoji aplikací .NET, jež mohou běžet v síťovém prostředí Internetu nebo intranetu byl zcela srozumitelný, neboť potřeba přistupovat k datům prostřednictvím síťových rozhraní se pro mnoho programátorů stala takřka nutností. Strategie Microsoft .NET s tímto fenoménem nejenomže počítala, ona byla na něm dokonce kompletně založena. Hlavním záměrem byl okamžitý přístup k požadovaným datům, a to kdekoliv, kdykoliv a z jakéhokoliv počítačového zařízení. Při vývoji webových aplikací a XML webových služeb programátorům velice dobře posloužila technologie ASP.NET, která nahradila technologii ASP (Active Server Pages) z předcházející verze Visual Studia. Za asistence ASP.NET mohli vývojáři vytvářet své webové aplikace s lehkostí, která byla srovnatelná s vývojem standardních aplikací pro systém Windows. Výborné bylo, že programátoři mohli při tvorbě ASP.NET webových stránek upotřebit vší sílu jazyka Visual Basic .NET (nebo Visual C# .NET), a nebyli tudíž omezování použitím skriptovacích jazyků typu VBScript nebo JScript.

Koncem dubna roku 2003 se na softwarový trh dostává inovovaná verze platformy .NET Framework s pořadovým označením 1.1, která připravuje půdu pro aktualizovanou kolekci vývojářských nástrojů Visual Studio .NET 2003. Sada Visual Studio .NET 2003 obsahovala tyto softwarové produkty: Visual Basic .NET 2003, Visual C# .NET 2003, Visual C++ .NET 2003 a nově také Visual J# .NET 2003, což je vývojářský nástroj pro programátory pracující v Javě. Odhlédneme-li od parciálních inovací integrovaného vývojového prostředí Visual Studia .NET, můžeme prohlásit, že zcela největší inovací byla přímá implementace vývoje aplikací .NET pro inteligentní zařízení jako jsou „chytré“ mobilní telefony a PDA pracující s operačním systémem Windows Mobile for Pocket PC. Softwaroví odborníci používající jazyky Visual Basic .NET 2003 a Visual C# .NET 2003 mohli začít ihned vytvářet svá řešení pro mobilní počítačová zařízení. V tomto směru jim pomocnou ruku poskytla vývojově-exekuční platforma .NET Compact Framework, kterou si můžete představit jako mladšího brášku „velikého“ .NET Frameworku. Všechny softwarové služby, které nabízela řízeným aplikacím platforma .NET Framework, pak byla

schopna nabídnout mobilním aplikacím běžícím na inteligentních zařízeních i platforma .NET Compact Framework. Nejlepší ovšem bylo, že vývojář vůbec nemusel měnit své zažité pracovní postupy a návyky: stále totiž pracoval v jazyce Visual Basic .NET 2003, byť na jiném aplikačním projektu. Vylepšena byla také technologie ASP.NET (nyní ve verzi 1.1), která přinesla podporu vývoje webových aplikací i pro zařízení třídy Pocket PC.

Ze syntaktické stránky dostal jazyk Visual Basic .NET 2003 do vínka dvě užitečné programové konstrukce: operátory bitového posunu a možnost explicitní deklarace řídící proměnné uvnitř cyklů `For...Next` a `For Each...Next`.

Nyní, v roce 2005, se nacházíme v prozatím poslední evoluční fázi jazyka Visual Basic. Je to proto, že společnost Microsoft připravila novou a opět radikálně inovovanou verzi tohoto programovacího nástroje, který si oblíbili houfy vývojářů, programátorů a IT odborníků na celém světě. Nový Visual Basic nese označení 2005, a v této souvislosti musíme poukázat na skutečnost, že jde o jednoznačně nejpropracovanější Visual Basic, jaký byl kdy vůbec stvořen. Společně s jazykem Visual Basic 2005 se k vývojářům dostává také nová verze vývojově-exekuční platformy .NET Framework, jejíž číselní označení je 2.0. Svého uvedení se dočkala také nová verze technologie ASP.NET 2.0, která nabízí vývojářům ještě větší možnosti a vyšší pracovní produktivitu při návrhu a vývoji webových aplikací a XML webových služeb. Abychom nezapomněli, společnost Microsoft uvádí rovněž nové verze svých dalších vývojářských produktů, mezi něž patří Visual C# 2005, Visual C++ 2005 a Visual J# 2005. Všechny vývojářské produkty jsou pak sdruženy v jednotné sadě Visual Studio 2005, která se na trh dostává v několika vyhotoveních, jež jsou zacílena na specifické cílové segmenty softwarových specialistů.

POZNÁMKA: Novinka v portfoliu vývojářských nástrojů: Visual Basic 2005 Express

V portfoliu softwarových produktů společnosti Microsoft pro vývoj aplikací .NET běžících na platformě .NET Framework 2.0, se objevuje nová produktová řada, která je tvořena nástroji řady Express. Každý vývojářský produkt ze sady Visual Studio 2005 disponuje svým Express protějškem. Nástroje řady Express nejsou primárně určeny pro profesionální využití, své uplatnění však naleznou u studentů informačních technologií, hobby vývojářů a fanoušků vývoje softwaru. Visual Basic 2005 Express je tedy jakousi „odlehčenou" verzí produktu Visual Basic 2005. Na jedné straně obsahuje Visual Basic 2005 Express plnou podporu nové jazykové specifikace Visual Basicu, ovšem na straně druhé je v jistých aspektech limitován (jde především o nemožnost vývoje jistých typů aplikací, menší možnosti při ladění kódu, či absenci pokročilých softwarových pomůcek, které mohou vývojáři ve Visual Basicu 2005 použít). Jednoznačnou výhodou Visual Basicu 2005 Express je jeho cena, která je ve srovnání s plnohodnotným Visual Basicem 2005 daleko nižší. Pokud byste se o produktu Visual Basic 2005 Express chtěli dozvědět podrobnější informace, nalistujte šestou kapitolu této publikace.

Visual Basic ve verzi 2005 je opravdovým progresivním skokem a nejedná se tak o pouhý evoluční produkt. Ve skutečnosti mají mnohé inovace a modifikace Visual Basicu 2005 charakter inovací vyššího řádu. Tato skutečnost je citelná zejména v okamžiku, kdy se poohlédnete po nově implementovaných vylepšeních: generické typy, přetěžování operátorů, explicitní dealokace systémových zdrojů pomocí příkazu `Using`, zavedení operátoru `IsNot`, příkaz `Continue` pro nastartování nové iterace cyklů, možnost definovat vlastnosti s variabilními přístupovými právy, použití neznaménkových datových typů pro lepší interoperabilitu s funkcemi Win32 API, separace programového kódu pomocí parciálních typů a vytváření uživatelsky definovaných událostí – to vše jsou nové a vzrušující programové elementy, které byly pečlivě zakomponovány do jazykové specifikace Visual Basicu 2005. Ano, jazyk Visual Basic 2005 je z pohledu syntaktických inovací bezpochyby nejsilnějším ze všech doposud vydaných Visual Basiců. Kdybychom se ale soustředili pouze na syntax, odhalili bychom jenom půlku tajemství nového Visual Basicu, protože radikálním „faceliftingem" prošlo taktéž integrované vývojové prostředí Visual Studia 2005. Podobně jako u minulých verzí Visual Studia .NET, také verze 2005 disponuje

integrovaným vývojovým prostředím, které je sdíleno všemi .NET-kompatibilními vývojářskými nástroji společnosti Microsoft. IDE je obdařeno dávkou umělé inteligence a zcela automaticky se přizpůsobuje vašim požadavkům a pracovním návykům. Dynamická nápověda nepřestajně sleduje vaše kroky a pohotově vám předkládá témata s informacemi, které právě potřebujete. Vylepšená technologie IntelliSense je schopna vám na požádání poskytnout seznam všech dostupných programových elementů jako jsou objekty či jejich vlastnosti, metody nebo události. IntelliSense je dokonce tak chytrá, že dokáže analyzovat nejběžněji používané prvky a nabízet vám je v samostatném seznamu. Vylepšení se dočkal také editor pro zápis zdrojového kódu, jenž nově zavádí podporu pro inteligentní značky (smart tags) a šablony kódu (code snippets). Jestliže jste pracovali v prostředí Microsoft Office XP nebo Microsoft Office System 2003, je vám přínos inteligentních značek jistě dobře známý. Ve Visual Basicu 2005 vám inteligentní značky pomohou zejména tehdy, kdy se při zápisu příkazu, klíčového slova nebo identifikátoru spletete a uvedete jeho chybné znění. V tuto chvíli se aktivuje inteligentní značka, která podezřelé místo okamžitě odhalí a nabídne vám možnosti pro nápravu zjištěné chyby. Inteligentní značky odvádějí při analýze a následní opravě programového kódu skutečně znamenitou práci, neboť pomocí nich jste schopni vhodně upravit i poměrně složité syntaktické konstrukce. Šablony kódu reprezentují předem navržené a připravené segmenty zdrojového kódu, které můžete okamžitě vkládat do svých aplikací.

Tím však výčet novinek integrovaného vývojového prostředí Visual Basicu 2005 ani zdaleka nekončí. Softwarové vývojáře čeká spousta dalších užitečných nástrojů, s nimiž bude vytváření aplikací .NET rychlejší, snadnější a produktivnější. Pokud byste se o IDE Visual Basicu 2005 rádi dozvěděli další informace, přečtěte si druhou kapitolu této vývojářské příručky.

UPOZORNĚNÍ: V této příručce je používán softwarový produkt Visual Studio 2005 Beta 2

Tato vývojářská příručka byla napsána použitím softwarového produktu Microsoft Visual Studio 2005 Beta 2. V době, kdy vznikaly jednotlivé stránky této publikace, byla programátorům, vývojářům a IT specialistům k dispozici teprve druhá beta verze balíku Visual Studio 2005. Všechny postupy demonstrované v této příručce, a také všechny snímky obrazovek proto pocházejí z Beta 2 verze. V ostré verzi Visual Studia 2005 mohou být některé prvky lehce pozměněny, což je důsledek stálého vývoje tohoto softwarového produktu. Ačkoliv by se nemělo jednak o nic zásadního, bude dobré, když budete mít tuto skutečnost na paměti.

Kapitola 2.: Směr Visual Basic 2005 – Instalace, konfigurace a integrované vývojové prostředí (IDE)

Dobrou zprávou pro vývojáře ve Visual Basicu 6.0 je, že Visual Basic 2005, jenž je součástí vývojářského kompletu Microsoft Visual Studio 2005, lze bez jakýchkoliv potíží nainstalovat na počítač, který již obsahuje instalaci Visual Basicu 6.0, resp. Visual Studia 6.0. Na svém počítači tedy můžete provést paralelní instalaci produktů Visual Basic 6.0 a Visual Basic 2005, aniž byste jakkoliv narušili funkčnost svých stávajících aplikací, nebo nějakým způsobem omezili nové aplikace .NET, které připravíte pomocí Visual Basicu 2005. Samozřejmě, paralelní instalace produktu Visual Basic 2005 je pro programátory migrující z Visual Basicu 6.0 nejlepším řešením, a to přinejmenším z níže uvedených důvodů:

1. Přítomnost vývojářského nástroje Visual Basic 6.0 na pracovním počítači je nepostradatelná, protože je velice pravděpodobné, že při inovování svých existujících aplikací budete muset provést v jejich syntaktické struktuře jisté změny. Visual Basic 2005 sice obsahuje poměrně mocného pomocníka pro inovaci s názvem **Visual Basic Upgrade Wizard**, ovšem v procesu migrace zejména sofistikovanějších aplikací se mohou vyskytnout potíže, které vyplývají jednak ze změny syntaxe, a také z jisté míry nekompatibility jazyka Visual Basic 2005 s jazykem Visual Basic 6.0. Když si na svém PC ponecháte instalaci produktu Visual Basic 6.0, budete moci nezbytné úpravy svých stávajících aplikací ihned provést. Jednoduše řečeno, získáte větší kontrolu a možnost okamžité zpětné vazby.

2. Pokud se rozhodnete přejít na nový Visual Basic 2005, bude vhodné, když budete mít Visual Basic 6.0 stále po ruce. Můžete tak lépe pochopit rozdíly, které nová verze jazyka přináší. Vzájemným porovnáváním integrovaných vývojových prostředí obou vývojářských produktů můžete rychleji odhalit ty aspekty, které jsou v nové verzi upraveny, modifikovány či zcela změněny. Tento způsob práce se někdy označuje jako „učení praxí" a je vhodný především pro ty programátory a vývojáře, kteří se raději učí na skutečných příkladech a postupech než na pouhé teorii.

3. Ačkoliv se tvůrci softwarového produktu Visual Basic 2005 snažili zachovat co možná největší míru zpětné kompatibility, je nutno na rovinu říci, že v mnoha podstatných aspektech programování je jazyk Visual Basic 2005 a potažmo také vývojově-exekuční platforma Microsoft .NET Framework 2.0 velice odlišná od původního pracovního schématu jazyka Visual Basicu 6.0. Co to pro vývojáře pracující v jazyce Visual Basic 6.0 znamená? Inu to, že některé aplikace nebo jejich části nebudou moci být implicitně převedeny do prostředí Visual Basicu 2005. Dostanete-li se do takovéto situace, je jenom na vás, jakou postupovou cestu zvolíte. Pokud nehodláte věnovat svůj čas a prostředky „manuální" konverzi stávající aplikace, nejlépe uděláte, když ponecháte danou aplikaci ve Visual Basicu 6.0. Abyste ji mohli nadále spravovat, je dobré mít tento nástroj na počítači nainstalován. Na druhé straně, budete-li chtít aplikaci jazyka Visual Basic 6.0 převést takříkajíc „za každou cenu", můžete přikročit k explicitní migraci aplikace do prostředí Visual Basicu 2005. Také v tomto případě se však přítomnost produktu Visual Basic 6.0 vyplatí: můžete pohotově provádět nezbytné modifikace ve zdrojovém kódu a ujistit se, že aplikace funguje tak, jak má.

TIP: Paralelní instalace více verzí produktu Visual Studio

Na vývojářském počítači mohou být paralelně nainstalovány následující softwarové produkty z dílny společnosti Microsoft:

1. Visual Studio 6.0 + Visual Studio .NET (2002)
2. Visual Studio 6.0 + Visual Studio .NET (2002) + Visual Studio .NET 2003
3. Visual Studio 6.0 + Visual Studio .NET (2002) + Visual Studio .NET 2003 + Visual Studio 2005

Visual Studio 6.0 může společně koexistovat se všemi novějšími verzemi Visual Studia (.NET, .NET 2003 a 2005). Soustředíme-li se na Visual Basic 6.0, můžeme říci, že programátoři mohou na jednom počítači paralelně vyvíjet jednak aplikace prostřednictvím tohoto jazyka, a jednak aplikace využívající výhod jazyků Visual Basic .NET (2002), Visual Basic .NET 2003 a Visual Basic 2005. Vzhledem k zaměření této příručky se budeme pochopitelně zabývat pouze vzájemným vztahem aplikací jazyků Visual Basic 6.0 a Visual Basic 2005. Nakoukneme-li pod roušku, zjistíme, že obě zmíněná prostředí pracují podle zásadně odlišného paradigmatu. Zatímco aplikace jazyka Visual Basic 6.0 využívají knihovny běhového prostředí tohoto jazyka, které jsou pro ně životně důležité, aplikace .NET připravené v jazyce Visual Basic 2005 jsou závislé na přítomnosti funkčních elementů vývojového rámce .NET Framework 2.0. Aplikace Visual Basicu 6.0 a řízené aplikace Visual Basicu 2005 tak využívají služeb naprosto samostatných a nezávislých vývojových prostředí, a proto není jejich paralelní vývoj ani exekuce nijak omezena.

POZNÁMKA: Koexistence produktů Visual Basic 6.0 a Visual Basic 2005 Express

Jak jsme si již řekli, novinkou v produktovém portfoliu společnosti Microsoft jsou vývojářské nástroje řady Express, které jsou zacíleny na fanoušky programování, studenty a hobby vývojáře. Jelikož je možné, že někteří vývojáři ve Visual Basicu 6.0 se budou chtít seznámit také s Express verzí produktu Visual Basic 2005, měli bychom si povědět, jaká je vzájemná interakce těchto produktů. Abychom vás dlouho nenapínali: Visual Basic 2005 Express můžete bez jakýchkoliv potíží nainstalovat na PC s Visual Basicem 6.0. Tyto nástroje si společně velmi dobře rozumí, podobně jako tomu bylo u produktu Visual Basic 2005. Visual Basic 2005 Express se však od svého „většího brášky" liší v mnoha oblastech. Jestliže byste se o rozdílech mezi vývojářskými nástroji Visual Basic 2005 Express a Visual Basic 2005 chtěli dovědět více, vše potřebné naleznete v šesté kapitole této publikace.

2.1 Hardwarové a softwarové požadavky pro instalaci produktu Visual Basic 2005

Co naplat, Visual Basic 2005, jenž je součástí rozsáhlé sady Visual Studio 2005, je moderním vývojářským nástrojem pro vytváření vyspělých aplikací 21. století, a tomu samozřejmě odpovídají i požadavky na hardwarové a softwarové vybavení hostitelského počítače. Podle oficiálních informací společnosti Microsoft budete pro instalaci a běh produktu Visual Studio 2005 potřebovat počítač s následující hardwarovou (tab. 2.1) a softwarovou (tab. 2.2) konfigurací.

Tab. 2.1: Hardwarové požadavky pro instalaci produktu Visual Studio 2005

Hardwarový komponent	Požadavek
Typ a taktovací frekvence procesoru (CPU)	Minimum: 766megahertzový procesor řady Intel Pentium, Intel Itanium, AMD Athlon nebo AMD Opteron Doporučeno: 1,5gigahertzový procesor řady Intel Pentium, Intel Itanium, AMD Athlon nebo AMD Opteron

Operační paměť (RAM)	Minimum: 256 MB Doporučeno: 512 MB
Kapacita pevného disku (HDD)	Bez instalace elektronické dokumentace MSDN: • 1 GB prostoru na systémovém diskovém oddílu, • 2,5 GB prostoru na instalačním diskovém oddílu. S instalací elektronické dokumentace MSDN: • 1,5 GB prostoru na systémovém diskovém oddílu, • 4,5 GB prostoru na instalačním diskovém oddílu.
Optická mechanika	Požadována je CD nebo DVD optická mechanika
Obrazovkové rozlišení zobrazovacího zařízení	Minimum: Monitor s rozlišením 800×600 pixelů a 8bitovou barevnou hloubkou (256 barev) Doporučeno: Monitor s rozlišením 1024×768 pixelů a 16bitovou barevnou hloubkou (65536 barev)
Polohovací zařízení	Požadována je klávesnice a myš

Tab. 2.2: Softwarové požadavky pro instalaci produktu Visual Studio 2005

Softwarový komponent	Požadavek
Operační systém	Kolekce vývojářských produktů Visual Studio 2005 může být nainstalována na těchto operačních systémech: • Microsoft Windows 2003 Server, • Microsoft Windows XP Professional, • Microsoft Windows XP Home Edition (zde není podporován vývoj lokálních webových aplikací), • Microsoft Windows 2000. Na 64bitových počítačích jsou požadovány tyto operační systémy: • Microsoft Windows Server 2003 X64 Edition (verze sestavení 1184 nebo aktuálnější), • Microsoft Windows XP Professional X64 Edition (verze sestavení 1184 nebo aktuálnější).
Softwarové součásti pro vývoj, ladění, testování a distribuci aplikací .NET pro inteligentní mobilní zařízení	• Microsoft ActiveSync 3.7, • Microsoft .NET Framework 1.1.

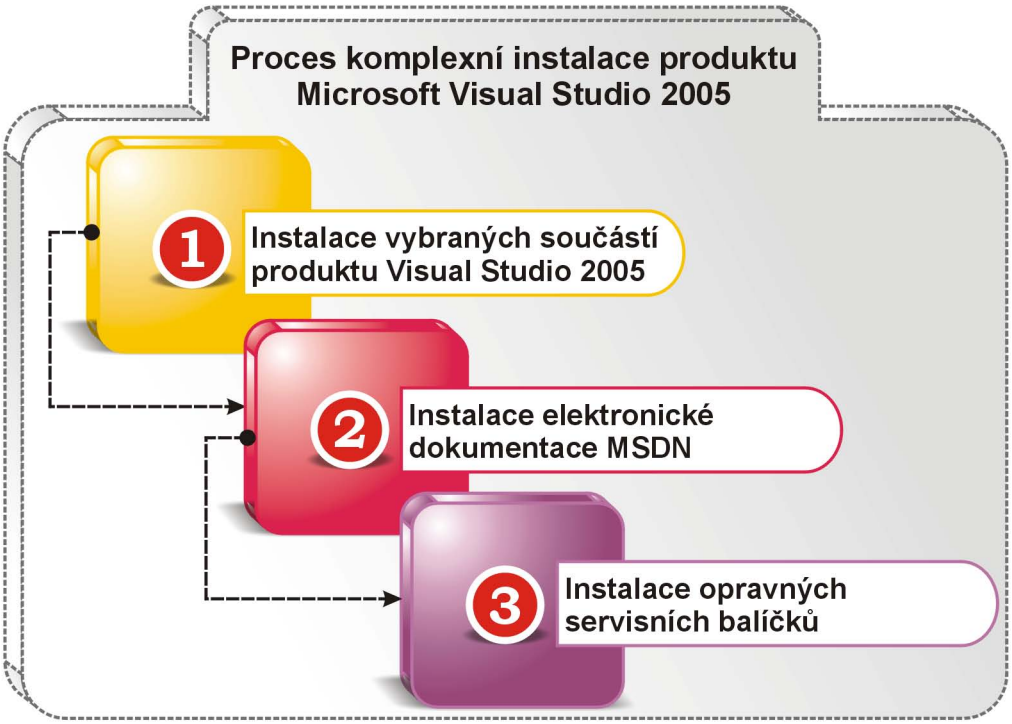
Uváděné hardwarové a softwarové předpoklady byste měli v zásadě brát jako nejnižší možný standard, který musíte splnit, abyste mohli na svém počítači vyvíjet aplikace ve Visual Basicu 2005. Co se týče hardwaru, v každém případě byste měli mít k dispozici rychlý procesor (nemusí být nutně s podporou 64 bitů), dostatek kapacity operační paměti (nejlépe 512 MB a více) a adekvátně velký pevný disk podle vašich potřeb. Kapacita pevného disku je důležitá zejména proto, abyste byli schopní instalovat také veškerou elektronickou dokumentaci k produktu Visual Basic 2005 (a když budete chtít, také k dalším nástrojům sady Visual Studio 2005), neboť je to právě doprovodná softwarová dokumentace, která obsahuje spoustu relevantních informací technického charakteru. Tyto informace mají pro vývojáře cenu zlata, a proto by byla veliká škoda je nevyužít.

Výčet softwarových požadavků názorně ukazuje, že produkt Visual Basic 2005 není možné instalovat na počítač se staršími operačními systémy řady Microsoft Windows 95, 98 a Me. Jako nejnižší základ pro vývoj aplikací .NET, pracujících na báze platformy .NET Framework 2.0, je nutná přítomnost instalace operačního systému Microsoft Windows 2000 s nejaktuálnějším servisním balíčkem (SP). Visual Basic 2005 může být rovněž nainstalován na počítačové stanice, na nichž běží operační systémy třídy Microsoft Windows XP a Microsoft Windows 2003 Server. I zde se doporučuje aplikovat nejnovější aktualizace a servisní balíčky.

2.2 Instalace produktu Microsoft Visual Studio 2005

Za správnou instalaci produktu Visual Studio 2005 je odpovědný softwarový instalátor, který uskutečňuje komplexní instalaci ve třech fázích, jimiž jsou:

1. Instalace vybraných součástí produktu Visual Studio 2005
2. Instalace elektronické dokumentace Microsoft Developer Network (MSDN)
3. Instalace opravných servisních balíčků

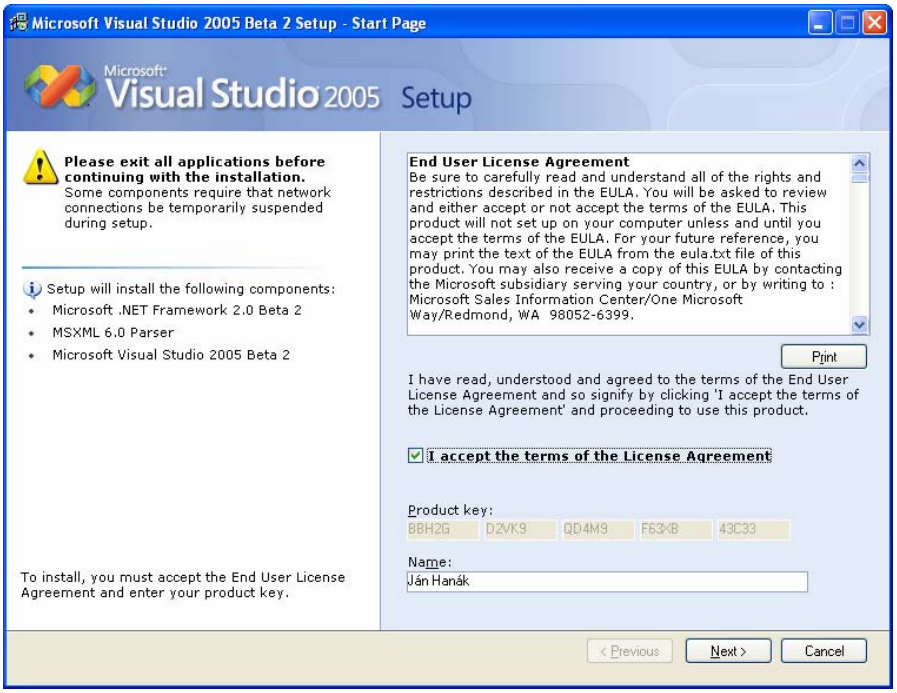


Obr. 2.1: Proces komplexní instalace produktu Microsoft Visual Studio 2005

Instalace Visual Studia 2005 bude spuštěna automaticky po vložení příslušného CD nebo DVD disku do vaší optické mechaniky. Pokud nemáte aktivované automatické oznamování, můžete instalaci zahájit manuálním spuštěním instalačního souboru *setup.exe* ze složky **vs**. Instalátor zobrazí informační dialog, který vás bude provázet celým procesem instalace. Zprvu je aktivní pouze první volba **Install Visual Studio 2005 Beta 2** (obr. 2.2).

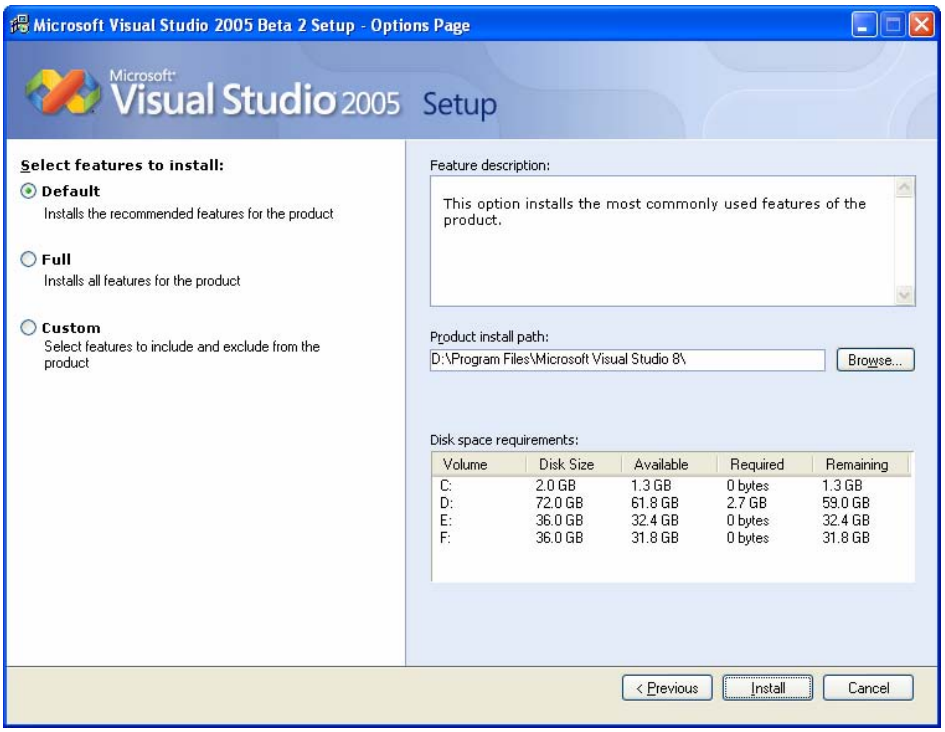
Obr. 2.2: Začátek instalačního procesu

Když na tuto volbu klepnete, instalátor načte instalační komponenty a provede analýzu softwarového prostředí vašeho počítače. Následně vám oznámí, které součástí musí být nevyhnutně nainstalovány a zeptá se vás, zda souhlasíte s licenčními podmínkami (obr. 2.3).



Obr. 2.3: Seznam základních komponent určených k instalaci

Poté vás instalátor uvede do hlavní instalační nabídky, kde si můžete vybrat jeden ze tří instalačních profilů, které nesou názvy **Default**, **Full** a **Custom** (obr. 2.4).



Obr. 2.4: Výběr instalačního profilu

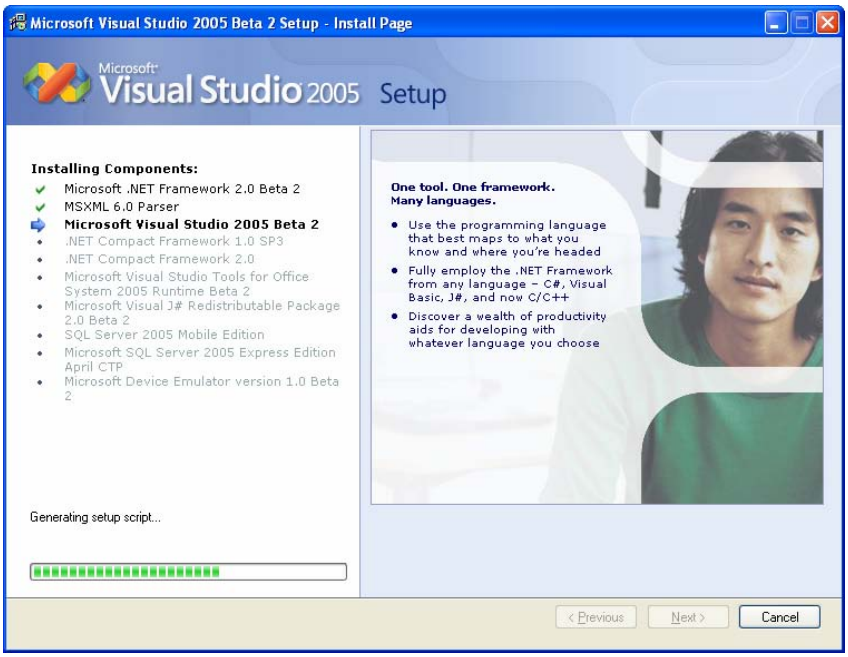
Většině uživatelů bude pravděpodobně zcela vyhovovat výchozí nastavení instalátoru, které garantuje volba **Default**. Ta do instalace zahrne všechny doporučené součásti, o kterých se předpokládá, že je budete při své práci potřebovat. Budete-li chtít nainstalovat softwarový produkt Visual Studio 2005 se všemi komponentami, nástroji a podpůrnými prvky, můžete zvolit instalační profil **Full**. Měli byste však pamatovat na to, že plná instalace je kapacitně nejnáročnější, a proto si z vašeho pevného disku odkrojí největší kousek. Mnoho vývojářů bude chtít možnosti instalace pečlivě konfigurovat. Je-li to také váš případ, vyberte volbu **Custom**. Pak budete moci z přehledné stromové struktury určit ty softwarové produkty a jejich součástí, o které máte zájem.
A naopak, z instalace můžete vyloučit ty komponenty, o nichž si myslíte, že je nebudete nezbytně potřebovat.

Chcete-li nainstalovat pouze Visual Basic 2005, můžete zvolit možnost **Custom** a následně ve skupině položek **Language Tools** ponechat zatrženou pouze položku **Visual Basic**. Zatržení dalších položek, jež reprezentují jiné programovací jazyky, můžete zrušit. Jestliže byste ovšem kromě Visual Basicu 2005 chtěli později vyzkoušet třeba také Visual C# 2005, můžete samozřejmě nainstalovat i tento produkt (zařazení produktu do seznamu produktů určených k instalaci provedete aktivací příslušné položky).

POZNÁMKA: Pozdější modifikace instalačních komponent Visual Studia 2005

Konfiguraci nainstalovaných součástí Visual Studia 2005 můžete kdykoliv upravit také poté, co jste tento produkt úspěšně nainstalovali na vaše PC. K tomuto účelu vám poslouží aplikace *Přidat nebo ubrat programy*, kterou spustíte z ovládacích panelů operačního systému Windows. Nejste-li proto pevně rozhodnuti, které komponenty Visual Studia 2005 byste měli, samozřejmě vyjma Visual Basicu 2005, instalovat, můžete toto rozhodnutí odložit až na později. Instalace produktu Visual Studio 2005 je plně konfigurovatelná, a proto můžete po nainstalování nástroje Visual Basic 2005 již provedenou instalaci v případě potřeby libovolně upravovat. To znamená, že můžete přidávat nové instalační součásti, upravovat ty stávající, nebo nepotřebné komponenty jednoduše odstranit a uvolnit tak místo na pevném disku.

Když má instalátor k dispozici dostatek informací, vygeneruje instalační skript a zahájí kopírování programových souborů pro specifické součásti Visual Studia 2005, které byly zvoleny uživatelem. Průběh instalace zachycuje obr. 2.5.

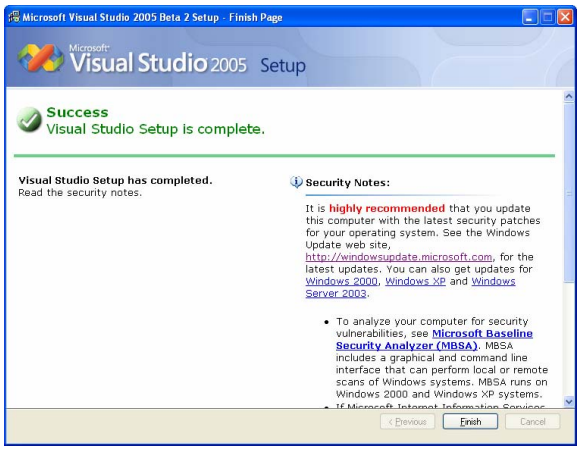


Obr. 2.5: Instalátor právě instaluje produkt Visual Studio 2005 Beta 2

Celková doba instalace Visual Studia 2005 je závislá na více faktorech, ovšem nejmarkantněji ji ovlivňuje především počet instalovaných softwarových komponent, rychlost procesoru, přenosová rychlost pevného disku i optické mechaniky, z níž je instalace prováděna.

TIP: Urychlení instalace produktu Visual Studio 2005	
	Pokud máte optickou mechaniku staršího data, která se vyznačuje nízkou přenosovou rychlostí, můžete průběh instalace Visual Studia 2005 urychlit tak, že obsah instalačních médií (CD nebo DVD) zkopírujete na váš pevný disk a instalaci budete provádět z pevného disku. Jelikož je přenosová rychlost pevného disku několikrát větší než přenosová rychlost optické mechaniky, měli byste zaznamenat citelné zvýšení rychlosti instalace jednotlivých součástí produktu Visual Studio 2005.

Když jsou všechny zvolené součásti Visual Studia 2005 nainstalovány, instalátor zobrazí finální zprávu, čímž je první etapa instalace ukončena (obr. 2.6).



Obr. 2.6: Finální zpráva instalátoru po ukončení instalace zvolených součástí produktu Visual Studio 2005

Instalace se v následujících okamžicích dostává do své druhé fáze, ve které dochází k instalaci elektronické dokumentace Microsoft Developer Network (MSDN). Ačkoliv je instalace elektronické MSDN fakultativní, vřele vám doporučuji ji provést. MSDN je totiž velice cenným zdrojem informací nejenom pro vývojáře a programátory, ale také pro softwarové analytiky a jiné IT specialisty. Při realizaci instalace elektronické dokumentace si můžete vybrat z několika instalačních alternativ, přičemž jako nejvhodnější se jeví rámcová instalace, při níž instalátor zjistí již nainstalované součásti Visual Studia 2005 a uskuteční instalaci dokumentace pouze zjištěných nástrojů. Disponujete-li dostatečným diskovým prostorem, můžete dát přednost úplné instalaci, nebo naopak, pokud vám záleží na každém „bajtu“, vaším favoritem bude instalace pouze základních komponent. Průběh instalace elektronické dokumentace MSDN k produktu Visual Studio 2005 můžete vidět na obr. 2.7.

Obr. 2.7: Právě se instaluje elektronická dokumentace MSDN

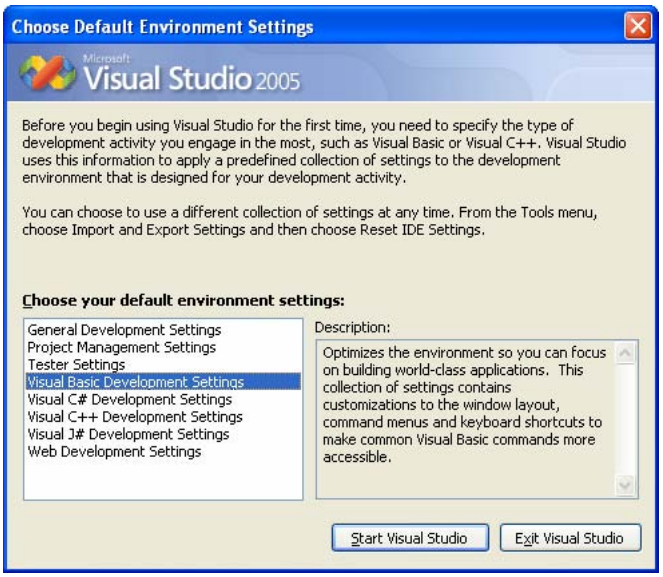
V poslední etapě komplexní instalace produktu Visual Studio 2005 vám instalátor nabídne možnost aktualizovat tento vývojářský komplet prostřednictvím sítě Internet. Provádění pravidelné aktualizace se všeobecně doporučuje, a proto byste se měli z času na čas podívat, zdali nejsou na webu k převzetí nové aktualizační balíčky.

Instalátor rovněž umístí do nabídky **Start** několik nových programových skupin:

1. Microsoft .NET Framework SDK v.2.0,
2. Microsoft Developer Network,
3. Microsoft SQL Server 2005 CTP,
4. Microsoft Visual SourceSafe,
5. Microsoft Visual Studio 2005 Beta 2.

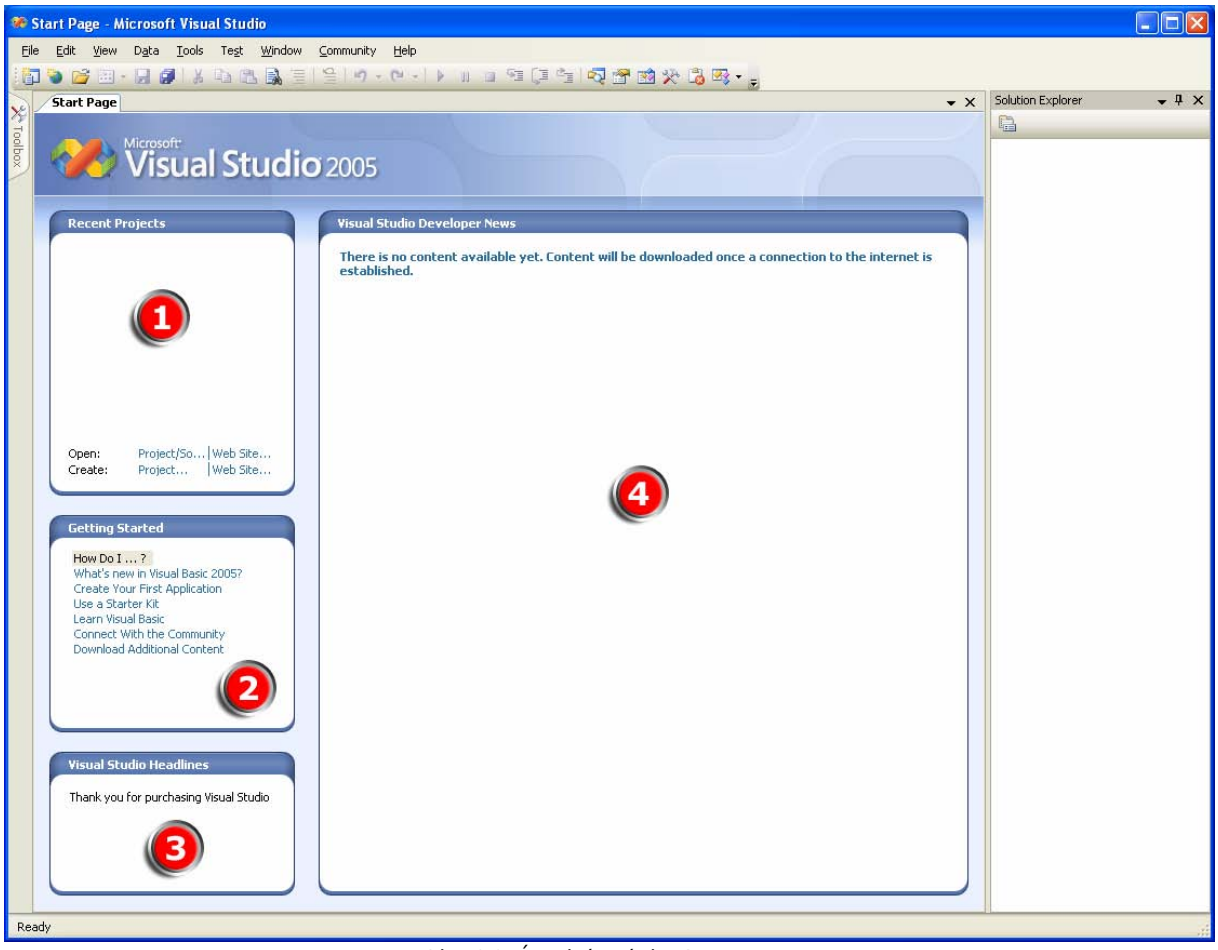
2.3 První spuštění a základní konfigurace produktu Visual Studio 2005

Když spustíte Visual Studio 2005 prvně, bude zobrazeno dialogové okno **Choose Default Environment Settings**, které vám dovoluje vybrat jeden z předdefinovaných konfiguračních profilů integrovaného vývojového prostředí. Výběr kýženého profilu se odvíjí od cílového programovacího nástroje, jehož pomocí hodláte vytvářet své aplikace .NET. Jelikož naším favoritem je Visual Basic, zvolíme ze seznamu **Choose your default environment settings** položku **Visual Basic Development Settings** a klepneme na tlačítko **Start Visual Studio**. Tím Visual Studiu 2005 nařídíme, aby provedlo takovou konfiguraci svého integrovaného prostředí, které bude nejlépe vyhovovat programátorům v jazyce Visual Basic.



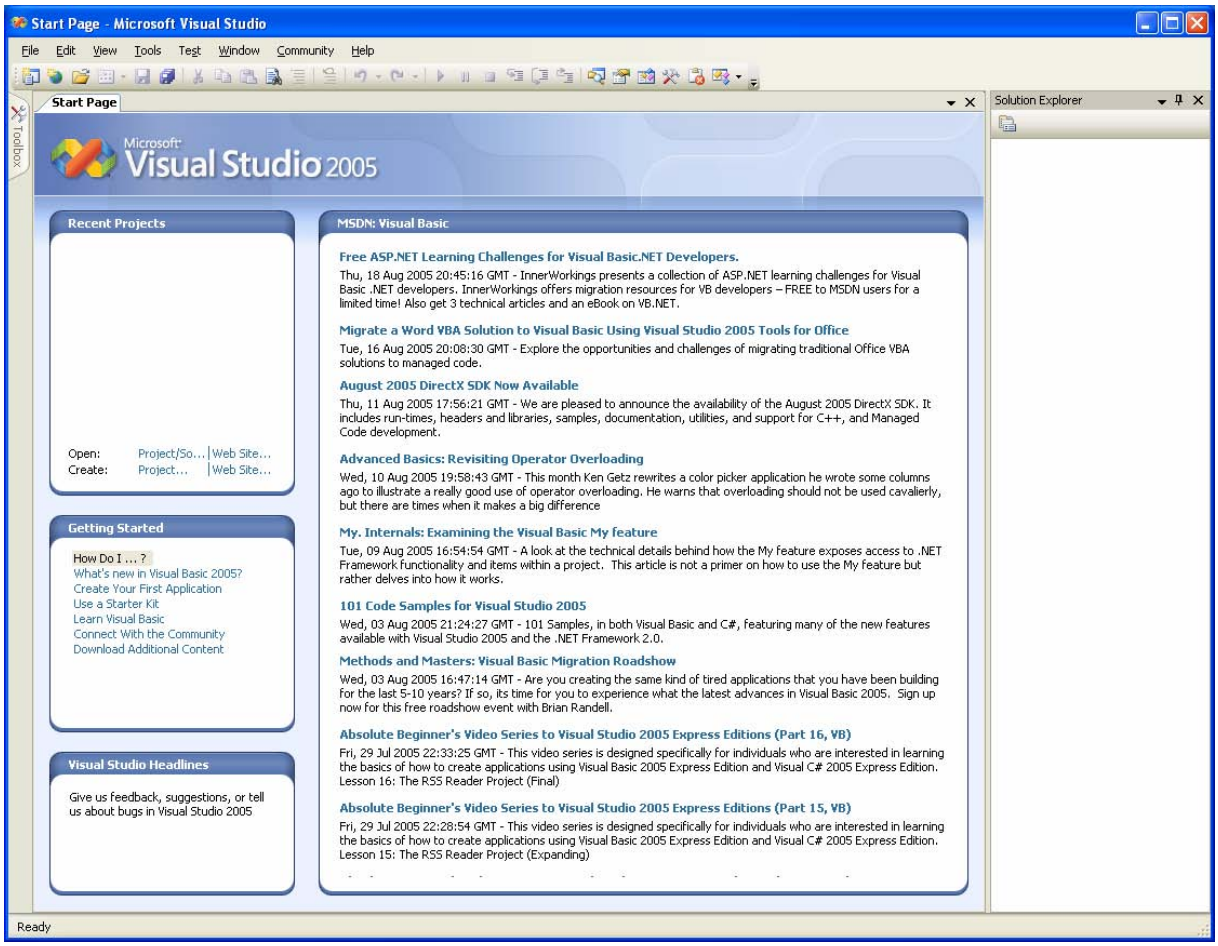
Obr. 2.8: Výběr konfiguračního profilu pro IDE Visual Studia 2005

Specifikovaný konfigurační profil si Visual Studio 2005 zapamatuje a bude vám ho automaticky nabízet také při budoucích spuštěních. Jakmile je profil zvolen, Visual Studio 2005 se nastartuje a uskuteční načtení všech hlavních součástí své pracovní plochy. Integrované vývojové prostředí Visual Studia 2005 je po startu implicitně prázdné, což znamená, že Visual Studio 2005 nevytvoří žádné nové aplikační řešení. Místo toho je zobrazena úvodní stránka **Start Page**, která nabízí rychlou cestu pro realizaci různých činností (obr. 2.9).



Obr. 2.9: Úvodní stránka **Start Page**

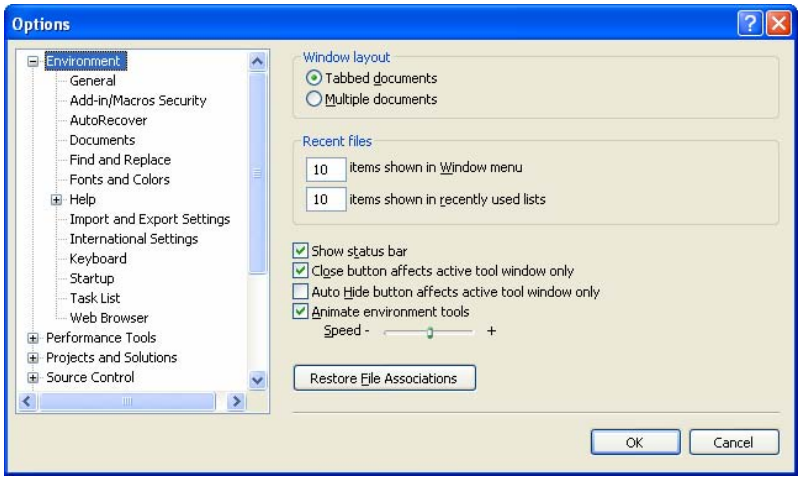
Stránka **Start Page** je složena ze čtyř podoken: **Recent Projects** (ikona č. 1), **Getting Started** (ikona č. 2), **Visual Studio Headlines** (ikona č. 3) a **Visual Studio Developer News** (ikona č. 4). Podokno **Recent Projects** plní tří úkoly: nabízí vám seznam posledně otevřených projektů a umožňuje vám rychle vytvářet nové projekty či otevírat jiné, již existující projekty. Pochopitelně, při úvodním spuštění Visual Studia 2005 je toto podokno prázdné, neboť jsme ještě žádný projekt nevytvořili a ani neotevřeli. Druhé podokno (**Getting Started**) sdružuje hypertextové odkazy, jejichž prostřednictvím můžete získat informace o programovacím jazyce Visual Basic 2005, spojit se dálkově s komunitou vývojářů nebo stáhnout doplňky ze sítě Internet. Visual Studio 2005 s vámi živě komunikuje, a to prostřednictvím zbývajících dvou podoken. Zatímco podokno **Visual Studio Headlines** je určeno pro zobrazování stručných zpráv a jako takové má spíše kompenzační charakter, enormní síla se ukrývá v podokně **Visual Studio Developer News**. Toto podokno představuje informační portál, jenž vám umožňuje získávat aktuální informace ze světa programování a vývoje softwaru pomocí Visual Studia 2005. Jste-li připojeni k Internetu, Visual Studio 2005 se spojí s dálkovým serverem společnosti Microsoft a stáhne informační záhlaví, jimiž vzápětí naplní podokno **Visual Studio Developer News** (obr. 2.10).



Obr. 2.10: Vždy čerstvé informace o programování v podokně **Visual Studio Developer News**

Pokud je k dispozici připojení k celosvětové síti, Visual Studio 2005 každou hodinu kontroluje, zda nebyl zveřejněn nový obsah ke stažení. Jestliže došlo na severu společnosti Microsoft k aktualizaci dat, Visual Studio 2005 tuto změnu diagnostikuje a nová data vám okamžitě nabídne k prohlédnutí. Tento styl komunikace je velice efektivní, přičemž se mu skutečně dobře daří simulovat inteligentní chování Visual Studia 2005.

Pracovní algoritmus Visual Studia 2005 však můžete ovlivnit prostřednictvím nastavení příslušných konfiguračních voleb, které naleznete v dialogovém okně **Options**. Okno **Options** zviditelníte následovně: Nejprve otevřete nabídku **Tools** a posléze klepněte na položku **Options**. Podobu dialogu **Options** můžete vidět na obr. 2.11.



Obr. 2.11: Dialogové okno **Options**

Okno **Options** vám umožňuje precizně nastavit všechny důležité konfigurační volby, které se pojí s pracovním stylem integrovaného vývojového prostředí Visual Studia 2005.
Můžete nastavit opravdu vše, na co si vzpomenete – rozvržením dialogových oken počínaje a editorem zdrojového kódu konče. Bohužel, není v našich silách, abychom se podrobně věnovali všem volbám, které můžete nastavit. Proto se budeme soustředit pouze na ty nejčastěji používané či vysloveně podstatné možnosti konfigurace.

Začneme stylem rozvržení oken dokumentů. Dokumenty, které v prostředí Visual Studia 2005 otevřete, jsou zobrazovány v integrovaném vývojovém prostředí, přičemž jejich názvy jsou automaticky přidávány do horizontálního pruhu záložek. Tento pracovní model je velice podobný způsobu, jakým upravujete své webové dokumenty v aplikaci Microsoft FrontPage 2003. Ve skutečnosti jde o užitečnou a flexibilní alternativu práce s dokumenty, protože dovoluje programátorům rychle se orientovat mezi všemi soubory, s nimiž právě pracují. Pokud byste ale raději přivítali rozvržení dokumentů ve stylu MDI (Multiple Document Interface), můžete v rámečku **Window Style** uzlu **Environment** vybrat volbu **Multiple documents**. Rozvržení MDI je vhodné zejména tehdy, potřebujete-li získat maximální možný prostor pro svou práci. Bohužel, v tomto rozvržení ztrácíte exaktní přehled o počtu právě otevřených dokumentů. Také nemůžete mezi jednotlivými dokumenty přecházet přímo prostřednictvím jednoho klepnutí myši – místo toho musíte použít klávesovou zkratku CTRL+TAB, nebo vybrat požadovaný dokument z nabídky **Window**.

Visual Studio 2005 přináší oproti Visual Studiu 6.0 podstatně inovovaný systém elektronické nápovědy. Revoluční dynamická nápověda (**Dynamic Help**) je nyní schopna sledovat kroky uživatele a pohotově se přizpůsobovat jeho pracovním návykům. Ačkoliv o dynamické nápovědě si budeme povídat v části vyhrazené bližšímu popisu integrovaného vývojového prostředí Visual Studia 2005, už teď byste měli vědět, že IDE může zobrazovat témata nápovědy ve dvou režimech: interním a externím. Ve výchozím nastavení je zvoleno interní zobrazování nápovědy (volba **Integrated Help Viewer** otevíracího seznamu **Show Help using**, který se zobrazí po klepnutí na uzel **Help**), což znamená, že témata budou zobrazována uvnitř IDE Visual Studia 2005. Na výběr máte také druhou možnost, a sice externí zobrazení témat nápovědy (položka **External Help Viewer**), kdy jsou témata zobrazována v samostatných „plovoucích" oknech.

Novinkou v oblasti nápovědy je možnost využívat vedle lokálních zdrojů informací také online nápovědu. Po klepnutí na uzel **Help** se vám zpřístupní další tří uzly (**General**, **Dynamic Help** a **Online**). Svou pozornost zaměřte na ten s názvem **Online**. V rámečku **When loading Help content** můžete určit, zda má Visual Studio 2005 nejprve prohledávat online nápovědu (volba **Try online first, then local**), nebo zda má upřednostnit lokální zdroje informací (volba **Try local first, then online**), anebo zda má vždy používat pouze nainstalovaná témata elektronické dokumentace (**Try local only, not online**).

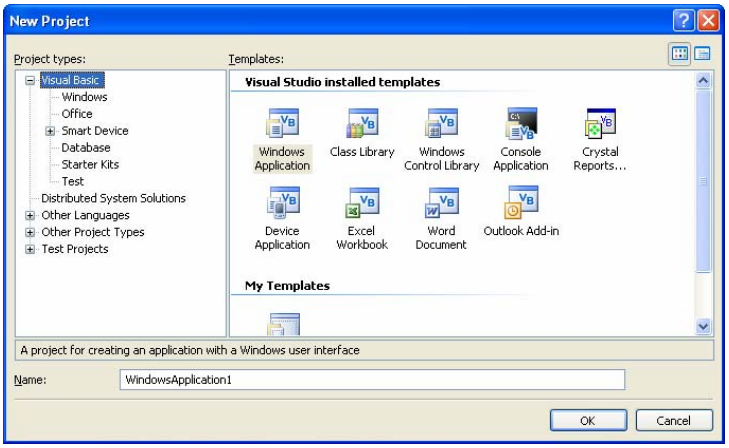
Další inovací integrovaného vývojového prostředí je jeho schopnost obnovit rozpracované dokumenty poté, co z nějakého vážného důvodu došlo k neočekávané chybě a zhroucení aplikace. Tato dovednost se jmenuje **AutoRecover** a konfigurovat ji můžete pomocí stejnojmenné položky, která se nachází v uzlu **Environment**. Aby mohlo Visual Studio 2005 obnovit dokumenty, musí disponovat korektními záchrannými daty. Tato data se podle „výrobního" nastavení ukládají každých pět minut (položka **Save AutoRecover information every minutes**) a jsou dostupné po dobu sedmi dnů (položka **Keep AutoRecover information for**).

Dialogové okno **Options** ukrývá skutečně nepřeberné množství prospěšných konfiguračních voleb, které vám dovolují upravit vzhled a chování IDE Visual Studia 2005 podle vašich potřeb a požadavků. Nebojte se proto s nastaveními experimentovat a zkoušet, co která volba dokáže. Nejdůležitější je totiž to, abyste se při své náročné práci v prostředí Visual Basicu 2005 cítili co možná nejlépe. Dialogové okno **Options** je v tomto směru skvělým startovním bodem.

2.4 Charakteristika integrovaného vývojového prostředí jazyka Visual Basic 2005

Abychom si mohli představit IDE Visual Basicu 2005 pod drobnohledem a porovnat jej s vývojovým prostředím jazyka Visual Basic 6.0, nejlépe uděláme, když vytvoříme první ukázkovou aplikaci. Postupujte, prosím, dle níže uvedených instrukcí:

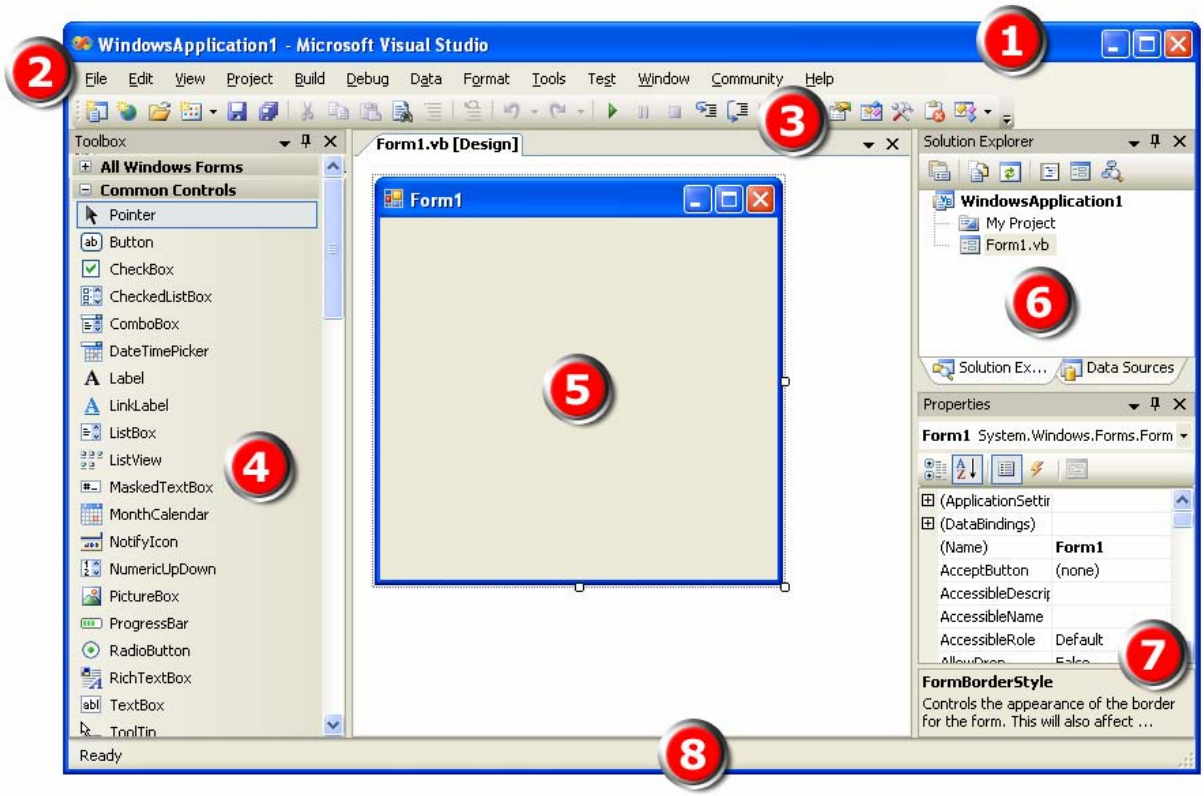
1. Jestliže jste tak ještě neučinili, spusťte Visual Studio 2005.
2. Otevřete nabídku **File** a klepněte na položku **New Project**. Jste-li příznivci klávesových zkratek, můžete stejného výsledku dosáhnout také použitím „dvojhmatu" CTRL+N.
3. V dialogu **New Project** si všimněte stromovou strukturu s názvem **Project Types**. Tato struktura seskupuje typy aplikačních projektů, které můžete vyvinout pomocí programovacích nástrojů .NET společnosti Microsoft. Jelikož nám jde pouze o Visual Basic, klepněte na stejnojmennou položku. V pravé části okna, v sekci **Templates**, se v tu chvíli zobrazí všechny projektové šablony, které jsou vám, jako vývojářům ve Visual Basicu 2005, k dispozici (obr. 2.12).



Obr. 2.12: Založení nového projektu

4. Vyhledejte šablonu s názvem **Windows Application**. Tato šablona představuje základní stavební kámen pro vývoj aplikací .NET určených pro běh v systémech řady Windows. (Šablona **Windows Application** je ekvivalentem projektu **Standard EXE**, jenž znáte z Visual Basicu verze 6.0.)
5. Poté, co jste projektovou šablonu **Windows Application** vyhledali, označte ji. Do textového pole **Name** můžete zadat název pro nově vytvářenou aplikaci, ovšem není to bezpodmínečně nutné. Pokud aplikaci nepojmenujete, Visual Basic 2005 ji přisoudí implicitní pojmenování (**WindowsApplication1**).
6. Klikněte na tlačítko **OK**.

Visual Basic 2005 sestaví plně funkční kostru standardní aplikace pro Windows. Aplikace bude mít jeden formulář, který bude zobrazen v okně vizuálního návrháře. Pohled na IDE Visual Basicu 2005 s nově založeným projektem přináší obr. 2.13.



Obr. 2.13: Integrované vývojové prostředí jazyka Visual Basic 2005

Nyní si jednotlivé součásti představíme blíže. Budeme přitom vycházet z jejich číselní identifikace, která je uvedena na obrázku.

2.4.1 Titulkový pruh aplikace

Titulkový pruh je standardní součástí jakékoliv aplikace pro Windows a nejinak je tomu i ve Visual Basicu 2005. Zdejší titulkový pruh obsahuje informace o názvu aplikace a jejím stavu (nehledejte v tom žádný jinotaj). Aplikace se může nacházet, podobně jako její kolegyně z Visual Basicu 6.0, ve třech základních stavech neboli režimech: jde o režim návrhu, režim běhu a režim přerušení, někdy označovaný také jako ladící režim. Režim návrhu je typický pro návrhovou fázi, v níž vytváříte formuláře, upravuje jejich vzhled a chování a pracujete s instancemi ovládacích prvků a komponent. V režimu běhu se aplikace Visual Basicu 2005 chová jako samostatně spustitelná softwarová jednotka. V tomto režimu s aplikací pracujete tak, jak s ní bude pracovat finální uživatel. Aplikace se může ocitnout také v režimu ladění, například tehdy, když kompilátor zastaví exekuci kódu aplikace na programové zarážce.

2.4.2 Panel nabídek

Ihned po vygenerování první ukázkové aplikace je panel nabídek IDE Visual Basicu 2005 tvořen celkem třinácti samostatnými nabídkami (**File**, **Edit**, **View**, **Project**, **Build**, **Debug**, **Data**, **Format**, **Tools**, **Test**, **Window, Community** a **Help**). Počet nabídek ovšem není konečný: jak totiž pracujete s jazykem Visual Basic 2005, vizuálním návrhářem nebo jinými komponenty IDE, náplň panelu nabídek se flexibilně mění podle charakteru právě realizovaných operací. Když si budete prohlížet jednotlivé nabídky, zcela jistě si povšimnete, že jejich grafické ztvárnění bylo od verze 6.0 Visual Studia znatelně vylepšeno. Všechny nabídky jsou obdařeny novými vizuálními styly, které přispívají

k větší přehlednosti a lepší orientaci v IDE, a také napomáhají tomu, aby byla vaše práce s Visual Basicem 2005 co nejproduktivnější.

2.4.3 Panel tlačítek

Pokud uskutečníme vzájemné porovnání pracovních prostředí jazyků Visual Basic 6.0 a Visual Basic 2005, zjistíme, že nová verze má k dispozici daleko větší počet panelů s tlačítky. Zatímco ve Visual Basicu 6.0 jste mohli pracovat pouze se čtyřmi panely tlačítek (**Debug**, **Edit**, **Form Editor** a **Standard**), v prostředí vývojového nástroje Visual Basic 2005 můžete vyzkoušet bezmála třicet různých panelů. Tento počet se může zdát obrovský, ovšem ve skutečnosti jsou mnohé panely využívány napříč všemi programovacími jazyky Visual Studia 2005, takže jejich použití je univerzálnější a není vázáno pouze na Visual Basic 2005. Panely tlačítek jsou konfigurovatelné – můžete měnit pořadí tlačítek, jejich pojmenování, dokonce můžete také upravovat vzhled ikon pro jednotlivá tlačítka. Řečeno jinými slovy, s panely tlačítek jazyka Visual Basic 2005 můžete zažít mnohá dobrodružství.

TIP: Panely tlačítek a umělá inteligence

Panely tlačítek, s nimiž se střetnete v prostředí jazyka Visual Basic 2005, nabyly od dřívějška na umělé inteligenci. Tak předně, posledním tlačítkem na každém panelu tlačítek je speciální tlačítko s názvem **Toolbar Options**, které vám poslouží hned dvakrát. Když na toto tlačítko klepnete, otevře se nabídka **Add or Remove Buttons**, jejíž pomocí můžete uživatelsky přívětivým způsobem upravovat kolekci tlačítek, která mají být na editovaném panelu viditelná. To ovšem není všechno, protože tlačítko **Toolbar Options** má také sekundární „mód" použití, jenž využijete ve chvíli, kdy nebude možné na panelu tlačítek zobrazit všechna dostupná tlačítka. Dojde-li k této situaci, na tlačítku **Toolbar Options** se objeví ikona dvojité šipky (>>). Je-li tato ikona viditelná, můžete klepnutím na tlačítko **Toolbar Options** zobrazit zbývající tlačítka, která nemohla být z důvodu nedostatečného prostoru na panelu tlačítek zobrazena. Použití tlačítka **Toolbar Options** uvádí obr. 2.14.

Obr. 2.14: Použití tlačítka **Toolbar Options**

S integrovaným vývojovým prostředím jazyka Visual Basic 2005 se vám již nemůže stát to, co bylo v IDE Visual Basicu 6.0 celkem běžné, a sice, že po zmenšení pracovní plochy tlačítka mnoha panelů jednoduše zmizela pryč a kromě opětovné modifikace rozměrů IDE neexistoval žádný přívětivý způsob, jak se k „zmizelým" tlačítkům dostat.

Panely tlačítek jsou implicitně ukládány do vymezeného prostoru, kterému se říká lišta panelů. Ačkoliv je lišta panelů vhodným kontejnerem pro úschovu panelů různého typu, při některých příležitostech jistě oceníte, že panely lze z lišty panelů také snadno vyjímat. Požadovaný panel vyjmete z lišty panelů jednoduše: nejprve jej uchopíte a pak jej tažením přemístíte do nové lokace. Když se panel tlačítek nachází nad všemi součástmi integrovaného vývojového prostředí, mluvíme o něm jako o plovoucím panelu.

2.4.4 Sada nástrojů (Toolbox)

Pokud bychom vyhlásili soutěž o nejvíce inovovanou součást IDE Visual Basicu 2005, jsem si jist, že sada nástrojů **Toolbox** by se umístila když ne rovnou na čelní, tak určitě na jedné z prvních tří příček. Sada nástrojů není totiž již pouhým panelem s ikonami ovládacích prvků a komponent. Spíš jde o kompletní programátorskou dílnu, která vám nabízí všechny důmyslné nástroje, jež budete potřebovat při stavbě svých aplikací .NET. Sada nástrojů nově sdružuje několik panelů nástrojů, přičemž jejich počet není konečný. Budete-li mít chuť, můžete sadu nástrojů rozšiřovat o další panely s vlastními kolekcemi ovládacích prvků a komponent. Standardně není sada nástrojů dokována, nýbrž je automaticky skrývána. Pokud budete chtít sadu nástrojů dokovat tak, aby byla stále „na dohled", proveďte toto:

1. Najeďte myší na záložku sady nástrojů **Toolbox**, která se nachází na levé straně integrovaného vývojového prostředí Visual Basicu 2005. IDE zjistí, že budete chtít sadu nástrojů použít, a proto ji prostřednictvím animace vysunutí zobrazí.

2. Je-li sada nástrojů viditelná, klepněte na tlačítko **Auto Hide**. Jedná se o malé tlačítko s ikonou připínáčku (), které se nachází v záhlaví sady nástrojů **Toolbox**. Po aktivaci uvedeného tlačítka bude sada nástrojů pevně umístěna podél levé strany integrovaného vývojového prostředí. To znamená, že se již nebude nadále automaticky ukrývat pokaždé, když ji kurzor myši opustí. Dokovaná podoba sady nástrojů **Toolbox** je užitečná zejména v etapě vývoje grafického uživatelského rozhraní aplikací .NET.

Souprava nástrojů **Toolbox** obsahuje mnoho panelů: **Crystal Reports**, **Data**, **Common Controls**, **Containers**, **All Windows Forms** a jiné. Pojednání o všech panelech by zabralo mnoho stránek, a proto svoji pozornost soustředíme pouze na panel **All Windows Forms**, jenž seskupuje přes šedesát ovládacích prvků a komponent, které můžete při vytváření svých softwarových řešení použít. Ovládací prvky a komponenty platformy .NET Framework 2.0 jsou pod dohledem společné objektově orientované knihovny tříd s názvem **Windows Forms**.

UPOZORNĚNÍ: Objektová knihovna **Windows Forms**	
	Bázová knihovna tříd platformy .NET Framework 2.0 definuje vyspělou kolekci tříd, které formují objektově orientovaný rámec pro vývoj všech elementů grafického uživatelského rozhraní aplikací .NET zacílených na operační systémy řady Microsoft Windows. Tyto třídy, jež jsou seskupeny v objektové knihovně **Windows Forms**, dovolují programátorům využívat pokročilých rysů platformy .NET Framework 2.0, mezi něž patří kompletní podpora pro objektově orientované modelování a programování, vizuální dědičnost a rozšiřitelnost, s cílem vytvářet působivá a snadno spravovatelná softwarová řešení. Objektovou knihovnu **Windows Forms** tvoří všechny třídy, které se nacházejí v prostoru jmen **System.Windows.Forms** bázové knihovny tříd platformy .NET Framework 2.0. Pomocí těchto tříd mohou vývojáři používat a vytvářet ovládací prvky, komponenty, formuláře a společná dialogová okna. Nadto mohou programátoři ze systémových tříd odvozovat své vlastní třídy, a tyto pak obohacovat o nové funkční prvky. Díky jazykové interoperabilitě mohou s třídami knihovny **Windows Forms**, a samozřejmě také s odvozenými třídami, pracovat všechna vývojová prostředí, která vyhovují standardům platformy Microsoft .NET. Když tedy vyvinete skvostný ovládací prvek v jazyce Visual Basic 2005, můžete jej například zaslat elektronickou poštou svému příteli, který pracuje v jazyku C#. Základní pilíře, na nichž je postavena architektura vývojově-exekuční platformy .NET Framework 2.0, zaručují, že váš přítel bude moci ovládací prvek používat ve svých aplikacích bez jakýchkoliv potíží či výkonnostních penalizací.

Porovnáte-li sadu ovládacích prvků a komponent mezi jazyky Visual Basic 6.0 a Visual Basic 2005, rychle dojdete k poznání, že mnohé z nich jsou zcela nové (**DataGridView**, **MenuStrip**), jiné jsou

modifikované (**PictureBox**, **ProgressBar**) či přejmenované (**DateTimePicker**, **RadioButton**) a některé byly ze soupravy nástrojů zcela odstraněny (**Image**, **Shape**, **Line**).

Ovládací prvky a komponenty, s nimiž můžete v prostředí Visual Basicu 2005 pracovat, jsou plně přizpůsobeny vývojově-exekuční platformě .NET Framework 2.0 a zcela vyhovují jejím standardům. To tedy znamená, že tyto entity nevystupují jako prvky ActiveX či komponenty, nepracují na báze objektového modelu COM a jejich programový kód není uložen v specifických .ocx souborech. Místo toho jsou všechny ovládací prvky a komponenty uloženy v nových logicko-funkčních aplikačních jednotkách, kterým se v řízeném prostředí říká sestavení (assemblies). Sestavení disponují řadou výhod, inovovanou interní strukturou počínaje a širokými možnostmi interoperability konče. Problematiku sestavení probereme ve třetí kapitole této příručky.

POZNÁMKA: Ovládací prvky ActiveX a souprava nástrojů ve Visual Basicu 2005

Pro úplnost je potřebné uvést, že do soupravy nástrojů **Toolbox** můžete ukládat také existující ovládací prvky ActiveX, které byly vytvořeny buď v jazyce Visual Basic 6.0 nebo v jiném COM-kompatibilním programovacím prostředí. Tento postup je však nutné předem dobře zvážit, neboť prvky ActiveX nebyly vytvářeny s ohledem na jejich potenciální budoucí použití v prostředí programovacích nástrojů .NET. Proto může při implementaci stávajících prvků ActiveX dojít k narušením v bezpečnostní politice aplikací .NET, což může vést k různým neblahým konsekvencím. Navíc se mohou objevit také potíže s výkonností aplikací .NET –prvky ActiveX totiž nepatří mezi řízené komponenty, a proto nejsou aplikace .NET schopny je přímo používat. Místo toho je nutné vybudovat řízenou obalovou vrstvu pro nativní kód prvků ActiveX – tato je vytvořena pomocí interoperabilní technologie COM Interop.

Instance ovládacích prvků můžete na formulář umísťovat podobně, jako jste to dělali ve Visual Basicu 6.0. Ovšem zatímco v jazyce Visual Basic 6.0 jste mohli instanci ovládacího prvku vložit na formulář jejím „nakreslením" nebo rychlým poklepáním na ikonu příslušného prvku, Visual Basic 2005 vám umožňuje vytvořit instanci také pomocí techniky „táhni a pusť" (*drag&drop*). Jakmile instanci ovládacího prvku vložíte na plochu formuláře, můžete upravovat její rozměry, vzhled a další charakteristiky pomocí dialogového okna **Properties** (o tomto okně pojednáme dále v textu). Podobně jako s instancemi ovládacích prvků můžete v prostředí Visual Basicu 2005 pracovat i s instancemi komponent. Na rozdíl od jazyka Visual Basic 6.0 nejsou instance komponent ukládány na plochu formuláře, ale své útočiště nacházejí ve speciálně vyhrazené oblasti, které se říká podnos komponent (**Component Tray**). Podnos komponent je reprezentován samostatným regionem, jenž je situován pod oknem formuláře. Popsané paradigma umožňuje vizuálně oddělit instance ovládacích prvků od instancí komponent, což je další pozitivní vlastnost integrovaného vývojového prostředí jazyka Visual Basic 2005. Instance různých typů se vám již nebudou plést, čehož důsledkem je nejenom vyšší přehlednost, ale také větší komfort při navrhování grafického uživatelského rozhraní aplikací .NET.

2.4.5 Vizuální návrhář (Windows Forms Designer)

Vizuální návrhář je spolehlivým pomocníkem, jenž je připraven podat vám pomocnou ruku kdykoliv se pustíte do vizuálního programování v jazyce Visual Basic 2005. Po vytvoření standardní aplikace pro systém Windows je v okně vizuálního návrháře zobrazen hlavní aplikační formulář. Jak si můžete všimnout, formulář implicitně využívá pokročilých vizuálních stylů systému Microsoft Windows XP, což znamená, že pro aktivaci těchto stylů nemusíte uskutečňovat žádné další operace. Vizuální návrhář implementovaný ve Visual Basicu 2005 vám pomůže s vkládáním instancí ovládacích prvků a komponent, a také s jejich upravováním a umísťováním. Díky inteligentním vodícím linkám můžete instance zarovnávat daleko snadněji, než tomu bylo ve Visual Basicu 6.0. Vetší míru volnosti v návrhové etapě vám poskytuje zcela nový způsob editace základních vlastností instancí, tzv. režim rychlého

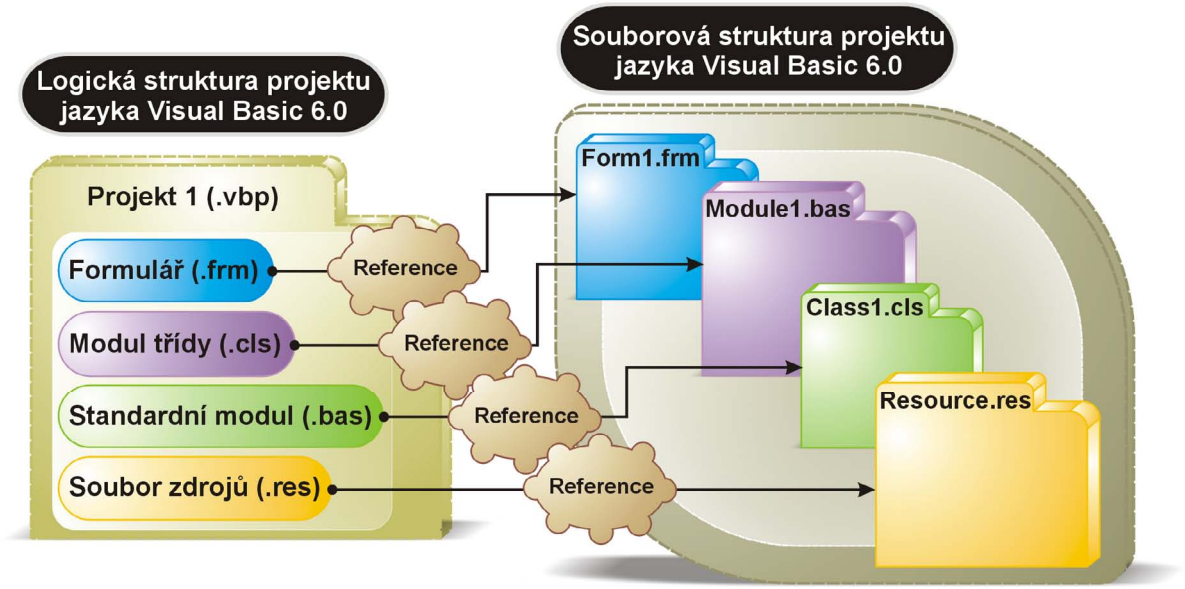
editování (**Quick Edit Mode**), jehož pomocí můžete měnit hodnoty vymezené skupiny vlastností instancí ovládacích prvků přímo na ploše formuláře. Vizuální návrhář představuje také bránu do světa editoru zdrojového kódu: Kdykoliv poklepete na libovolnou instanci ovládacího prvku či komponenty, vizuální návrhář vás přenese do editoru zdrojového kódu, kde na vás již bude čekat vygenerovaná programová kostra zpracovatele události **Click** této instance. Editor zdrojového kódu, jenž je úzce propojen s vizuálním návrhářem, můžete zobrazit několika způsoby: můžete otevřít nabídku **View** a klepnout na položku **Code**, anebo můžete kliknout na tlačítko **View Code** na panelu tlačítek okna **Solution Explorer**.

2.4.6 Průzkumník řešení (Solution Explorer)

Průzkumník řešení je centrem projektového managementu v integrovaném vývojovém prostředí jazyka Visual Basic 2005. Umožňuje vám pečlivě spravovat všechny relevantní součástí aplikačního projektu nebo skupiny projektů, která se nově označuje termínem řešení. Průzkumník řešení, s nímž se střetáváme v jazyce Visual Basic 2005, je nástupcem Průzkumníka projektu (**Project Explorer**) z Visual Basicu 6.0. Jelikož se proces řízení projektů od dob Visual Basicu 6.0 podstatně změnil, musíme si o uskutečněných změnách povědět přece jenom více.

Projektový management Visual Basicu 6.0 byl založen na práci s jednotlivými softwarovými projekty. Projekt byl samostatným celkem, který byl reprezentován jedním projektovým souborem (jednalo se o soubor s extenzí .vbp). Projektový soubor definoval odkazy na všechny ostatní součásti projektu jazyka Visual Basic 6.0. Těmito součástmi mohly být soubory formulářů (.frm), binární datové soubory (.frx), moduly tříd (.cls) standardní moduly (.bas) a další komponenty. Projektový soubor měl ve skutečnosti textovou interní strukturu, což znamenalo, že vývojáři mohli obsah tohoto souboru poměrně snadno upravovat i mimo integrovaného vývojového prostředí. Jazyk Visual Basic 6.0 nabízel programátorům práci s více projekty současně – stačilo vybrat volbu **Add Project** z menu **File** a založit nový projekt podle předdefinované projektové šablony. Visual Basic 6.0 nově vytvořený projekt automaticky přidal ke stávajícími projektu a aktualizoval obsah okna Průzkumníka projektu. Rovněž byl vytvořen soubor skupiny projektů (Project Group) s extenzí .vbg, který sdružoval reference na všechny projekty umístěné v jedné skupině projektů. V této souvislosti je velice důležité, abyste si uvědomili, že Průzkumník projektu jazyka Visual Basic 6.0 byl založen na referenčním pracovním modelu. Řečeno jinými slovy, Průzkumník projektu deklaroval stromovou strukturu referencí, které byly nasměrovány na cílové soubory, jež tvořily nosné prvky softwarového projektu. Tyto soubory tak byly prostřednictvím referencí přímo používány.

Když jste kupříkladu přidali do projektu již existující formulář, Průzkumník projektu jednoduše přidal do své stromové struktury referenci na tento soubor. Jestliže jste formulář nějakým způsobem pozměnili nebo upravili, a poté jste vydali povel pro uložení uskutečněných modifikací, Visual Basic 6.0 provedené úpravy promítl přímo do struktury cílového souboru s extenzí .frm. Pokud jste chtěli později formulář z projektu odstranit, použili jste příkaz **Remove NázevSouboru.frm** z nabídky **Project**, čímž jste sice odstranili referenci na soubor s formulářem, ovšem tato změna se nijak nedotkla cílového souboru s formulářem. Ten zůstal i nadále přítomen na vašem pevném disku v té složce, do které jste ho uložili. Logickou a souborovou strukturu standardního projektu jazyka Visual Basic 6.0 můžete vidět na obr. 2.15.



Obr. 2.15: Logická a souborová struktura standardního projektu jazyka Visual Basic 6.0

Ve Visual Basicu 2005 je situace poněkud odlišná. Ačkoliv také zde pracují programátoři s projekty různých typů, základní funkční jednotkou již není projekt, ale softwarové řešení. Ačkoliv si toto řešení můžete představit jako ekvivalent skupiny projektů z jazyka Visual Basic 6.0, takováto analogie by nebyla zcela přesná. Zatímco ve Visual Basicu 6.0 jste mohli do skupiny projektů přidávat pouze projekty jazyka Visual Basic 6.0, do řešení jazyka Visual Basic 2005 můžete, samozřejmě kromě projektů tohoto jazyka, přidávat rovněž projekty dalších jazyků platformy .NET, C# a C++/CLI nevynímaje. Podle uvedeného schématu můžete naplno využít všech dovedností společné jazykové interoperability, která vám otevírá pomyslné dveře do světa vývoje sofistikovaných softwarových modulů a komponent. Řešení tak můžeme zjednodušeně chápat jako kontejner pro více softwarových projektů různých typů, které mohou mezi sebou úzce kooperovat. Pokud bychom se však na řešení podívali zblízka, zjistili bychom, že kromě projektů mohou řešení spoluvytvářet i samostatně působící položky, které sice nepatří do žádného projektu, ale jejichž aplikovatelnost se váže k celému řešení. Informace o každém řešení, které ve Visual Basicu 2005 vytvoříte, jsou uschovány ve dvou samostatných souborech s extenzemi .sln a .suo. V obou případech se jedná o definiční soubory, které obsahují metadata o softwarovém řešení. V souboru .sln se nacházejí informace o všech projektech, které do daného řešení patří, dále informace o samostatně působících položkách řešení jakožto i informace o konfiguraci sestavení jednotlivých projektů řešení. V souboru .suo jsou pak uložena data, která popisují uživatelská nastavení pojící se se softwarovým řešením.

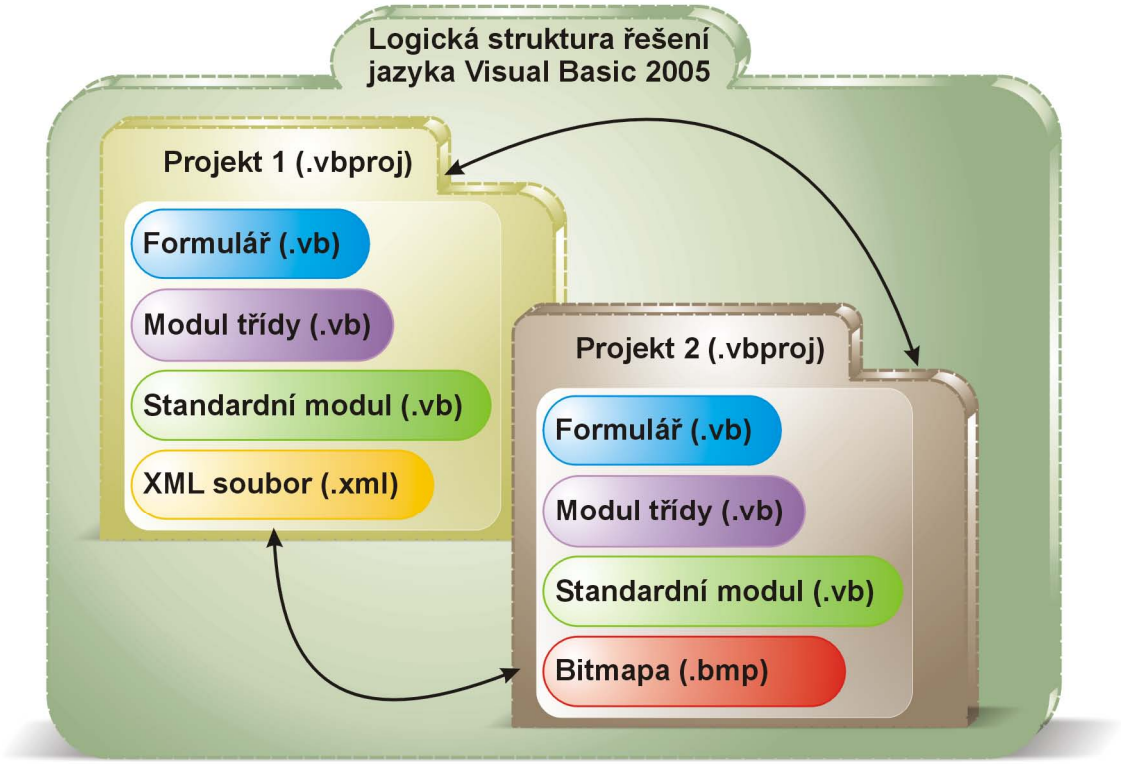
Průzkumník řešení, s nímž se ve Visual Basicu 2005 setkáváte, pracuje na rozdíl od svého kolegy z jazyka Visual Basic 6.0 na bázi složkového modelu. Všechny součásti řešení jsou ukládány do samostatných složek, tj. každý projekt obsahuje svou vlastní složku, ve které je uložena kompletní sada projektových souborů. Projektové složky jsou dále hierarchicky členěny, a proto může být každá složka tvořena jistým počtem podsložek. Kupříkladu, projektová složka standardní aplikace pro Windows, kterou můžete sestrojit pomocí šablony **Windows Application**, disponuje podsložkou s názvem **Bin**, v níž je po sestavení a přeložení aplikace uložen její spustitelný soubor (.exe). Průzkumník řešení Visual Basicu 2005 používá jiný druh správy projektových součástí než Průzkumník projektu v jazyce Visual Basic 6.0. Když totiž do projektu Visual Basicu 2005 přidáte nějakou novou součást, povězme dynamicky linkovou knihovnu, Visual Basic 2005 vytvoří kopii původní knihovny DLL a tuto kopii následně umístí do příslušné projektové složky. Průzkumník řešení tuto situaci identifikuje a referenci na zkopírovanou dynamicky linkovanou knihovnu umístí do své stromové struktury. Jestliže budete chtít knihovnu DLL z projektu odstranit, můžete aktivovat příkaz **Delete** z nabídky **Edit**. Visual Basic 2005 vás upozorní, že tímto úkonem bude soubor natrvalo smazán a bude požadovat váš souhlas, zdali chcete tento soubor skutečně zlikvidovat.

Budete-li souhlasit, Visual Basic 2005 odstraní kopii knihovny DLL z projektové složky, ovšem tato operace nebude mít vůbec žádný vliv na původní dynamicky linkovou knihovnu (ta zůstane zcela nedotčena).

POZNÁMKA: Model správy projektových součástí Průzkumníka řešení

Jak sami vidíte, model správy projektových součástí Průzkumníka řešení se ponáší na způsob předávání argumentů hodnotou. V tomto případě je cílové metodě předána kopie původního argumentu a ačkoliv metoda může hodnotu této kopie modifikovat, není schopna pozměnit původní argument. Když nahradíme argumenty projektovými součástmi a metodu projektem, získáme zdařilou analogii práce Průzkumníka řešení.

Schématické zobrazení logické struktury řešení jazyka Visual Basic 2005, které je tvořeno dvěma samostatnými projekty přináší obr. 2.16.



Obr. 2.16: Logická struktura řešení jazyka Visual Basic 2005

Každý projekt jazyka Visual Basic 2005 obsahuje definiční projektový soubor s extenzí .vbproj. Tento soubor obsahuje metadata o projektu ve formátu značkovacího jazyka XML. Definiční projektový soubor aplikace .NET se tedy zásadně liší od projektového souboru (.vbp) aplikace Visual Basicu 6.0. Nově jsou všechna metadata zakódována v XML a nikoliv v textové podobě, jak tomu bylo v minulé verzi Visual Basicu. Schéma projektového souboru .vbproj je automaticky generované integrovaným vývojovým prostředím Visual Basicu 2005 a nedoporučuje se, abyste jej jakkoliv měnili či upravovali. Jak je z obr. 2.16 patrné, softwarový projekt může být tvořen mnoha různorodými součástmi: standardně se jedná o soubory s formuláři, moduly tříd nebo standardní moduly. Všimněte si, že nyní mají soubory s formuláři, třídami a moduly shodnou koncovku – .vb. Po pravdě řečeno, stejnou extenzi mají také další soubory, s nimiž můžete v prostředí jazyka Visual Basic 2005 pracovat. Za všechny jmenujme třeba soubory s COM třídami, uživatelskými prvky či rozhraními. Kromě definičního projektového souboru s extenzí .vbproj obsahuje každý projekt rovněž definiční soubor s kombinovanou koncovkou .vbproj.user. Tento soubor je tvořen metadaty, která definují uživatelská nastavení projektu provedená vývojářem.

2.4.7 Okno Properties

Jako programátorům přicházejícím z Visual Basicu 6.0 vám okno **Properties** nemusím zcela jistě zdlouhavě představovat. V tomto okně se zobrazují vlastnosti všech objektů, s nimiž v prostředí vizuálního návrháře pracujete. K těmto objektům patří především formuláře a instance ovládacích prvků a komponent. Kromě toho vám ovšem okno **Properties** poslouží také při úpravě vlastností projektových položek, které jsou zobrazeny v Průzkumníkovi řešení (**Solution Explorer**). Tato nová dovednost okna **Properties** vám umožňuje modifikovat požadované atributy souborů, ve kterých jsou programové objekty (jako třeba formuláře a instance ovládacích prvků) uloženy. Abyste si popsané dva režimy okna **Properties** vyzkoušeli v praxi, proveďte následující malé cvičení:

1. Klepněte levým tlačítkem myši na formulář, čímž na něj přenesete zaměření.
2. Když se v tuto chvíli podíváte do okna **Properties**, uvidíte, že jsou zde zobrazeny vlastnosti formuláře, které můžete podle libosti upravovat.
3. Nyní zaměřte svou pozornost na okno Průzkumníka řešení, ve kterém klepnete na položku **Form1.vb**.
4. Okno **Properties** se zaplní vlastnostmi, jejichž prostřednictvím můžete upravovat atributy projektového souboru, v němž je uložen zdrojový kód formuláře.

Položky, jež jsou zobrazeny v okně **Properties**, mohou být řazeny podle abecedy (tlačítko **Alphabetic**) nebo podle kategorií (tlačítko **Categorized**). Další novinkou okna **Properties** ve Visual Basicu 2005 je tlačítko **Events**, jehož pomocí můžete snadno vytvářet zpracovatele událostí zvolených programových objektů. Když aktivujete tlačítko **Events**, v okně **Properties** se zjeví seznam všech událostí, na které je zvolený objekt schopen reagovat. Podobně jako vlastnosti, také události mohou být filtrovány pomocí abecedy nebo kategorického rozvržení. Když budete chtít, aby Visual Basic 2005 vygeneroval zpracovatele jisté události, stačí, když na název této události poklepete levým tlačítkem myši. O vše ostatní se již postará samotný Visual Basic 2005. Ve spodní části okna **Properties** je umístěn obdélníkový panel **Description**, který vám nabízí stručnou nápovědu k vybrané vlastnosti, respektive události. Budete-li chtít maximalizovat prostor pro zobrazení seznamu vlastností nebo událostí, můžete panel **Description** skrýt. Panel ukryjete tak, že na něj klepnete pravým tlačítkem myši a v kontextové nabídce zrušíte zatržení u položky **Description**.

2.4.8 Stavový pruh

Integrované vývojové prostředí jazyka Visual Basic 2005 zobrazuje ve stavovém pruhu informační zprávy, které se vážou k právě realizovaným programovým operacím. Kupříkladu, když vydáte pokyn na překlad aplikace, IDE bude ve stavovém pruhu zobrazovat aktuální informace o kompilačním procesu.

2.4.9 Dynamická nápověda

Visual Studio 2005 přichází v oblasti elektronické dokumentace se zbraní opravdu těžkého kalibru. Ano, nemýlíte se, jde o dynamickou nápovědu, která přináší revoluci v způsobu, jakým vývojáři a programátoři pracující v integrovaném vývojovém prostředí. Pokud jste chtěli v IDE Visual Basicu 6.0 získat informace o jistém programovém příkazu, měli jste v zásadě dvě možnosti: Mohli jste buď zadat hledaný příkaz do systému nápovědy HTML, anebo umístit kurzor na daný příkaz (jestliže jste jej již zapsali) a následně aktivovat klávesu F1. Ačkoliv druhá varianta byla rychlejší, byli jste to pořád vy, kdo inicioval dialog se systémem elektronické dokumentace. Dynamická nápověda implementovaná ve Visual Studiu 2005 staví tuto letitou koncepci takříkajíc vzhůru nohama. Vaše kroky v integrovaném vývojové prostředí jsou totiž od nynějška pod pečlivým dohledem dynamické nápovědy, která vám

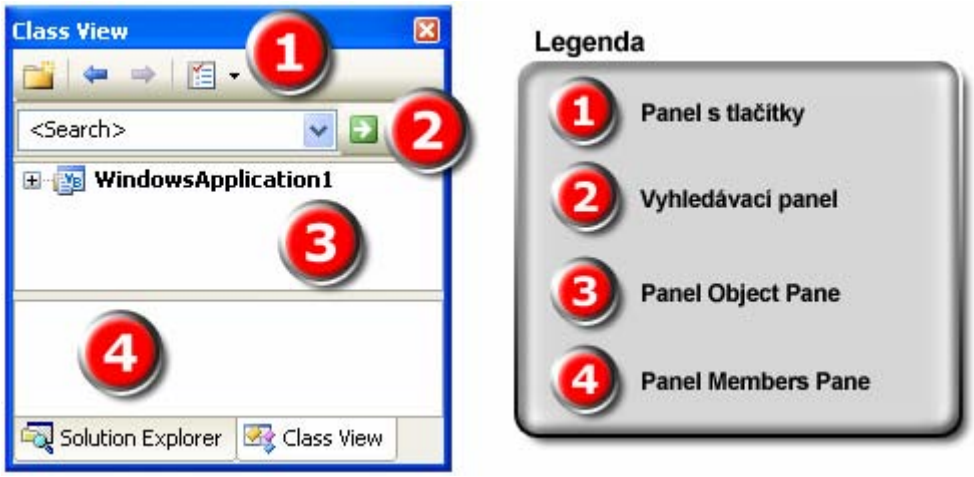
podle stylu vaší práce automaticky nabízí ta témata nápovědy, o nichž předpokládá, že by vás mohla zajímat. Okno dynamické nápovědy můžete zobrazit aktivací položky **Dynamic Help** z nabídky **Help** (rovněž můžete upotřebit klávesovou zkratku CTRL+ALT+F4). Podobu dynamické nápovědy uvádí obr. 2.17.

Obr. 2.17: Okno dynamické nápovědy (**Dynamic Help**)

Kdykoliv označíte instanci ovládací prvku, zadáte klíčové slovo jazyka Visual Basic 2005 do editoru zdrojového kódu, nebo provedete jinou činnost, kterou dynamická nápověda rozezná, bude vám doručen seznam doporučených témat nápovědy, která byste neměli minout. Z vlastní zkušenosti můžu říct, že algoritmus, podle něhož dynamická nápověda pracuje, je skutečně až obdivuhodně inteligentní. Výsledným efektem je přímo raketový nárůst pracovní produktivity vývojáře, neboť ten se může ke kýženým informacím dopracovat daleko rychleji, než tomu bylo v prostředí jazyka Visual Basic 6.0. Při standardním rozvržení součástí integrovaného vývojového prostředí se okno dynamické nápovědy rozprostírá v pravém dolní časti, přičemž sdílí vyhrazený prostor s oknem **Properties**.

2.4.10 Okno Class View

Přehled o datových typech a jejich členech, které jsou definovány v softwarovém řešení jazyka Visual Basic 2005, můžete získat prostřednictvím okna **Class View**. Okno **Class View** je implicitně situováno v pravém horním segmentu IDE jazyka Visual Basic 2005. Pokud jej nevidíte, zobrazíte jej přes nabídku **View → Other Windows → Class View**. Za asistence okna **Class View** můžete analyzovat hodnotové a odkazové datové typy, které jsou v právě editovaném softwarovém řešení definovány. Vizuálně je okno **Class View** tvořeno panelem s tlačítky, vyhledávacím panelem a dalšími dvěma samostatnými panely s názvy **Object Pane** a **Members Pane**, které zabírají většinu plochy dialogového okna (obr. 2.18).



Obr. 2.18: Okno **Class View** a jeho součásti

V panelu **Object Pane** se nachází hierarchicky uspořádaná stromová struktura projektů softwarového řešení, ve které přislouchá každému projektu právě jeden uzel. Jelikož naše ukázkové řešení obsahuje pouze jeden projekt standardní aplikace pro systém Windows, je v panelu **Object Pane** zobrazen právě tento projekt. Když rozvinete uzel projektu, objeví se seznam dostupných jmenných prostorů, v našem případě půjde pouze o jediný jmenný prostor, jenž má stejný název jako aplikační projekt. Jmenné prostory představují logické kontejnery, které sdružují programové entity podobného funkčního zaměření, jako jsou například třídy či standardní moduly. Rozvinete-li uzel s jmenným prostorem, Visual Basic 2005 vám nabídne seznam programových entit, které jsou v daném jmenném prostoru uloženy. Pokud vybraná entita reprezentuje datový typ, třeba třídu, můžete na ní klepnout a v panelu **Members Pane** se objeví datové členy této třídy. Ve jmenném prostoru **WindowsApplication1** je umístěna ikona třídy formuláře (**Form1**). Když na tuto ikonu kliknete, uvidíte datové členy, které třída **Form1** definuje.

POZNÁMKA: Filtrování dostupných datových členů tříd v panelu **Members Pane**

Okno **Class View** vám umožňuje provádět účinnou filtraci datových členů tříd, které se zobrazují v panelu **Members Pane**. Klepnutím na šipku u tlačítka **Class View Settings** zviditelníte kontextovou nabídku, v jejíž spodní části jsou umístěny položky, pomocí nichž můžete určovat, které datové členy mají být zobrazeny. Implicitně se v panelu **Members Pane** zobrazují všechny veřejně přístupné (položka **Show Public Members**), chráněné (položka **Show Protected Members**) a soukromé (položka **Show Private Members**) datové členy tříd. Budete-li chtít zobrazit také zděděné datové členy, můžete provést rovněž aktivaci položky **Show Inherited Members**. O dědičnosti jakožto i dalších principech objektově orientovaného programování bude blíže pojednávat 5. kapitola této publikace.

TIP: Rychlé zjištění seznamu datových členů determinovaného datového typu

Když budete pracovat s rozsáhlým projektem jazyka Visual Basic 2005, který definuje desítky tříd a datových členů, pravděpodobně zjistíte, že postupné vyšetřování dostupných datových typů prostřednictvím stromové struktury panelu **Object Pane** není tou nejrychlejší cestou, jak se dostat ke kýženým informacím. Ve chvílích, kdy budete chtít rychle zjistit množinu datových členů, které definuje jistý datový typ, můžete s povděkem využít možností vyhledávacího panelu okna **Class View**. Do textového okna vyhledávače zadejte textový řetězec představující název cílového datového typu a stiskněte klávesu ENTER, nebo klepněte na tlačítko **Search** ( ). Takřka okamžitě bude panel **Members Pane** naplněn kolekcí datových členů specifikovaného datového typu.

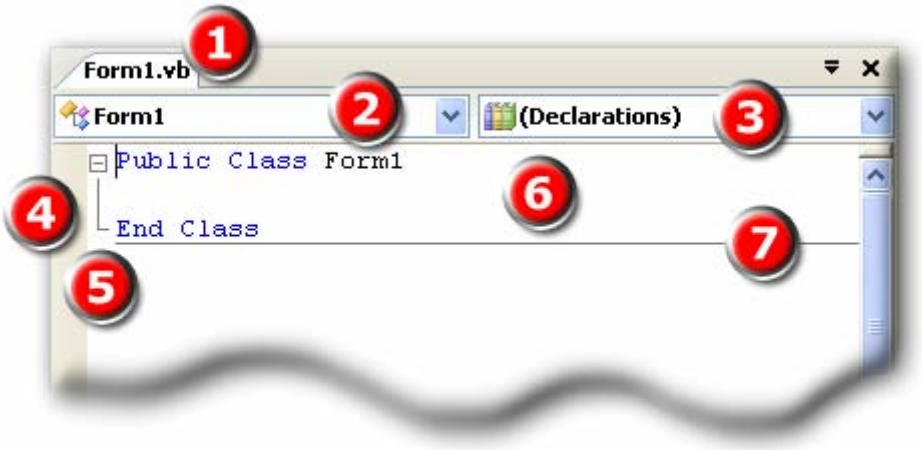
2.4.11 Editor zdrojového kódu

Editor zdrojového kódu jazyka Visual Basic 2005 představuje vyspělou softwarovou laboratoř, v níž budete trávit jistě velké množství času. Podobně jako mají vědci při svém bádání k dispozici inteligentní a sofistikované nástroje, vývojáři ve Visual Basicu 2005 mají na dosah ruky pokročilé nástroje pro psaní syntaktických programových konstrukcí a fragmentů zdrojového kódu. Tyto nástroje jsou obsaženy v jednom balíku, jemuž se říká editor zdrojového kódu. Ačkoliv na první pohled se může editor zdrojového kódu jevit jako standardní textový procesor, již po několika okamžicích zjistíte, že je plně přizpůsoben pro náročnou vývojářskou práci.

Editor zdrojového kódu můžete zobrazit několika způsoby:

1. Otevřete nabídku **View** a klepněte na položku **Code**.
2. V okně Průzkumníka řešení (**Solution Explorer**) klepněte na ikonu **View Code**.
3. Klepněte pravým tlačítkem myši na plochu formuláře nebo na plochu instance libovolného ovládacího prvku nebo komponenty a z kontextové nabídky vyberte příkaz **View Code**.
4. Poklepejte na instanci libovolného ovládacího prvku nebo komponenty. Visual Basic 2005 vygeneruje syntaktickou kostru zpracovatele implicitní události instance a tuto zobrazí v okně editoru zdrojového kódu.

Editor zdrojového kódu jazyka Visual Basic 2005 vypadá jinak než jeho protějšek, jenž znáte z Visual Basicu 6.0. Podobu editoru zachycuje obr. 2.19.



Obr. 2.19: Editor zdrojového kódu jazyka Visual Basic 2005

Popišme si nyní jednotlivé části editoru zdrojového kódu:

1. **Záložka s názvem souboru, jehož obsah je právě zobrazen v editoru zdrojového kódu.** Integrované vývojové prostředí používá pro rychlou orientaci mezi soubory různých typů systém záložek. Záložky se zobrazují v horním horizontálním pruhu, kde jsou nejenom dobře viditelné, ale také rychle přístupné. Mezi záložkami můžete přepínat a prohlížet tak obsahy všech otevřených souborů v IDE jazyka Visual Basic 2005.

2. **Otevírací seznam Class Name**. V tomto seznamu se zobrazují názvy tříd, které jsou základními pilíři pro stavbu instancí ovládacích prvků a komponent, s nimiž pracujete v prostředí vizuálního návrháře. Když kupříkladu zobrazíte zdrojový kód formuláře **Form1**, zjistíte, že formulář je ve skutečnosti reprezentován stejnojmennou třídou. Visual Basic 2005 se může pochlubit plnou podporou objektově orientovaného programování, což znamená, že s třídami a jejich

instancemi se budete v tomto prostředí střetávat daleko častěji, než tomu bylo v jazyce Visual Basic 6.0. Pomocí otevíracího seznamu **Class Name** můžete přistupovat také k seznamu událostí, na které je schopna instance dané třídy reagovat. Budete-li chtít zobrazit události formuláře **Form1**, udělejte toto: Ze seznamu **Class Name** vyberte položku **(Form1 Events)**, a poté z otevíracího seznamu **Method Name**, jenž je umístěn napravo od seznamu **Class Name**, vyberte požadovanou událost. Jakmile je událost vybrána, Visual Basic 2005 sestaví syntaktickou kostru jejího zpracovatele a tuto umístí do editoru zdrojového kódu. Ačkoliv roli zpracovatelů událostí sehrávají procedury Sub podobně jako ve Visual Basicu 6.0, syntaxe těchto obslužných procedur se podstatně změnila. Více se o syntaktických a sémantických modifikacích jazyka Visual Basic 2005 dozvíte ve čtvrté kapitole.

3. **Otevírací seznam Method Name.** Primárním účelem tohoto seznamu je poskytovat na požádání kolekci metod, kterými třída specifikovaná v otevíracím seznamu **Class Name** disponuje. Jak jsme si ovšem uvedli výše, seznam **Method Name** vám dobře poslouží také při generování zpracovatelů událostí instance třídy.

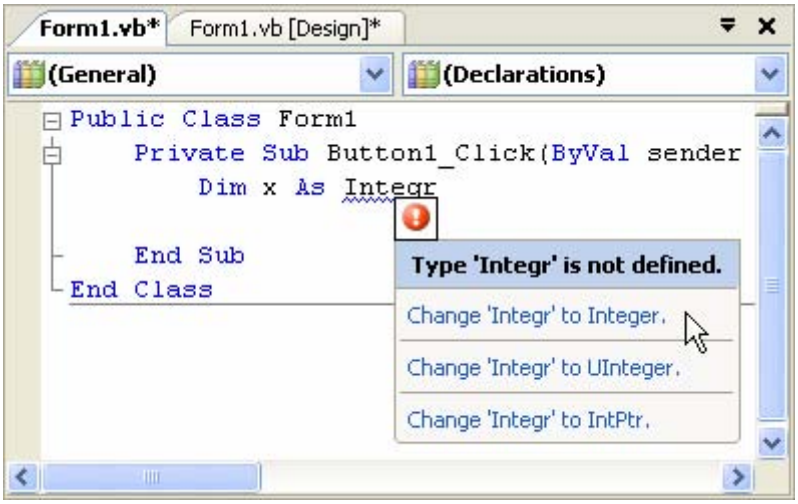
4. **Indikátorový pruh.** Indikátorový pruh je vizuálně reprezentován vertikálním pruhem šedé barvy, jenž zastává svoji pozici v levé části okna editoru zdrojového kódu. Do tohoto pruhu můžete umísťovat programové indikátory, ke kterým patří zejména body přerušení (někdy nazývané také jako programové zarážky) a záložky. Když klepnete levým tlačítkem myši na plochu indikátorového pruhu, Visual Basic 2005 umístí na příslušný řádek se zdrojovým kódem lokální bod přerušení ( ). Systém bodů přerušení byl od minulosti podstatně přepracován, čehož důsledkem je, že nyní mohou vývojáři s programovými zarážkami pracovat flexibilněji a rovněž je mohou lépe konfigurovat. Velice důležité je ovšem to, že nově mohou programátoři exaktně určit časový okamžik, kdy má bod přerušení vykonat svou činnost, tedy pozastavit exekuci kódu, jenž běží na specifikovaném programovém vláknu (zpravidla se jedná o vlákno aktivní). Bod přerušení může být aktivován při mnoha příležitostech, kupříkladu pokaždé, když se exekuce kódu přesune na řádek, na němž je bod přerušení umístěn, když je splněna předem určená podmínka, anebo když byl bod přerušení aktivován několikrát po sobě. Kromě bodů přerušení lze indikátorový pruh využít i k zakládání záložek. Postup pro přidání záložky se od přidání bodu přerušení liší. Budete-li chtít aplikovat na vybraný řádek zdrojového kódu záložku, postupujte následovně:

    1. Umístěte kurzor myši na řádek s kódem, na který chcete záložku přidat.
    2. Otevřete nabídku **Edit**, rozviňte podnabídku **Bookmarks** a klepněte na položku **Toggle Bookmark**.
    3. Do indikátorového pruhu bude přidán symbol záložky ( ).

5. **Výběrový pruh.** Po pravé straně od indikátorového pruhu se nachází poněkud těžko rozpoznatelný prostor se svislým pruhem bíle barvy. Tento je znám jako výběrový pruh, neboť vám dovoluje provádět selekci celých řádků se zdrojovým kódem. Když najedete kurzorem myši na výběrový pruh, standardní kurzor myši se zrcadlově převrátí, a bude mít tedy tuto podobu . Nyní můžete tažením kurzoru přes více řádků provést jejich výběr do jednoho souvislého bloku.

   Tím ovšem nejsou možnosti použití výběrového pruhu vyčerpány. Ve výběrovém pruhu se totiž zobrazují také vizuální symboly ve tvaru uzlů, které označují hranice regionů, v nichž je uložen zdrojový kód různých programových elementů, jako jsou třeba procedury Sub, funkce nebo třídy. Pomocí těchto uzlů můžete kód uvedených programových elementů svinout a ušetřit tak prostor v editoru zdrojového kódu.

6. **Prostor pro psaní zdrojového kódu.** Tato hlavní část editoru zdrojového kódu slouží pro zadávání klíčových slov, příkazů, programových elementů a syntaktických konstrukcí jazyka Visual Basic 2005. Editor zdrojového kódu vývojářům nabízí mnoho pokročilých funkcí, s jejichž pomocí se fáze psaní kódu stává produktivnější a efektivnější. Samozřejmostí je barevné formátování speciálních segmentů programového kódu, jakými jsou například klíčová slova, modifikátory či komentáře. Za asistence technologie IntelliSense můžete provádět rychlou selekci metod a vlastností objektů a získávat informace o parametrech a signaturách procedur Sub a funkcí. IntelliSense si poradí také s kompletizací identifikátorů, automatickou kontrolou závorkových párů či zobrazováním syntaktické nápovědy při zadání programových příkazů se specifickým významem (například **Exit**, **Implements** nebo **Declare**). Nově se v editoru zdrojového kódu objevují i inteligentní značky, ne nepodobné těm, které znáte z prostředí sady kancelářských aplikací Microsoft Office System 2003. Když se kupříkladu spletete a chybně zapíšete název datového typu proměnné, editor toto nedopatření identifikuje a špatně zadaný název datového typu formátuje vlnovkou modré barvy. Když na chybný text najedete kurzorem myši, objeví se inteligentní značka, která vám nabídne možnosti vyřešení vzniklé problémové situace (obr. 2.20).



Obr. 2.20: Inteligentní značka v prostředí editoru zdrojového kódu

Revoluční novinkou editoru zdrojového kódu, která se u vývojářů přicházejících z jazyka Visual Basic 6.0 dočká bezesporu velikého ohlasu, je možnost pracovat se šablonami programového kódu technologie IntelliSense. Tyto šablony působí jako samostatně stojící funkční bloky programového kódu, které řeší jistý specifický úkol, a které můžete vkládat do kódu svých aplikací. Šablony programového kódu byly připraveny tvůrci jazyka Visual Basic 2005 tak, aby byla zabezpečena jejich široká aplikovatelnost a opětovná použitelnost. Výbava jazyka Visual Basic 2005 je v oblasti šablon programového kódu skutečně bohatá: jistě budete souhlasit, že takřka pět set šablon je opravdu úctyhodné číslo. Každá šablona řeší jistý úkol: zatímco jedna vám pomůže s vytvořením bitové mapy za běhu aplikace, jiná za vás vypočte hodnotu trigonometrické funkce. Výborné je, že šablony si s sebou vezou veškerý zdrojový kód, jenž je potřebný k dosažení cíle. Jediné, co musíte udělat vy, je upravit vstupní data, nebo modifikovat implicitně dodané hodnoty. Za několik málo okamžiků pak máte k dispozici zcela funkční fragment kódu, který odvádí svou práci na jedničku.

Abychom si představili praktické použití šablon programového kódu, ukážeme si, jak za pomoci šablony získat kód, jenž bude schopen spustit námi určenou aplikaci. Řiďte se níže uvedenými instrukcemi:

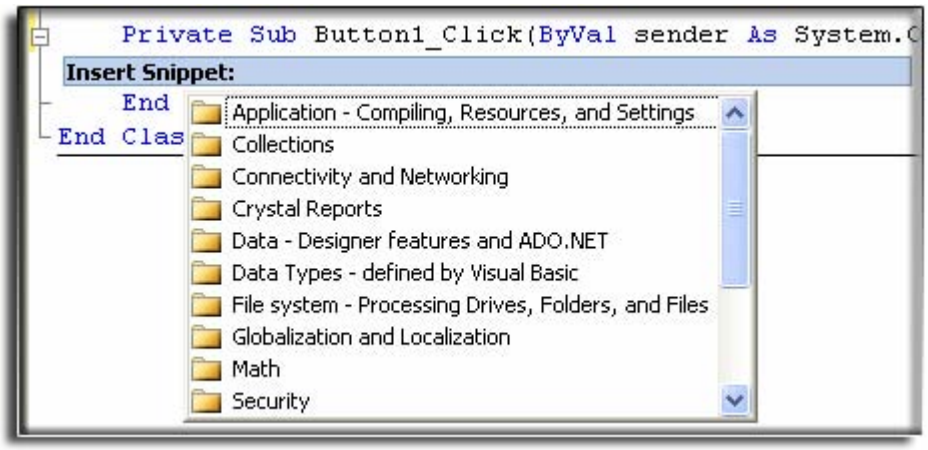
1. Vytvořte projekt standardní aplikace pro systém Windows (**Windows Application**).

2. Na aplikační formulář přidejte jednu instanci ovládacího prvku `Button`, která bude hrát roli tlačítka.

3. Když na tlačítko poklepáte, Visual Basic 2005 vygeneruje programovou kostru zpracovatele události `Click` tlačítka.

4. Kurzor myši nasměrujte do těla vytvořeného zpracovatele události a aktivujte pravé tlačítko myši.

5. V zobrazivší se kontextové nabídce klepněte na příkaz **Insert Snippet...**. Po aktivaci příkazu se v prostředí editoru zdrojového kódu objeví modrý čtyřúhelník s návěstím **Insert Snippet:**. Pod čtyřúhelníkem bude zobrazen seznam složek, v němž každá složka reprezentuje jednu skupinu, v níž se nacházejí šablony programového kódu podobného funkčního zaměření (obr. 2.21).

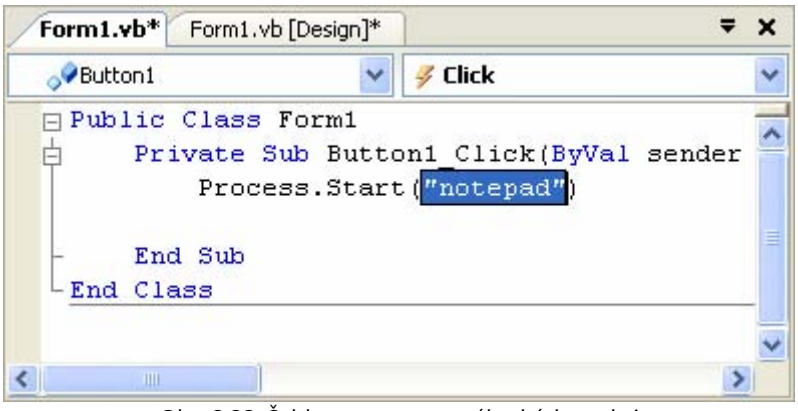


Obr. 2.21: Vkládání šablony programového kódu

6. Ze seznamu složek vyhledejte složku s názvem **Windows Operating System** a poklepejte na ni. Na obrazovce se objeví vnořený seznam složek, který sdružuje skupiny šablon programového kódu nižší úrovně.

7. Poklepejte na složku **Processes**, čímž se dostanete k třem šablonám programového kódu.

8. Ze seznamu dostupných šablon poklepejte na šablonu s názvem **Start an Application**. Visual Basic 2005 vloží do editoru zdrojového kódu kód specifikované šablony, který vypadá následovně:



Obr. 2.22: Šablona programového kódu v akci

9. Kód šablony volá sdílenou metodou `Start` třídy `Process` z jmenného prostoru `System.Diagnostics`, které předává název aplikace pro spuštění ve formě textového řetězce. Šablona implicitně počítá s tím, že budete chtít spustit aplikaci *Poznámkový blok*. Jestliže byste ovšem rádi spustili jinou aplikaci, můžete cestu k jejímu spustitelnému souboru (.exe) předat metodě `Start`. Ve skutečnosti je metoda `Start` třídy `Process` velmi flexibilní, neboť vám dovoluje iniciovat spuštění libovolné aplikace dle vašich požadavek.

10. Stiskněte klávesu **F5**, čímž vydáte povel na přeložení kódu a spuštění ukázkové aplikace. Jakmile klepnete na tlačítko, bude spuštěna vámi určená aplikace.

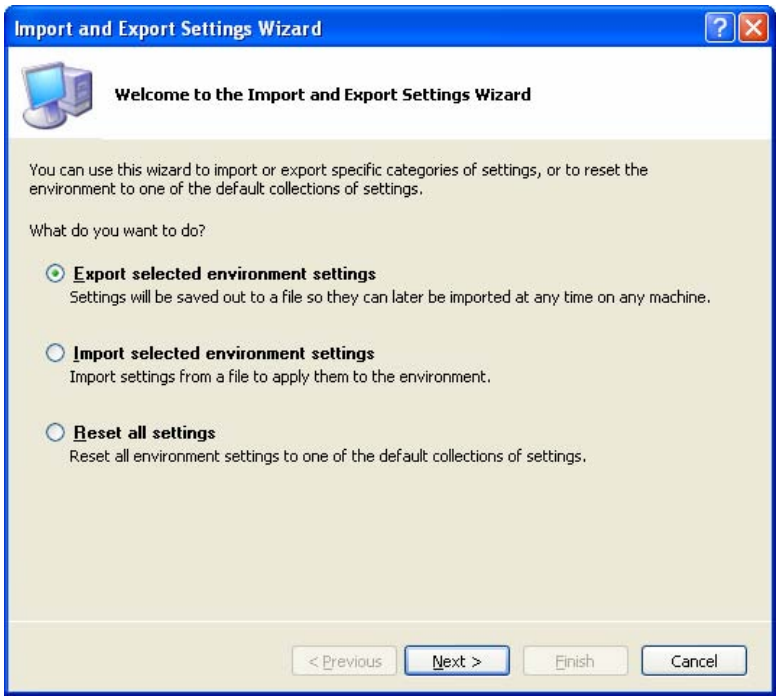
7. **Oddělovač programových elementů.** Editor zdrojového kódu analyzuje vámi zapsaný kód a mezi jednotlivé programové elementy, jakými jsou například procedury Sub, funkce nebo třídy, umísťuje horizontální oddělovače. Oddělovače jsou vizuálně reprezentovány tenkými linkami černé barvy. Podíváme-li se zpět do minulosti, zjistíme, že editor zdrojového kódu jazyka Visual Basic 6.0 nabízel programátorům dva pohledy na zdrojový kód: **Procedure View** a **Full Mode View**. Implicitně byl zvolen pohled **Full Mode View**, který zobrazoval všechny programové elementy umístěné v jednom modulu. Na druhé straně, pohled **Procedure View** umožňoval zobrazit v jeden okamžik pouze jeden programový element, tedy jednu proceduru Sub nebo jednu funkci. Mezi pohledy mohli programátoři přepínat pomocí dvou tlačítek, jež byly situovány v levém dolním rohu editoru zdrojového kódu. Oddělovače programových elementů byly dostupné již ve Visual Basicu 6.0, ovšem pouze tehdy, byl-li zvolen pohled **Full Procedure View**. (Tato skutečnost byla logická, protože při aktivaci pohledu **Procedure View** mohl být vždy zobrazen pouze jeden programový element, a tudíž nebylo co „oddělovat".)

2.5 Exportování konfiguračních nastavení IDE Visual Studia 2005

Vysněnou novinkou sady vývojářských nástrojů Visual Studio 2005 je možnost uložit všechna konfigurační nastavení integrovaného vývojového prostředí do jednoho souboru s extenzí .vssettings. Tento konfigurační soubor obsahuje specifikované volby, které definují vzhled a způsob chování integrovaného vývojového prostředí Visual Studia 2005, a které lze aplikovat také na jiné instalace tohoto softwarového produktu. Řečeno jinými slovy, když provedete export nastavení integrovaného vývojového prostředí do souboru .vssettings, získáte přesný otisk stylu práce, podle něhož se IDE řídí. Získaný konfigurační soubor .vssettings můžete použít na jiných počítačových stanicích, na nichž je instalace produktu Visual Studio 2005 přítomna. Nastavení z konfiguračního souboru můžete importovat stejně snadno, jako jste provedli jejich export. Po importu nastavení bude IDE nakonfigurováno podle předdefinovaných voleb konfiguračního souboru. Tento proces je zcela

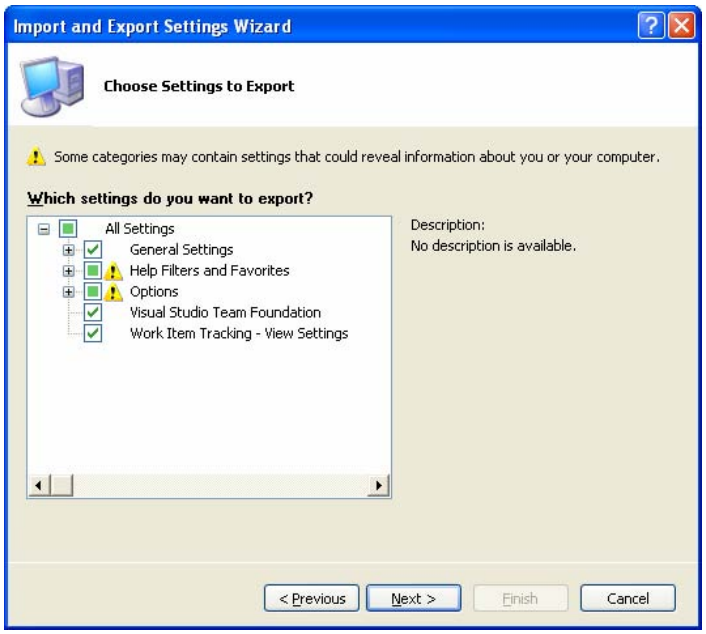
automatický, čehož důsledkem je ušetření značného množství času a energie, neboť se můžete okamžitě pustit do své práce a nemusíte se zabývat konfigurací pracovního prostředí. Budete-li chtít uložit konfigurační nastavení integrovaného vývojového prostředí Visual Studia 2005 do souboru .vssettings, postupujte následovně:

1. Jestliže jste tak ještě neučinili, spusťte Visual Studio 2005.
2. Otevřete nabídku **Tools** a klepněte na položku **Import and Export Settings...**.
3. Na obrazovce se objeví průvodce, jenž vám pomůže s exportem, respektive importem konfiguračních voleb IDE (obr. 2.23).



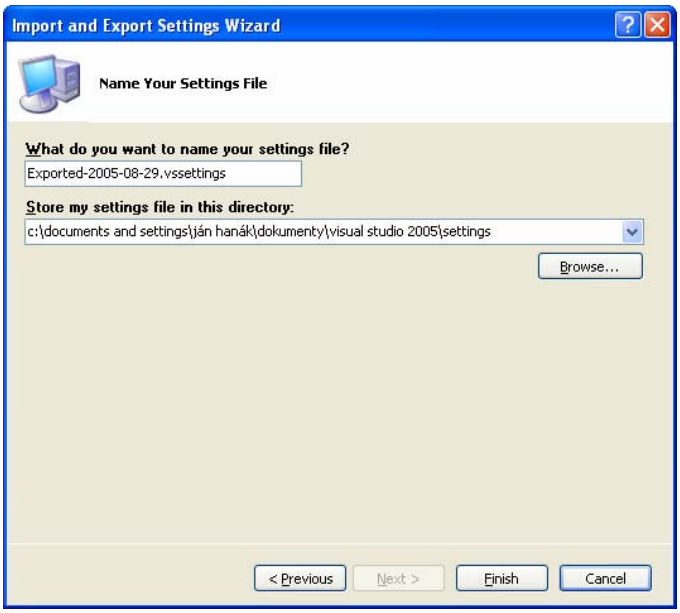
Obr. 2.23: Dialogové okno **Import And Export Settings Wizard**

Průvodce se vás ptá na váš záměr: konfigurační volby můžete jednak exportovat (**Export selected environment settings**), jednak importovat (**Import selected environment settings**), a také obnovit do původní podoby (**Reset all settings**). Jelikož nám jde o exportování nastavení, vybereme první volbu a klepneme na tlačítko **Next**. Ve druhém kroku po nás průvodce chce, abychom upřesnili ta nastavení, která hodláme exportovat (obr. 2.24).



Obr. 2.24: Výběr nastavení pro export

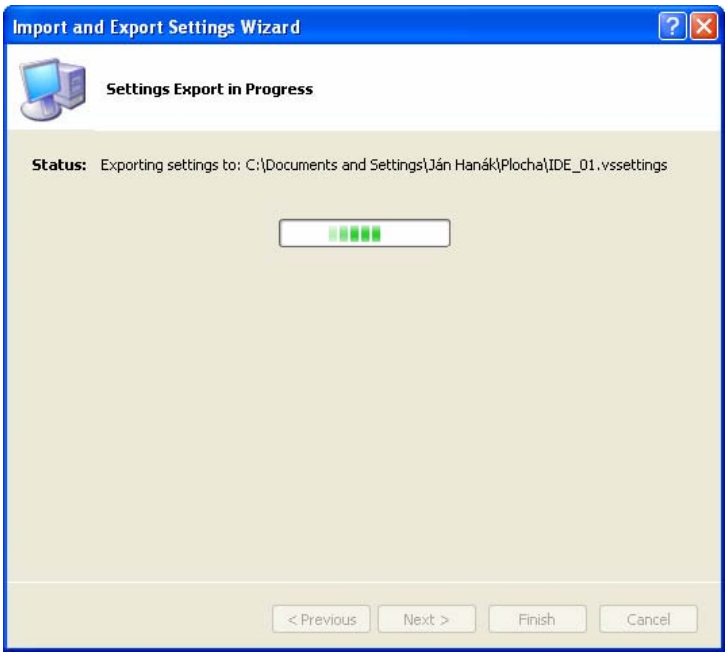
Ze stromové struktury, která se nachází pod návěstím **Which settings do you want to export?**, můžete vybrat všechna nastavení, jež byste rádi uložili do externího konfiguračního souboru .vssettings. Jak si můžete všimnout, máte možnost provádět selekci všech relevantních voleb, které se vážou k činnosti integrovaného vývojového prostředí Visual Studia 2005. Poté, co jste vymezili množinu požadovaných konfiguračních nastavení pro export, můžete přistoupit k určení názvu a lokace konfiguračního souboru. Tento výběr se odehrává ve třetím kroku průvodce (obr. 2.25).



Obr. 2.25: Určení pojmenování a zařazení konfiguračního souboru

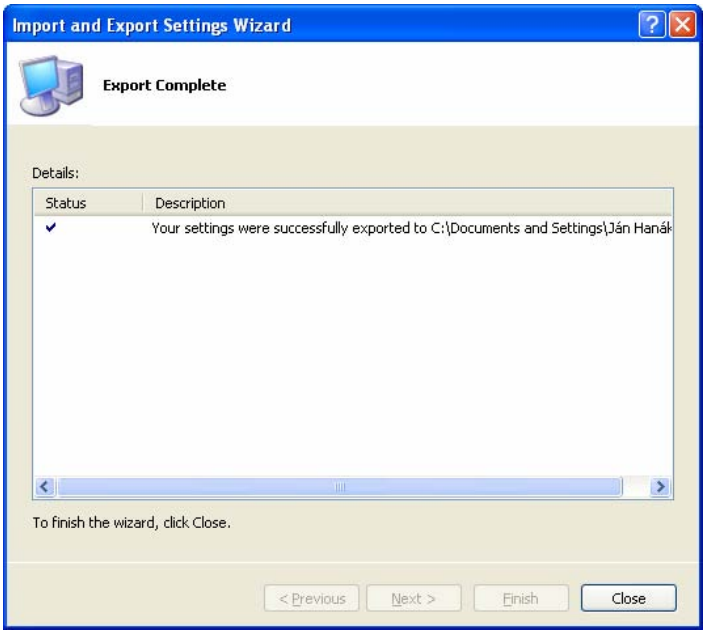
Zatímco název konfiguračního souboru můžete zapsat do textového pole **What do you want to name your settings file?**, pro určení umístění souboru na disku vám poslouží otevírací seznam **Store my settings file in this directory** s tlačítkem **Browse...**. V okamžiku, kdy jste s uskutečněnými modifikacemi spokojeni, můžete klepnout na tlačítko **Finish**. Tím zahájíte

proces exportování konfiguračních nastavení integrovaného vývojového prostředí Visual Studia 2005 do samostatného souboru s extenzí .vssettings (obr. 2.26).



Obr. 2.26: Probíhá export konfiguračních voleb do souboru

4. Jestliže celá operace proběhla úspěšně, integrované vývojové prostředí zobrazí informační zprávu, kterou můžete vidět na obr. 2.27.



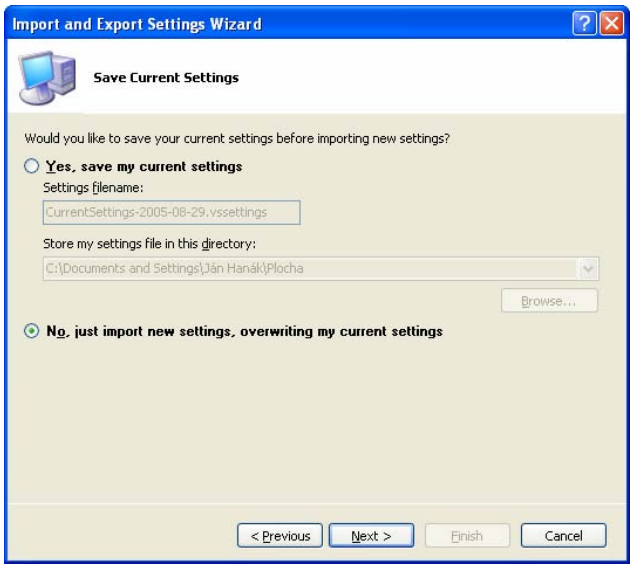
Obr. 2.27: Proces exportování dat do konfiguračního souboru .vssettings byl úspěšně ukončen

2.6 Importování konfiguračních nastavení IDE Visual Studia 2005

Když máte k dispozici konfigurační soubor .vssettings, v němž jsou uložena nastavení pro integrované vývojové prostředí produktu Visual Studio 2005, můžete data z tohoto souboru načíst do dalších instalací Visual Studia 2005. Může se přitom jednat o novou instalaci na stejném PC, anebo o stávající

instalace na jiných počítačových stanicích vývojářského týmu. Import konfiguračních nastavení provedete takto:

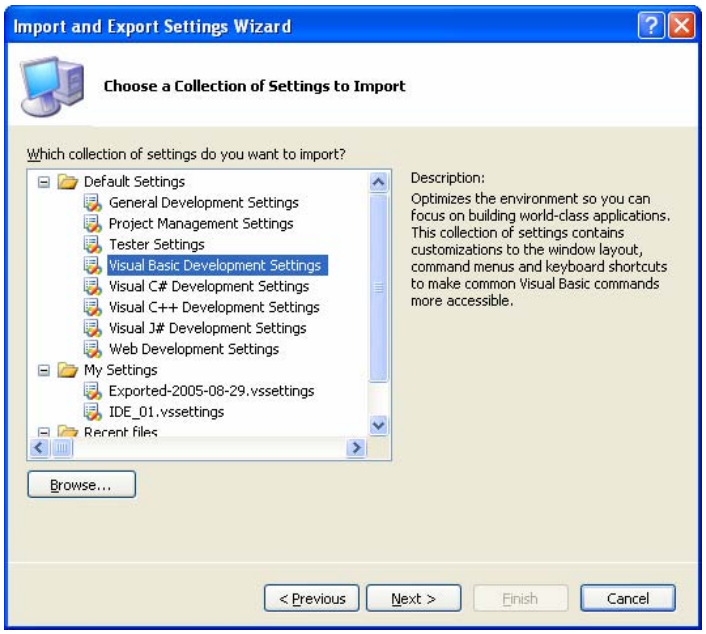
1. Na cílovém PC spusťte produkt Visual Studio 2005.
2. Rozviňte nabídku **Tools** a aktivujte příkaz **Import and Export Settings...**.
3. Jakmile se objeví průvodce, vyberte volbu **Import selected environment settings** a klepněte na tlačítko **Next**.
4. V dalším kroku se vás průvodce zeptá, zda chcete před importem nových konfiguračních nastavení zálohovat ta stávající (obr. 2.28).



Obr. 2.28: Otázka průvodce směřující k požadavku zálohování stávající konfigurace IDE

Pokud je vaše odpověď kladná, označte volbu **Yes, save my current settings** a zadejte název souboru společně s jeho umístěním na pevném disku. Naopak, nemáte-li zájem o uložení existující konfigurace IDE, můžete vybrat volbu **No, just import new settings, overwriting my current settings**. Je-li tato volba aktivní, průvodce nebude současnou konfiguraci integrovaného vývojového prostředí ukládat, ale jednoduše ji nahradí nově importovaným konfiguračním profilem. My zvolíme druhou možnost a klepneme na tlačítko **Next**.

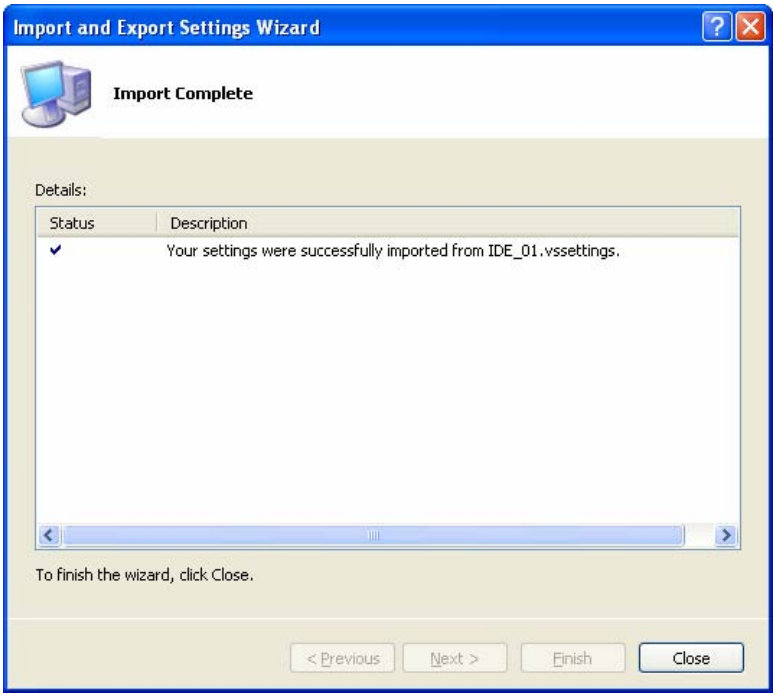
5. Posléze se dostáváme k nejdůležitější etapě procesu importování konfiguračních dat, která nese jméno **Choose a Collection of Settings to Import**. Průvodce chce vědět, kterou kolekci nastavení bychom rádi importovali. K dispozici jsou tři oblasti, z nichž můžeme vybírat. Tyto oblasti jsou prezentovány v přehledné stromové struktuře a jejich znění je následovní: standardní nastavení (uzel **Default Settings**), uživatelská nastavení (uzel **My Settings**) a posledně editované konfigurační soubory (uzel **Recent Files**). Vizuální ilustraci sady kolekcí znázorňuje obr. 2.29.



Obr. 2.29: Volba konfigurační kolekce

Možností volby je opravdu požehnaně. Kromě vestavěných konfiguračních profilů můžete využít své vlastní profily, anebo klepnout na tlačítko **Browse** a najít externí konfigurační soubor s příponou .vssettings. My zvolíme náš původní konfigurační soubor, který jsme získali v procesu exportování dat v předcházející podkapitole.

6. Poté stiskněme tlačítko **Next** a vybereme ta nastavení, která si přejeme importovat. Po klepnutí na tlačítko **Finish** dojde k načtení konfiguračních nastavení integrovaného vývojového prostředí Visual Studia 2005 ze specifikovaného souboru. Nová nastavení budou po svém načtení okamžitě aplikována a jestliže celý proces proběhne úspěšně, Visual Studio 2005 zobrazí potvrzující dialogové okno (obr. 2.30).



Obr. 2.30: Import konfiguračních nastavení ze souboru .vssettings se zdařil

POZNÁMKA: Obnovení původní konfigurace pracovního prostředí jazyka Visual Basic 2005

Jestliže budete chtít obnovit původní konfigurační nastavení pro pracovní prostředí jazyka Visual Basic 2005, můžete postupovat takto:

1. V dialogu **Import and Export Settings** zvolte možnost **Reset all Settings**.
2. V okně s návěstím **Choose a Default Collection of Settings** vyberte kolekci nastavení s názvem **Visual Basic Development Settings**.
3. Klepněte na tlačítko **Finish**.

Po této úpravě bude mít IDE Visual Basicu 2005 stejnou podobu jako při svém prvním spuštění s vybraným profilem pro tento programovací jazyk.

2.7 Charakteristika aplikací .NET, které lze vyvíjet v jazyce Visual Basic 2005

Programovací jazyk Visual Basic 2005 obsahuje kompletní podporu nástrojů pro vývoj, testování a distribuci moderních aplikací .NET využívajících dovedností vývojově-exekuční platformy Microsoft .NET Framework 2.0. Základním kamenem pro vývoj softwarových aplikací jsou projektové šablony, s kterými jste se již setkali při zakládání nového typu projektu. Jako programátoři přicházející z jazyka Visual Basic 6.0 byste měli vědět, že aplikace .NET, které sestrojíte ve Visual Basicu 2005, běží a pracují na zcela jiném principu než jejich protějšky napsané v minulé verzi tohoto jazyka. Ačkoliv podrobnější rozpravu na téma „Architektura a model exekuce aplikace .NET" si ponecháme až na třetí kapitolu, je důležité vědět, že na aplikace .NET není možné nahlížet pouze jako na další evoluci aplikací, jež jsou vám známy z prostředí jazyka Visual Basic 6.0. Platforma .NET Framework 2.0 ve společnosti s Visual Basicem 2005 přinášejí zbrusu nové technologické zázemí, které nelze srovnávat s předchozími modely vývoje počítačového softwaru. Důsledkem tohoto tvrzení je skutečnost, kterou si všimnete ihned, jakmile otevřete dialog pro založení nového aplikačního projektu: složení, počet a názvosloví projektových šablon se citelně změnily. Stupeň modifikací je dokonce tak vysoký, že nejeden migrující vývojář se může v novém prostředí cítit nejistě a zmateně. Abychom vnesli více světla do této problematiky, představíme si základní typy aplikací .NET, s nimiž budete pracovat v jazyce Visual Basic 2005 (tab. 2.3).

Tab. 2.3: Přehled základních aplikací .NET, které lze připravit v jazyce Visual Basic 2005

Název aplikace .NET	Ikona projektové šablony aplikace .NET	Charakteristika aplikace .NET
Windows Application (Standardní aplikace pro systém Windows)		Standardní aplikace pro systém Windows je pravděpodobně nejběžnějším typem aplikací .NET, které budete vytvářet. Jde o aplikaci s přepracovaným grafickým uživatelským rozhraním, jehož podstatnými součástmi jsou formuláře a instance ovládacích prvků a komponent. Standardní aplikace pro systém Windows může vystupovat jako samostatní entita, anebo může být propojena s webovými aplikacemi či službami. Programový kód standardní aplikace pro systém Windows je uložen v spustitelném souboru (.exe). Exekuce kódu je podřízena bezpečnostním pravidlům vývojově-exekuční platformy Microsoft .NET Framework 2.0. Nově lze standardní aplikaci pro systém Windows nakonfigurovat tak, aby byla schopna se sama v pravidelných časových intervalech aktualizovat prostřednictvím sítě Internet.

Class Library (Knihovna tříd)		Pokud budete chtít vytvořit knihovnu tříd pro opětovné použití v různých projektech, můžete sáhnout po této projektové šabloně aplikace .NET. Na rozdíl od standardní aplikace pro systém Windows (**Windows Application**), nedisponuje tento aplikační projekt žádným grafickým uživatelským rozhraním. Jedná se tedy o monolitní celek programového kódu, který řeší jistý vymezený programátorský úkol. Jednou sestrojenou knihovnu tříd mohou vývojáři používat v dalších projektech, anebo ji sdílet s kolegy či přáteli. Finálním produktem projektové šablony **Class Library** je soubor dynamicky linkované knihovny (.dll), jehož interní struktura se liší od klasických knihoven DLL, které můžete znát z prostředí Win32 API.
Windows Control Library (Knihovna ovládacích prvků pro systém Windows)		Projektová šablona s názvem **Windows Control Library** umožňuje vývojářům vytvářet uživatelské ovládací prvky, které lze používat v prostředí standardních aplikací pro systém Windows. Programátoři mohou své vlastní ovládací prvky zhotovovat podle různých scénářů, kupříkladu mohou kombinovat a rozšiřovat stávající ovládací prvky, nebo napsat zcela nové ovládací prvky. Ovládací prvky zpravidla disponují grafickým uživatelským rozhraním a lze je umísťovat do soupravy nástrojů **Toolbox** integrovaného vývojového prostředí jazyka Visual Basic 2005. Všechny nezbytné součásti této aplikace .NET jsou uloženy v sestavení (assembly), které má podobu dynamicky linkované knihovny (.dll). Knihovny ovládacích prvků pro systém Windows mohou být opětovně používány a sdíleny, což ve velké míře napomáhá opětovné použitelnosti jednou napsaného programového kódu.
Web Control Library (Knihovna ovládacích prvků pro webové aplikace)		Zatímco prostřednictvím výše charakterizované projektové šablony můžete sestavovat vlastní ovládací prvky pro použití ve standardních aplikacích pro systém Windows, šablona s názvem **Web Control Library** vám nabízí vše potřebné pro vývoj ovládacích prvků, které budou využity v prostředí webových aplikací běžících s pomocí softwarové technologie ASP.NET 2.0. ASP.NET 2.0 je nástupkyní předchozí technologie ASP (Active Server Pages), přičemž poskytuje přehršel inovativních prvků, které přispívají k snadnějšímu vývoji, ladění, testování a správě webových aplikací pracujících na vzdálených serverech. Navenek vystupuje knihovna ovládacích prvků pro webové aplikace jako dynamicky linkovaná knihovna (.dll). Samozřejmostí je opětovná použitelnost této knihovny DLL v jiných aplikacích, jimiž jsou především aplikace .NET s grafickým uživatelským rozhraním běžící na webovém serveru.

Console Application (Konzolová aplikace)		Konzolová aplikace představuje jednoduchou aplikaci bez grafického uživatelského rozhraní, která běží v textovém režimu příkazového řádku. Konzolová aplikace se proto velice ponáší na aplikace systému MS-DOS, ovšem nenechte se mýlit. Všechny aplikace, které vytvoříte za asistence projektové šablony **Console Application**, mohou využívat všech dovedností bázové knihovny tříd a společného běhového prostředí platformy Microsoft .NET Framework 2.0. Konzolová aplikace může spolupracovat rovněž s jinými typy aplikací .NET, standardní aplikace pro systém Windows a webové aplikace nevyjímaje. Vzhledem k tomu, že konzolová aplikace je spouštěna z příkazového řádku, lze jí předávat textové parametry, které mohou předem definovaným způsobem ovlivňovat algoritmus jejího běhu. Programový kód konzolové aplikace je přetransformován do samostatného spustitelného souboru. V neposlední řadě je potřebné zmínit, že konzolová aplikace je vhodným startovním bodem pro studium programovacího jazyka, neboť programátoři se mohou plně soustředit na psaní kódu a nemusejí se zabývat tvorbou grafického uživatelského rozhraní.
Windows Service (Služba systému Windows)		Služba systému Windows je speciální typ softwarové aplikace, který se vyznačuje následujícími charakteristikami: 1. Služba systému Windows je dlouho běžící aplikace .NET, jejíž životní cyklus je úzce spojen se životním cyklem operačního systému. Služba je zpravidla spouštěna automaticky po startu relace operačního systému a ukončena těsně před tím, než dojde k ukončení relace operačního systému. 2. Služba systému Windows nedisponuje grafickým uživatelským rozhraním, a tudíž nemůže s uživatelem přímo komunikovat. Služba však může zasílat uživateli informační zprávy, které mohou být archivovány v souborech s protokoly. 3. Služba systému Windows může být pozastavena a opětovně rozběhnuta. Služba může být spouštěna buď automaticky operačním systémem, nebo manuálně prostřednictvím uživatele. 4. Služba systému Windows může pracovat podle bezpečnostních nařízení specifického uživatelského účtu.

		Vzhledem ke své povaze se služby systému Windows využívají nejčastěji pro monitorování událostí a analyzování výkonnostních charakteristik operačního systému a aplikačních procesů, přičemž zjištěné výsledky jsou ukládány do vymezených souborů pro další zpracování. Problematika správy služeb systému Windows patří do „vyšší školy" programování v jazyce Visual Basic 2005, ovšem při řešení jistých úkolů jsou služby systému Windows nepostradatelné.

2.8 Převodový můstek: Aplikace jazyka Visual Basic 6.0 a jejich protějšky v jazyce Visual Basic 2005

Když budete zakládat nový aplikační projekt v jazyce Visual Basic 2005, zcela jistě si povšimnete, že názvy a zaměření jednotlivých projektových šablon se od dob Visual Basicu 6.0 změnily. Jelikož je cílem této příručky usnadnit vám přechod na jazyk Visual Basic 2005, uvádíme níže převodový můstek, který porovnává vybrané typy softwarových aplikací mezi jazyky Visual Basic 6.0 a Visual Basic 2005. Pomocí převodového můstku byste měli získat základní představu o modifikacích v oblasti projektových šablon a aplikací, které přináší Visual Basic 2005. Převodový můstek by vám měl rovněž sloužit jako průvodce, jenž poukazuje na možné varianty postupu při výběru a plánování nového softwarového projektu v jazyce Visual Basic 2005.

Tab. 2.4: Převodový můstek: Komparace aplikací jazyka Visual Basic 6.0 a jejich protějšků v jazyce Visual Basic 2005

Aplikace jazyka Visual Basic 6.0	Ekvivalentní aplikace jazyka Visual Basic 2005	Poznámky
 Standard EXE	 Windows Application	Projektová šablona standardní spustitelné aplikace jazyka Visual Basic 6.0 sloužila pro vytváření formulářových aplikací s grafickým uživatelským rozhraním. Aplikace typu **Standard EXE** mohly implementovat tři různé druhy rozvržení svých dokumentů: SDI, MDI a dialogové rozvržení. V prostředí nové verze Visual Basicu je standardní spustitelná aplikace nahrazena standardní aplikací pro systém Windows, kterou můžete sestrojit pomocí projektové šablony **Windows Application**. Pokud tedy budete chtít vytvářet formulářové aplikace s bohatým grafickým uživatelským prostředím, zvolte tuto projektovou šablonu jazyka Visual Basic 2005.
Standardní spustitelná aplikace	Standardní aplikace pro systém Windows	

<table>
<tr><td> 
ActiveX EXE</td><td> 
Class Library</td><td rowspan="2">V jazyce Visual Basic 6.0 mohli vývojáři prostřednictvím projektové šablony ActiveX EXE vytvářet mimoprocesovou serverovou komponentu pracující na bázi technologie ActiveX. Tato komponenta byla načítána do svého vlastního procesu a mohla poskytovat své služby několika na sobě zcela nezávislým klientským aplikacím. Programový kód mimoprocesové serverové komponenty ActiveX byl uložen v samostatném spustitelném souboru (.exe), což znamenalo, že aktivace komponenty mohla být kromě klientské aplikace iniciována také uživatelem. Jazyk Visual Basic 2005 vytváření mimoprocesových komponent ActiveX nepodporuje. Místo nich mohou vývojáři ukládat kód svých tříd do jednotné knihovny tříd (projektová šablona Class Library), kterou může sdílet větší počet klientských aplikací .NET.</td></tr>
<tr><td>Komponenta ActiveX</td><td>Knihovna tříd</td></tr>
<tr><td> 
ActiveX DLL</td><td> 
Class Library</td><td rowspan="2">Pomocí projektové šablony ActiveX DLL bylo ve Visual Basicu 6.0 možné sestrojit vnitroprocesovou serverovou komponentu, která využívala schopností a dovedností technologie ActiveX. Kód této serverové komponenty byl uložen v dynamicky linkované knihovně (.dll). Když klientská aplikace vygenerovala požadavek na použití vnitroprocesové komponenty ActiveX, kód příslušné knihovny DLL byl načten do procesu klientské aplikace. Podobně se komponenta chovala i ve vztahu k jiným klientským aplikacím, což znamenalo, že součástí procesu každé klientské aplikace byl také kód dynamicky linkované knihovny vnitroprocesové komponenty ActiveX. Jazyk Visual Basic 2005 vytváření vnitroprocesových ActiveX komponent nepodporuje. Vhodnou substitucí se jeví použití projektové šablony Class Library, jejíž pomocí vytvoříte knihovnu tříd, která může být opětovně používána a sdílena mnoha aplikacemi .NET.</td></tr>
<tr><td>Komponenta ActiveX</td><td>Knihovna tříd</td></tr>
<tr><td> 
ActiveX Control</td><td> 
Windows Control Library</td><td rowspan="2">Ovládací prvky ActiveX byly v jazyce Visual Basic 6.0 velice oblíbené, neboť umožňovaly zapouzdřovat programovou funkcionalitu do samostatných softwarových entit s grafickým uživatelským rozhraním, které bylo možné vkládat do projektů standardních spustitelných aplikací. Jakmile byl ovládací prvek ActiveX vložen do projektu uvedeného typu, rozšířil portfolio ovládacích prvků v soupravě nástrojů Toolbox. Vývojáři pak mohli s ovládacím prvkem ActiveX pracovat stejně jako s mnoha jinými dostupnými prvky. Ačkoliv Visual Basic 2005 již nepracuje s technologií ActiveX, také v tomto prostředí můžete vytvářet své vlastní ovládací prvky. Skvělým startovním bodem je projektová šablona Windows Control Library, která představuje základní kámen pro programování nového ovládacího prvku. Hotový ovládací prvek můžete sdílet s mnoha vývojáři, dokonce i s těmi, kteří při své práci používají jiný programovací jazyk než Visual Basic 2005.</td></tr>
<tr><td>Ovládací prvek ActiveX</td><td>Knihovna ovládacích prvků pro systém Windows</td></tr>
</table>

<table>
<tr>
<td> 
</td>
<td> 
Visual Basic 2005 nedisponuje žádným přímým ekvivalentem. Můžete použít projektovou šablonu ASP.NET Web Site.

</td>
<td rowspan="2">Jestliže jste chtěli v prostředí jazyka Visual Basic 6.0 vyvíjet internetové aplikace běžící na straně serveru, měli jste k dispozici projektovou šablonu IIS Application. Již z názvu je patrné, že tento typ aplikací intenzivně využíval služeb Internet Information Serveru, jenž musel být na webové počítačové stanici nainstalován. Aplikace běžící na serveru komunikovala s klienty na základě přijímání, zpracování a odesílání požadavků. Internetová aplikace IIS Application implementovala technologii Active Server Pages (ASP), jejíž pomocí bylo možné realizovat programové operace ve vícevrstvém modelu typu klient-server. Jazyk Visual Basic 2005 také podporuje vývoj webových aplikací, ovšem na kvalitativně vyšší úrovni, kterou zabezpečuje implementace technologie ASP.NET 2.0. ASP.NET 2.0 umožňuje vývojářům vytvářet dynamické webové aplikace, jež běží na webovém serveru. Ačkoliv by se mohlo zdát, že technologie ASP.NET 2.0 je pouhým evolučním produktem dřívější ASP, není tomu tak. Inovace ve vývoji webových aplikací jsou natolik hluboké, že jejich podrobný výčet by zabral celou knihu. Pro programátory je ovšem důležitá skutečnost, že technologie ASP.NET 2.0 jim nabízí širší možnosti ve vývoji, ladění, testování, distribuci a správě webových aplikací a XML webových služeb pracujících na základech vývojově-exekuční platformy Microsoft .NET Framework 2.0.</td>
</tr>
<tr>
<td>Internetová aplikace využívající IIS</td>
<td>Webová aplikace využívající ASP.NET 2.0</td>
</tr>
</table>

<table>
<tr>
<td> 
</td>
<td> 
Visual Basic 2005 nedisponuje žádným přímým ekvivalentem.

Můžete použít projektovou šablonu Visual Studio Add-in (pro vytvoření doplňku pro Visual Studio) anebo projektovou šablonu Shared Add-in (pro vytvoření sdíleného doplňku).

 
</td>
<td rowspan="2">Rozšiřování integrovaného vývojového prostředí jazyka Visual Basic 6.0 bylo možné realizovat vytvářením samostatných softwarových součástek, jimž se říkalo doplňky (add-ins). Doplněk byl ve své podstatě kompilovanou serverovou komponentou ActiveX, která mohla působit buď uvnitř procesu hostitelské aplikace (.dll) anebo ve svém vlastním procesu (.exe). Visual Basic 6.0 rozlišoval několik typů doplňků, generickými doplňky počínaje a průvodci (wizards) konče. Proces vývoje doplňku byl těsně navázán na rozšiřitelný objektový model IDE Visual Basicu 6.0. Tak bylo možné vytvářet nová dialogová okna, panely s tlačítky či programově reagovat na různé události, k nimž v integrovaném vývojovém prostředí docházelo. Ačkoliv doplněk bylo možné sestavit i pomocí projektové šablony pro komponenty ActiveX, Visual Basic 6.0 nabízel vývojářům samostatnou šablonu s názvem Addin. Výhodou použití této šablony bylo přednastavené vývojové prostředí a snadný přístup k pokročilým konfiguračním nastavením. Pokud jste si oblíbili vytváření doplňků, pravděpodobně budete zvědaví, zda můžete tyto softwarové součástky vytvářet také v prostředí jazyka Visual Basic 2005. Ano, je to možné, což je jistě dobrá zpráva. V portfoliu projektových šablon byste ovšem tu pro vývoj doplňků hledali marně: Je to způsobeno tím, že nyní lze doplňky zhotovovat nejenom pro Visual Basic, ale také pro další programovací nástroje, které sídlí v IDE Visual Studia 2005. Pro vývoj tohoto typu programového doplňku je k dispozici šablona Visual Studio Add-in, po jejímž výběru se aktivuje přehledný průvodce, který vás provede procesem založení nového doplňku stylem „krok za krokem“. Pomocí Visual Basicu 2005 ovšem můžete jít ještě dál a pustit se rovněž do vývoje sdílených programových doplňků, které bude možné využít také v prostředí jiných aplikací (vyjma Visual Studia 2005) jako je třeba Microsoft Office Word 2003 nebo Microsoft Office Excel 2003. Rozhodnete-li se pro vývoj sdíleného doplňku, můžete si zvolit projektovou šablonu Shared Add-in.</td>
</tr>
<tr>
<td>Doplněk pro Visual Basic 6.0</td>
<td>Doplněk pro Visual Studio, Sdílený doplněk</td>
</tr>
</table>

<table>
<tr><td> 


</td><td> 
 </td><td rowspan="2">Podpora technologie ActiveX byla v jazyce Visual Basic 6.0 opravdu pečlivě implementována: Kromě kompilovaných serverových komponent ActiveX a ovládacích prvků ActiveX měli programátoři možnost vytvářet rovněž dokumenty ActiveX. Dokumenty ActiveX byly ve své podstatě inteligentní objekty, které mohly být umísťovány do hostitelských aplikací typu Microsoft Internet Explorer nebo Microsoft Office Binder. Dokumenty ActiveX se v jazyce Visual Basic 6.0 lišily stupněm zapouzdření, neboť mohly být ztělesňovány jak dynamicky linkovanými knihovnami ActiveX (.dll), tak samostatně spustitelnými soubory (.exe).
V prostředí jazyka Visual Basic 2005 se s dokumenty ActiveX již nesetkáte. Důvodem je skutečnost, že nová verze jazyka Visual Basic nepodporuje přímé vytváření dokumentů ActiveX, a proto ani neposkytuje žádnou adekvátní projektovou šablonu. Jestliže tedy budete chtít i nadále vytvářet své dokumenty ActiveX, budete muset zůstat u Visual Basicu 6.0.</td></tr>
<tr><td>Dokumenty ActiveX</td><td>Žádný přímý ekvivalent</td></tr>
<tr><td rowspan="2"> 
</td><td> 
Visual Basic 2005 nedisponuje žádným přímým ekvivalentem. Můžete použít projektovou šablonu **ASP.NET Web Site**.
 
</td><td rowspan="2">Aplikace DHTML patřily společně s IIS aplikacemi do množiny internetových aplikací, které mohli vývojáři v jazyce Visual Basic 6.0 vyvíjet. Mezi uvedenými typy internetových aplikací ovšem existovaly zásadní rozdíly. Zatímco IIS aplikace reprezentovaly webové aplikace založené na technologii ASP a běžící na Internet Information Serveru, DHTML aplikace byly aplikacemi pracujícími na straně klienta (mohly však spolupracovat také se vzdáleným webovým serverem). DHTML neboli dynamické HTML umožňovalo prostřednictvím programových skriptů reagovat na události v prohlížeči WWW, měnit formátování elementů stránek HTML a flexibilně přizpůsobovat rozvržení těchto stránek potřebám finálních uživatelů. Startovním bodem jazyka Visual Basic 6.0 pro vývoj aplikací DHTML byla projektová šablona **DHTML Application**. Ačkoliv Visual Basic 2005 nenabízí přímý ekvivalent šablony **DHTML Application**, vývojáři v tomto prostředí mohou vytvářet webové aplikace, jež pohání technologie ASP.NET 2.0.</td></tr>
<tr><td>Internetová aplikace využívající DHTML</td><td>Webová aplikace využívající ASP.NET 2.0</td></tr>
</table>

Kapitola 3.: Architektura vývojově-exekuční platformy Microsoft .NET Framework 2.0

Vývojově-exekuční platforma Microsoft .NET Framework 2.0 představuje komplexní rámec pro vytváření, ladění, testování, exekuci, distribuci a správu novodobých aplikací .NET. Softwarovou platformu .NET Framework 2.0 si můžete představit jako sofistikované prostředí, v němž se uskutečňují všechny důležité operace, které se pojí s celým životním cyklem počítačových aplikací vyhovující standardům technologie .NET. Jde především o vývoj aplikací pomocí .NET-kompatibilních programovacích nástrojů, jejich běh v chráněném, tzv. řízeném prostředí, jako i další služby, mezi nimiž nechybí automatická správa alokované paměti či dohlížení na bezpečnou exekuci programových instrukcí aplikací. Platforma .NET Framework 2.0 byla zkonstruována tak, aby mohla nejenom výrazně pozvednout kvalitativní úroveň vývoje moderních aplikací 21. století, ale také aby byla schopna vyřešit drtivou většinu palčivých omezení, s nimiž se vývojáři softwarových aplikací po dlouhý čas potýkali. Návrháři společnosti Microsoft ve svých laboratořích vytvořili skutečně objemný mechanizmus, jenž má všechny předpoklady pro to, aby učinil mnohdy náročný a složitý proces vývoje, testování a správy aplikací flexibilnějším a lépe zvládnutelným. Navíc, platforma .NET Framework 2.0 přináší skutečnou softwarovou interoperabilitu, která může být uplatněna tak v prostředí jednotlivých vývojových nástrojů, jako i napříč různými technologickými platformami. Již není problémem, aby aplikace .NET mezi sebou úzce spolupracovaly, vyměňovaly si důležité informace a ty následně zpracovávaly. Díky platformové interoperabilitě mohou řízené aplikace dokonce bezproblémově komunikovat s komponenty COM, nativními funkcemi Win32 aplikačního programového rozhraní či „starými" dynamicky linkovanými knihovnami napsanými v jazycích C a C++.

Problematiku vývoje aplikací ovšem není možné v dnešních časech zužovat pouze na tvorbu aplikací běžících na lokálních počítačových stanicích. Na všech frontech se totiž objevují požadavky na ničím neomezený přístup k informacím, a to kdykoliv a z jakéhokoliv místa na naší planetě. Je jenom pochopitelné, že vývojáři v Redmondu se rozhodli vyslyšet tento moderní trend a implementovali do platformy .NET Framework 2.0 inovativní technologii ASP.NET 2.0, s jejíž pomocí mohou vývojáři sestrojovat vyspělé webové aplikace a XML webové služby, ke kterým lze přistupovat skutečně odkudkoliv. Vskutku, onen důraz na distribuované zpracování dat je v prostředí platformy .NET Framework 2.0 patrný v daleko větší míře, než tomu bylo kdykoliv předtím. Vývojáři mají na dosah ruky nástroje, se kterými se mohou směle pustit do vytváření i těch nejnáročnějších vícevrstvých korporativních aplikací, které budou běžet v kritickém prostředí podnikových serverů. Webové aplikace se těší pořád většímu zájmu, a proto se nelze divit, že právě vytváření aplikací .NET, jež pohání technologie ASP.NET 2.0, se stane pro mnoho vývojářů jejich denním chlebem.

Ti z vás, kdo sledují vývoj v informačních technologiích, určitě postřehli, že novým fenoménem se stávají inteligentní mobilní zařízení. Tvůrci platformy .NET Framework 2.0 nezapomněli ani na tuto kategorii počítačových pomocníků, a přivedli na svět vývojově-exekuční platformu .NET Compact Framework, která je zacílena primárně na „malá a chytrá PC". Z technologického hlediska je .NET Compact Framework podmnožinou platformy .NET Framework 2.0, která disponuje nástroji pro vývoj aplikací .NET pro systémy Windows Mobile for Pocket PC či Windows CE .NET. Platforma .NET Framework 2.0 ovšem potěší také příznivce programování počítačové grafiky, neboť definuje zcela nový grafický subsystém GDI+ pro práci s 2D-grafikou. Fanoušci 3D-multimediálních aplikací a počítačových her si zase oblíbí vestavěnou podporu pro vytváření dech beroucích aplikací využívajících všech dovedností progresivního grafického aplikačního rozhraní DirectX 9. Oblastí použití aplikací .NET a vývojově-exekuční platformy Microsoft .NET Framework 2.0 je skutečně mnoho a kdybychom je měli všechny postihnout, jistě bychom popsali nespočet stránek papíru. Pro programátory a IT specialisty je důležité vědět, že prostřednictvím platformy .NET Framework 2.0 dostávají do svých rukou prozatím

nejdokonalejší prostředí pro vývoj moderních softwarových jednotek. Je přitom zcela nepodstatné, které oblasti programování se právě věnujete – platforma .NET Framework 2.0 vám nabízí vše, co ke své práci skutečně potřebujete.

3.1 Charakteristika a architektura vývojově-exekuční platformy Microsoft .NET Framework 2.0

Platforma .NET Framework 2.0 definuje řízené, typově bezpečné softwarové prostředí, v němž se uskutečňuje vývoj a exekuce aplikací .NET. Při bližším pohledu na architektonickou strukturu této vývojově-exekuční platformy zjistíme, že je tvořena níže uvedenými komponentami:

1. **Společné běhové prostředí (Common Language Runtime, CLR).** Společné běhové prostředí neboli běhové prostředí CLR představuje prostor, v němž se uskutečňuje exekuce softwarových aplikací, které vyhovujících kriteriím technologie .NET. Společné běhové prostředí je odpovědné za realizaci všech fundamentálních nízkoúrovňových softwarových operací, které jsou pro běh aplikací .NET nepostradatelné. Jedná se především o kompilaci programového kódu, alokaci paměťových jednotek, správu programových vláken a procesů a automatickou kontrolu životních cyklů vytvořených softwarových objektů. Jelikož každá aplikace .NET běží uvnitř společného běhového prostředí, říkáme, že aplikace je tímto prostředím řízena. Všechny aplikace, které napíšete v jazyce Visual Basic 2005, potřebují pro svůj bezproblémový chod běhové prostředí CLR, a jsou tudíž charakterizovány jako řízené aplikace. Všechny řízené aplikace jsou pod správou a neustálou kontrolou společného běhového prostředí, které těmto aplikacím poskytuje v pravý čas požadované služby. Vztah mezi aplikací .NET a společným běhovým prostředím je natolik silný, že jednoduše není možné zabezpečit start této aplikace, aniž by bylo aktivováno běhové prostředí CLR.

2. **Společná knihovna tříd (.NET Framework 2.0 Class Library, FCL).** Společná knihovna tříd vývojově-exekuční platformy .NET Framework 2.0 (někdy také nazývána bázová knihovna tříd) představuje hierarchicky organizovanou soustavu datových typů, které mohou programátoři při vytváření svých aplikací použít. Všechny datové typy, které jsou přítomny v společné knihovně tříd, se dělí do dvou základních skupin: hodnotové datové typy a odkazové (neboli referenční) datové typy. Datové typy, které jsou sdruženy v společné knihovně tříd byly navrženy s důrazem na jejich objektově orientovanou povahu, efektivnost použití, snadnou rozšiřitelnost a jazykovou interoperabilitu. Důsledkem takto promyšleného návrhu je skutečnost, že všechny datové typy lze bez potíží použít z jakéhokoliv .NET-kompatibilního vývojového prostředí, ať už se jedná o Visual Basic 2005, C# nebo C++/CLI. Knihovna tříd platformy .NET Framework 2.0 definuje několik tisíc datových typů, které pokrývají širokou paletu vývoje softwaru. Pro lepší představu vzpomeňme třídy pro práci s ovládacími prvky a komponenty, programování rovinné počítačové grafiky, přístup k bázovým aplikačním elementům jakými jsou procesy a vlákna či třídy pro komunikaci v síťovém prostředí. Jelikož je dostupných datových typů skutečně veliké množství, jsou všechny pro lepší přehlednost a rychlejší orientaci členěny do tzv. jmenných prostorů neboli prostorů jmen. Jmenné prostory jsou interně homogenní soubory typů s podobnou funkcionalitou, které se vzájemně doplňují. Jmenné prostory mohou být navzájem vnořovány, což znamená, že jeden jmenný prostor může být umístěn v jiném jmenném prostoru.

   Datové typy, jejichž definice je uložena ve společné knihovně tříd platformy .NET Framework 2.0, jsou přístupné z kterékoliv .NET-programovacího jazyka. Řečeno jinými slovy, dostupné datové typy mohou být sdíleny mezi libovolnými jazyky, které vyhovují standardům technologie .NET. Možnost využívat univerzální datové typy napříč různými aplikacemi .NET

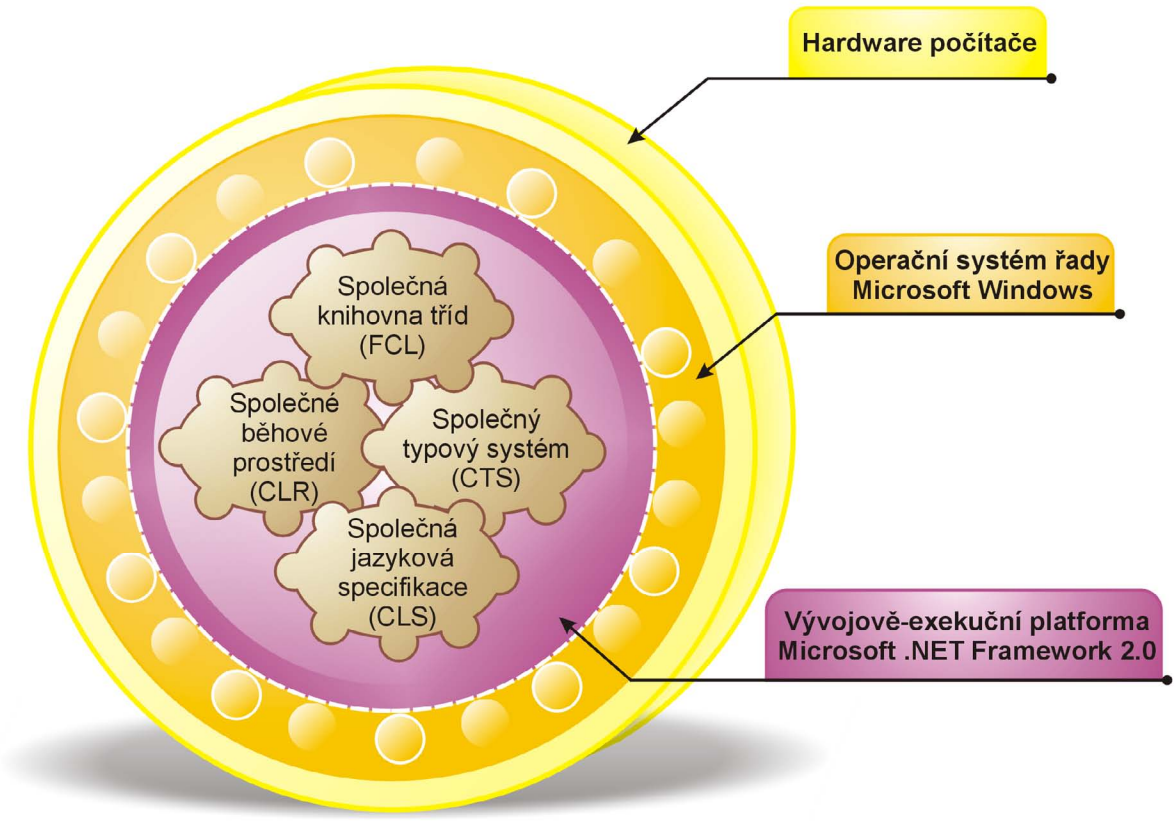
a programovacími jazyky, je prvním, ovšem velice důležitým krůčkem k naplnění vize skutečné jazykové a typové interoperability.

3. **Společná jazyková specifikace (Common Language Specification, CLS).** Společná jazyková specifikace vývojově-exekuční platformy Microsoft .NET Framework 2.0 definuje soubor syntaktických a sémantických pravidel a standardů, kterým musí vyhovovat všechny .NET-kompatibilní vývojové nástroje a jejich kompilátory. Jestliže jistý programovací jazyk splňuje kriteria daná společnou jazykovou specifikací, může být označen jako .NET-kompatibilní. Aplikace .NET vytvořené v tomto jazyce mohou nejenom využívat všech dovedností platformy .NET Framework 2.0, ale je rovněž garantováno, že se dokáží domluvit s dalšími softwarovými aplikacemi, které byly vytvořeny prostřednictvím jiných .NET-kompatibilních programovacích jazyků. Společná jazyková specifikace tak výrazně napomáhá opravdovému využití jazykové interoperability mezi aplikacemi či jejich částmi, které byly připraveny v různých vývojářských nástrojích platformy .NET Framework 2.0.

4. **Společný typový systém (Common Type System, CTS).** Společný typový systém charakterizuje soustavu datových typů, které mohou počítačové aplikace vytvořené v .NET-kompatibilních programovacích jazycích používat. Společný typový systém provádí následující činnosti:

    - vytváří hierarchickou klasifikaci řízených datových typů,
    - nařizuje, jak mají být vytvářeny a používány instance datových typů,
    - definuje unifikovanou sadu datových typů, které mohou být bez jakýchkoliv potíží sdíleny mezi různými programovacími jazyky operujícími na platformě .NET,
    - zabezpečuje typovou bezpečnost a jazykovou interoperabilitu,
    - přispívá k vyšší výkonnosti a efektivnější exekuci programových instrukcí řízených aplikací,
    - spolu se společnou jazykovou specifikací umožňuje provádět verifikaci datových typů,
    - je založen na plné podpoře objektově orientovaného programování, což znamená, že datové typy implementované pomocí CTS mohou využívat všech dovedností koncepce OOP (abstrakce, zapouzdření, skrývání dat, dědičnost, polymorfizmus, opětovná použitelnost programového kódu).

Grafickou ilustraci vzájemné interakce mezi jednotlivými součástmi platformy .NET Framework 2.0, operačním systémem řady Windows a hardwarem počítače můžete vidět na obr. 3.1.



Obr. 3.1: Ilustrace vztahu mezi vývojově-exekuční platformou .NET Framework 2.0, operačním systémem řady Windows a hardwarem počítače

Programovací jazyky vývojově-exekuční platformy .NET Framework 2.0, mezi nimiž má své nezastupitelné místo také Visual Basic 2005, mohou používat dvě základní skupiny datových typů, o kterých jsme se zmínili již při povídání o společné knihovně tříd. Jedná se o hodnotové a odkazové (referenční) datové typy. Na tomto místě je nutno poukázat na skutečnost, že definice uvedených primárních kategorií datových typů je uskutečňována právě prostřednictvím společného typového systému. Jestliže bude aplikace .NET využívat datové typy společného typového systému, respektive typy, jež vznikly odvozením od těchto typů, je zaručeno, že s takovýmito datovými typy budou moci na nízké úrovni pracovat i jiné řízené aplikace. Ano, mluvíme o typové kompatibilitě napříč aplikacemi a programovacími jazyky, která je veskrze důležitou součástí komplexní jazykové interoperability, což je jeden z aspektů vize .NET.

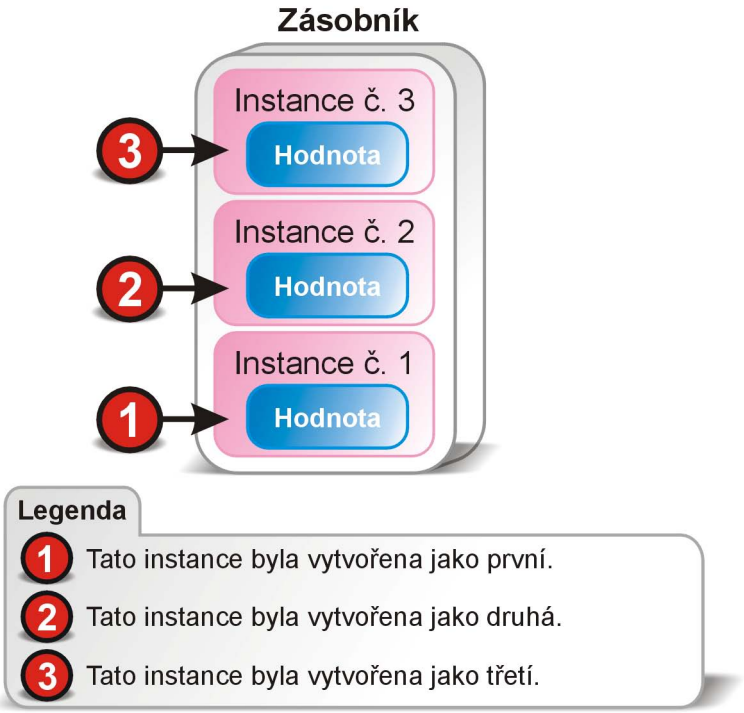
3.2 Hodnotové a odkazové datové typy

Pro programátory přicházející z prostředí jazyka Visual Basic 6.0 je naprosto nezbytné pochopit rozdíly mezi hodnotovými a odkazovými datovými typy, které definuje společný typový systém (CTS) a implementuje společná knihovna tříd (FCL). Obě skupiny datových typů jsou si podobné v tom smyslu, že charakterizují data, která můžete analyzovat, zpracovávat, konvertovat, nebo použít k provádění mnoha různorodých programových operací. Na druhé straně, hodnotové a odkazové typy se v mnoha oblastech zásadně liší, paměťovou reprezentací počínaje a způsobem vytváření a používání svých instancí konče. Na následujících řádcích si hodnotové a odkazové datové typy popíšeme podrobněji.

Obr. 3.2: Datové typy společného typového systému

3.2.1 Hodnotové datové typy

Instance hodnotových datových typů dovedou přímo uchovávat jistá data. Jakmile programátor deklaruje proměnnou hodnotového typu, vytváří tím ve skutečnosti instanci tohoto typu, do které mohou být explicitně uloženy hodnoty z definičního oboru determinovaného datového typu. Z tohoto pohledu můžeme pojmy „instance hodnotového typu" a „proměnná hodnotového typu" považovat za ekvivalentní. Pokaždé, když dojde k deklaraci proměnné hodnotového typu, dochází také k sestrojení instance tohoto typu. Instance hodnotového datového typu je ukládána do speciální vyhrazené sekce operační paměti, které se říká zásobník (stack). Zásobník reprezentuje automaticky spravovanou datovou strukturu, která pracuje na principu LIFO (Last-In-First-Out, Poslední-Dovnitř-První-Ven). Podle uvedeného modelu může být ze zásobníku vyjmuta pouze ta instance hodnotového typu, která byla na zásobník umístěna jako poslední. Správa zásobníku je nenáročná a velice rychlá, a proto je implicitní alokace instancí hodnotových typů na zásobníku z výkonnostního hlediska značně efektivní. Grafickou ilustraci zásobníku se třemi instancemi hodnotových datových typů můžete vidět na obr. 3.3.



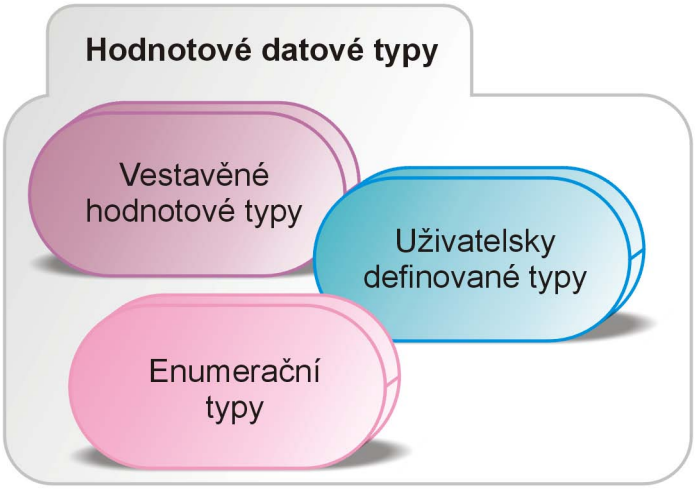
Obr. 3.3: Zásobník se třemi instancemi hodnotových datových typů

Struktura zásobníku odráží životní cykly jednotlivých instancí, které jsou na něm umístěné. Instance hodnotového typu, tedy proměnná, začíná svůj život tehdy, když ji programátor deklaruje. V tuto chvíli je instance hodnotového typu umístěna na zásobník programového vlákna. (V závislosti od stylu práce použitého .NET-kompatibilního programovacího jazyka může být instance hodnotového typu po své deklaraci implicitně inicializována, ovšem nejde o podmínku. Visual Basic 2005 však instance hodnotových typů implicitně inicializuje.) Je-li vytvořena další instance hodnotového typu, i tato je umístěna na zásobník, ovšem „nad" již přítomnou instanci. Jestliže se instance hodnotového typu dostane mimo svůj obor platnosti, je ze zásobníku odstraněna a její obsah je podroben destrukci. Jelikož likvidace instancí hodnotových typů je přesně definována, říkáme, že tyto instance podléhají deterministické finalizaci.

Dále byste si měli zapamatovat, že kdykoliv dojde k přiřazení hodnoty uložené v jedné instanci hodnotového typu do jiné instance hodnotového typu, kompilátor uskutečňuje kopírování hodnot, tedy samotných dat. Abychom si tuto situaci objasnili, povězme, že máme k dispozici dvě proměnné, `Proměnná1` a `Proměnná2`, přičemž v první proměnné je uložena jistá celočíselní hodnota. Když provedeme přiřazovací příkaz `Proměnná2 = Proměnná1`, bude nastartován kopírovací proces, který vytvoří kopii dat uložených ve zdrojové proměnné (`Proměnná1`) a tato data umístí do cílové proměnné (`Proměnná2`). Výsledkem přiřazovacího procesu bude naplnění druhé proměnné stejnými daty, jaká jsou uložena v první proměnné. Každá proměnná bude obsahovat „svá vlastní" data, takže proměnné budou na sobě zcela nezávislé. Jestliže se rozhodnete později modifikovat hodnotu proměnné `Proměnná1`, tato změna nijak neovlivní data, která jsou uložena ve druhé proměnné (`Proměnná2`).

Společný typový systém platformy .NET Framework 2.0 rozlišuje následující kategorie hodnotových datových typů:

1. Vestavěné hodnotové datové typy
2. Uživatelsky definované hodnotové datové typy
3. Enumerační (výčtové) datové typy



Obr. 3.4: Klasifikace hodnotových datových typů CTS/CLS

3.2.1.1 Vestavěné hodnotové datové typy

Vestavěné hodnotové datové typy jsou předdefinované typy, které mohou programátoři okamžitě používat ve svých aplikacích. To tedy znamená, že vývojář nemusí provádět definici vestavěného hodnotového datového typu, neboť tento úkol již uskutečnili tvůrci společného typového systému a společné knihovny tříd platformy .NET Framework 2.0. Místo toho se může programátor

soustředit na samotné použití požadované typu ve svém zdrojovém kódu. Mezi vestavěné hodnotové datové typy patří:

- integrální hodnotové datové typy pro práci s celými čísly (`System.Byte`, `System.SByte`, `System.Int16`, `System.UInt16`, `System.Int32`, `System.UInt32`, `System.Int64`, `System.UInt64`),
- hodnotové datové typy pro práci s čísly s pohyblivou řádovou čárkou (`System.Single`, `System.Double`, `System.Decimal`),
- hodnotový datový typ `System.Boolean` pro realizaci operací s logickými hodnotami `True` a `False`,
- hodnotový datový typ `System.Char` pro uchovávání znaků ze znakové sady Unicode.

Bližší pohled na integrální hodnotové datové typy přináší tab. 3.1.

Tab. 3.1: Přehled integrálních hodnotových datových typů společného typového systému (CTS)

Název datového typu	Obor hodnot datového typu	Charakter datového typu	Nominální alokační kapacita instance datového typu (B)	Konformita se společnou jazykovou specifikací (CLS)	
System.Byte	<0, 255>	8bitový neznaménkový datový typ	1 bajt	✓	Ano
System.SByte	<-128, 127>	8bitový znaménkový datový typ	1 bajt	✗	Ne
System.Int16	<-32768, 32767>	16bitový znaménkový datový typ	2 bajty	✓	Ano
System.UInt16	<0, 65535>	16bitový neznaménkový datový typ	2 bajty	✗	Ne
System.Int32	$<-2^{31}, 2^{31}-1>$	32bitový znaménkový datový typ	4 bajty	✓	Ano
System.UInt32	$<0, 2^{32}-1>$	32bitový neznaménkový datový typ	4 bajty	✗	Ne
System.Int64	$<-2^{63}, 2^{63}-1>$	64bitový znaménkový datový typ	8 bajtů	✓	Ano
System.UInt64	$<0, 2^{64}-1>$	64bitový neznaménkový datový typ	8 bajtů	✗	Ne

Integrální hodnotové datové typy můžete využít při realizaci operací s celými čísly. Společný typový systém definuje několik integrálních datových typů, které se liší svým rozsahem (neboli definičním oborem), nominální alokační kapacitou a konformitou s pravidly společné jazykové specifikace. Podle oboru hodnot můžeme integrální hodnotové datové typy dále členit na typy znaménkové a neznaménkové.

Znaménkové integrální datové typy jsou všechny ty, jejichž instance mohou uchovávat kladná i záporná celá čísla (samozřejmě včetně nulové hodnoty). Když se podíváte na tab. 3.1, zjistíte, že CTS definuje čtyři znaménkové integrální hodnotové typy: `System.SByte`, `System.Int16`, `System.Int32` a `System.Int64`. Z těchto typů ovšem jeden (`System.SByte`) nevyhovuje standardům společné jazykové specifikace. Jestliže programátor použije ve svém kódu uvedený CLS-nekompatibilní integrální datový typ, není garantováno, že s tímto typem budou moci pracovat také aplikace napsané v jiných programovacích jazycích platformy .NET Framework 2.0.

Neznaménkové integrální datové typy nejsou schopny pojmout záporné celočíselné hodnoty, a tudíž je jejich definiční obor tvořen hodnotami z intervalu <0, n>, kde *n* je maximální kladné celé číslo, které ještě lze do instance neznaménkového integrálního typu uložit. Společný typový systém definuje množinu čtyř neznaménkových integrálních hodnotových datových typů (`System.Byte`, `System.UInt16`, `System.UInt32` a `System.UInt64`), z nichž ovšem jenom jeden datový typ (`System.Byte`) je CLS-kompatibilním datovým typem.

Přehled hodnotových datových typů pro práci s čísly s pohyblivou řádovou čárkou je uveden v tab. 3.2.

Tab. 3.2: Přehled hodnotových datových typů pro práci s čísly s pohyblivou řádovou čárkou

Název datového typu	Obor hodnot datového typu	Charakter datového typu	Nominální alokační kapacita instance datového typu (B)	Konformita se společnou jazykovou specifikací (CLS)	
System.Single	$<-3{,}4\times10^{38}$, $3{,}4\times10^{38}>$	32bitová číselní hodnota s pohyblivou řádovou čárkou s jednoduchou přesností	4 bajty	✓	Ano
System.Double	$<-1{,}79\times10^{308}$, $1{,}79\times10^{308}>$	64bitová číselní hodnota s pohyblivou řádovou čárkou s dvojitou přesností	8 bajtů	✓	Ano
System.Decimal	$<-7{,}9\times10^{28}$, $7{,}9\times10^{28}>$	128bitová číselní hodnota s pohyblivou řádovou čárkou s dvojitou přesností	16 bajtů	✓	Ano

Společný typový systém nabízí vývojářům tři hodnotové datové typy, jejichž instance si dokáží poradit i s desetinnými čísly. Jde o datové typy `System.Single`, `System.Double` a `System.Decimal`. Všechny zmíněné datové typy si rozumí se standardy společné jazykové specifikace, takže mohou být sdíleny napříč všemi .NET-kompatibilními programovacími jazyky. Nejúspornějším typem je `System.Single`, jehož instance je představována jako 32bitový kontejner pro uchování číselných hodnot s jednoduchou přesností. Pokud budete potřebovat vyšší přesnost výpočtu, můžete využít služeb datového typu `System.Double`, který sice alokuje v paměti dvojnásobek prostoru než typ `System.Single`, ovšem zato se může pochlubit větším rozsahem a věrohodnější interpretací náročných výpočtů. Poslední z představovaných typů, `System.Decimal`, najde své využití zejména při vědeckých výpočtech či finančních kalkulacích. Nominální alokační kapacita každé instance tohoto datového typu je 16 bajtů, což je nejvíc ze všech vestavěných hodnotových typů.

Abychom učinili náš výklad úplným, nesmíme zapomenout na datové typy `System.Boolean` a `System.Char`, které spadají rovněž do skupiny vestavěných hodnotových datových typů (tab. 3.3).

Tab. 3.3: Přehled hodnotových datových typů System.Boolean a System.Char

Název datového typu	Obor hodnot datového typu	Charakter datového typu	Nominální alokační kapacita instance datového typu (B)	Konformita se společnou jazykovou specifikací (CLS)	
System.Boolean	Logické hodnoty True a False	Vyjadřuje logické hodnoty	2 bajty	✓	Ano
System.Char	Znaky sady Unicode	16bitová celočíselní hodnota z intervalu <0, 65535>	2 bajty	✓	Ano

Tyto datové typy jsou specifické, neboť každý z nich byl navržen pro jinou oblast použití. Zatímco hodnotový typ `System.Boolean` slouží pro realizaci programových operací s logickými hodnotami `True` (pravda) a `False` (nepravda), do instancí typu `System.Char` lze ukládat textové znaky znakové

sady Unicode. Tyto znaky jsou však interně reprezentovány 16bitovými celočíselnými hodnotami z intervalu <0, 65535>.

POZNÁMKA: Textové znaky a textové řetězce

Pokud budete chtít provádět programové operace s textovými znaky sady Unicode, můžete použít vestavěný hodnotový datový typ `System.Char`. Typ `System.Char` vám však nedovoluje pracovat s textovými řetězci, tedy se soubory navzájem propojených textových znaků. Pro práci s textovými řetězci je určen typ `System.String`, který ovšem nepatří k hodnotovým, ale k odkazovým datovým typům. O typu `System.String` si budeme povídat dále v textu, kdy budeme rozebírat charakteristiku a použití odkazových datových typů společného typového systému platformy .NET Framework 2.0.

3.2.1.2 Vestavěné hodnotové datové typy v jazyce Visual Basic 2005

Vestavěné hodnotové datové typy, které definuje společný typový systém a společná jazyková specifikace, patří k systémovým datovým typům. Ve skutečnosti jsme doposud mluvili pouze o systémových datových typech, k nimž mohou přistupovat všechny programovací jazyky, které vyhovují standardům CTS a CLS. Systémové datové typy poznáte velice snadno, protože v jejich názvosloví vždy vystupuje klíčové slovo `System`. Definice všech systémových datových typů jsou uloženy ve společné knihovně tříd platformy .NET Framework 2.0. Přesněji řečeno, definice těchto datových typů jsou umístěny v jmenném prostoru `System`, což je kořenový jmenný prostor společné knihovny tříd. Kromě jmenného prostoru `System` obsahuje CLS tucty dalších jmenných prostorů, a to jak vnořených, tak samostatně působících.

Ačkoliv vestavěné systémové hodnotové datové typy lze bez jakýchkoliv potíží používat z prostředí libovolného .NET-kompatibilního programovacího jazyka, je nutno vzpomenout, že mnoho jazyků vývojově-exekuční platformy .NET Framework 2.0 definuje své vlastní ekvivalenty pro množinu systémových datových typů. Podobně se chová i Visual Basic 2005, který obsahuje vlastní klíčová slova pro jednotlivé vestavěné hodnotové datové typy. Tab. 3.4 představuje vestavěné hodnotové datové typy jazyka Visual Basic 2005 a jejich vztah vůči systémovým datovým typům společného typového systému.

Tab. 3.4: Vztah mezi vestavěnými hodnotovými datovými typy jazyka Visual Basic 2005 a systémovými datovými typy, které definuje společný typový systém platformy .NET Framework 2.0

Název vestavěného hodnotového datového typu jazyka Visual Basic 2005	Ekvivalentní systémový datový typ společného typového systému (CTS)
Boolean	System.Boolean
Byte	System.Byte
Char	System.Char
Date	System.DateTime
Decimal	System.Decimal
Double	System.Double
Integer	System.Int32

Long	System.Int64
SByte	System.SByte
Short	System.Int16
Single	System.Single
UInteger	System.UInt32
ULong	System.UInt64
UShort	System.UInt16

Po pravdě řečeno, editoru zdrojového kódu jazyka Visual Basic 2005 je zcela jedno, zdali programátor deklaruje proměnnou pomocí datového typu `Integer` nebo `System.Int32`. Klíčové slovo `Integer` jazyka Visual Basic 2005 je totiž ve skutečnosti pouhým aliasem neboli referencí, která směruje na původní systémový hodnotový datový typ. Při překladu zdrojového kódu aplikací .NET jsou všechna klíčová slova jazyka Visual Basic 2005, jež označují datové typy, nahrazena odpovídajícími „systémovými" protějšky.

3.2.2 Odkazové datové typy

Ačkoliv hodnotové datové typy jsou oblíbené pro svou kapacitní nenáročnost a jednoduchou správu, existují situace, jejichž charakter vyžaduje mnohem větší programovou flexibilitu. V těchto chvílích mohou vývojáři sáhnout po odkazových (referenčních) datových typech, které tvoří druhou polovinu spektra datových typů společného typového systému platformy .NET Framework 2.0. CTS definuje následující kategorie odkazových datových typů:

1. Samopopisné odkazové datové typy
2. Ukazatele
3. Rozhraní

3.2.2.1 Samopopisné odkazové datové typy

Samopopisné odkazové datové typy představují poměrně rozvětvenou skupinu typů, která je tvořena níže uvedenými elementy:

1. Třídy
    - vestavěné třídy,
    - uživatelsky definované třídy,
    - boxované hodnotové datové typy,
    - delegáti.
2. Pole

Třídy jsou základním kamenem objektově orientovaného programování na platformě Microsoft .NET. Třída je exaktně definovanou programovou šablonou pro generování objektů jistého typu: v tomto směru se třídy .NET nijak neliší od tříd, s nimiž jste se setkali v jazyce Visual Basic 6.0. Programátoři mohou při své práci využívat jednak vestavěné třídy, které tvoří nedílnou součást společné knihovny tříd, anebo si mohou podle svých preferencí napsat třídy vlastní. Vzhledem k tomu, že jazyk Visual Basic 2005 implementuje plnou

podporu dědičnosti, lze velice rychle vytvářet nové třídy jejich odvozováním z již existujících tříd. Tyto nové třídy je dále možné obohatit o požadovanou komplementární funkcionalitu, což je skvělý recept, jehož výsledkem je příprava chutného pokrmu v podobě dobře navržené třídy.

Instancí třídy je objekt, jenž je vytvořen a alokován ve speciální oblasti paměti aplikace .NET, které se říká řízená hromada (managed heap). Řízená hromada je globální úložiště všech objektů, které vaše aplikace .NET sestrojí. Při požadavku na instanciaci třídy je běhovým prostředím CLR alokován na řízené hromadě dostatečně veliký prostor pro uložení nově vytvářeného objektu. Aby byl ovšem takto vytvořený objekt dosažitelný, je odkaz na něj uložen do tzv. odkazové proměnné. Odkazová, nebo také referenční proměnná je proměnná, která obsahuje odkaz na objekt, jenž se nachází na řízené hromadě. V této souvislosti je důležité si uvědomit, že objekt je přístupný pouze prostřednictvím objektové reference, která je uložena v odkazové proměnné. Neexistuje žádný jiný způsob, jak se objektu dostat – musíme použít adekvátní objektovou referenci.

Zatímco objekt samotný si hoví na řízené hromadě, odkazová proměnná je uložena na zásobníku vlákna, a tudíž pro její životní cyklus platí všechna pravidla, o nichž jsme se zmínili při proměnných hodnotových datových typů. To tedy znamená, že odkazová proměnná je vytvořena v době své deklarace a její životnost je ohraničena oborem platnosti, v němž se proměnná nachází. Jestliže srovnáme hodnotovou proměnnou s jejím odkazovým protějškem, dojdeme k poznání, že zatímco hodnotová proměnná je instancí hodnotového typu a obsahuje konkrétní data, odkazová proměnná je naplněna objektovou referencí, která „ukazuje" na objekt na řízené hromadě. Objektovou referenci můžeme charakterizovat jako přestrojený ukazatel, tedy adresu paměťové lokace, na níž se nachází náš objekt. Programátoři v jazyce Visual Basic nejsou zvyklí pracovat s ukazateli, což na jedné straně sice zabraňuje efektivněji vykonávat jisté programové operace, ovšem na straně druhé vede k bezpečnějšímu stylu programování. Visual Basic 2005 tuto tradici dodržuje, a proto během své vývojářské práce nikdy nezískáte přímý přístup k ukazateli, respektive objektové referenci.

Pokud umístíte dvě odkazové proměnné do přiřazovacího příkazu, dojde ke kopírování objektové reference ze zdrojové proměnné do proměnné cílové. Po provedení příkazu tak budou obě odkazové proměnné nasměrovány na jeden a tentýž objekt. Když uložíte do inicializované odkazové proměnné hodnotu `Nothing`, zlikvidujete tím objektovou referenci, která byla v této proměnné uložena. Řečeno jinými slovy, pomocí dané odkazové proměnné již nebudete moci přistupovat k objektu uloženému na řízené hromadě.

Ačkoliv životní cyklus odkazové proměnné je přímočarý a dobře determinovatelný, o životním cyklu objektu to samé říct nemůžeme. Je to způsobeno tím, že management životnosti objektů je na platformě .NET realizován jinak, než tomu bylo v dobách Visual Basicu 6.0. V jazyce Visual Basic 6.0 byl životní cyklus objektů postaven na mechanizmu počítání referencí, které byly na daný objekt navázány. Každý objekt proto obsahoval interní počítadlo referencí, které bylo inkrementováno vždy, když byla vytvořena nová objektová reference směřující na tento objekt. Podobně, jestliže reference zanikla, hodnota počítadla se snížila. Když hodnota interního počítadla referencí spadla na nulu, objekt zjistil, že již neexistuje žádná reference, která by na něj byla navázána. V tuto chvíli objekt usoudil, že je správný čas pro samodestrukci. Je jistě více než pravděpodobné, že jako pokročilí programátoři v jazyce Visual Basic 6.0 jste se s uvedeným „mechanizmem likvidace objektů podle COM" již setkali. Nicméně, při přechodu do prostředí platformy .NET byste měli vědět, že zde nejsou životní cykly objektů řízené mechanizmem počítání referencí. Objekty .NET se nekontrolují sami, ale jejich životnost je v rukou automatického správce paměti, jenž sleduje dění na řízené hromadě a nepotřebné, respektive nedosažitelné objekty automaticky uvolňuje.

Automatický správce paměti bedlivě analyzuje aktuální strukturální složení řízené hromady, přičemž pro každý objekt vytváří tzv. strom objektových referencí. Strom referencí si můžete představit jako

standardní stromovou strukturu všech objektových referencí, které na daný objekt směřují. (Každá jedna reference je přitom uložena v samostatní odkazové proměnné.) Pomocí stromu referencí je správce paměti schopen zjistit, zdali je ten-který objekt využíván zdrojovým kódem aplikace .NET či nikoliv. Pokud existují objektové reference, které ukazují na jistý objekt, říkáme, že tento objekt je aktivní. Aktivní objekt nesmí být správcem paměti uvolněn, protože se pořád nachází v „dosahu" zdrojového kódu aplikace .NET. Jestliže ovšem zaniknou všechny reference ukazující na tento objekt, objekt se stane nedosažitelným a v této chvíli se může stát soustem pro automatického správce paměti.

Přínos automatického správce paměti je jasně zřetelný již na první pohled, neboť oprošťuje vývojáře od potřeby explicitní dealokace vytvořených objektů. Tato skutečnost je jistě chvályhodná, ovšem pro vývojáře ve Visual Basicu nepředstavuje žádnou zásadní revoluci, protože tito softwaroví tvůrci jsou na automatický paměťový management zvyklí. Naopak, pro programátory pracující v jazyku C++ je zařazení automatického správce paměti velikou devizou, protože za jeho asistence dochází k eliminaci mnoha programátorských chyb, které byly zapříčiněny špatným systémem explicitní dealokace instancí tříd.

Mluvíme-li o správci paměti, neměli bychom zapomenout poukázat na skutečnost, že tento mechanizmus pracuje s tzv. nedeterministickou finalizací. To znamená, že programátor není za běžných podmínek schopen přesně určit časový okamžik, kdy dojde k destrukci kýženého objektu a uvolnění zdrojů, které tento objekt alokoval.

Boxované hodnotové datové typy

Společný typový systém definuje pro každý hodnotový datový typ odpovídající referenční datový typ, který se nazývá boxovaný datový typ. Důležitost existence boxovaných typů je podmíněna požadavkem uskutečňovat s instancemi hodnotových typů stejné operace jako s instancemi odkazových typů, tedy s objekty. Z uvedeného plyne, že v případě potřeby lze s libovolnou instancí hodnotového typu pracovat jako s objektem. Ptáte se, jak je to možné? Klíčem k úspěchu jsou právě boxované hodnotové datové typy. Pokud vznikne potřeba použít instanci hodnotového typu v roli objektu, je vytvořena instance příslušného boxovaného hodnotového typu, do které je následně uložena původní hodnota, která byla obsažena v instanci hodnotového datového typu. Popsanému procesu se říká mechanizmus sjednocení typů. Jeho výsledkem je vytvoření objektu boxovaného typu na řízené hromadě a uložení kopie dat zdrojové instance hodnotového typu do tohoto objektu.

Delegáti

Delegáti jsou velice specifickým odkazovým datovým typem, jehož činnost se nejvíce podobá na pracovní algoritmus funkčních ukazatelů z jazyků C a C++. Protože nemohu předpokládat, že umíte programovat kromě Visual Basicu také v C++, pokusím se vám vysvětlit podstatu delegátu tak, aniž bych byl nucen se ponořit do ukazatelové džungle. Pro naše potřeby můžeme delegáta definovat jako objekt, který dokáže uchovávat paměťovou adresu instanční nebo sdílené procedury (případně funkce) jiné objektu anebo třídy. Pomocí delegáta lze pak onu proceduru aktivovat nepřímo, tedy zprostředkovaně. Delegát je dobře aplikovatelný v mnoha situacích, ke kterým patří zejména synchronní a asynchronní aktivace cílových metod.

POZNÁMKA: Synchronní a asynchronní aktivace metod pomocí delegátů

Pokud je pomocí delegáta aktivovaná cílová metoda synchronně, dochází k exekuci kódu této metody na stejném programovém vlákně, na kterém byl sestrojen delegát. Při synchronní exekuci kódu metody je proto běh aplikace pozastaven až do té doby, než je zpracován všechen kód aktivované metody. Synchronní aktivační model je standardní a uplatňuje se pokaždé, když voláte jednu metodu z těla jiné metody.

Kromě synchronní aktivace metody však existuje i další způsob, jak metodu nastartovat. Ano, máte pravdu, jde o asynchronní aktivaci metody, která je typická tím, že exekuce kódu metody se odehrává na samostatném (novém) programovém vlákně. Běhové prostředí CLR v tomto případě poskytne delegátovi nové vlákno ze skladu vláken, naplní tohle vlákno kódem cílové metody a zabezpečí její přímou exekuci. O výsledku činnosti asynchronně aktivované metody se můžeme dovědět pomocí metody zpětného volání (callback), a to tak, že ve chvíli, kdy bude exekuce asynchronně aktivované metody u konce, dojde k zavolání metody zpětného volání, která uskuteční požadovanou operaci.

Působnost delegátů je však mnohem širší: delegáti sehrávají stěžejní roli v procesu vícevláknového programování, či při vytváření vzájemných vazeb mezi událostí a jejím zpracovatelem za běhu aplikace .NET.

Pole

Rovněž pole patří do kategorie samopopisných odkazových datových typů. Na rozdíl od polí, jež znáte z Visual Basicu 6.0, ta zdejší jsou reprezentována řízenými objekty. Každé pole je samostatnou jednotkou, která může uchovávat předem definovaný počet elementů hodnotových nebo odkazových datových typů. Programátoři proto mohou vytvářet pole instancí hodnotových typů, například celých čísel typu `Integer`. Anebo mohou zkonstruovat pole, které bude pozůstávat s objektů specifikované třídy jakožto zástupce skupiny odkazových datových typů.

Zatímco pole jazyka Visual Basic 6.0 byla indexována od jedničky, spodní index všech polí na platformě .NET Framework 2.0 je roven nule. Vývojáři mohou pracovat s poli různých dimenzí: k dispozici jsou standardní jednorozměrná (vektorová) pole, dvojrozměrná, a také trojrozměrná pole. Ačkoliv tři dimenze jsou dostatečné i pro řešení náročných programátorských úkolů, jazyk Visual Basic 2005 vám dovoluje zkonstruovat pole až o dvaatřiceti rozměrech. Popři pravidelných polích mohou vývojáři programovat také pole nepravidelná.

3.2.2.2 Ukazatele

Aby byla dodržena úplnost a celistvost výkladu, je nutno vzpomenout speciální soubor odkazových datových typů, které tvoří ukazatele. Jak jsme si již řekli, v jazyce Visual Basic 2005 na ukazatele nikdy nenarazíte. S ukazateli se však můžete setkat v případě, že se rozhodnete programovat v jiných .NET-kompatibilních vývojových nástrojích, jakými jsou C# a Visual C++. V jazyku C# mohou programátoři využívat ukazatele pouze v zřetelně označených blocích tzv. nezabezpečeného programového kódu. V těchto „ukazatelových ostrůvcích“ pak vývojáři mohou použít programový kód, jenž je takřka identický s kódem jazyka C. Ačkoliv existují situace, ve kterých je použití ukazatelů velmi žádoucí, je nevyhnutné mít neustále na paměti fakt, že ukazatele jsou mocnou zbraní, se kterou je zapotřebí zacházet velice obezřetně. Mnoho programátorů se totiž přesvědčilo, že nevhodná implementace ukazatelů jim může způsobit mnohé trampoty. I z tohoto důvodu se přímé použití ukazatelů v jazyce C# příliš nedoporučuje. A jestliže se bez ukazatelů nemůžete za žádnou cenou obejít, C# vás přinutí, abyste všechny segmenty zdrojového kódu s ukazateli explicitně označili.

Jediným programovacím jazykem z trojlístku Visual Basic, C# a Visual C++, který nabízí přímou podporu pro práci s ukazateli, je samozřejmě C++. Ukazatele jsou totiž pro céčko se dvěma plusy charakteristické asi tak jako záhony zářivých tulipánů pro krajinu větrných mlýnů. Někteří programátoři dokonce říkají, že ukazatele jsou srdcem jazyků C a C++. Ať tak nebo tak, pravdou je, že v prostředí jazyka Visual C++ 2005 mohou vývojáři pracovat jednak s neřízenými čili nativními ukazateli, a také s řízenými ukazateli. Mezi oběma skupinami ukazatelů panují mnohé rozdíly: zatímco řízené ukazatele si můžete představit jako objektové reference, nativní ukazatele jsou ryzími paměťovými adresami, které identifikují data uložená v různých zákoutích operační paměti počítače.

3.2.2.3 Rozhraní

Rozhraní je tvořeno sadou deklaračních prototypů vlastností, metod a událostí, mezi kterými existují vzájemné vazby. Rozhraní má čistě deklarativní charakter, což znamená, že rozhraní podává informace pouze o signaturách svých členů, ovšem nijak se nezabývá jejich implementačními detaily. Definice členů rozhraní provádí až třída, která se rozhodne dané rozhraní implementovat. Z tohoto hlediska můžeme na rozhraní nahlížet jako na kontrakt: každá třída, která bude chtít rozhraní implementovat, musí provést definici všech členů, z nichž rozhraní pozůstává.

Vývojáři používají rozhraní často zejména proto, aby naučili své třídy chovat se polymorfně. Této technice se mezi odborníky často říká polymorfizmus implementovaný prostřednictvím rozhraní. Jejím cílem je vytvořit odlišnou implementaci stejného rozhraní napříč několika třídami. Výsledkem je soustava tříd, které provádějí rozdílnou implementaci jednoho rozhraní. Uživatelé kterékoliv třídy z této soustavy se mohou spolehnout na stabilní rozhraní, a to i přesto, že každá třída uskutečňuje definici všech členů rozhraní podle svých vlastních potřeb.

Programátoři v jazyce Visual Basic 6.0 byli schopní vytvářet třídy, které mohly implementovat větší počet různých rozhraní. Vývojáři mohli také vytvářet své vlastní rozhraní, ovšem jednalo se o nepříliš přívětivou techniku, protože vytváření rozhraní bylo aplikováno prostřednictvím sestavování speciálních abstraktních tříd, které obsahovaly vlastnosti anebo metody s prázdnými těly. Vstoupíte-li do světa Visual Basicu 2005, spatříte nový styl práce s rozhraními. Rozhraní můžete nyní vytvářet s ohromnou rychlostí, neboť jazyková specifikace obsahuje přímý příkaz `Interface`, jenž rozhraní explicitně zakládá. Jednou vytvořené rozhraní můžete implementovat v mnoha třídách, a naopak, jedna třída může implementovat libovolný počet rozhraní.

3.2.2.4 Vestavěné odkazové datové typy jazyka Visual Basic 2005

Programovací jazyk Visual Basic 2005 umožňuje vývojářům přímou práci se dvěma vestavěnými odkazovými datovými typy, jimiž jsou typy `String` a `Object`. Základní informace o obou uvedených odkazových datových typech můžete najít v tab. 3.5.

Tab. 3.5: Charakteristika odkazových datových typů String a Object

Název datového typu	Obor hodnot datového typu	Systémový datový typ	Nominální alokační kapacita instance datového typu (B)	Konformita se společnou jazykovou specifikací (CLS)	
String	Textové řetězce složené z textových znaků sady Unicode	System.String	Variabilní	✓	Ano
Object	Jakákoliv platná hodnota jakéhokoliv hodnotového nebo odkazového datového typu	System.Object	4 bajty	✓	Ano

Datový typ `String`, jehož systémovým ekvivalentem je typ `System.String`, slouží pro uchovávání textových řetězců, které jsou složeny z textových znaků znakové sady Unicode. Každý textový znak je interně reprezentován 16bitovým celým číslem bez znaménka z intervalu <0, 65535>. Textový řetězec může pozůstávat z více než dvou miliard textových znaků (celkově se jedná o 2^{31} znaků).

Datový typ `Object` je veskrze univerzálním datovým typem, protože do odkazové proměnné tohoto typu lze uložit jakoukoliv platnou hodnotu kteréhokoliv jiného hodnotového nebo odkazového datového typu, a to jak vestavěného do společného typového systému, tak uživatelsky definovaného. Reprezentantem datového typu `Object` na nižší úrovni je typ `System.Object`. Proměnná typu `Object` alokuje 4 bajty, které ve skutečnosti představují 32bitovou paměťovou adresu, která determinuje místo, na němž se nachází instance hodnotového nebo referenčního datového typu.

3.2.3 Mapování datových typů jazyků Visual Basic 6.0 a Visual Basic 2005

Poté, co jsme si představili hodnotové a odkazové datové typy, které definuje společný typový systém a s nimiž lze pracovat na platformě Microsoft .NET Framework 2.0, můžeme přikročit ke vzájemnému porovnání popsaných datových typů a datových typů zastoupených v jazyce Visual Basic 6.0. Tab. 3.6 představuje mapovací schéma, pomocí kterého mohou programátoři znalí dřívější verze jazyka Visual Basic pochopit vzájemný vztah mezi datovými typy jazyků Visual Basic 6.0 a Visual Basic 2005.

Tab. 3.6: Mapování datových typů mezi jazyky Visual Basic 6.0 a Visual Basic 2005

Datový typ jazyka Visual Basic 6.0	Ekvivalentní datový typ jazyka Visual Basic 2005	Charakter datového typu v jazyce Visual Basic 2005	Ekvivalentní systémový datový typ podle CTS
Boolean	Boolean	Hodnotový datový typ	System.Boolean
Byte	Byte	Hodnotový datový typ	System.Byte
Currency	Decimal	Hodnotový datový typ	System.Decimal
Date	Date	Hodnotový datový typ	System.DateTime
Double	Double	Hodnotový datový typ	System.Double
Integer	Short	Hodnotový datový typ	System.Int16
Long	Integer	Hodnotový datový typ	System.Int32
Object	Object	Odkazový datový typ	System.Object
Single	Single	Hodnotový datový typ	System.Single
String	String	Odkazový datový typ	System.String
Variant	Object	Odkazový datový typ	System.Object

Při psaní svých prvních programů ve Visual Basicu 2005 bude pro vás tato převodní tabulka jistě užitečná. Jak si můžete všimnout, některé datové typy jazyka Visual Basic 6.0 se v prostředí nové verze již nevyskytují (`Currency`, `Variant`), zatímco jiným typům byl přiřazen odlišný rozsah a změnily se také jejich názvy: „staré" typy `Integer` a `Long` byly nahrazeny modernějšími protějšky `Short` a `Integer`. Speciální pozornost si zaslouží právě typ `Integer`, neboť se jedná o plnokrevní 32bitový integrální datový typ, použití kterého je na dvaatřicetibitových procesorech nejefektivnější ze všech datových typů. Typ `Long` je nyní schopen pojmout 64 bitů, což je dvakrát tolik jako jeho jmenovec z Visual Basicu 6.0.

V jazyku Visual Basic 2005 již nenarazíte na univerzální typ `Variant` – jeho pozici úspěšně zaujal odkazový datový typ `Object`. Pokud tedy budete potřebovat vytvořit odkazovou proměnnou, do které budete chtít uložit libovolnou hodnotu vestavěného nebo uživatelsky definovaného hodnotového, respektive odkazového typu, datový typ `Object` je vám k dispozici. Přitom však musíte mít na paměti skutečnost, že každá proměnná typu `Object` alokuje 4 bajty. Celková alokační kapacita je ovšem vyšší, neboť se do ní započítávají také bajty, které obsazuje instance cílového hodnotového nebo odkazového typu.

Vhodnou substituci datových typů budete muset zvolit nejenom při deklarování nových proměnných, ale také při práci s metodami stávajících ovládacích prvků ActiveX a komponent či při explicitním volání funkcí aplikačního programového rozhraní Win32 API. Vzhledem k tomu, že Visual Basic 2005 podporuje také neznaménkové datové typy (`UShort`, `UInteger` a `Ulong`), jako i znaménkový typ `SByte`, je vzájemná spolupráce s nativními funkcemi snadnější, než tomu bylo ve Visual Basicu 6.0.

3.3 Proces sestavení, Just-In-Time kompilace a řízené exekuce kódu aplikace .NET

V této podkapitole si blíže představíme proces sestavení a běhu aplikace .NET. Uvidíte, jakou roli v tomto procesu sehrává společné běhové prostředí CLR a Just-In-Time kompilátor, který je zodpovědný za překlad MSIL kódu do formy strojového kódu příslušné instrukční sady CPU počítače. Rovněž objevíte půvab základní logicko-funkční jednotky aplikace .NET, která nese jméno sestavení (assembly).

Když v jazyce Visual Basic 2005 napíšete svou aplikaci a vydáte pokyn na její sestavení, kompilátor zdrojový kód analyzuje, zpracuje a přetransformuje do kódu jazyka MSIL. MSIL je akronymem pro Microsoft Intermediate Language, což je nízkoúrovňový objektově orientovaný mezijazyk, jenž byl vyvinut speciálně pro potřeby vývojově-exekuční platformy .NET Framework 2.0 a .NET-kompatibilních vývojových prostředků. Kód jazyka MSIL je nezávislý od použitého programovacího nástroje vyšší úrovně, a proto můžeme prohlásit, že stejný kód napsaný ve dvou různých vyšších programovacích jazycích platformy .NET (například Visual Basic a C#), bude přeložen do stejných symbolických instrukcí jazyka MSIL. Mezijazyk je velice efektivní z pohledu své nezávislosti ve vztahu k instrukční sadě CPU, na níž bude aplikace .NET spuštěna. Díky společnému typovému systému jazyka MSIL byl učiněn veliký pokrok ve vzájemné interoperabilitě mezi aplikacemi připravenými v rozdílných programovacích jazycích.

Kromě vygenerování nezbytných porcí jazyka MSIL je úkolem kompilátoru vytvořit také metadata, která instrukce jazyka MSIL blíže popisují. Na metadata můžete nahlížet jako na podrobnější informace o datových typech, které vaše aplikace .NET používá, a to buď přímo nebo zprostředkovaně. Protože metadata podrobněji charakterizují aplikované datové typy, často se označují také termínem typová metadata.

Princip zařazení metadat má velice široký dosah na způsob, podle něhož moderní aplikace .NET pracují a komunikují. Vzhledem k tomu, že typová metadata podávají přesné informace o všech typech použitých v aplikaci, aplikace se stává zcela samostatní a nezávislou programovou jednotkou. To mimo jiné znamená také to, že aplikace není závislá od žádných externích součástí, jako jsou třeba typové a objektové knihovny, které obsahují definice použitých datových typů (vzpomínáte na doby technologie COM?). Pomocí typových metadat mohou programátoři provádět mnoho zajímavých věcí: kupříkladu mohou využít mechanizmus reflexe, který jim na základě metadat dovoluje realizovat kompilaci kódu jazyka MSIL za běhu programu, okamžitě zakládat nové objekty a volat jejich metody, či sestavovat seznamy všech datových typů, které aplikace .NET definuje a využívá při své činnosti.

Instrukce jazyka MSIL a typová metadata jsou za asistence kompilátoru uložena do základní logicko-funkční jednotky aplikace .NET, která se nazývá sestavení (assembly). Ačkoliv o sestavení bude pojednáno dále, již nyní bychom měli na sestavení nahlížet jako na kontejner, v němž jsou uloženy všechny nezbytné součásti aplikace .NET. Zatím jsme mluvili pouze o dvou – instrukcích jazyka MSIL a typových metadatech. Jak se záhy dozvíte, sestavení může být tvořeno také zdroji (grafickými soubory, soubory s aplikačními zdroji a libovolnými soubory jiných typů) a dalšími součástmi. Každé sestavení rovněž obsahuje svůj manifest, jenž sdružuje informace o jménu, verzi, kultuře a dalších atributech, které jednoznačně charakterizují dané sestavení vůči vnějšímu světu.

Nyní se však těmito podrobnosti prozatím nezabývejme a předpokládejme, že sestavení naší imaginární aplikace .NET tvoří pouze manifest, kód jazyka MSIL a typová metadata. Když spustíte aplikaci .NET z průzkumníka, bude aktivován proces řízené exekuce, který pozůstává z níže uvedených etap:

1. Operační systém vytvoří pro právě aktivovanou aplikaci .NET samostatný proces, do něhož bude v dalším kroku načteno společné běhové prostředí CLR, které odpovídá za celkovou správu řízeného kódu aplikace. Proces, jenž je vytvořen operačním systémem, lze chápat jako fyzickou vrstvu, pomocí které je jedna běžící aplikace oddělena od jiných spuštěných aplikací. Ačkoliv jsou fyzické procesy vhodnou a mezi programátory zcela jistě oblíbenou metodou pro segmentaci procesů, v souvislosti s vývojově-exekuční platformou Microsoft .NET Framework 2.0 je nutno zmínit, že běhové prostředí CLR nabízí aplikacím .NET ještě vyšší míru abstrakce, kterou opatřují tzv. aplikační domény. Aplikační doména je samostatní oblast operační paměti uvnitř fyzického procesu, přičemž platí, že každý fyzický proces aplikace .NET obsahuje nejméně jednu aplikační doménu. Ve skutečnosti hraje aplikační doména roli logického procesu, jenž operuje v područí procesu fyzického, za jehož vytvoření je zodpovědný operační systém. Pokud je to zapotřebí, může programátor explicitně vytvořit i další aplikační domény: programové entity (například objekty), které pracují v kontextu jedné aplikační domény mohou pomocí pokročilých technik překračovat hranice své „domovské" aplikační domény, a být tak efektivně sdíleny.

2. Společné běhové prostředí CLR je načteno do procesu aplikace .NET. Poté, co je běhové prostředí CLR náležitě inicializováno a je vytvořena aplikační doména, dochází k aktivaci Just-In-Time (JIT) kompilátoru, jehož úkolem je na požádání přeložit kód jazyka MSIL vstupního bodu aplikace .NET do podoby strojového kódu. Asistence JIT kompilátoru je nezbytná, protože MSIL je mezijazyk, kterému instrukční sada procesoru počítače nerozumí, a proto není ani schopna jeho instrukce přímo zpracovat. Úkolem JIT kompilátoru je kompilace MSIL kódu za běhu aplikace, přičemž finálním produktem kompilačního procesu je strojový kód, který již může být nabídnout k explicitní exekuci CPU. Nesmírně důležitou součástí konverze kódu jazyka MSIL je verifikační proces, jehož cílem je ověřit bezpečnost překládaného programového kódu. Verifikační proces se řídí pravidly aktuálně dostupné bezpečnostní politiky pro aplikace .NET (tato pravidla nejsou neměnná, a tudíž mohou být upravena administrátorem anebo uživatelem, jenž disponuje příslušnými přístupnými právy). V rámci verifikačního procesu je

analyzován nejenom MSIL kód, ale také dostupná metadata. Na základě realizovaných operací lze přijat rozhodnutí, zda je daný kód z hlediska bezpečnostních zásad použitelný či nikoliv. Pokud není v procesu verifikace kód shledán jako „nezávadný", běhové prostředí CLR ukončí započatou exekuci aplikace .NET. Tímto způsobem je počítač chráněn před potenciálně nežádoucím softwarem. Ovšem, jak běhové prostředí CLR pozná, že kód té-které aplikace .NET není zcela v pořádku? Stěžejní úlohu v tomto směru hrají metadata a, samozřejmě, také MSIL kód. Podle platných regulí lze při ověřování kódu vycházet z několika následujících premis:

- Programový kód může přímo používat pouze ty paměťové oblasti, ke kterým má garantovaný přístup. Není kupříkladu možné, aby se kód snažil „skenovat" paměťové segmenty jiných aplikací, a to buď řízených běhovým prostředím CLR anebo nativních.
- Instance datových typů (hodnotových i odkazových) mohou aktivovat pouze ty metody a využívat pouze ty vlastnosti, které uvedené typy explicitně definují. Společný typový systém platformy .NET Framework nařizuje striktní závislost mezi typy, jejich instancemi a objektovými referencemi, jež jsou na tyto instance navázané. V programovacích jazycích jako je třeba C++ bylo možné pracovat s ukazatelovými proměnnými a ukazatele libovolně konvertovat tak, aby je bylo možné použít s více instancemi různorodých datových typů. Tato benevolence ve světě .NET neexistuje: každá objektová reference je typově silná, což znamená, že může být nasměrována pouze na instance odpovídajícího datového typu, anebo na instanci typu, jenž byl odvozen od původního datového typu.
- Objekty mohou vykonávat pouze ty programové operace, pro jejichž realizaci byly vytvořeny.

3. Když JIT kompilátor úspěšně přetransformuje MSIL kód vstupního bodu aplikace .NET do strojového kódu, je zahájen proces řízené exekuce. Do předem připraveného primárního aplikačního vlákna je „napumpováno" první sousto zkompilovaného MSIL kódu.

POZNÁMKA: Primární aplikační vlákno a vícevláknové aplikace .NET

Každá aplikace .NET disponuje primárním aplikačním vláknem, které představuje základní jednotku exekuce působící uvnitř aplikační domény fyzického procesu aplikace .NET. Podle standardního scénáře je veškerý aplikační kód vykonáván právě na primárním aplikačním vlákně. Na rozdíl od aplikací, jež byly připraveny pomocí jazyka Visual Basic 6.0, moderní řízené softwarové moduly mohou relativně snadno využívat výhod vícevláknového pracovního prostředí. Po pravdě řečeno, zrod nového vlákna si nyní v ničem nezadá s přímočarým vytvořením nové instance třídy `Thread` z jmenného prostoru `System.Threading`. A i když jsou věci „pod povrchem" poněkud složitější, než se na první pohled jeví, práce s více vlákny je ve Visual Basicu 2005 mnohem přívětivější než v kterémkoliv předchozím Visual Basicu.

Podpora vývoje aplikací .NET, jejichž digitální těla jsou složena z více vláken, je pro programátory jistě výtečnou zprávou. Jednotlivá vlákna mohou nezávisle na sobě zpracovávat programový kód takřka v reálnem čase, čehož důsledkem je nejenom vyšší rychlost uskutečňovaných činností, ale také daleko větší citlivost na vstupy uživatele. Zapojíte-li do práce další vlákno, na němž budete provádět výpočetně náročnější operace, můžete i nadále pohotově reagovat na změny vzniklé v uživatelském prostředí. Aplikace tak zůstává čilá i za náročných podmínek.

Když se budete zabývat problematikou vývoje vícevláknových aplikací déle, zjistíte, že implementace tohoto programového mechanizmu vyžaduje pečlivější programovací styl. Snad nejdůležitější je adekvátně zvládnout management jednotlivých vláken a náležitě ošetřit vznik situací, kdy by mohlo docházet k paralelnímu přístupu ke sdíleným datům. Vzhledem k tomu, že v jednu chvíli může sdílené datové členy obsluhovat pouze jedno vlákno, je potřebné dávat pozor na to, aby se o přístup k těmto členům nepokusilo v daný okamžik také jiné vlákno.

4. Aplikace .NET je nastartována a JIT kompilátor překládá další a další metody, které kód této aplikace volá. Přeložený strojový kód však není po svém zpracování procesorem zlikvidován, ale dochází k jeho uchování v operační paměti počítače. Tím je zabezpečena opětovná použitelnost jednou přeložených MSIL instrukcí a, samozřejmě, rychlost, se kterou lze jednou zkompilované porce MSIL kódu znovu aktivovat. Z algoritmu práce JIT kompilátoru plyne další nepostradatelná výhoda, a sice, že překládán je pouze ten MSIL kód, který je skutečně volán z prostředí aplikace .NET. Řečeno jinými slovy, kód metod, které nejsou nikdy volány, není JIT kompilátorem nikdy překládán. Tímto přístupem je práce JIT kompilátoru optimalizována, protože kompilovány jsou pouze aktivní části kódu aplikace .NET.

POZNÁMKA: JIT kompilátor a kritické omezení systémových zdrojů

V některých situacích se může stát, že kód jazyka MSIL, který byl jednou přeložen prostřednictvím JIT kompilátoru, nemůže být uchován v operační paměti počítače pro opětovní použití. Ačkoliv pravděpodobnost reálnosti tohoto scénáře není nijak vysoká, může k němu přesto dojít, a to zejména v případě nedostatku volně alokovatelné paměti, tedy ve stavu, jenž je označován jako kritické omezení systémových zdrojů. V této situaci je jednou přeložený MSIL kód z paměti odstraněn. Bude-li se aplikace .NET znovu odkazovat na daný segment MSIL kódu, ke slovu se zase dostane JIT kompilátor, který předmětné sousto instrukcí jazyka MSIL přeloží ještě jednou.

5. Životní cyklus aplikace .NET je v dalších etapách plně pod kontrolou běhového prostředí CLR. Běhové prostředí CLR provádí management programových vláken, inicializaci programových datových struktur, monitorování využití paměti a implicitní dealokaci nepotřebných objektů pomocí automatického správce paměti. Společné běhové prostředí CLR je alfou a omegou procesu řízené exekuce: aplikace .NET je tímto prostředím plně řízena a kontrolována.

6. V okamžiku, kdy dojde k ukončení aplikace, a to buď zásahem uživatele nebo poručením programové logiky, dochází k paměťovému úklidu vytvořených aplikačních zdrojů, likvidaci programových vláken a procesu, v němž aplikace působila. Rovněž je terminována činnost

společného běhového prostředí CLR. Tím je životní cyklus a proces řízené exekuce aplikace .NET u konce.

JIT kompilátor přináší novou úroveň flexibility, kterou mohou aplikace .NET dosáhnout. V některých případech, kdy jsou na rychlost spuštění a exekuci kódu aplikace .NET kladeny nesmírně vysoké požadavky, však může být lepším řešením uskutečnit kompilaci všech částí MSIL kódu aplikace .NET již v době její instalace. Této technice se říká překlad v době instalace a její podstata spočívá v převodu MSIL kódu aplikace .NET do nativního kódu okamžitě poté, co je tato aplikace nainstalována na cílovou počítačovou stanici uživatele. Je-li uskutečněn překlad v době instalace, není již nutné během exekuce aplikace aktivovat JIT kompilátor, čímž se minimalizuje výkonnostní penalizace, která je s činností JIT kompilátoru spjata.

Předkompilované sestavení s nativním kódem můžete vytvořit pomocí nástroje *Native Image Generator* (Ngen.exe), který se spouští z příkazové řádky Visual Studia 2005. Nabídnete-li programu *Ngen* platné sestavení aplikace .NET, generátor zkonstruuje sestavení s nativním obrazem, tedy se strojovým kódem, jenž vznikl překladem původních instrukcí jazyka MSIL. Program zabezpečí také umístění seskupení s nativním obrazem do nativní mezipaměti sestavení. Nativní mezipaměť sestavení je vyhrazenou oblastí mezipaměti sestavení, což je globální úložiště sdílených sestavení, ke kterým mohou přistupovat všechny aplikace .NET nainstalované na daném počítačovém systému. Je-li jednou vytvořeno sestavení s nativním obrazem, společné běhové prostředí CLR použije při požadavku na aktivaci aplikace .NET právě toto sestavení. Tím bude maximalizována výkonnost a rychlost provádění kódu aplikace .NET v co možná největší míře.

POZNÁMKA: Sestavení s nativním obrazem

Pokud se sestavení s nativním obrazem nachází v nativní mezipaměti sestavení, společné běhové prostředí CLR bude preferovat toto sestavení před sestavením s MSIL kódem a metadaty. Je však důležité poukázat na skutečnost, že na počítači se musí nacházet rovněž původní sestavení s řízeným programovým kódem a metadaty – to pro případ, že by běhové prostředí CLR nebylo schopné najít, nebo použít sestavení s nativním obrazem. Existuje několik situací, kdy se sestavení s nativními obrazy mohou stát nepoužitelnými. Za všechny jmenujme kupříkladu instalaci nové verze vývojově-exekuční platformy Microsoft .NET Framework nebo inovaci operačního systému či rodiny procesoru.

3.4 Sestavení aplikace .NET pod drobnohledem

Již víte, že sestavení aplikace .NET je jejím základním seskupením z hlediska funkcionality a logické struktury. Nyní nadešel čas, abychom se na sestavení podívali blíže. Začneme však pozvolna, přičemž si představíme nejjednodušší sestavení typické aplikace .NET. Takovéto sestavení se skládá z následujících částí:

1. **Manifest obsahující metadata popisující sestavení.** Manifest je velmi důležitým elementem, který musí být vždy v sestavení přítomen. (Pokud by tomu tak nebylo, nemohli bychom mluvit o skutečném sestavení aplikace .NET.) Manifest charakterizuje sestavení a rovněž obsahuje informace o všech součástech, z nichž je sestavení složeno. Manifest definuje podrobný popis sestavení, jenž je znám jako identita sestavení. Identitu sestavení formují níže uvedené informační prvky:

    - **Pojmenování sestavení.** Každé sestavení musí disponovat svým jménem, které je reprezentováno smysluplným textovým řetězcem.

- **Číslo verze sestavení.** Libovolné sestavení lze vždy rozeznat na základě jedinečného čísla verze. Číslo verze nepředstavuje jeden číselný identifikátor, ale je tvořeno souhrnem hned čtyř identifikátorů: jedná se o hlavní a vedlejší číslo verze, dále o revizní číslo verze a konečně o číslo sestavení.
- **Kultura sestavení.** Jednotlivá sestavení mohou disponovat jazykově závislými systémovými zdroji. Na bázi těchto zdrojů lze sestavení přisoudit kulturu, tedy specifikátor použité jazykové konfigurace. Implicitně mají vytvořená sestavení neutrální kulturu, a tudíž nejsou přímo svázána s žádnou specifickou verzí jazyka.
- **Informace o silném jménu sestavení.** Jestliže jako vývojář potřebujete ujištění, že vaše sestavení bude vždy jednoznačně identifikovatelné jako produkt vaší tvůrčí činnosti, můžete jej opatřit silným jménem. Silné jméno je možné vygenerovat na požádání, přičemž jako takové pozůstává z několika ingrediencí. Základními surovinami jsou pojmenování sestavení, jeho verze a kultura, které jsou za asistence kryptograficky silného klíčového páru a případně také digitálního podpisu použity v procesu generování silného jména sestavení. Pokud je jistému sestavení jednou přiřazeno silné jméno, takovéto sestavení se stává jedinečným.
- **Seznam všech souborů, které tvoří sestavení.** Manifest jako nejdůležitější element sestavení uchovává informace o všech ostatních souborech, z nichž je sestavení poskládáno. Jedno sestavení může obsahovat libovolný počet souborů, ovšem tyto musí být umístěny ve stejné složce jako soubor s manifestem.
- **Odkazy na jiná sestavení.** Specifikace vývojově-exekuční platformy Microsoft .NET Framework 2.0 dovolují, aby jedno sestavení mohlo využívat datové typy uložené v jiných sestaveních. V případě, že sestavení pracuje s „cizími" datovými typy, je nutné, aby manifest sdružoval informace o všech sestaveních, na které se zdrojové sestavení odkazuje. Tato referenční sestavení jsou identifikována pomocí svých metadat (tedy jména, verze, kultury a veřejného klíče v případě, že referenční sestavení disponuje svým silným jménem).
- **Informace o odkazovaných datových typech.** Při použití externích datových typů jsou do sestavení uloženy informace, jejichž pomocí lze bezpečně najít a identifikovat odkazované datové typy, jež jsou uloženy v jiných sestaveních.

2. **Typová metadata.** Nepřehlédnutelnou výhodou sestavení je skutečnost, že si „nesou s sebou" všechny relevantní informace o definovaných datových typech. Když programátor v aplikaci jazyka Visual Basic 2005 vytvoří novou třídu nebo strukturu, její syntaktická podoba bude po sestavení aplikačního projektu převedena do typových metadat. Typová metadata jsou uložena přímo v sestavení, a věřte nebo ne, takovéto uspořádání přináší řadu výhod. Tak především, definované datové typy jsou vždy po ruce, což oprošťuje vývojáře od nutnosti navazovat kontakt s externími součástmi. Dalším plusem je, že typová metadata dovolují programátorům provádět fantastické operace pomocí mechanizmu reflexe. Reflexe je termín označující dynamické získávání informací o datových typech, na základě kterých lze aktivovat metody a vlastnosti těchto typů, nebo zakládat jejich instance a ty pak používat pro uskutečnění zamýšlených operací. Z vlastní zkušenosti vám mohu říci, že reflexe je v ponímání platformy .NET Framework ohromně silnou zbraní: věděli jste kupříkladu, že prostřednictvím reflexe můžete dynamicky kompilovat kód jazyka MSIL a dělat spoustu

jiných doposud neuvěřitelných věcí? Po pravdě řečeno, s reflexí byste měli začínat poněkud opatrněji, nebo se vám může ze schopností této mašinérie snadno zatočit hlava!

3. **Programový kód jazyka MSIL.** Microsoft Intermediate Language je takový „dotnetový assembler" a ačkoliv se jedná o jazyk nízké úrovně, lze v něm psát plnohodnotné řízené aplikace, jejichž zpracování má na starosti běhové prostředí CLR. Sestavení standardní aplikace .NET zpravidla obsahuje stránky s programovými instrukcemi jazyka MSIL. Již víte, že kód jazyka MSIL je analyzován JIT kompilátorem, jehož výsledným produktem jsou nativní instrukce pro specifickou sadu CPU. Použití mezijazyka je výhodné zejména z hlediska vzájemné spolupráce softwarových jednotek připravených v různých vývojářských nástrojích splňujících požadavky společné jazykové specifikace a společného typového systému. Zapojení mezijazyka MSIL je významným krokem k zajištění společné jazykové interoperability, která se tak stává skutečností až na platformě Microsoft .NET Framework.

4. **Zdroje.** Programátoři přicházející z jazyka Visual Basic 6.0 se se zdroji již jistě setkali: může jít o grafické soubory či tabulky textových řetězců, které jsou uloženy separátně a které lze používat při stavbě grafického uživatelského rozhraní aplikací. V prostředí Visual Basicu 2005 můžete se zdroji rovněž pracovat. Ba co víc, operace se zdroji (jako je jejich přidávání, modifikace či odstraňování) jsou nyní mnohem lépe provedeny, a to díky sofistikovanějším podpůrným nástrojům.

Sestavení aplikace .NET může být složeno pouze z jednoho souboru: typickým příkladem je sestavení uložené v samostatně spustitelném souboru (.exe) formátu PE, anebo sestavení v podobě dynamicky linkované knihovny (.dll).

POZNÁMKA: Sestavení a souborové formáty .exe a .dll

Ačkoliv použitý souborový formát by mohl navozovat jistou příbuznost se „starými známými" spustitelnými soubory či knihovnami DLL, nenechte se zmást. Architektonická stavba sestavení je zcela odlišná od struktury souborových formátů hojně rozšířených v minulosti. Ovšem na druhou stranu, aby bylo možné používat moderní sestavení i na stávajících operačních systémech řady Microsoft Windows, jsou jejich těla uložena v již zavedených souborových formátech typu .exe a .dll.

Grafickou ilustraci jednosouborového sestavení aplikace .NET můžete vidět na obr. 3.5.

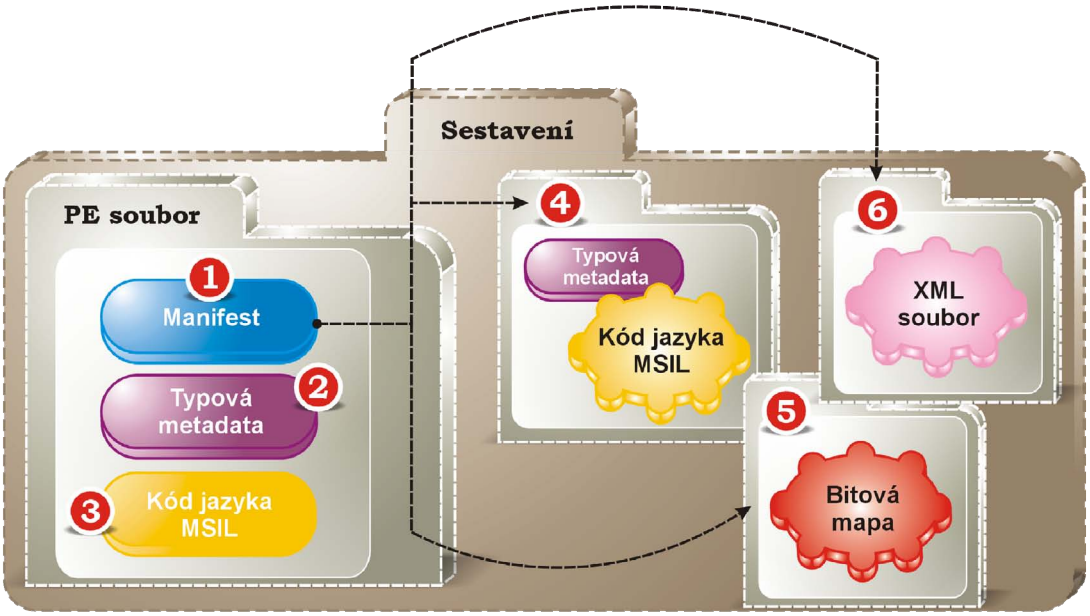
Obr. 3.5: Jednosouborové sestavení

Sestavení aplikace .NET, jehož součásti jsou uloženy v jednom spustitelném souboru PE, je typickým příkladem sestavení, které získáte při překladu standardní aplikace .NET pro systém Windows (tuto aplikaci vytvoříte pomocí programové šablony **Windows Application**). Uvedený typ sestavení je přítomen na pevném disku (v souboru .exe), zpravidla ve složce **Bin** hlavní projektové složky. Taková sestavení, která mají svoji fyzickou reprezentaci na pevném disku, se nazývají statická sestavení. Kromě statických sestavení známe také sestavení dynamická – tato ovšem na rozdíl od svých protějšků „žijí" pouze v operační paměti počítače.

POZNÁMKA: Dynamická sestavení

Ačkoliv dynamická sestavení operují obvykle pouze v operační paměti počítače, lze je konvertovat do statické podoby. Tato konverze spočívá v uložení obrazů dynamických sestavení do odpovídajících souborů (formátu PE), které budou situovány na pevném disku. Takto upravená dynamická sestavení se stávají ve skutečnosti sestaveními statickými, z čehož plyne, že pro ně platí všechna pravidla jako pro statická sestavení.

Sestavení aplikace .NET však může seskupovat i více než pouhý jeden soubor. Je-li tomu tak, máme do činění s vícesouborovým sestavením, jehož ilustraci přináší obr. 3.6.



Obr. 3.6: Vícesouborové sestavení

Ukázkové vícesouborové sestavení obsahuje celkem čtyři samostatné soubory:

1. PE soubor s manifestem, typovými metadaty a instrukcemi mezijazyka Microsoft Intermediate Language,
2. řízený modul (soubor s extenzí .netmodule), jenž obsahuje typová metadata a další programový kód mezijazyka MSIL,
3. grafický soubor s příponou .bmp, v němž jsou uložena data bitové mapy, která slouží jako aplikační zdroj,
4. XML soubor působící jako dodatečný zdroj dat pro aplikaci .NET.

POZNÁMKA: Řízené moduly a moduly

V textu jsme zavedli nový pojem řízený modul, který představuje samostatný fyzický soubor s implicitní příponou .netmodule. Ve všeobecnosti se pod pojmem řízený modul chápe soubor s MSIL kódem, jenž neobsahuje manifest sdružující metadata o sestavení. Ačkoliv standardní extenzí řízeného modulu je .netmodule, tato extenze není neměnná, což znamená, že programátor ji může v případě potřeby upravit podle své chuti. Vedle řízených modulů existují také další součásti sestavení, jako jsou třeba grafické či jiné soubory, s nimiž aplikace pracuje, anebo je při své práci frekventovaně využívá. Tyto ostatní soubory se někdy označují termínem moduly. Některé literární zdroje dokonce pokládají za moduly všechny součásti sestavení. Abychom ovšem předešli terminologickým nejasnostem, budeme v této příručce používat termín „řízený modul" pro označení souboru s MSIL kódem a extenzí .netmodule, zatímco použití termínu „modul" omezíme na ostatní součásti sestavení, které neobsahují instrukce mezijazyka MSIL.

Manifest umístěný v PE souboru vícesouborového sestavení uchovává reference na všechny ostatní komponenty sestavení – vzájemná závislost je na obr. 3.6 znázorněna spojnicemi vykreslenými přerušovanou čárou. Ve skutečnosti pouze PE soubor (pomocí svého manifestu) „ví", které soubory tvoří celé sestavení. Zbývající soubory touto informací nedisponují, a proto z fyzického hlediska působí jako zcela nezávislé a samostatné softwarové jednotky. Jak je vidět, manifest zabezpečuje propojení jednotlivých částí sestavení na logické úrovni. Toto poznání je velice důležité, neboť z pohledu fyzické reprezentace souborů sestavení se jedná o independentní jednotky.

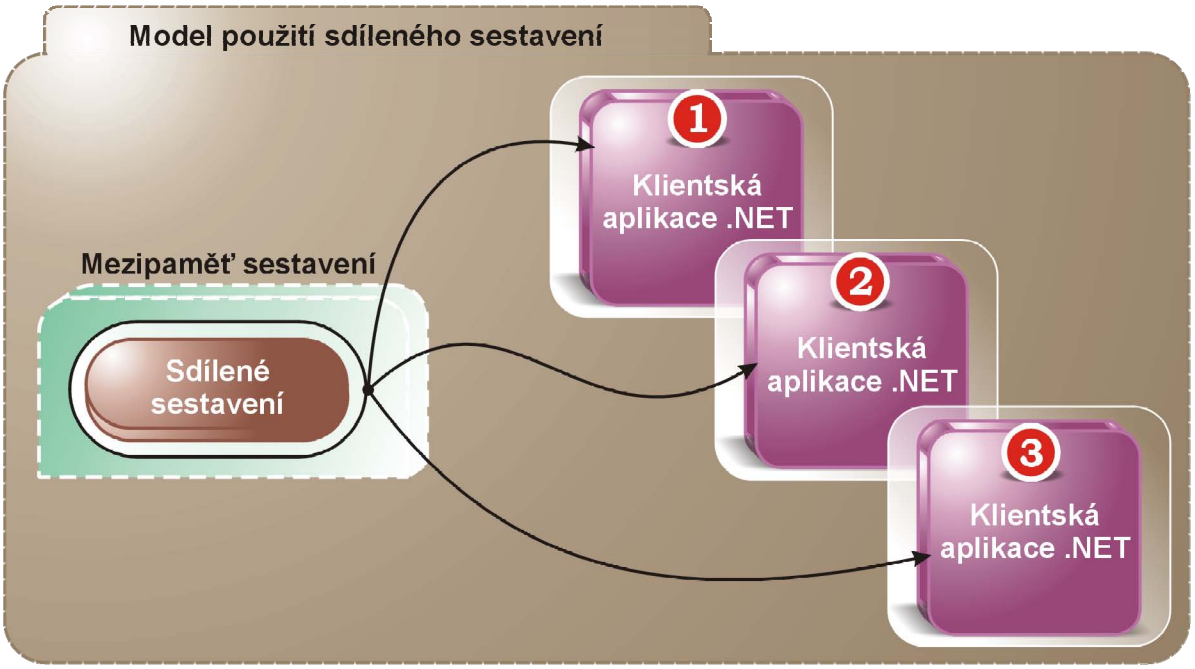
3.4.1 Soukromá a sdílená sestavení

Vývojáři pracující v řízeném prostředí platformy Microsoft .NET Framework 2.0 se mohou setkat se dvěma typy sestavení: jde o sestavení privátní (soukromá) a sdílená. Privátní sestavení je takové sestavení, které poskytuje své služby pouze jednomu klientovi, tedy jedné cílové aplikaci .NET. Vztah mezi aplikací .NET a soukromým sestavením lze popsat pomocí modelu 1:1. Privátní sestavení je proto přímo přístupné pouze jedné aplikaci .NET, většinou té, které je součástí. Představme si kupříkladu knihovnu matematických funkcí, která byla napsána v jazyce Visual Basic 2005. Tato knihovna obsahuje řízené funkce, které pro svůj bezproblémový běh potřebují zázemí vývojově-exekuční platformy .NET Framework 2.0. Imaginární knihovna funkcí se po svém vytvoření stává soukromým sestavením. Nyní popojeďme v našich úvahách o kousek dále a představme si, že připravíme aplikaci .NET, která bude chtít za účelem výpočtu některých goniometrických funkcí použít vytvořenou knihovnu tříd. Tento styl vzájemné spolupráce je samozřejmě možný: stačí, abychom sestavení knihovny tříd umístili (nebo překopírovali) do složky, v níž se nachází sestavení (v podobě samostatně spustitelného souboru) klientské aplikace .NET. Poté už stačí vytvořit odkaz na vložené sestavení a můžeme využívat výhod naprogramovaného matematického aparátu. Je důležité, abyste si uvědomili, že sestavení knihovny funkcí je privátní v tom smyslu, že je může explicitně využívat pouze jedna cílová aplikace.

Obr. 3.7: Model použití soukromého sestavení

Pokud je knihovna matematických funkcí napsána skvěle, pravděpodobně se s ní budete chtít pochlubit kolegům v práci. Jestliže se zalíbí také jim, možná ji budou chtít použít v rámci svého vývojového týmu. Povězme, že jeden vývojářský tým se skládá z pěti programátorů, kteří používají pět počítačových systémů. Kdyby naše knihovna vystupovala jako soukromá, bylo by nutné ji neustále kopírovat do složek těch aplikací, které by se ji rozhodli používat. Jestliže by kupříkladu pouze dvě aplikace na každém počítači potřebovali volat matematické funkce knihovny, bylo by nutno opatřit celkem deset kopií stejné knihovny. Pravděpodobně sami cítíte, že takovýto postup není to pravé ořechové. Výrazným negativem je snížení celkové produktivity, jakožto i celkem zbytečná nutnost duplikace jednoho souboru. Aby byly aplikace .NET pracující pod křídly platformy .NET Framework 2.0 schopné čelit tomuto problému, dochází k zavedení sdílených sestavení.

Sdílená sestavení jsou na rozdíl od těch soukromých přístupná pro všechny aplikace .NET, které hodlají využívat jejich služby. Na sdílená sestavení jsou však kladeny vyšší požadavky: každé sestavení tohoto typu musí disponovat svým silným jménem, které je vygenerováno na základě kryptograficky silného klíčového páru. Takto podepsané sestavení musí být dále nainstalováno do mezipaměti sestavení (Global Assembly Cache, GAC). V okamžiku, kdy je instalace sestavení do GAC úspěšná, se ze sestavení stává sdílená softwarová jednotka. Sdílené sestavení pracuje podle modelu 1:N, a tudíž dovede splnit přání libovolného množství cílových aplikací. Vzhledem k tomu, že sdílené sestavení je situováno v mezipaměti sestavení, není již nutné provádět jeho kopírování do složek klientských aplikací. Jednoduše řečeno, je-li jednou sestavení sdílené, je veřejně přístupné.



Obr. 3.8: Model použití sdíleného sestavení

Kapitola 4.: Syntaktické a sémantické inovace jazyka Visual Basic 2005

Když společnost Microsoft uváděla v roce 1998 na trh svůj nový softwarový produkt Visual Studio 6.0, mnozí vývojáři se cítili jako v pohádce. Vývojářský balík, jenž seskupoval mocné pomocníky s názvy Visual Basic, Visual C++, Visual InterDev, Visual J++ a Visual FoxPro, se rázem stal novým králem ve vývoji softwarových aplikací pro operační systémy Microsoft Windows a web. Ačkoliv byl Visual Basic šesté verze uveden v krátkém časovém sledu po Visual Basicu 5.0 (ten spatřil světlo světa v roce 1997), přinesl spoustu inovativních prvků, ať už z oblasti jazykové syntaxe a sémantiky, přístupu k datům, programování ovládacích prvků ActiveX, vývoje internetových a jiných typů aplikací. Vývojáři si velice rychle zvykli na precizně implementované vývojové prostředí s mnoha integrovanými součástmi, které zavádělo systém rychlého vývoje aplikací (známý akronym *RAD – Rapid Application Development*). Zkrátka a dobře, Visual Basic 6.0 byl skvělý produkt, který přišel ve vhodném čase. Tato skutečnost ho pasovala do role nepsaného krále vývojářských nástrojů s podporou vizuálního programování pod systémy Microsoft Windows. O pravdivosti tohoto tvrzení svědčí stále početná skupina programátorů, vývojářů a IT specialistů, kteří v jazyce Visual Basic 6.0 vyvíjejí svá řešení dodnes.

Na přelomu dvacátého a jednadvacátého století zaznamenal svět vývoje softwaru novou výzvu, kterou se stalo distribuované zpracování dat prostřednictvím počítačových sítí různého typu, rozsahu a topologie. Pro tvůrce softwaru se otevírá nová dimenze, kterou ovládají inteligentní aplikace pracující na vzdálených webových serverech a obsluhující stovky, ne-li tisíce uživatelů. Postupem času se začíná objevovat další fenomén, jenž je zhmotněn v podobě doposud nevídaných produktů: na svět přicházejí personální digitální asistenti (PDA) a chytré mobilní telefony, které ještě více umocňují myšlenku pro přístup k datům kdekoliv a odkudkoliv. Je zcela pochopitelné, že mobilní technologie a distribuované zpracování dat představují oblasti, které jsou úzce propojeny. Stejně tak je logické, že na nově vznikající trendy musela reagovat i společnost Microsoft, a také jiné firmy vykonávající svoji podnikatelskou činnost v softwarové branži.

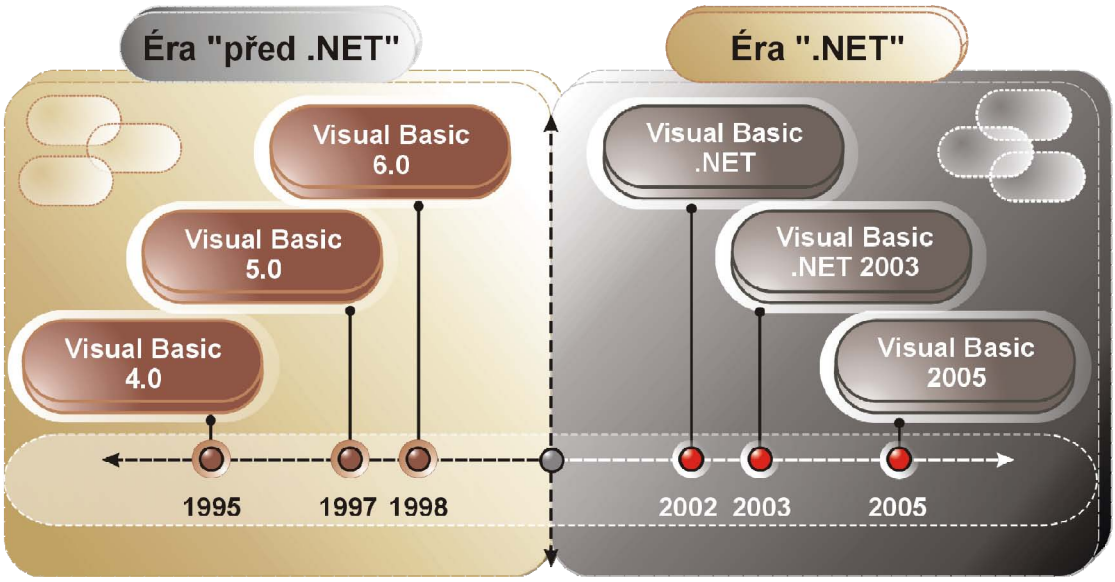
V dobře utajených redmonských laboratořích se začala pozvolna rodit nová vize, která dostala označení Microsoft .NET. Zjednodušeně řečeno, šlo o vybudování zcela nové vývojově-exekuční platformy, která by byla schopna uspokojit nové požadavky vývojářů a uživatelů, a přinést doposud netušené možnosti ve vzájemné interakci mezi člověkem a počítačem. Hlavním cílem platformy Microsoft .NET bylo poskytnout logicky navržené a zcela kompaktní řešení, které by dovedlo vývojářům nabídnout všechny potřebné nástroje pro vývoj širokého spektra softwarových aplikací nové generace. Zásadní novinkou bylo, že kromě vývoje standardních „okenních" aplikací byl kladen velký důraz také na webové aplikace a XML webové služby, které byly za pomoci nejnovějších technologií schopny nabídnout uživatelům novou úroveň funkcionality a přidané hodnoty. Práce s daty takříkajíc „na dálku" již neměla být hrozbou, která by programátory strašila. Právě naopak, distribuované využívání datových zdrojů mělo být snadnější než kdykoliv předtím.

Jednou z několika součástí skládanky s názvem Microsoft .NET byla i kompletně přepracovaná sada Microsoft Visual Studio .NET, která se k vývojářům dostala začátkem roku 2002. Visual Basic .NET, jenž byl plnohodnotnou součástí tohoto kolosu, znamenal pro spoustu vývojářů vstup do neznámé komnaty. Ano, stále to byl Visual Basic, ovšem v úplně novém provedení, které bylo doslova přeplněno novými prvky, modifikacemi a inovacemi. Vývojáři znalí Visual Basicu 6.0 se dokonce v novém Basicu cítili ztraceni a je nutno vzpomenout, že jim chvilku trvalo než se notně vylepšené verzi vývojářského stroje cítili jako doma. O rok později vstoupil na scénu Visual Basic .NET s přídomkem 2003, který adresoval lepší podporu vývoje mobilních aplikací, přičemž zlehka obohatil také syntaktickou strukturu jazyka (nově přibyla možnost deklarovat řídící proměnné cyklů přímo v jejich tělech, potěšila

i dostupnost operátorů pro realizaci bitového posunu). Ať tak či onak, inovativní progres mezi Visual Basicem verze 2002 a 2003 nebyl až natolik vypouklý, z čehož těžili především ti programátoři, kteří již byli s verzí 2002 obeznámeni.

Nyní jsme v roce 2005 a společně očekáváme další přírůstek do „rodiny Visual Basiců". Na nejnovější Visual Basic, jenž se hrdě pyšní číslovkou 2005 ve svém názvu, však musíme nahlížet jako na revoluci ve světě programování. Zcela určitě nejde pouze o další evoluční stupínek, protože vylepšení, změny a novinky, jež Visual Basic 2005 vsází do hry, mají hluboký dosah na způsob, jakým tvoříme náš software.

Vývojovou chronologii jazyka Visual Basic můžeme rozdělit na dvě základní etapy, přičemž dělící linkou je příchod platformy Microsoft .NET. Obr. 4.1 ukazuje grafickou podobu dosavadní cesty jazyka Visual Basic.



Obr. 4.1: Geneze vývoje jazyka Visual Basic v „před .NET éře" a v „éře .NET"

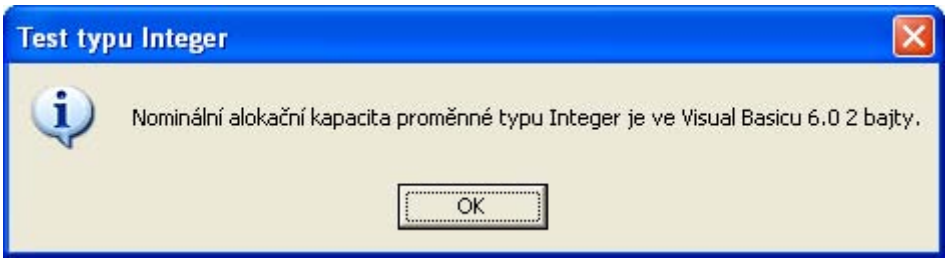
Úkolem této kapitoly je představit vám nejpodstatnější modifikace, úpravy a inovační elementy programovacího jazyka Microsoft Visual Basic 2005. Je však nutné, abyste si uvědomili, že mezi Visual Basicem 6.0 a jeho verzí 2005 existuje sedmiletá časová prodleva, a jak jistě sami uznáte, 7 let je ve sféře vývoje softwaru a softwarových technologií proklatě dlouhá doba. Možná se vám bude zdát, že Visual Basic 2005 je tak robustním nástrojem, že jej snad ani nebudete schopni pochopit a zvládnout. Přece jenom změny na straně syntaxe a sémantiky jednotlivých příkazů a programových konstrukcí jsou natolik revoluční, že zprvu se vám může jevit, jako byste ani nepracovali s Visual Basicem. Ano, je jistě nepopiratelné, že Visual Basic 2005 je společně s knihovnou FCL platformy .NET Framework 2.0 plný mnoha překvapení. Na druhou stranu, pořád jde o Visual Basic, a proto jestliže jste pokročilými programátory ve Visual Basicu 6.0, nikdy byste neměli zcela ztratit niť. Budete se však muset připravit na zvládnutí migračního procesu, v rámci kterého se seznámíte s novými programovacími paradigmaty, nově zavedenými mechanizmy a celkově moderní architektonickou strukturou jazyka Visual Basic. Cílem této kapitoly je podat vám v tomto směru pomocnou ruku a vysvětlit vám podstatu stěžejných pilířů, na nichž Visual Basic 2005 stojí. Jak již bylo zmíněno, změny jsou patrné takřka na každém kroku, a proto byste se měli obrnit trpělivostí a chuti přiučit se novým věcem. Věřte mi, na konci této poutě se vám Visual Basic 2005 natolik zalíbí, že se budete divit, jak jste mohli své aplikace doposud vyvíjet jinak.

4.1 Změny v datových typech

Visual Basic 2005 definuje rozsáhlou kolekci hodnotových a odkazových datových typů, které jsou programátorům přímo dostupné. Jak jste se dověděli ve třetí kapitole této příručky, datové typy nového Visual Basicu se od těch, které jste používali v minulé verzi tohoto jazyka, poněkud liší. Odlišnosti nacházíme zejména ve dvou oblastech, jimiž jsou: nominální alokační kapacita proměnných datových typů a rozsah datových typů. Kupříkladu proměnná datového typu `Integer` byla v jazyce Visual Basic 6.0 dlouhá 16 bitů (2 bajty), přičemž jeho rozsah byl dán číselným intervalem <–32 768, 32,767>. Ekvivalentní integrální hodnotový datový typ `Integer`, jenž je přítomen ve Visual Basicu 2005, je odlišný: nominální alokační kapacita proměnné tohoto typu je 4 a ne 2 bajty, z čehož vyplývá, že délka typu `Integer` je plných 32 bitů. Pro zjištění nominální alokační kapacity proměnné specifikovaného datového typu můžeme (tak ve Visual Basicu 6.0 jako i ve Visual Basicu 2005) použít vestavěnou funkci `Len`.

```
'Kód jazyka Visual Basic 6.0.
Dim a As Integer, NAK As Integer
NAK = Len(a)
MsgBox "Nominální alokační kapacita proměnné typu Integer " & _
"je ve Visual Basicu 6.0 " & NAK & " bajty.", _
vbInformation, "Test typu Integer"
```

Výsledkem práce tohoto fragmentu zdrojového kódu je zobrazení následujícího dialogového okna (obr. 4.2).



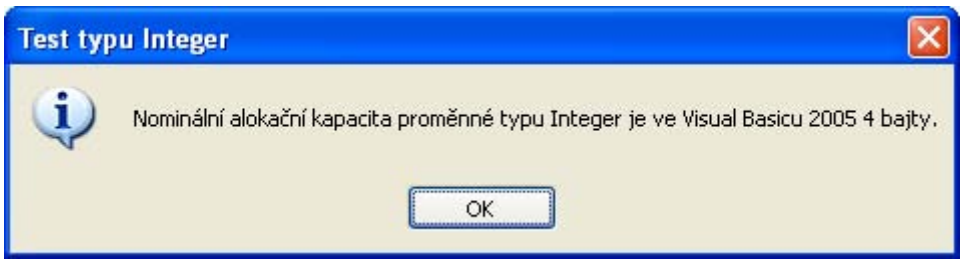
Obr.4.2: Dialog s informací o nominální alokační kapacitě proměnné typu `Integer` v jazyce Visual Basic 6.0

Abychom probádali situaci v prostředí Visual Basicu 2005, nemusíme psát nový kód, ale vystačíme si s již napsanými programovými instrukcemi. Tyto jednoduše pomocí systémové schránky zkopírujeme a vzápětí vložíme do editoru zdrojového kódu jazyka Visual Basic 2005.

```
'Kód jazyka Visual Basic 2005.
Dim a As Integer, NAK As Integer
NAK = Len(a)
MsgBox("Nominální alokační kapacita proměnné typu Integer " & _
"je ve Visual Basicu 6.0 " & NAK & " bajty.", _
vbInformation, "Test typu Integer")
```

Po úspěšném transportu kódu do editoru Visual Basicu 2005 si můžete všimnout pečlivějšího barevného formátování. Již na první pohled postřehnete, že textové řetězce jsou formátovány pomocí hnědé barvy, což je novinka, kterou Visual Basic 6.0 nedisponuje. Partie kódu s textovými řetězci jsou tak od nynějška lépe viditelné, takže je můžete ihned identifikovat a vizuálně je odlišit od jiných syntaktických programovacích konstrukcí. Samozřejmě, nové barevné schéma potěší, ovšem daleko důležitější je skutečnost, že kód, který jsme napsali ve Visual Basicu 6.0, bude bez potíží fungovat i v nové verzi tohoto jazyka. Jediné, co musíme aktualizovat, je verze jazyka v textovém řetězci, který předáváme

funkci `MsgBox`, vše ostatní může zůstat nezměněno. Upravený kód způsobí zobrazení dialogu s velikostí proměnné datového typu `Integer` v jazyce Visual Basic 2005 (obr. 4.3).



Obr. 4.3: Nominální alokační kapacita proměnné typu `Integer` je ve Visual Basicu 2005 čtyři bajty

Ptáte se, jak jsme mohli použít „starý" kód jazyka Visual Basic 6.0 v novém prostředí tak snadno, když ze všech stran slyšíte, jak je Visual Basic 2005 změněný a plný nových věcí? Ano, je sice pravda, že nový Visual Basic je jiný, ovšem jak se můžete sami přesvědčit, je to pořád ten samý programovací jazyk, s kterým jste kdysi pracovali, ovšem na kvalitativně vyšší úrovni. Nahlédneme-li pod pokličku, zjistíme, že Visual Basic 2005 definuje mnoho funkcí, které mají svůj původ v „před .NET éře". Jednou z nich je také funkce `MsgBox`, jejíž definice se nachází ve třídě `Interaction` umístěné v jmenném prostoru `Microsoft.VisualBasic`.

POZNÁMKA: Užitečná třída **Interaction** a jmenný prostor **Microsoft.VisualBasic**

Z pohledu programátora v jazyce Visual Basic 6.0 je třída `Interaction` velice užitečná, protože sdružuje hojný počet „starých známých" procedur a funkcí. Třída `Interaction` plní úlohu jakéhosi průvodce kompatibility, neboť vám dovoluje přímo aktivovat mnoho populárních procedur. Tak můžete pomocí funkce `Shell` okamžitě spouštět externí aplikace, anebo za asistence funkce `CreateObject` realizovat automatizaci aplikací Microsoft Office System na bázi technologie COM. Třída `Interaction` není jediným zdrojem funkcí jazyka Visual Basic 6.0, které lze ve verzi 2005 použít. Na dosah rukou je vám totiž celý jmenný prostor `Microsoft.VisualBasic`, který je studnicí dalších programových entit, známých z Visual Basicu verze 6.

Přestože můžeme bez jakýchkoliv potíží použít „zděděný" kód jazyka Visual Basic 6.0, zkusme se podívat, jak lze tento kód efektivněji zapsat pomocí vylepšení začleněných do Visual Basicu 2005.

```
'Upravený kód jazyka Visual Basic 2005.
Dim a, NAK As Integer
NAK = Len(a)
MessageBox.Show("Nominální alokační kapacita proměnné typu Integer " & _
"je ve Visual Basicu 2005 " & NAK & " bajty.", "Test typu Integer", _
MessageBoxButtons.OK, MessageBoxIcon.Information)
```

První důležitou změnou je použití deklaračního příkazu `Dim`, jehož pomocí provádíme deklaraci dvou proměnných typu `Integer`. Mějte prosím na paměti, že zde skutečně dochází k deklaraci dvou celočíselných proměnných, které jsou odděleny symbolem čárky. Pokud bychom příkaz `Dim` v uvedené podobě použili ve Visual Basicu 6.0, uskutečnili bychom deklaraci jedné proměnné typu `Variant` (proměnná `a`) a jedné proměnné typu `Integer` (proměnná `NAK`).

UPOZORNĚNÍ: Datový typ **Variant** není v jazyce Visual Basic 2005 podporován

Datový typ `Variant` byl ve Visual Basicu 6.0 vpravdě univerzálním typem – jeho proměnné mohly uchovávat jakékoliv hodnoty jiných datových typů. Pokročilí programátoři mohli pomocí funkce `VarType` třídy `Information` objektové knihovny `VBA`, která byla uložena v dynamicky linkované knihovně MSVBVM60.DLL, pohodlně zjišťovat podtyp proměnné typu `Variant`. Podívejme se na následující výpis programového kódu:

```
Private Sub Command1_Click()
Dim a As Variant
a = 10
ZjistitPodtyp (a)
a = "deset"
ZjistitPodtyp (a)
a = 10#
ZjistitPodtyp (a)
End Sub

Private Sub ZjistitPodtyp(ByRef v As Variant)
Select Case VarType(v)
Case vbInteger
MsgBox "V proměnné typu Variant je uloženo celé číslo."
Case vbString
MsgBox "V proměnné typu Variant je uložen textový řetězec."
Case vbDouble
MsgBox "V proměnné typu Variant je uloženo desetinné číslo."
End Select
End Sub
```

Kód deklaruje proměnnou `a` typu `Variant`, do které jsou ve třech postupných krocích uloženy tři různé hodnoty. Po exekuci každého přiřazovacího příkazu je zavolána soukromá procedura Sub s názvem `ZjistitPodtyp`, jejíž cílem je provést analýzu variantní proměnné a určit podtyp použitého datového typu. Na základě konstant, jež tvoří návratovou hodnotu funkce `VarType`, může rozhodovací příkaz `Select Case` spolehlivě determinovat podtyp proměnné typu `Variant`. Tak se z informačních zpráv dozvíme, že proměnná `a` obsahuje nejprve celé číslo, pak textový řetězec a nakonec desetinné číslo typu `Double`.

Visual Basic 2005 typ `Variant` již nepodporuje. Nicméně k dispozici je nový datový typ `Object`, který je schopen dřívější typ `Variant` plně zastoupit. `Object` je odkazovým datovým typem s univerzální povahou, a tudíž není problémem do proměnné tohoto typu uložit jakoukoliv platnou hodnotu jiných hodnotových či odkazových datových typů. Jestliže je do proměnné typu `Object` uložena hodnota hodnotového datového typu (například `Byte`, `Integer` nebo `Boolean`), je nastartován mechanizmus sjednocení typů, jehož výsledkem je vytvoření speciálního objektu boxovaného typu na řízené hromadě společného běhového prostředí CLR. Do vytvořeného objektu je následně umístěna kopie hodnoty použité hodnotové proměnné.
V případě, že je do odkazové proměnné typu `Object` přiřazena hodnota jiné odkazové proměnné (například proměnné typu `String`), není mechanizmus sjednocení typů aktivován. Za těchto okolností dochází pouze k prostému kopírování objektové reference ze zdrojové odkazové proměnné do cílové odkazové proměnné typu `Object`.

Nominální alokační kapacita 32bitových celočíselných proměnných typu `Integer` je ve Visual Basicu 2005 rovna čtyřem bajtům. Tuto hodnotu nám na požádání nabídne funkce `Len` pocházející ze třídy `Strings` jmenného prostoru `Microsoft.VisualBasic`. Získanou velikost proměnné však nezobrazujeme pomocí funkce `MsgBox`, ale její použití jsme vhodně nahradili zavoláním metody `Show` třídy `MessageBox` z jmenného prostoru `System.Windows.Forms`. Třída `MessageBox` je nepostradatelným pomocníkem, jehož služeb můžete využít vždy, když budete chtít zobrazit dialogové okno s hlavním textem, textem v záhlaví, informační ikonou a tlačítky. Třída `MessageBox` definuje přetíženou sdílenou metodu `Show`, která je přímo zodpovědná za zobrazování informačních dialogů.

UPOZORNĚNÍ: Přetížené metody v jazyce Visual Basic 2005

Přetěžování metod je nová a velice užitečná technika, kterou jazyk Visual Basic 2005 nabízí migrujícím vývojářům. Podstatou techniky přetěžování je vytvoření několika rozdílných definic jedné metody, které se od sebe navzájem odlišují. Pro vícenásobné definice přetížené metody platí následující pravidla:

1. Definice přetížené metody musí sdílet stejný název.
2. Definice přetížené metody se musí lišit seznamem parametrů, přesněji počtem parametrů, jejich pořadím či použitými datovými typy parametrů.

Když pak zavoláte metodu a předáte ji argument jistého datového typu, Visual Basic 2005 na základě charakteru předávaného argumentu vyhledá příslušnou verzi přetížené metody a zpracuje její programový kód. Skutečnost, že se jedná o přetíženou metodu, může být vizuálně umocněna použitím klíčového slova `Overloads`. O přetížených metodách si povíme více v samostatní podkapitole.

4.1.1 Explicitní inicializace proměnných v deklaračním příkazu Dim

Novinkou jazyka Visual Basic 2005 je možnost uskutečňovat explicitní inicializaci deklarované proměnné ať už hodnotového nebo odkazového datového typu přímo v deklaračním příkazu `Dim`. Podívejme se nejprve na explicitní inicializaci hodnotové proměnné typu `Byte`:

```
'Proměnná x je explicitně inicializována hodnotou 10.
Dim x As Byte = 10
```

Příkaz `Dim` provádí dvě operace: jednak deklaruje novou proměnnou `x` typu `Byte` a jednak do nově vytvořené proměnné ukládá celočíselnou inicializační hodnotu 10.

Explicitní inicializaci lze stejně dobře použít také s proměnnými odkazových datových typů. Níže je kupříkladu znázorněn postup pro explicitní inicializaci odkazové proměnné typu `String`:

```
'Explicitní inicializace dovoluje programátorovi uložit
'do proměnné typu String textový řetězec již v rámci deklaračního příkazu.
Dim JménoOS As String = "Microsoft Windows XP Home Edition"
MessageBox.Show("Jméno operačního systému je " & JménoOS, _
"Jméno operačního systému", MessageBoxButtons.OK, _
MessageBoxIcon.Information)
```

TIP: Zjištění uživatelsky přívětivého jména operačního systému pomocí jmenného prostoru **My** a přidružených objektů

Jméno operačního systému, na němž běží vaše aplikace .NET, můžete s bleskovou rychlostí vypátrat pomocí speciálního jmenného prostoru `My`. Jmenný prostor `My` seskupuje několik členů, které se navenek jeví jako pouhé vlastnosti, ovšem ve skutečnosti se jedná o specializované objekty. Pomocí těchto objektů mohou programátoři získávat informace o:

- samotné aplikaci .NET (objekt `My.Application`),
- počítačovém systému (objekt `My.Computer`),
- aplikačních formulářích (objekt `My.Forms`),
- nastaveních aplikace (objekt `My.Settings`),
- aktivním uživateli (objekt `My.User`),
- XML webových službách, na které se aktivní aplikace odkazuje (objekt `My.WebServices`).

Pro vyřešení našeho úkolu využijeme dovedností objektu `My.Computer`. Pomocí tohoto objektu se dostaneme k dalšímu, vnořenému objektu, jehož jméno je `Info`. Pokud zavoláme vlastnost `OSFullName` objektu `Info`, získáme uživatelsky přívětivé pojmenování běžícího operačního systému.

Více vám prozradí následující fragment zdrojového kódu:

```
'Rychlé zjištění jména OS pomocí vlastnosti OSFullName
'objektu My.Computer.Info.
Dim JménoOS As String = My.Computer.Info.OSFullName
MessageBox.Show("Jméno operačního systému je " & JménoOS & ".", _
"Použití objektu My.Computer.Info", MessageBoxButtons.OK, _
MessageBoxIcon.Information)
```

Obr. 4.4: Informace o operačním systému lze ve Visual Basicu 2005 najít skutečně rychle pomocí jmenného prostoru `My` a příslušných objektů

Odkazové proměnné však mohou být na rozdíl od svých hodnotových protějšků explicitně inicializovány jakýmikoliv platnými objektovými referencemi. Další ukázka programového kódu demonstruje tento postup v praxi:

```
'Instanciace třídy Button. Vytvořená instance představuje nové tlačítko.
Dim Tlačítko1 As New Button
'Modifikace vlastností zrozené instance.
With Tlačítko1
    .Location = New Point(150, 10)
    .Height = 50
    .Width = 100
    .Text = "Tlačítko"
End With
'Instance třídy Button je přidána do kolekce prvků aktivního formuláře.
Me.Controls.Add(Tlačítko1)
'Odkazová proměnná Tlačítko2 třídy Button je explicitně inicializovaná
'objektovou referencí, která je uložena v již existující
'odkazové proměnné Tlačítko1.
Dim Tlačítko2 As Button = Tlačítko1
```

Programový kód ukazuje vytvoření nové instance třídy `Button` z jmenného prostoru `System.Windows.Forms`. Nové instance odkazových typů jsou v jazyce Visual Basic 2005 zakládány pomocí operátoru `New`. V naší ukázce aplikujeme operátor `New` společně s klausulí `As` a určením třídy, z níž bude instance vytvořena. Jakmile je instance skutečně vybudována, je umístěna na řízenou hromadu, přičemž objektová reference směrující na tuto instanci je uložena do připravené odkazové proměnné `Tlačítko1`. V této souvislosti je důležité, abyste si uvědomili, že k vytvořenému objektu třídy `Button` můžeme přistupovat jenom prostřednictvím deklarované odkazové proměnné. Je to právě tato proměnná, kdo vystupuje v roli prostředníka mezi vámi a virtuálním objektem ležícím na řízené hromadě. Objektová reference, která se nachází v odkazové proměnné, je typově silná, což znamená, že může „ukazovat" pouze na objekty třídy `Button`, respektive na objekty tříd odvozených ze třídy `Button`.

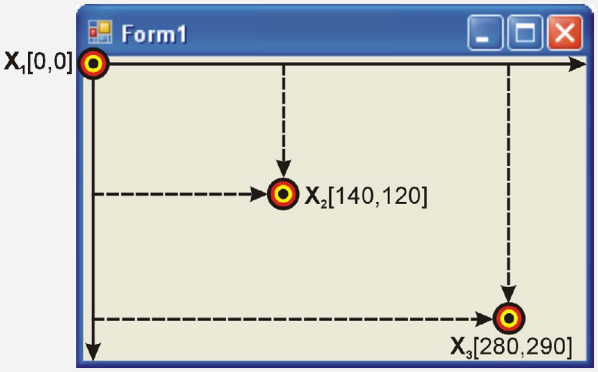
Poté, co jsme vytvořili novou instanci třídy `Button`, přistupujeme k pozměnění vlastností `Location`, `Height`, `Width` a `Text` instance. Věřím, že aplikace vlastností `Height`, `Width` a `Text` je vám velice dobře známá, a proto se přistavím pouze při první vlastnosti s názvem `Location`. Tato vlastnost udává pozici instance třídy `Button`, tedy tlačítka, na ploše nadřazeného neboli hostitelského objektu. Tímto objektem je v našem případě hlavní aplikační formulář, k němuž získáme přístup pomocí klíčového slova `Me`. Jak ovšem pozici určíme? Pro tento účel vytvoříme novou instanci struktury `Point`

z jmenného prostoru `System.Drawing`. Instance struktury reprezentuje bod, jehož pozice je v souřadnicovém systému jazyka Visual Basic 2005 jednoznačně určena pomocí x-ové a y-ové souřadnice.

POZNÁMKA: Souřadnicový systém a grafické metody ve Visual Basicu 2005

Souřadnice udávají polohu v rámci použitého souřadnicového systému, jehož implicitní nastavení se řídí podle těchto kriterií:

1. Levý horní roh souřadnicového systému je určen bodem se souřadnicemi [0,0].
2. Směrem zleva doprava dochází k inkrementaci x-ové souřadnice.
3. Směrem seshora dolů dochází k inkrementaci y-ové souřadnice.



Obr. 4.5: Grafické znázornění umístění tří bodů (X_1, X_2 a X_3) v souřadnicovém systému formuláře Visual Basicu 2005

Souřadnicový systém, jenž používá Visual Basic 2005, je odlišný od toho, který znáte z jazyka Visual Basic 6.0. Zcela nejpodstatnější změnou je to, že základní logickou zobrazovací jednotkou již nejsou twipy, nýbrž pixely. S pixely se nyní přímo pracuje ve všech grafických operacích. Zatímco aplikace napsané v jazyce Visual Basic 6.0 prováděli grafické operace pomocí metod, jež přímo volaly rozhraní GDI (Graphics Device Interface), nové aplikace .NET již spolupracují s novější reinkarnací tohoto rozhraní nesoucí označení GDI+. Inovace grafickému rozhraní velice pomohly: nejenom, že se pozvedne produktivita vaší vývojářské práce, ale také budete schopni používat mnohé grafické techniky, jejichž aplikace byla ve Visual Basicu 6.0 když ne zcela nemožná, tak alespoň hodně obtížná.

Nové tlačítko přidáme do kolekce ovládacích prvků formuláře pomocí metody `Add` vlastnosti `Controls` aktivní instance formuláře (`Me`). (Vlastnost `Controls` formuláře vrací odkaz na kolekci `ControlCollection`, která zahrnuje všechny instance třídy `Control`.) Poté, co metoda `Add` přidá nové tlačítko do kolekce prvků nacházejících se na formuláři, dojde rovněž k jeho „zviditelnění“.

Velice zajímavý je také poslední řádek výpisu kódu. Abyste nemuseli listovat zpět, připomeňme si jej ještě jednou.

```
Dim Tlačítko2 As Button = Tlačítko1
```

V tomto kódu je deklarovaná a explicitně inicializovaná další odkazová proměnná typu `Button`, tentokrát s názvem `Tlačítko2`. Přiřazovací příkaz provádí kopírování objektové reference ze zdrojové odkazové proměnné (`Tlačítko1`) do cílové odkazové proměnné (`Tlačítko2`). To znamená, že po provedení tohoto příkazu lze přistupovat k metodám a vlastnostem nového tlačítka prostřednictvím obou odkazových proměnných, nakolik obě dvě uchovávají své objektové reference směřující na jednu a tu samou instanci.

4.1.2 Použití příkazů Option Explicit a Option Strict

Pro důkladnou kontrolu syntaktické efektivnosti kódu slouží dva příkazy jazyka Visual Basic 2005 s názvy `Option Explicit` a `Option Strict`. První z nich, `Option Explicit`, byl uveden již ve Visual Basicu šesté verze, přičemž jeho úkolem bylo kontrolovat, zdali jsou všechny proměnné (použité v daném modulu) náležitě deklarovány pomocí deklaračního příkazu `Dim`. Stejné poslání má příkaz `Option Explicit` i v prostředí nového Visual Basicu: je-li aktivní (`Option Explicit On`), neúnavně sleduje všechny použité proměnné, přičemž identifikuje ty, které nebyly platně deklarované.

UPOZORNĚNÍ: Syntaktická komparace příkazu **Option Explicit** v jazycích Visual Basic 6.0 a Visual Basic 2005

Ačkoliv ve Visual Basicu 6.0 stačilo pro aktivaci kontroly realizace deklarace proměnných zapsat příkaz `Option Explicit` na první řádek modulu s programovým kódem, v jazyce Visual Basic 2005 je podoba zmíněného příkazu jiná. Syntaktická skořápka příkazu `Option Explicit` je totiž rozšířena o speciální klíčové slovo `On` nebo `Off` podle toho, zda je příkaz aktivní (`Option Explicit On`) či nikoliv (`Option Explicit Off`). Jazyk Visual Basic 2005 je však inteligentní: zadáte-li pouze část příkazu v podobě `Option Explicit`, technologie IntelliSense doplní slovíčko `On`, čímž je příkaz doplněn a následně samozřejmě také aktivován.

Na rozdíl od Visual Basicu 6.0 je v jazyce Visual Basic 2005 příkaz `Option Explicit` implicitně aktivní, takže již během kompilace zdrojového kódu budou vyhledány všechny nedeklarované proměnné. Jestliže se pokusíte použít proměnnou, aniž byste ji patřičně deklarovali, kompilátor toto nedopatření odhalí a název předmětné proměnné formátuje pomocí modré vlnovky. Jestliže najedete na takto označenou proměnnou kurzorem myši, objeví se bublinové okno se zprávou o výskytu nedeklarované proměnné (obr. 4.6).

Obr. 4.6: Příkaz `Option Explicit` pomáhá v identifikaci nedeklarovaných proměnných

Jinak je způsob použití příkazu `Option Explicit` podobný jeho aplikaci v kódu jazyka Visual Basic 6.0: příkaz musí stát na prvním řádku souboru se zdrojovým kódem, tedy před všemi procedurami, funkcemi, globálními proměnnými a dalšími programovými entitami. Mezi počtem výskytů příkazů `Option Explicit` a počtem souborů s programovým kódem je poměr 1:1, z čehož vyplývá, že v rámci jednoho souboru lze použít pouze jeden příkaz `Option Explicit`.

Jestliže se ohlédneme do historie jazyka Visual Basic, dojdeme k poznání, že to byl právě tento programovací nástroj, jenž dovoloval vývojářům používat nedeklarované proměnné. Tato vlastnost Visual Basicu byla rozporuplná: zatímco jedna skupina programátorů ji považovala za přínos a znak větší „programátorské svobody“, jiní tvůrci softwaru, zejména z řad zástupců jazyků C a C++, se jenom škodolibě usmívali. Ať tak či onak, pravdou je, že použití nedeklarovaných proměnných představuje techniku, jejíž upotřebení nelze doporučit. Když totiž neprovedete správnou deklaraci proměnné, ztrácíte schopnost určit její datový typ. Jestliže datový typ proměnné nevyberete vy, je tato činnost svěřena do rukou Visual Basicu, který se v tomto ohledu chová jinak ve verzi 6.0 a jinak ve verzi 2005. Všem nedeklarovaným proměnným vyskytujícím se v kódu jazyka Visual Basic 6.0 je implicitně přiřazen univerzální datový typ `Variant`. Visual Basic 2005 typ `Variant` nepodporuje, v tomto prostředí se uplatňuje nový datový typ `Object`, do proměnných kterého je

možné ukládat jakákoliv data. Sami asi tušíte, že tak `Variant` jako i `Object` jsou objemnými datovými typy a jejich přílišné používání nemá nic společného s psaním optimalizovaného programového kódu. Oba typy byste měli používat jenom v opravdu odůvodněných případech, takže z tohoto hlediska je aplikace příkazu `Option Explicit` jistě přínosná.

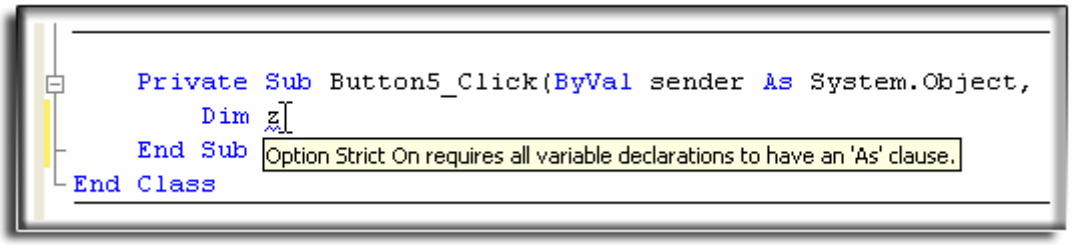
Bohužel, schopnosti příkazu `Option Explicit` jsou poněkud omezené. Přestože je příkaz schopen vynutit si deklaraci proměnných, není natolik pečlivý, aby u deklarovaných proměnných vyžadoval také přítomnost datového typu. Vzniká tak paradoxní situace, protože deklarovaná proměnná, která je prosta datového typu, je stejně nebezpečná jako její nedeklarovaná kolegyně. Níže uvedený fragment programového kódu představuje tuto situaci v jazyce Visual Basic 2005:

```
'Proměnná x je deklarována bez použití datového typu.
Dim x
x = 10
'Proměnná y je nedeklarovaná proměnná.
y = 20
```

Jak můžete vidět, proměnná `x` je deklarována pomocí příkazu `Dim`, ovšem není určen její datový typ. Ze specifikace jazyka Visual Basic 2005 je zřejmé, že když proměnná nedisponuje svým datovým typem, ke slovu se dostává generická substituce typem `Object`. Druhá proměnná `y` není vůbec explicitně deklarována a také jí je přiřazen „náhradní" datový typ `Object`.

Omezení příkazu `Option Strict` se úspěšně snaží odstranit nově uváděný příkaz `Option Strict`. Příkaz `Option Strict` je zbraní, jejíž pomocí se Visual Basic 2005 snaží uvést do praxe produktivnější a současně bezpečnější styl programování. Podobně jako příkaz `Option Explicit`, také příkaz `Option Strict` se může vyskytovat ve dvou stavech (zapnuto = `Option Strict On`, vypnuto = `Option Strict Off`). Jestliže se vývojář rozhodne využít služby příkazu `Option Strict`, musí jej zapsat na samý začátek souboru se zdrojovým kódem. A co vlastně tenhle příkaz dělá? Po pravdě řečeno, použití příkazu `Option Strict` je duální: jedna oblast aplikace příkazu se váže k problematice kontroly přítomnosti datových typů u deklarovaných proměnných, zatímco v další sféře se příkaz soustřeďuje na bezpečnou realizaci konverzních operací mezi různými datovými typy.

Je-li příkaz `Option Strict` aktivní, jsou na všechny deklarované proměnné kladeny daleko vyšší požadavky. Již tedy nestačí, aby byly všechny proměnné řádně deklarovány, každá proměnná musí být navíc opatřena jistým datovým typem. Přesněji řečeno, příkaz `Option Strict` vyžaduje, aby byla součástí deklaračního příkazu také klausule `As`, za níž následuje specifikace použitého datového typu. Pokud kompilátor při překladu zdrojového kódu zjistí, že tato podmínka není splněna, zahájí generování chybové zprávy (obr. 4.7).



Obr. 4.7: Pomocí příkazu `Option Strict On` lze identifikovat deklarované proměnné, jež jsou prosty svého datového typu

V souvislosti s péčí o proměnné se doporučuje kombinovat dovednosti příkazů `Option Strict` a `Option Explicit`. Je to proto, že osamocený příkaz `Option Strict` neumí „bojovat" proti nedeklarovaným proměnným. Řečeno jinými slovy, použití nedeklarovaných proměnných příkaz

`Option Strict` nezakazuje, a proto je nutné výskyt takovýchto proměnných eliminovat prostřednictvím příkazu `Option Explicit`.

4.1.2.1 Příkaz Option Strict a implicitní konverze

Příkaz `Option Strict` zamezuje provádění implicitních konverzních operací, při nichž by mohlo dojít ke ztrátě dat. K této situaci dochází například při pokusu umístit obsah proměnné s větším rozsahem do proměnné s menším rozsahem. Aktivací příkazu `Option Strict` mohou vývojáři zapnout účinnou kontrolu, která bdí nad realizací datových konverzí. Jestliže se bude jistá část kódu pokoušet provést potenciálně nebezpečné přetypování dat, lze toto pochybení odhalit již v době kompilace. Povězme, že v editoru zdrojového kódu jazyka Visual Basic 2005 máme tento fragment programového kódu:

```
Dim i As Integer = 1000
Dim j As Byte
j = i
```

Řádek s přiřazovacím příkazem uskutečňuje riskantní operaci, neboť se snaží umístit hodnotu 1000 do proměnné datového typu `Byte`, jehož rozsah je dán intervalem <0, 255>. Tento příkaz jazyk Visual Basic 2005 nedokáže provést. Pokud se na začátku souboru s kódem nachází příkaz `Option Strict On`, kompilátor vzniklý nesoulad typů odhalí, přičemž vás přesně nasměruje na řádek, jenž vyžaduje vaši pozornost. Tímto způsobem můžete kód vhodně korigovat už po přeložení kódu aplikace.

POZNÁMKA: Překlad kódu aplikace .NET v IDE jazyka Visual Basic 2005

Visual Basic 6.0 neumožňoval vývojářům separovat fázi překladu zdrojového kódu aplikace a fázi exekuce již přeloženého kódu. Když jste v tomto prostředí stiskli klávesu F5, Visual Basic 6.0 provedl kompilaci kódu a za předpokladu, že vše proběhlo bez potíží, došlo k samotnému spuštění aplikace. V jazyce Visual Basic 2005 jsou obě zmíněné etapy životního cyklu aplikace důsledně odděleny, což znamená, že kompilaci zdrojového kódu můžete uskutečnit, aniž by bylo nutné aplikaci také spouštět. Překlad kódu aplikace .NET můžete realizovat kdykoliv budete chtít. Postupujte dle níže uvedených instrukcí:

1. V integrovaném vývojovém prostředí jazyka Visual Basic 2005 otevřete nabídku **Build**.
2. Vyberte příkaz **Build NázevAplikace**.

Co by se ovšem stalo, kdyby příkaz `Option Strict` nebyl aktivní? Inu, nic dobrého. V tomto případě by kompilátor číhající hrozbu neodhalil a chyba by tak byla nalezena až během exekuce aplikačního kódu. Při pokusu o přiřazení hodnoty proměnné typu `Integer` do proměnné typu `Byte` by došlo ke vzniku chybové výjimky `System.OverflowException`. Tato by nám říkala, že výsledkem prováděné aritmetické operace je chyba přetečení. Je zcela pochopitelné, že odhalení chyby v době kompilace je mnohem přívětivější, než generování nepříjemné chybové výjimky za běhu programu.

Z uvedené ukázky by se mohlo zdát, že příkaz `Option Strict` sleduje hodnoty zdrojové a cílové proměnné a na základě použitých datových typů rozhoduje, zdali je provedení programové operace riskantní či nikoliv. Ač by to bylo jistě přínosné, takto příkaz nepracuje: ve skutečnosti jsou porovnávány pouze aplikované datové typy a ne aktuální hodnoty, které jsou v proměnných těchto typů uloženy. Nevěříte-li, pokusím se vás přesvědčit dalším názorným příkladem. Předpokládejme, že výše uvedený kód upravíme takto:

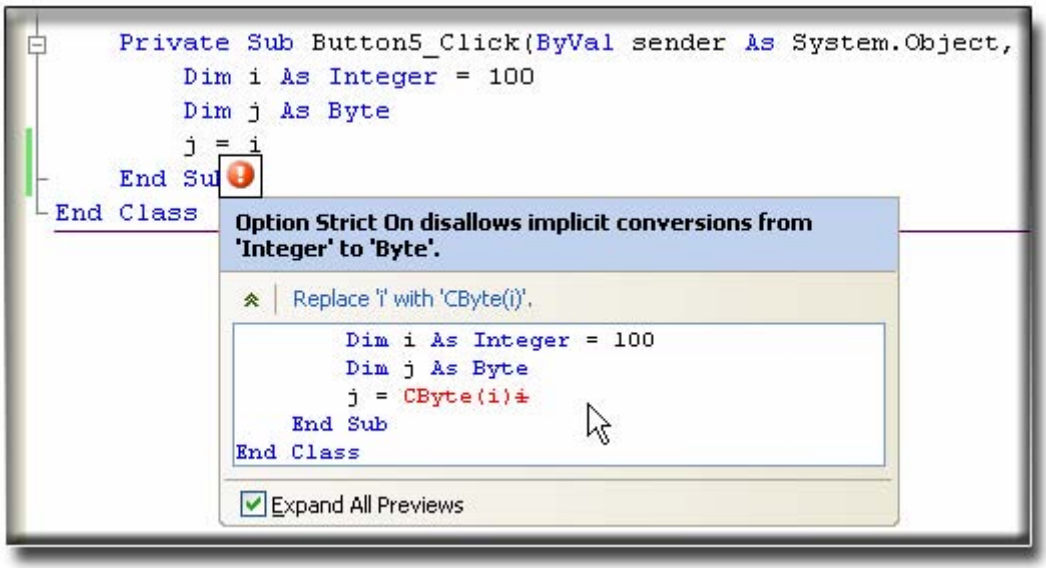
```
Dim i As Integer = 100
Dim j As Byte
j = i
```

Jak si můžete všimnout, nyní se v proměnné `j` typu `Integer` nachází hodnota 100. Tato hodnota není tak velká, aby se nevešla do definičního oboru datového typu `Byte`, ovšem navzdory tomu Visual Basic 2005 nedovolí přiřazovací příkaz uskutečnit a ohlásí chybu. Příčina tohoto chování tkví v tom, že jazyk bere do úvahy pouze přítomné datové typy a již se nezatěžuje zkoumáním aktuálních hodnot proměnných těchto typů. Visual Basic 2005 tedy odmítne uskutečnit námi požadovanou implicitní konverzi. My se však nevzdáme a přikročíme ke konverzi explicitní. Zde se nám nabízejí dvě cesty postupu:

- Problematický kód můžeme manuálně upravit tak, aby v příhodné chvíli aktivoval konverzní funkci `CByte`. Tato funkce dovede přetypovat 32bitovou celočíselnou hodnotu typu `Integer` do podoby 8bitové reprezentace typu `Byte`. Upravená verze kódu bude vypadat následovně:

  ```
  Dim i As Integer = 100
  Dim j As Byte
  j = CByte(i)
  MessageBox.Show("Hodnota proměnné j je " & j & ".", _
  "Použití konverzní funkce CByte", MessageBoxButtons.OK, _
  MessageBoxIcon.Information)
  ```

- Veskrze příjemnou novinkou editoru zdrojového kódu jazyka Visual Basic 2005 je přítomnost inteligentních značek. Tyto značky jsou velice podobné těm, které znáte z prostředí aplikací sady Microsoft Office XP či Microsoft Office 2003, takže se s nimi rychle sžijete. Inteligentní značky sledují styl vaší práce a nabízejí vám pohotová řešení na základě vámi prováděných činností. Jednou z velice užitečných inteligentních značek je značka s názvem **Error Correction Options**, která vám pomůže při opravování syntakticky nesprávných částí zdrojového kódu. Když kompilátor zjistí potenciální nebezpečí, formátuje podezřelé části kódu tenkou červenou linkou (▭). Najedete-li na tuto linku kurzorem myši, objeví se symbol inteligentní značky (!). Po aktivaci tohoto symbolu se otevře lokální dialog, v němž vás Visual Basic 2005 upozorňuje na programovou chybu a současně prezentuje řešení, jak se nalezené chyby zbavit. V naší ukázce se inteligentní značka objeví pod proměnnou `i` v řádku s přiřazovacím příkazem. Poté, co rozvinete inteligentní značku, bude vaše obrazovka vypadat jako ta z obr. 4.8.



Obr. 4.8: Inteligentní značka **Error Correction Options** v plném nasazení

V informačním dialogu můžete vidět náhled řešení, které vám Visual Basic 2005 doporučuje. Pro opravení stávajícího programového kódu můžete klepnout na návěstí s textem **Replace 'i' with 'CByte(i) '**. Jakmile tak učiníte, Visual Basic 2005 provede všechny nezbytné úpravy. Inteligentní značky jsou opravdu mocnými pomocníky, takže se na ně můžete docela dobře spolehnout.

4.1.2.2 Příkaz Option Strict a pozdní vázání objektů

Příkaz `Option Strict` zakazuje používat techniku pozdního vázání objektů. Pozdní vazba vzniká v okamžiku, kdy je reference na objekt jistého typu umístěna do generické odkazové proměnné typu `Object`. Příklad použití pozdního vázání objektu si můžeme demonstrovat na jednoduchém fragmentu kódu jazyka Visual Basic 6.0, který předvádí automatizaci aplikace Microsoft Excel 2003:

```
Dim obj_Excel As Object
Set obj_Excel = CreateObject("Excel.Application")
obj_Excel.Workbooks.Add
obj_Excel.Visible = True
```

UPOZORNĚNÍ: Přidání reference na objektovou knihovnu Microsoft Excel 11.0 Object Library v jazyce Visual Basic 6.0

Aby mohl napsaný kód správně fungovat, je nutno přidat do projektu jazyka Visual Basic 6.0 odkaz na objektovou knihovnu aplikace Microsoft Excel 2003 s názvem Microsoft Excel 11.0 Object Library. Odkaz na tuto knihovnu přidáte v integrovaném vývojovém prostředí Visual Basicu 6.0 následovně:

1. Otevřete nabídku **Project**.
2. Klepněte na položku **References...**.
3. Když se objeví stejnojmenné dialogové okno, zaměřte svoji pozornost na seznam s dostupnými knihovnami (**Available References**).
4. V seznamu vyhledejte objektovou knihovnu **Microsoft Excel 11.0 Object Library**.
5. Zatrhněte pole s názvem knihovny, čímž tuto přidáte do projektu jazyka Visual Basic 6.0.
6. Nakonec aktivuje tlačítko **OK**.

Nový objekt aplikace Microsoft Excel 2003 vytvoříme pomocí funkce `CreateObject`, které poskytneme název zdrojové třídy (`Excel.Application`), z níž má být objekt sestrojen. Funkce `CreateObject` založí primární objekt `Application` aplikace Microsoft Excel 2003 a objektovou referenci na tento objekt uloží do přichystané proměnné `obj_Excel` typu `Object`. Jelikož proměnná `obj_Excel` působí jako generická, Visual Basic 6.0 v době psaní kódu neví, jaké vlastnosti a metody bude mít objekt, na který bude nasměrovaná objektová reference uložená v této proměnné. Když budete zadávat tento kód do editoru, všimněte si, že po zadání „tečkového" operátoru vám technologie IntelliSense nenabídne žádné metody ani vlastnosti. Je to logické, neboť, jak jsme si již pověděli, Visual Basic 6.0 neumí předvídat charakter budoucího obsahu generické proměnné typu `Object`. Skutečný typ objektu je zjištěn až za běhu programu, tedy přesněji v okamžiku, kdy daný objekt funkce `CreateObject` sestrojí. Do působnosti funkce `CreateObject` patří rovněž vytvoření objektové reference, ovšem také tato činnost se uskutečňuje až „za běhu". Podstatné je, abyste pochopili algoritmus práce pozdního vázání objektů – stěžejní je v tomto směru poznání, že pozdní vazba mezi cílovým objektem a příslušnou proměnnou je vytvořena (na základě objektové reference) až v režimu exekuce kódu softwarové aplikace.

Pozdní vázání objektů není efektivní programovací technikou. Proto byste ji měli používat jenom v opravdu nezbytných případech, kdy ve fázi psaní kódu nedokážete přesně určit typ cílového objektu. Bohužel, mnozí programátoři využívají techniku pozdního vázání objektů nadmíru, což klade vyšší nároky na množství prováděných programových operací. Důsledkem takovéhoto chování je

výkonnostní penalizace, tedy jev, jemuž se snažíme, pokud možno, vyhnout velikým obloukem. Jazyk Visual Basic 2005 se společně s příkazem `Option Strict On` snaží tento neduh vymítit. Když v tomto prostředí zapíšeme do editoru zdrojového kódu následující řádky programových instrukcí, po překladu bude vygenerováno chybové hlášení (obr. 4.9).

```
Private Sub Button5_Click(ByVal sender As System.Object,
    Dim obj_Excel As Object
    obj_Excel = CreateObject("Excel.Application")
    obj_Excel.Visible = True
End   Option Strict On disallows late binding.
End Class
```

Obr. 4.9: Vytvářet pozdní vazbu mezi objektem a odkazovou proměnnou není při aktivním příkazu `Option Strict` možné

UPOZORNĚNÍ: Přidání reference na objektovou knihovnu Microsoft Excel 11.0 Object Library v jazyce Visual Basic 2005

Aby bylo možné realizovat automatizaci aplikace Microsoft Excel 2003 z prostředí aplikace .NET napsané v jazyce Visual Basic 2005, je zapotřebí přidat do aplikačního projektu referenci na objektovou knihovnu Microsoft Excel 11.0 Object Library. Postup pro přidání knihovny je však ve Visual Basicu 2005 odlišný od toho, jenž je vám známý z jazyka Visual Basic 6.0. Budete-li chtít přidat odkaz na objektovou knihovnu, proveďte toto:

1. Rozviňte nabídku **Project** a klepněte na položku **Add Reference...**.
2. V dialogu **Add Reference** vyberte záložku **COM**.
3. V seznamu vyhledejte knihovnu **Microsoft Excel 11.0 Object Library** a stiskněte tlačítko **OK**.

Ačkoliv výše popsaný postup může navozovat dojem, že do projektu je přidán odkaz na objektovou knihovnu, není tomu ve skutečnosti tak. Objektové knihovny jsou nativními soubory, které obsahují definice datových typů. Objektová knihovna aplikace Microsoft Excel 2003 seskupuje definice datových typů pro potřeby této aplikace. Bohužel, potíž je v tom, že společné běhové prostředí CLR není schopno přímo pracovat s nativními datovými typy uloženými v klasických objektových, respektive typových knihovnách. Běhové prostředí CLR společně s jazykem Visual Basic 2005 tvoří řízenou infrastrukturu, která si rozumí s řízenými datovými typy. Jelikož přímé použití objektové knihovny není možné, do hry musí vstoupit řízená mezivrstva v podobě primárního sestavení vzájemné spolupráce (Primary Interop Assembly, PIA). Aplikace Microsoft Excel 2003 disponuje svým vlastním sestavením PIA s názvem Microsoft.Office.Interop.Excel.dll, které je uloženo v mezipaměti sestavení. Když pomocí funkce **Add Reference** přidáte odkaz na objektovou knihovnu Excelu, Visual Basic 2005 zjistí, zdali je k dispozici odpovídající sestavení vzájemné spolupráce. Pokud je hledání úspěšné, je použito nalezené sestavení PIA s řízenými definicemi datových typů. V opačném případě se Visual Basic 2005 pokusí vygenerovat nové sestavení vzájemné spolupráce (Interop Assembly, IA). Zkonstruování nového sestavení IA je zpravidla bezproblémové, ačkoliv sestavení IA není zcela srovnatelné se sestavením PIA. Primární sestavení vzájemné spolupráce je opatřeno silným jménem a digitálním certifikátem, který jednoznačně identifikuje původ a producenta tohoto sestavení. Sestavení PIA může existovat pouze v jednom exempláři, zatímco sestavení IA může mít i několik variant.

Kompilátor nás nepustí, a proto budeme-li chtít kód „rozeběhnout", musíme si pomoci vytvořením časné vazby, asi takto:

```
Dim obj_Excel As Microsoft.Office.Interop.Excel.ApplicationClass
obj_Excel = CreateObject("Excel.Application")
obj_Excel.Visible = True
```

Teď je typem odkazové proměnné třída `ApplicationClass`, která je uložena v jmenném prostoru `Microsoft.Office.Interop.Excel`. Po uskutečněné změně můžeme konstatovat, že odkazová proměnná `obj_Excel` již není generická, nýbrž jde o proměnnou konkrétního datového typu. Přeložíte-li kód opět, zjistíte, že příkaz `Option Strict` má výhrady ke druhému řádku, v němž dochází k přiřazení návratové hodnoty funkce `CreateObject` do odkazové proměnné `obj_Excel`. Zárodkem problémů je skutečnost, že vzpomínaná funkce vrací hodnotu typu `Object` a příkaz `Option Strict` nepovoluje implicitní přetypování hodnoty tohoto typu na typově silnou objektovou referenci ukazující na instanci třídy `ApplicationClass`. Abychom celou situaci vyřešili, provedeme explicitní konverzi za asistence nové konverzní funkce `CType`:

```
Dim obj_Excel As Microsoft.Office.Interop.Excel.ApplicationClass
obj_Excel = CType(CreateObject("Excel.Application"), _
Microsoft.Office.Interop.Excel.ApplicationClass)
obj_Excel.Visible = True
```

Funkce `CType` je velice užitečná, protože pro nás uskuteční kýženou konverzi. Ve chvíli, kdy funkce `CreateObject` vrátí svou návratovou hodnotu, je tato podrobena explicitnímu přetypování, jehož výsledkem je uložení „opravdové" objektové reference do proměnné `obj_Excel`.

Časnou vazbu lze samozřejmě použít i v kódu jazyka Visual Basic 6.0. Stačí, když datový typ `Object` nahradíme třídou `Application` z knihovny `Excel`:

```
Dim obj_Excel As Excel.Application
Set obj_Excel = CreateObject("Excel.Application")
obj_Excel.Workbooks.Add
obj_Excel.Visible = True
```

Na rozdíl od Visual Basicu 2005, jazyk Visual Basic 6.0 od vás nebude vyžadovat explicitní konverzi návratové hodnoty funkce `CreateObject`.

Jak již naznačili prezentované výpisy zdrojového kódu, zakládání instancí tříd se v jazyce Visual Basic 2005 řídí jiným scénářem, než tomu bylo v minulé verzi tohoto jazyka.

4.1.3 Změny datového typu String

Provádění operací s textovými řetězci patří mezi činnosti, které vývojáři v jazyce Visual Basic realizují velice často, ne-li každý den. Proto je důležité, abychom si ozřejmili, jaké změny v tomto směru přináší Visual Basic 2005.

První změna spočívá v samotné povaze datového typu `String`: jedná se o odkazový datový typ, jehož základem je systémový odkazový typ `System.String`. Textový řetězec typu `String` je v jazyce Visual Basic 2005 chápán jako posloupnost objektů třídy `System.Char` o variabilní délce, která vytváří jeden celistvý objekt, a tedy textový řetězec. Ve většině případů platí, že jeden objekt třídy `System.Char` představuje jeden znak textového řetězce, ovšem mohou nastat také případy, kdy je jeden textový znak reprezentován hned dvěma instancemi třídy `System.Char`. Všechny textové řetězce, se kterými budete v prostředí jazyka Visual Basic 2005 pracovat, jsou implicitně vytvářeny jako řetězcové literály s proměnlivou délkou a schopností pracovat se znakovou sadou Unicode. Skutečná délka řetězce je určena těsně před vznikem instance třídy `System.String`. Textový řetězec, jenž je zapouzdřen v objektu třídy `System.String`, je kdykoliv dosažitelný pomocí objektové reference, kterou lze uložit do příslušné odkazové proměnné.

Druhou důležitou změnou je, že Visual Basic 2005 nepodporuje vytváření textových literálů o fixní délce. To znamená, že níže uvedený kód jazyka Visual Basic 6.0 není v nové verzi aplikovatelný:

```
Dim Řetězec As String * 5
Řetězec = "123"
MsgBox Řetězec
```

Kód vytváří proměnnou `Řetězec`, která dovede uchovat textový literál o maximální délce pěti znaků. Když do této proměnné uložíme textový řetězec s pouhými třemi znaky, Visual Basic 6.0 doplní na zbývající pozice textového řetězce mezery. Na druhou stranu, pokud by délka inicializačního textového řetězce přesahovala přípustný limit, Visual Basic 6.0 by řetězec jednoduše ořezal.

Kdybyste zkusili deklarovat proměnnou `Řetězec` v uvedené podobě v jazyce Visual Basic 2005, kompilátor by vám záhy pověděl, že něco není v pořádku. Stručně řečeno, pro fixní textové řetězce nenachází Visual Basic 2005 uplatnění a je-li takovýto řetězec rozpoznán, je okamžitě hlášena chyba.

POZNÁMKA: Simulace textového literálu o fixní délce pomocí atributu **VBFixedStringAttribute**

Třebaže je pravda, že jazyk Visual Basic 2005 nedisponuje interní podporou pro fixní textové řetězce, existuje způsob, jak variabilně dlouhý textový literál vystavovat navenek tak, jako by šlo o řetězec s fixní délkou. Klíčem k úspěchu je použití atributu `VBFixedStringAttribute` z jmenného prostoru `Microsoft.VisualBasic`. Aplikace atributu má speciální syntaxi, podle které je atribut uzavřen v špičatých závorkách před názvem proměnné. Atribut `VBFixedStringAttribute` pracuje s jediným parametrem (ten je umístěn v standardních kulatých závorkách za jménem atributu), který determinuje počet znaků, z nichž je fixní textový literál složen. Deklarace odkazové proměnné typu `String` společně s atributem `VBFixedStringAttribute` by mohla mít následující podobu:

```
'Symbioza atributu VBFixedStringAttribute a odkazové proměnné
'typu String.
<VBFixedString(5)> Dim Řetězec As String
```

Atribut nelze použít ve spojení s lokálními proměnnými, a proto i deklarace proměnné `Řetězec` v naší ukázce nesmí být uložena v těle nějaké procedury.

Aby ovšem nedošlo k mýlce: použití atributu neznamená, že původně variabilní textový řetězec se změní na literál o fixní délce. Atribut provádí účelnou kamufláž – API funkcím a metodám, které budou pracovat s obsahem proměnné typu `String` říká, že mají do činění s fixním textovým řetězcem, ačkoliv to ve skutečnosti není docela tak.

4.1.4 Změny v uživatelsky definovaných datových typech (UDT)

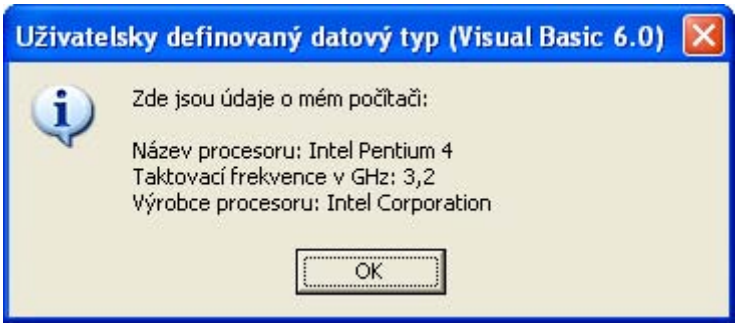
Programátoři v jazyce Visual Basic 6.0 mohli definovat uživatelsky definované datové typy pomocí příkazu `Type`. Uživatelsky definované datové typy (některými vývojáři zjednodušeně označované také jako záznamy) představovali samostatní kategorii datových typů, jež sdružovaly několik proměnných různých typů do jednoho celku. Ten byl samostatní a nezávislý jak po logické, tak po syntaktické a sémantické stránce. Definice každého UDT se nacházela v programovém bloku, jenž byl ohraničen příkazy `Type` a `End Type`. Programový blok s definicí UDT se mohl nacházet pouze na úrovni modulu, přičemž přístup k typu samotnému, jakožto i jeho členům, byl implicitně veřejný. Níže můžete vidět zdrojový kód jednoduchého, ovšem plně funkčního uživatelsky definovaného datového typu s názvem `Počítač`, jehož definice je umístěna ve standardním modulu aplikačního projektu jazyka Visual Basic 6.0.

```
Option Explicit
'Definice UDT v jazyce Visual Basic 6.0.
Type Počítač
  NázevCPU As String
  Producent As String
  TaktovacíFrekvence As Single
End Type
```

Poté, co jsme uskutečnili definici uživatelsky definovaného datového typu `Počítač`, můžeme se zaměřit na deklaraci a inicializaci proměnné tohoto typu:

```
'Deklarace a inicializace proměnné UDT.
Dim MůjPC As Počítač
With MůjPC
  .NázevCPU = "Intel Pentium 4"
  .Producent = "Intel Corporation"
  .TaktovacíFrekvence = 3.2
End With
'Zobrazení informací o proměnné UDT v informačním dialogu.
MsgBox "Zde jsou údaje o mém počítači:" & vbCrLf & vbCrLf & _
"Název procesoru: " & MůjPC.NázevCPU & vbCrLf & _
"Taktovací frekvence v GHz: " & MůjPC.TaktovacíFrekvence & vbCrLf & _
"Výrobce procesoru: " & MůjPC.Producent, vbInformation, _
"Uživatelsky definovaný datový typ (Visual Basic 6.0)"
```

Po spuštění kódu se objeví následující dialogové okno:



Obr. 4.10: Informace o instanci UDT v jazyce Visual Basic 6.0

V oblasti programování s uživatelsky definovanými datovými typy nenechal jazyk Visual Basic 2005 takříkajíc „kámen na kameni", a proto bude nutné, abyste této problematice věnovali zvýšenou pozornost. První významná změna se týká samotného názvosloví: v nové verzi se již přímo nemluví o uživatelsky definovaných typech, nýbrž o strukturách. Terminologická obměna je však k prospěchu věci, protože uživatelsky definovanými datovými typy nebyly v jazyce Visual Basic 6.0 pouze záznamy, ale také kupříkladu třídy. Jazyk Visual Basic 2005 považuje struktury za hodnotové datové typy, což znamená, že jejich instance jsou ukládány na zásobník a ne na řízenou hromadu, jako je to v případě instancí odkazových datových typů. Příkaz `Type`, jehož pomocí jste prováděli definici UDT ve Visual Basicu 6.0, již není podporován. Není ovšem důvod ke smutku, neboť jeho roli úspěšně zastupuje nový příkaz `Structure`, který společně s opozitním příkazem `End Structure` vymezuje blok sloužící pro definici struktury.

Definice každé struktury musí obsahovat alespoň jeden člen, jenž je reprezentován hodnotovou anebo odkazovou proměnnou. Každý člen struktury musí mít explicitně určený modifikátor přístupu. Jestliže programátor místo modifikátoru přístupu použije deklarační příkaz `Dim`, viditelnost daného členu struktury je implicitně nastavena na veřejnou (`Public`).

Uživatelsky definovaný typ `Počítač` se ve Visual Basicu 2005 změní na stejnojmennou strukturu:

```
'Definice struktury Počítač v jazyce Visual Basic 2005.
Public Structure Počítač
    Dim NázevCPU As String
    Dim TaktovacíFrekvence As Single
    Dim Producent As String
End Structure
```

UPOZORNĚNÍ: Změna viditelnosti jednotlivých členů struktury

Všechny proměnné uvnitř naší struktury jsou deklarované pomocí příkazu `Dim`, což znamená, že všechny se vyznačují implicitním veřejným přístupem. Pokud byste chtěli viditelnost té-které proměnné omezit, můžete ve spojení s kýženou proměnnou použít vhodný modifikátor přístupu. V tomto případě již není nutné uvádět příkaz `Dim`, ale pouze samotný modifikátor přístupu:

```
Public Structure Počítač
'Proměnná NázevCPU je soukromá. Vnější kód k ní nemůže přistupovat.
    Private NázevCPU As String
    Dim TaktovacíFrekvence As Single
    Dim Producent As String
End Structure
```

Když budete chtít strukturu použít, musíte nejprve deklarovat novou hodnotovou proměnnou, jejíž typem bude právě struktura. Jestliže JIT kompilátor uvidí příkaz deklarující tuto proměnnou, bude generovat instrukce pro založení nové instance struktury na zásobníku programového vlákna.

```
'Vytvoření instance struktury.
Dim MůjPC As Počítač
'Inicializace členů zrozené instance.
With MůjPC
    .NázevCPU = "Intel Pentium M"
    .Producent = "Intel Corporation"
    .TaktovacíFrekvence = 1.6
End With
'Zobrazení informací o CPU prostřednictvím metody Show třídy MessageBox.
MessageBox.Show("Zde jsou údaje o mém notebooku:" & vbCrLf & vbCrLf & _
"Název procesoru: " & MůjPC.NázevCPU & vbCrLf & _
"Taktovací frekvence v GHz: " & MůjPC.TaktovacíFrekvence & vbCrLf & _
"Výrobce procesoru: " & MůjPC.Producent, "Struktura (Visual Basic 2005)", _
MessageBoxButtons.OK, MessageBoxIcon.Information)
```

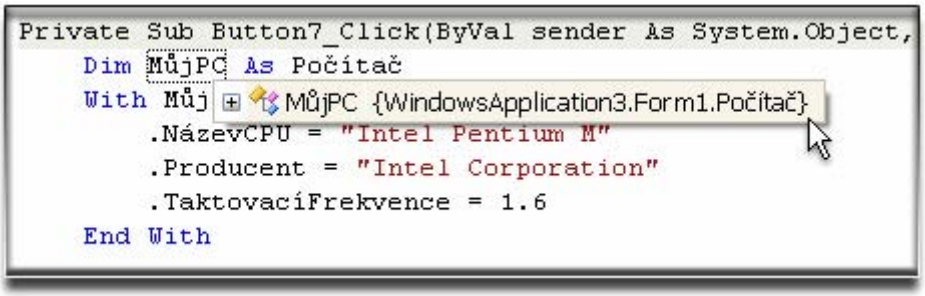
Výsledek práce kódu přibližuje obr. 4.11.

Obr. 4.11: Informace o instanci struktury v jazyce Visual Basic 2005

Abychom výklad poněkud odlehčili, podíváme se na následujících řádcích na nový způsob prohlížení obsahu vytvořené instance struktury. Integrované vývojové prostředí Visual Studia 2005 třímá v náručí

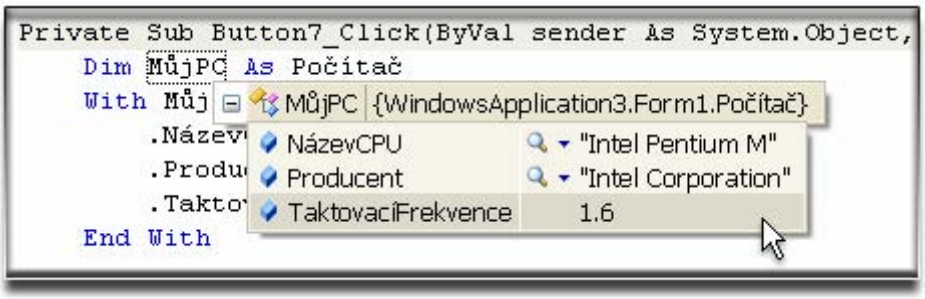
kupodivu velké množství výborných pomůcek, které v nebývalé míře usnadňují práci vývojáře. My se podíváme na takzvaný vizualizátor dat, jehož pomocí můžeme v době přerušení exekuce kódu prozkoumávat obsahy proměnných a zjišťovat tak informace o programových entitách, které jsou za proměnnými ukryty. Postupujte, prosím, dle těchto pokynů:

1. Výše uvedený programový kód, který provádí vytvoření instance struktury `Počítač` umístěte do zpracovatele události `Click` tlačítka.
2. Umístěte kurzor myši na řádek zdrojového kódu, ve kterém dochází k volání metody `Show` třídy `MessageBox`.
3. Stiskněte klávesu **F9**, čímž na tento řádek umístíte programovou zarážku. Programová zarážka vytváří bod přerušení: když je tento bod JIT kompilátorem dosažen, pozastavuje se exekuce programových instrukcí aplikace .NET. Jestliže jste zarážku umístili správně, v levém panelu se objeví symbol červeného kolečka ( ) a celý řádek s voláním metody `MessageBox.Show` bude mít červené pozadí.
4. Stiskněte klávesu **F5** a klepněte na tlačítko. Aplikace .NET bude uvedena do režimu přerušení.
5. Když kurzorem myši najedete na název proměnné `MůjPC` v deklaračním příkazu `Dim`, objeví se vizualizátor dat (obr. 4.12).



Obr. 4.12: Vizualizátor dat nám pomáhá s inspekcí instance struktury

6. Kurzor myši přemístěte na symbol se znaménkem plus (+), který se nachází před názvem instance struktury v okně vizualizátoru dat. Sledujte, jak se rozvine další informační okno, v němž můžete vidět aktuální hodnoty všech členů instance struktury `Počítač` (obr. 4.13).



Obr. 4.13: Zobrazení členů instance struktury a jejích hodnot

4.2 Implicitní a explicitní konverze

Jazyk Visual Basic byl snad od svých začátků neslavně proslulý tím, že dovoloval vývojářům provádět takřka všechny konverze implicitně, tedy bez zásahu programátora. Pokud bylo možné konverzi uskutečnit implicitně, Visual Basic tak učinil, což na jedné straně oprošťovalo vývojáře od psaní dodatečných konverzních příkazů, ovšem na straně druhé se mnohdy stávalo, že výsledkem realizované implicitní konverze bylo něco zcela jiného, než programátor původně očekával.

Implicitní konverze můžeme rozdělit na dvě skupiny: zužující a rozšiřující konverze. Zatímco rozšiřující konverze jsou bezpečné a jejich protějšcích se to říci nedá. Ve všeobecnosti můžeme říci, že při provádění rozšiřujících konverzí jsou hodnoty přetransformovávány z datových typů s menším rozsahem a nominální alokační kapacitou do datových typů s větším rozsahem a větší nominální alokační kapacitou. Příkladem rozšiřující konverze je převod celého čísla typu `Short` do podoby celého čísla typu `Integer`. Jelikož rozsah typu `Integer` je měřen v šířce 32 bitů, nikdy nebude problém uložit do proměnné tohoto typu 16bitovou hodnotu typu `Short`. Implicitní rozšiřující konverze proto nepředstavují postrach v tom smyslu, že při nich může dojít k závažné chybě (samozřejmě za předpokladu, že jsou respektovány rozsahy obou zúčastněných datových typů). Daleko obezřetnější však musíme být při uskutečňování implicitních zužujících konverzí. Zužující konverze jsou takové konverzní operace, při nichž dochází k přetypování hodnot mezi dvěma datovými typy, přičemž zdrojový datový typ disponuje větším rozsahem než cílový datový typ. Je proto nutné dbát na to, aby se konverze nestala zdrojem potenciálně závažných chyb, jako je kupříkladu přetečení. Nebudeme daleko od pravdy, když prohlásíme, že zužující implicitní konverze jsou poněkud riskantní technikou. Je proto bezpodmínečně nutné, abychom tyto konverze programovali tak, aby nemohlo dojít ke vzniku softwarových chyb.

Jazyk Visual Basic 6.0 je, co se týče konverzních operací, velice benevolentní, což může někdy vést k takřka zapeklitým situacím. Po pravdě řečeno, tento jazyk je schopen uskutečnit téměř všechny požadavky na přetypování zcela implicitně, bez sebemenšího zásahu ze strany vývojáře. Uveďme si několik příkladů:

```
'Konverzní operace č. 1.
Dim a As Integer, b As Long
a = 10000
b = a
MsgBox a
```

První ukázka je zářným příkladem implicitní rozšiřující konverze, která bude ze strany Visual Basicu 6.0 provedena bez jakýchkoliv potíží. Jakákoliv platná hodnota datového typu `Integer` (pamatujte, zde jde o 16bitový typ) může být reprezentována typem `Long`, jehož délka je 32 bitů. Ke druhé konverzi dochází při volání funkce `MsgBox`, která převádí celočíselnou hodnotu typu `Integer` na textový řetězec. Dodejme, že také tato konverze je bezpečná: implicitně lze z jakékoliv hodnoty integrálního datového typu získat příslušný řetězcový ekvivalent.

```
'Konverzní operace č. 2.
Dim a As Integer, b As Long
b = 100000
a = b
MsgBox a
```

Zde se již dostáváme do problémů. Ačkoliv do proměnné typu `Long` můžeme přiřadit číselnou hodnotu o šesti cifrách, přiřazovací příkaz `a = b` selže a za běhu programu bude hlášen vznik programové chyby č. 6 – Přetečení (overflow). Chyba přetečení je typickým příkladem toho, že jsme se pokusili provést operaci, při níž měla být hodnota datového typu s větším rozsahem umístěna do proměnné typu s menším rozsahem. Na reprezentaci čísla 100000 nestačí pouhých 16 bitů proměnné typu `Integer`, a proto dochází k chybě. Co je ale velice zajímavé je to, že Visual Basic 6.0 při překládání uvedeného zdrojového kódu zužující implicitní konverzi povolí. Bohužel, trpkým výsledkem takovéhoto počínání je generování chybové výjimky během exekuce aplikačního kódu. Možná si myslíte, že nápravu můžeme zabezpečit provedením explicitní konverze, třeba takto:

```
Dim a As Integer, b As Long
b = 100000
a = CInt(b)
MsgBox a
```

Když zavoláme konverzní funkci `CInt` a nabídneme ji obsah proměnné `b`, dojde opět k chybě přetečení. Ta je zapříčiněna pracovním algoritmem funkce `CInt`: přesahuje-li vstupní hodnota argumentu funkce meze datového typu `Integer`, funkce selže a iniciuje přerušení běhu aplikace ruku v ruce se zobrazením chybového hlášení. Jednoduše řečeno, tuto konverzní operaci nelze provést, a to ani pomocí explicitní konverze. Řešením je buď uložit do zdrojové proměnné typu `Long` menší hodnotu (takovou, kterou dovede interpretovat také typ `Integer`), anebo zaměnit typ cílové proměnné (`Integer`) za typ s větším rozsahem.

Postupně jsme se propracovali až k explicitním konverzím, jejichž realizaci mají na svědomí speciální konverzní funkce. Jazyk Visual Basic 6.0 jich zná celkem hodně, přičemž každá konverzní funkce přebírá argument jistého datového typu a vrací odpovídající přetypovanou hodnotu. Vzájemné porovnání konverzních funkcí pro realizaci explicitních konverzí můžete vidět v tab. 4.1.

Tab. 4.1: Komparace konverzních funkcí v jazycích Visual Basic 6.0 a Visual Basic 2005

Konverzní funkce v jazyce Visual Basic 6.0	Ekvivalentní funkce v jazyce Visual Basic 2005
CBool	CBool
CByte	CByte
CCur	**Žádný ekvivalent**
CDate	CDate
CDbl	CDbl
CDec	CDec
CInt	CInt
CLng	CLng
CSng	CSng
CStr	CStr
CVar	**Žádný ekvivalent**
CVErr	**Žádný ekvivalent**

Ve srovnání s verzí 6.0 je Visual Basic 2005 po stránce konverzních funkcí vybaven o něco lépe, protože obsahuje několik nových exemplářů, které přibližuje tab. 4.2.

Tab. 4.2: Nové konverzní funkce jazyka Visual Basic 2005

Název konverzní funkce	Charakteristika funkce
CChar	Funkce provádí přetypování prvního znaku textového řetězce typu `String` na hodnotu typu `Char`.
CObj	Funkce uskutečňuje přetypování jakékoliv hodnoty platného datového typu nebo jakéhokoliv smysluplného výrazu na hodnotu typu `Object`.
CSByte	Funkce realizuje konverzi celočíselné hodnoty z intervalu <–128, 127>. Výsledným produktem konverzní operace je hodnota typu `SByte`.
CShort	Funkce konvertuje celočíselné hodnoty z intervalu <-32 768, 32 767> do podoby hodnoty typu `Short` (16bitové celé číslo).
CUInt	Pomocí funkce `CUInt` lze vstupní argument konvertovat na „unsigned integer“, tedy celé číslo bez znaménka.
CULng	Funkce zabezpečuje přetypování celočíselné hodnoty na hodnotu typu `ULong` – celé číslo bez znaménka dlouhé 64 bitů.
CUShort	Funkce dovede zkonvertovat celá čísla z intervalu <0, 65535> tak, že se z nich stanou 16bitové celočíselné hodnoty bez znaménka.

Kromě výše popsaných explicitních konverzních funkcí má Visual Basic 2005 v záloze ještě jedno eso: jedná se o speciální funkci `CType`, se kterou jsme se již letmo setkali. Nyní nadešel čas, abychom se na ní podívali blíže. Funkce `CType` je opravdu mocná: dokáže totiž zastoupit všechny explicitní konverzní funkce! Její prototyp vypadá následovně:

```
CType(Výraz, DatovýTyp)
```

Výrazem je jakýkoliv výraz jazyka Visual Basic 2005, jenž je správný po syntaktické i sémantické stránce. Funkce `CType` nejprve určí hodnotu tohoto výrazu, a poté se pokusí zjištěnou hodnotu přetypovat na hodnotu datového typu, který určuje položka `DatovýTyp`. Návratovou hodnotou konverzní funkce `CType` je konvertovaná hodnota typu `DatovýTyp`, která může být uložena do proměnné příslušného datového typu. Kdybychom chtěli převést datum (hodnota typu `Date`) na textový řetězec typu `String`, postupovali bychom takhle:

```
Dim DnešníDatum As Date = DateTime.Today
Dim DatumJakoText As String
DatumJakoText = CType(DnešníDatum, String)
MessageBox.Show("Dneska je " & DatumJakoText & ".")
```

Použití funkce `CType` se ovšem neomezuje pouze na jednoduché konverze vestavěných hodnotových a odkazových datových typů. Tato funkce si stejně dobře poradí také s konverzemi uživatelsky definovaných typů a tříd. Podívejme se na další ukázku:

```
Dim Formulář As Form1 = Me
Dim DalšíFormulář As Form = CType(Formulář, Form)
DalšíFormulář.Text = My.User.Name
```

V kódu jsou deklarovány dvě odkazové proměnné: `Formulář` třídy `Form1` a `DalšíFormulář` třídy `Form`. Do první proměnné jsme okamžitě po deklaraci přiřadili objektovou referenci, která je nasměrována na aktivní instanci třídy `Form1`. Co je však uloženo ve druhé proměnné? Funkce `CType`

od nás obdržela objektovou referenci ukazující na instanci třídy `Form1`. Tuto referenci funkce posléze podrobila konverznímu procesu, jehož výsledkem byl vznik objektové reference, pomocí které lze přistupovat k instanci třídy `Form`. Tato konverze je možná, protože třída `Form` je mateřskou třídou třídy `Form1`. Poslední řádek kódu demonstruje změnu vlastnosti `Text` aktivní instance třídy `Form1` pomocí odkazové proměnné třídy `Form`.

4.3 Operátory

Sada operátorů dostupná v jazyce Visual Basic 2005 je citelně rozšířena. V nejnovější verzi je Visual Basic domovem těchto skupin operátorů: aritmetické operátory, operátory pro přiřazení, porovnávací operátory, operátory zřetězení, logické operátory, bitové operátory, operátory bitového posunu a speciální operátory `AddressOf`, `GetType` a `TypeOf`. V nastávajících podkapitolách prozkoumáme nejdůležitější nové operátory, které můžete využívat v prostředí Visual Basicu 2005.

4.3.1 Logické operátory AndAlso a OrElse

Visual Basic 2005 kráčí po stopách svých „.NET" předchůdců, přičemž nabízí vývojářům použití logických operátorů `AndAlso` a `OrElse`, které uskutečňují zkrácený výpočet logické konjunkce a logické disjunkce. Představme si nejprve operátor `AndAlso`. Pro zjednodušení předpokládejme, že operátor `AndAlso` bude pracovat pouze se dvěma operandy, jejichž datovým typem bude `Boolean`. Jak jsme si již řekli, operátor `AndAlso` realizuje logickou konjunkci. Z algoritmu práce této logické operace vyplývá, že pravdivostní hodnota `True` bude vrácena pouze v tom případě, že je hodnota obou operandů současně rovna `True`. Při testování svých operandů se operátor `AndAlso` podívá nejprve na první z nich. Je-li hodnota tohoto prvního operandu rovna `False`, hodnota druhého operandu již není zjišťována, ale místo toho vrací operátor `AndAlso` návratovou hodnotu `False`.

```
'Ukázka použití logického operátoru AndAlso.
Dim a As Boolean
Dim b As Boolean = True
Dim c As Boolean
c = a AndAlso b
MessageBox.Show("Hodnota proměnné c typu Boolean je " & c & ".", _
"Použití operátoru AndAlso", MessageBoxButtons.OK, _
MessageBoxIcon.Information)
```

Vzpomínanou situaci simuluje uvedený výpis programového kódu. Vzhledem k tomu, že proměnná `a` typu `Boolean` je implicitně inicilizována logickou hodnotou `False`, operátor `AndAlso` provede zkrácené vyhodnocení logické konjunkce. Ve finále získáme pomocí operátoru `AndAlso` hodnotu `False`, která bude také zobrazena i v informačním dialogu.

Druhým logickým operátorem, o němž si budeme povídat, je `OrElse`. Tento operátor se soustřeďuje na realizaci logické disjunkce – operátor vrací hodnotu `True` pokaždé, je-li alespoň jeden z operandů pravdivý, tedy vyhodnocen na hodnotu `True`. `OrElse` si však svoji práci inteligentně usnadňuje: pokud je hodnota již prvního operandu rovna `True`, druhý operand nebude vůbec analyzován, nýbrž ihned bude vrácena logická hodnota `True`. Konec konců, je to celkem pochopitelné – ať by už hodnota druhého operandu byla jakákoliv, nemá žádný vliv na konečný výsledek logické disjunkce. Ukažme si, jak si operátor `OrElse` vede v praxi.

```
'Ukázka použití logického operátoru OrElse.
Dim a As Boolean = True
Dim b, c As Boolean
c = a OrElse b
MessageBox.Show("Hodnota proměnné c typu Boolean je " & c & ".", _
"Použití operátoru OrElse", MessageBoxButtons.OK, _
MessageBoxIcon.Information)
```

4.3.2 Logické operátory Is a IsNot

Ačkoliv s operátorem `Is` jste mohli pracovat i v jazyce Visual Basic 6.0, operátor `IsNot` je novinkou, jejíž půvab objevíte teprve po vstupu do světa Visual Basicu 2005. Pomocí operátoru `Is` mohou programátoři zjišťovat, zda dvě objektové proměnné obsahují reference mířící na jeden a tentýž objekt.

```
'Kód jazyka Visual Basic 6.0.
Dim obj_1 As Form1
Dim obj_2 As Form1
Set obj_1 = New Form1
obj_1.Show: obj_1.Move 100
Set obj_2 = obj_1
If obj_1 Is obj_2 Then
MsgBox "Reference jsou shodné."
Else
MsgBox "Reference nejsou shodné."
End If
```

Představený fragment programového kódu předvádí aplikaci operátoru `Is`. Pomocí něho zkoumáme, zda proměnné `obj_1 a obj_2` uchovávají stejné objektové reference. V našem případě tyto proměnné skutečně stejné reference uchovávají, takže v informačním dialogu bude zobrazena zpráva o jejich shodnosti.

Logický operátor `IsNot` vznikl kombinací operátorů `Is` a `Not`. Jeho použití není vůbec složité: operátor zkoumá nerovnost dvou objektových referencí. Řečeno jinak, pomocí operátoru `IsNot` můžete snadno zjistit, zda dvě odkazové proměnné obsahují reference ukazující na dva různé objekty. Když se pokusíme přepsat předcházející fragment kódu jazyka Visual Basic 6.0 pro potřeby verze 2005 s využitím operátoru `IsNot`, dojdeme k následujícím výsledkům:

```
Dim obj_1 As Form1
Dim obj_2 As Form1
obj_1 = New Form1
obj_1.Show()
obj_1.DesktopLocation = New Point(100, 100)
obj_2 = obj_1
If obj_1 IsNot obj_2 Then
   MessageBox.Show("Objektové reference nejsou shodné.")
Else
   MessageBox.Show("Objektové reference jsou shodné.")
End If
```

Tento kód odkrývá více změn, nejenom použití operátoru `IsNot`. Začněme vytvořením instance třídy `Form1`. Jak vidíte, již není třeba, abychom v přiřazovacím příkazu používali klíčové slovo `Set`. Novou instanci třídy `Form1` zobrazíme zavoláním metody `Show`. Ačkoliv je název metody stejný jako v jazyce Visual Basic 6.0, její zápis se liší: nyní je nutné za názvem metody uvádět také kulaté závorky. (Jestliže je nezadáte sami, Visual Basic 2005 je za vás doplní.) Zcela jinak se po novém nastavuje pozice formuláře na pracovní ploše. Metodu `Move` vystřídala vlastnost `DesktopLocation`, která za asistence nové

instance struktury `Point` z jmenného prostoru `System.Drawing` determinuje přesnou polohu formuláře na ploše, ovšem nikoliv ve twipech, ale v pixelech. A kdepak je náš operátor `IsNot`? Ten si našel své místečko v rozhodovacím příkazu `If...Then...Else`, který porovnává objektové reference uložené v odkazových proměnných `obj_1 a obj_2`. V tomto případě vrátí operátor `IsNot` hodnotu `True`, což znamená, že byla nalezena shoda. Obě proměnné se totiž odkazují na jeden objekt třídy `Form1`.

4.3.3 Operátory bitového posunu

Pakliže přicházíte z jazyka Visual Basic 6.0, budete se pravděpodobně o operátorech bitového posunu chtít dozvědět víc. Naopak, jste-li znalí Visual Basicu .NET 2003, žádné nové překvapivé objevy vás nečekají. Ale pojďme pěkně po pořádku. Operátory bitového posunu provádějí operace s jednotlivými bity integrálních datových typů `Byte`, `Short`, `Integer` a `Long`. Visual Basic 2005 definuje dva operátory pro bitový posun: jedná se o operátor bitového posunu doleva (<<) a operátor bitového posunu doprava (>>). Oba operátory posouvají bity celočíselné proměnné o jistý počet pozic v zadaném směru, ovšem jejich pracovní algoritmus se liší. Operátory začínají svoji činnost tím, že hodnotu numerického datového typu převedou do binární podoby. Pak dochází k přeskupení bitů doleva nebo doprava podle přesně stanovených instrukcí. Nakonec je získána nová celočíselná hodnota, která je výsledkem práce operátorů bitového posunu.

Operátor bitového posunu doleva (<<) pracuje následovně: Poté, co je vytvořen binární ekvivalent celočíselné hodnoty, začne operátor s posouváním bitů na jednotlivých pozicích směrech doleva. Bity na uvolněných pozicích jsou přitom zaplněny nulami. Přepokládejme, že bychom chtěli na hodnotu 100, kterou máme uloženou v proměnné typu `Integer`, aplikovat operátor bitového posunu doleva, který přeskupí všechny bity o dvě pozice určeným směrem.

```
'Ukázka použití operátoru bitového posunu doleva (<<).
Dim Operand_1 As Integer, Operand_2 As Integer
Operand_1 = 100
Operand_2 = Operand_1 << 2
MessageBox.Show(CStr(Operand_2))
```

Když převedeme číslo 100 do dvojkové soustavy, získáme hodnotu 1100100.

POZNÁMKA: Hodnoty v desítkové a binární soustavě

Jestliže si nejste celkem jisti, jak jsme přišli k závěru, že binární hodnota 1100100 se rovná číslu 100 v desítkové soustavě, zde je malý úvod do teorie konverze hodnot mezi různými číselnými soustavami.

Když chceme hodnotu decimální soustavy vyjádřit v její binární podobě, postupujeme tak, že tuto hodnotu neustále dělíme dvěma, přičemž zaznamenáváme zbytky po dělení, které mohou nabývat pouze dvou hodnot, a to 0 nebo 1. Když pak zbytky po dělení přečteme inverzně, získáme finální binární hodnotu, která odpovídá vstupnímu číslu v desítkové soustavě. Vrátíme-li se k našemu příkladu s číslem 100, můžeme celý matematický postup znázornit takto:

Matematická operace	Celá část výsledku	Zbytek po dělení
100 : 2	50	0
50 : 2	25	0
25 : 2	12	1
12 : 2	6	0
6 : 2	3	0
3 : 2	1	1
1 : 2	0	1
Binární hodnota: 1100100		

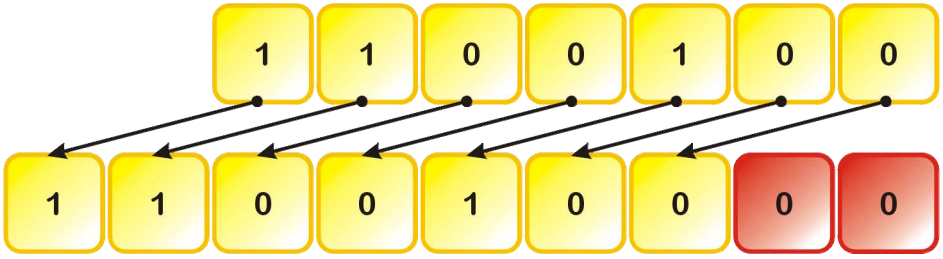
Jak se ovšem přesvědčíme o korektnosti prezentovaného převodu? Inu, pomůžeme si zkouškou správnosti, jejíž pomocí najdeme desítkový ekvivalent vypočtené binární hodnoty. Bude-li výsledkem číslo 100, náš postup byl zcela správný.

Decimální protějšek binární hodnoty určíme podle následující vzorce:

$1100100 = 1*2^6 + 1*2^5 + 0*2^4 + 0*2^3 + 1*2^2 + 0*2^1 + 0*2^0$
$1100100 = 64 + 32 + 4$
$1100100 = 100$

Jak je vidět, počínali jsme si bez nejmenších chybiček. Nyní již víte, jak číselní konverze ve skutečnosti fungují. Stejným postupem pracuje i Visual Basic 2005, když je zapotřebí provádět konverze hodnot desítkové a binární soustavy.

Ve chvíli, kdy disponujeme binární hodnotou čísla 100, nastartuje operátor `<<` posun jednotlivých bitů o 2 pozice směrem doleva (obr. 4.14).



Obr. 4.14: Grafická ilustrace posunu bitů hodnoty 1100100 o dvě pozice doleva

Každý bit je posunut o dvě pozice, což s sebou nese dva důsledky:

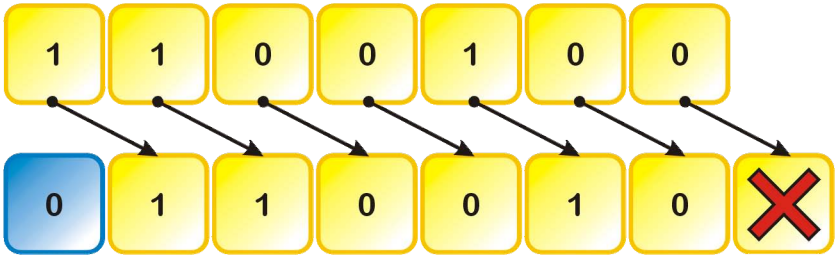
1. Získáváme 2 rozšiřující bity na levé straně.
2. Uvolněné bitové pozice napravo jsou zaplněny nulami (na obrázku jsou znázorněny červenými políčky).

Produktem operátoru bitového posunu doleva je tedy binární hodnota 110010000, která je v typu `Integer` reprezentována celočíselnou hodnotou 400. Tato hodnota se objeví také v informačním dialogu.

Operátor bitového posunu doprava (>>) se zabývá modifikací pozicí bitů hodnot datových typů `Byte`, `Short`, `Integer` a `Long` směrem doprava. Bity, jejichž pozice je posunuta za pozici bitu, jenž se nachází nejvíce napravo, jsou odstraněny. Na druhou stranu, uvolněné bitové pozice jsou zaplněny jedničkami nebo nulami, podle toho, zda je hodnota záporná nebo kladná. Je-li hodnota kladná, uvolněné bitové pozice jsou zaplněny nulami. Naopak, pokud je hodnota záporná, do uvolněných pozic se dostanou jedničky. Pro lepší názornost si pomozme příkladem. V níže uvedeném kódu používáme operátor >> pro posun bitů celočíselné hodnoty 100 o jednu pozici směrem doprava:

```
'Ukázka použití operátoru bitového posunu doprava (>>).
Dim Operand_1, Operand_2 As Integer
Operand_1 = 100
Operand_2 = Operand_1 >> 1
MessageBox.Show(Operand_2.ToString)
```

Dokážete nyní povědět, co bude zobrazeno v okně se zprávou? Říkáte-li, že číslo 50, máte pravdu. Podívejme se však, proč je tomu tak. Podobně jako v minulé ukázce, i v té zdejší je nejprve převedena hodnota 100 do své binární podoby. Již víte, že tato má podobu 1100100. Má-li operátor >> k dispozici binární hodnotu, začne s přeskupováním bitů o 1 pozici směrem doprava, tak jak to nařizuje programový kód. Zatímco přesahující bity jsou zlikvidovány, uvolněné bitové pozice jsou zaplněny nulami (je to proto, že hodnota 100 je kladná). Grafické znázornění práce operátoru bitového posunu doprava přináší obr. 4.15.



Obr. 4.15: Grafická ilustrace posunu bitů hodnoty 1100100 o jednu pozici doprava

Z obrázku je zřejmé, že po aplikaci operátoru bitového posunu doprava získáváme binární hodnotu 0110010, která odpovídá celému číslu 50 v desítkové soustavě ($0*2^6 + 1*2^5 + 1*2^4 + 0*2^3 + 0*2^2 + 1*2^1 + 0*2^0$).

Operátory bitového posunu realizují aritmetický bitový posun, což znamená, že operace přesouvání bitů není cyklická: přesahující bity z jedné strany binární hodnoty nejsou opětovně zařazovány na její začátek.

4.3.4 Kombinované přiřazovací operátory

Kombinované přiřazovací operátory představují velkou inovaci, která vám dovede znamenitě urychlit práci. Jazyk Visual Basic 2005 definuje skupinu o celkem devíti kombinovaných operátorech pro přiřazení, která je složena ze šesti aritmetických operátorů, dvou operátorů bitového posunu a jednoho operátoru pro zřetězení. Přehled kombinovaných operátorů přináší tab. 4.3.

Tab. 4.3: Přehled kombinovaných operátorů v jazyce Visual Basic 2005	
Syntaxe operátoru	**Druh operátoru**
+=	Kombinované přiřazovací aritmetické operátory
–=	
***=**	
/=	
\=	
^=	
<<=	Kombinované přiřazovací operátory bitového posunu
>>=	
&=	Kombinovaný přiřazovací operátor pro zřetězení

Kombinované operátory pracují ve dvou krocích:

1. Nejprve je na operand stojící napravo od kombinovaného přiřazovacího operátoru aplikován přítomný aritmetický operátor, operátor bitového posunu nebo operátor pro zřetězení.

2. Poté, co je změněna hodnota pravého operandu, je tato pomocí operátoru pro přiřazení (=) uložena do operandu, jenž stojí po levé straně kombinovaného přiřazovacího operátoru.

Povězme, že máme následující útržek zdrojového kódu:

```
'Ukázka použití kombinovaného přiřazovacího operátoru +=.
Dim op As Long
op += 10
```

Středem našeho zájmu je operátor += čili kombinovaný přiřazovací aritmetický operátor. Operátor nejprve uskuteční inkrementaci hodnoty proměnné op o 10 jednotek a posléze tuto modifikovanou hodnotu do uvedené proměnné přiřadí. Kdybychom chtěli, mohli bychom pracovní postup operátoru += rozepsat takto:

```
'Pomocí tohoto kódu můžeme simulovat chování operátoru +=.
Dim op As Long
op = op + 10
```

Ano, oba dva fragmenty zdrojového kódu jsou stejné v tom smyslu, že se dopátrají ke shodným výsledkům. Přímé použití kombinovaného přiřazovacího operátoru je ovšem rychlejší a efektivnější. Navíc kód, v němž se nachází operátor +=, působí syntakticky příjemně, přičemž se ponáší na kód jazyků C# nebo C++.

V tab. 4.4 můžete vidět všechny kombinované přiřazovací operátory a jejich ekvivalentní zápisy.

Tab. 4.4: Kombinované přiřazovací operátory a jejich ekvivalenty

Operátor	Použití kombinovaného operátoru pro přiřazení	Ekvivalentní zápis
`+=`	`Dim Hodnota As Integer` `Hodnota += 100`	`Dim Hodnota As Integer` `Hodnota = Hodnota + 100`
`-=`	`Dim Hodnota As Integer` `Hodnota -= 100`	`Dim Hodnota As Integer` `Hodnota = Hodnota - 100`
`*=`	`Dim Hodnota As Integer` `Hodnota *= 5`	`Dim Hodnota As Integer` `Hodnota = Hodnota * 5`
`/=`	`Dim Hodnota As Double` `Hodnota /= 2`	`Dim Hodnota As Double` `Hodnota = Hodnota / 2`
`\=`	`Dim Hodnota As Integer` `Hodnota \= 3`	`Dim Hodnota As Integer` `Hodnota = Hodnota \ 3`
`^=`	`Dim Hodnota As Double = 2.2` `Hodnota ^= 3`	`Dim Hodnota As Double = 2.2` `Hodnota = Hodnota ^ 3`
`<<=`	`Dim Hodnota As Integer = 22` `Hodnota <<= 2`	`Dim Hodnota As Integer = 22` `Hodnota = Hodnota << 2`
`>>=`	`Dim Hodnota As Integer = 22` `Hodnota >>= 2`	`Dim Hodnota As Integer = 22` `Hodnota = Hodnota >> 2`
`&=`	`Dim Text As String = "Ahoj, "` `Text &= "Visual Basicu 2005!"`	`Dim Text As String = "Ahoj, "` `Text = Text & "Visual Basicu 2005!"`

4.4 Cykly

Iterační příkazy neboli cykly jsou pevnou součástí každého programovacího jazyka a nejinak je tomu i ve Visual Basicu. Verze 2005 však i do této rádoby stabilní oblasti uvádí několik změn, o nichž pojednáme více. V novém Visual Basicu se můžete střetnout s následujícími typy cyklů:

1. Cyklus `For...Next`
2. Cyklus `For Each...Next`
3. Cyklus `Do...Loop` a jeho modifikace:
   - `Do-While...Loop`
   - `Do-Until...Loop`
   - `Do...Loop-While`
   - `Do...Loop-Until`
4. Cyklus `While...End While`

Změny, o nich byste měli vědět, si uvedeme v přehledném seznamu:

1. Cyklus `While...Wend` je nahrazen cyklem `While...End While`.
2. Řídící proměnné cyklů `For...Next` a `For Each...Next` lze nyní deklarovat přímo v hlavičkách těchto cyklů.
3. Proměnné deklarované v tělech cyklů jsou dostupné pouze v těchto oblastech. Nelze je tudíž použít mimo programového kódu cyklu.
4. Nový příkaz `Continue` dovoluje programátorům opustit jednu a započít novou iteraci cyklu.
5. Editor zdrojového kódu jazyka Visual Basic 2005 provádí automatické doplňování syntaktické konstrukce cyklů.

Všechny popsané modifikace si představíme blíže dále v textu této kapitoly.

4.4.1 Cyklus While...End While

Pro tentokrát začneme nezvykle od konce, a sice od cyklu `While...End While`. Tento cyklus nahrazuje cyklus `While...Wend`, který dobře znáte z jazyka Visual Basic 6.0. Změna v cyklu je ovšem ryze syntaktická: uzavírací příkaz `End While` byl zvolen proto, aby byla dosažena konformita s inovovanou syntaktickou jazykovou specifikací. Jinak se cyklus `While...End While` chová přesně tak jako jeho starší bratříček. To znamená, že příkazy nacházející se v těle cyklu jsou vykonávány stále dokola, až dokud se testovací podmínka stane neplatnou.

```
'Cyklus While...Wend v jazyce Visual Basic 6.0.
Dim x As Integer
x = 1
While x <= 100
List1.AddItem ("Toto je " & x & ". položka seznamu.")
x = x + 1
Wend
```

Cyklus `While...Wend` zjišťuje hodnotu proměnné `x` typu `Integer` a je-li tato hodnota menší nebo rovna než 100, přidává do instance ovládacího prvku `ListBox` (implicitně pojmenované jako `List1`) příslušnou položku. Aby mohl cyklus plynule běžet, provádíme po přidání položky do seznamu inkrementaci hodnoty proměnné `x` o 1.

Pokud bychom chtěli přepsat uvedený cyklus do adekvátní podoby, kterou by bylo možné použít v prostředí jazyka Visual Basic 2005, postupovali bychom podle následujícího vzoru:

```
'Cyklus While...End While v jazyce Visual Basic 2005.
Dim x As Integer = 1
While x <= 100
    ListBox1.Items.Add("Toto je " & x & ". položka seznamu.")
    x += 1
End While
```

TIP: Novinky jazyka Visual Basic 2005 nám urychlují a zefektivňují práci

Jestliže odhlédneme od použití nové syntaktické formy cyklu `While...End While` a zbystříme pozornost, zjistíme, že díky Visual Basicu 2005 můžeme zefektivnit kód ve dvou směrech:

1. Explicitní inicializace proměnné `x` v rámci deklaračního příkazu `Dim`.
2. Použití kombinovaného přiřazovacího aritmetického operátoru `+=` pro zvětšení hodnoty proměnné `x` o 1.

Kromě aplikace cyklu `While...End While` za zmínku stojí také použití instance ovládacího prvku `ListBox`. Tato instance nese standardní jméno `ListBox1`. Liší se však postup přidávání položek do instance. Zatímco ve Visual Basicu 6.0 jsme volali metodu `AddItem`, které jsme předali argument v podobě textového řetězce typu `String`, ve Visual Basicu 2005 postupujeme jinak. Nejprve pomocí vlastnosti `Items` instance získáme přístup ke kolekci položek, a poté aktivujeme metodu `Add`, která nám pomůže s přidáním položky nové. Na rozdíl od metody `AddItem` metoda `Add` pracuje s formálním parametrem typu `Object`, z čehož plyne, že vstupní hodnotou může být cokoliv, na co lze „ukázat" objektovou referencí. Po přidání položky vrací metoda `Add` celočíselnou návratovou hodnotu typu `Integer`, která udává index položky v celém seznamu.

4.4.2 Deklarace řídících proměnných v hlavičkách cyklů For...Next a For Each...Next

Pokud chtěli vývojáři v jazyce Visual Basic 6.0 použít v hlavičce cyklu `For...Next` řídící proměnnou, museli provést její deklaraci ještě před kódem samotného cyklu.

```
'VB 6.0: Řídící proměnnou cyklu For...Next je nutno deklarovat
'před jejím použitím v hlavičce cyklu.
Dim i As Integer
For i = 1 To 10
MsgBox i
Next i
```

V tomto fragmentu zdrojového kódu hraje roli řídící proměnné proměnná `i` typu `Integer`. Proměnnou chceme použít pro řízení cyklu `For...Next`, a proto ji musíme deklarovat před tímto cyklem. Jazyk Visual Basic 2005 nabízí programátorům větší míru volnosti, neboť jim dovoluje odložit deklaraci řídící proměnné cyklu `For...Next` do okamžiku jejího skutečného použití. Ve Visual Basicu 2005 proto můžeme bez jakýchkoliv potíží upotřebit tento programový kód:

```
'VB 2005: Řídící proměnnou je nyní možné deklarovat přímo
'v hlavičce cyklu For...Next.
For i As Integer = 1 To 10
    MessageBox.Show(CStr(i))
Next
```

Řídící proměnná je deklarována v hlavičce cyklu `For...Next`, přičemž je určen také její datový typ. Aby vše pracovalo v pořádku, nesmí být stejná proměnná deklarována v programovém kódu, jenž se nachází před tímto cyklem. Obor platnosti takto deklarované řídící proměnné je ohraničen tělem cyklu. Zjednodušeně řečeno, proměnnou lze použít kdykoliv mezi příkazy `For` a `Next` cyklu, ovšem již ne mimo kódu cyklu. Ačkoliv je datovým typem řídící proměnné v našem případě vestavěný hodnotový typ `Integer`, lze stejně dobře použít také odkazový typ, a to jak vestavěný (`String`, `Object`) tak uživatelsky definovaný.

```
'Datovým typem řídící proměnné může být i zástupce odkazových
'datových typů. V tomto případě je to typ Object.
For o As Object = 1 To 5
    o = New Object
Next
```

Zaměřme se nyní na cyklus `For Each...Next`. Síla tohoto cyklu se dostává do popředí především tehdy, když je zapotřebí prozkoumávat rozsáhle kolekce elementů různých typů. Když jsme chtěli v jazyce Visual Basic 6.0 zjistit názvy všech instancí ovládacích prvků umístěných na ploše formuláře, napsali jsme následující řádky programového kódu:

```
'VB 6.0: Cyklus For Each...Next zjišťuje jména všech instancí
'ovládacích prvků nacházejících se na formuláři. Řídící proměnná
'je deklarována před svým použitím v cyklu.
Dim c As Control
For Each c In Me.Controls
MsgBox c.Name
Next c
```

Přehupneme-li se zpět do Visual Basicu 2005, můžeme zrod řídící proměnné cyklu `For Each...Next` přesunout přímo do hlavičky tohoto cyklu.

```
'VB 2005: Deklarace řídící proměnné se odehrává v hlavičce cyklu
'For Each...Next.
For Each c As Control In Me.Controls
    MessageBox.Show(c.Name)
Next c
```

Cyklus `For Each...Next` uskutečňuje procházení kolekcí instancí ovládacích prvků inverzním směrem, což znamená, že v informačním dialogu se jako první objeví název posledně přidané instance.

4.4.3 Obor platnosti proměnné deklarované v těle cyklu

S příchodem Visual Basicu 2005 se mění chápání oboru platnosti proměnných deklarovaných uvnitř iteračních příkazů. Po dlouhé doby mohli programátoři takto deklarované proměnné využívat i mimo cyklů, neboť jejich oborem platnosti byla celá procedura, v níž byl cyklus uložen. Následující ukázka předvádí, že k proměnné `j` typu `Integer` bylo možné v prostředí jazyka Visual Basic 6.0 přistupovat i poté, co byl opuštěn kontext cyklu.

```
'VB 6.0: Proměnná deklarovaná v těle cyklu je přístupná i mimo něj.
Dim i As Byte
For i = 0 To 255 - 1
Dim j As Integer
j = i
Next i
j = 10
MsgBox j
```

Přestože je sledovaná proměnná deklarována uvnitř cyklu, lze s ním pracovat i mimo cyklu. Funkce `MsgBox` proto zobrazí hodnotu 10, která byla do proměnné `j` přiřazena v posledním kroku.

Jakmile do editoru zdrojového kódu Visual Basicu 2005 zadáme kód, v němž se budeme snažit o použití proměnné deklarované v těle cyklu i mimo něj, kompilátor nás obeznámí s chybovým hlášením. Toto hlášení bude říkat, že proměnná není deklarována.

```
'VB 2005: Pozdější přístup k proměnné j není povolen.
Dim i As Byte
For i = 0 To Byte.MaxValue - 1
    Dim j As Integer
    j = i
Next i
'S následujícími dvěma řádky nebude kompilátor spokojen.
j = 10
MessageBox.Show(CStr(j))
```

Visual Basic 2005 nám nedovolí inicializovat proměnnou `j`, protože tato byla deklarována uvnitř cyklu, což znamená, že jejím oborem platnosti je právě tento cyklus. Ačkoliv životní cyklus proměnné `j` se kryje se životním cyklem procedury, v níž je cyklus umístěn, proměnná je použitelná pouze v kontextu cyklu.

POZNÁMKA: Struktura **Byte** a její vlastnosti **MinValue** a **MaxValue**

Instance hodnotové struktury `Byte` z jmenného prostoru `System` představuje 8bitové celé číslo bez znaménka z intervalu <0, 255>. Struktura definuje dvě veřejně přístupné datové položky `MinValue` a `MaxValue`, které na požádání vrací nejmenší, respektive největší hodnotu, kterou lze do proměnné typu `Byte` uložit.

4.4.4 Nová iterace cyklu pomocí příkazu Continue

Jednou ze silných zbraní, které tvoří arzenál nového Visual Basicu, je i příkaz `Continue`. Tento příkaz dovede v příhodném okamžiku ukončit jednu iteraci cyklu a nastartovat iteraci novou. Příkaz `Continue` lze aplikovat ve společnosti cyklů z rodin `Do...Loop`, `For...Next` a `While...End While`. Nachází-li se v těle některého z uvedené rodiny cyklů příkaz `Continue`, exekuce programového kódu se neprodleně přesouvá k hlavičce cyklu a dochází k testování podmínky, která určuje běh cyklu. Je-li podmínka splněna, je vykonána nová iterace cyklu, a to aniž by byla ta předchozí zcela dokončena. Jestliže máte programátorský „background" v jazycích jako je třeba C# nebo C++, pracovní princip příkazu `Continue` (v uvedených prostředích s malým „c" v názvu) vám jistě není neznámý. Výborné je, že nyní mají takový příkaz k dispozici i vývojáři ve Visual Basicu!

```
'VB 2005: Symbioza příkazu Continue a cyklu For...Next.
For a As Byte = Byte.MinValue To Byte.MaxValue - 1
    If a >= 100 And a <= 150 Then Continue For
    ListBox1.Items.Add(a)
Next
```

Cyklus provádí své iterace tak dlouho, dokud je hodnota řídící proměnné v intervalu vymezeném výrazy `Byte.MinValue` a `Byte.MaxValue - 1`. Pokud je ale aktuální hodnota proměnné z intervalu <100, 150>, je aktivován příkaz `Continue For`, který ukončí exekuci stávající iterace cyklu, přesune zaměření na test podmínky a je-li tato splněna, realizuje novou iteraci cyklu. To v praxi znamená, že do seznamu instance `ListBox1` nebudou přidána čísla od 100 po 150.

Všimněte si, že syntaktická podoba příkazu `Continue` se mění podle toho, v jakém typu cyklu je tento příkaz přítomen. Možné varianty příkazu `Continue` jsou sdruženy v tab. 4.5.

Tab. 4.5: Syntaktické varianty příkazu Continue

Cyklus	Podoba příkazu Continue
For...Next	Continue For
For Each...Next	Continue For
Do...Loop a jeho modifikace: • **Do-While...Loop** • **Do-Until...Loop** • **Do...Loop-While** • **Do...Loop-Until**	Continue Do
While...End While	Continue While

Další fragment programového kódu poukazuje na získání kolekce aktivních procesů, které běží v operačním systému Microsoft Windows XP. K procesům a jejím názvům se dostaneme pomocí metody `GetProcesses` třídy `Process` z jmenného prostoru `System.Diagnostics` a cyklu `For Each...Next`.

```
Dim PočetProcesů As Short
ListBox1.Sorted = True
For Each Proces As Process In Process.GetProcesses
    ListBox1.Items.Add(Proces.ProcessName)
    PočetProcesů += CShort(1)
Next
MessageBox.Show("Celkový počet procesů: " & PočetProcesů & ".", _
"Kolekce procesů běžících v operačním systému", MessageBoxButtons.OK, _
MessageBoxIcon.Information)
```

Předpokládejme, že nebudeme chtít analyzovat procesy aplikací Word a Excel. Za těchto okolností můžeme do těla cyklu `For Each...Next` vložit rozhodovací příkaz `If...Then` s klauzulí `Continue For`:

```
Dim PočetProcesů As Short
ListBox1.Sorted = True
For Each Proces As Process In Process.GetProcesses
    If Proces.ProcessName = "WINWORD" Or _
    Proces.ProcessName = "EXCEL" Then Continue For
    ListBox1.Items.Add(Proces.ProcessName)
    PočetProcesů += CShort(1)
Next
MessageBox.Show("Celkový počet procesů: " & PočetProcesů & ".", _
"Kolekce procesů běžících v operačním systému", MessageBoxButtons.OK, _
MessageBoxIcon.Information)
```

Příkaz `Continue` může najít své uplatnění i při programování virtuálního hlídače, který bude ověřovat, zda uživatel zadal správné heslo pro vstup do počítačového systému. Ještě než se společně podíváme na následující výpis programového kódu, podotknu, že všechny nezbytné operace se odehrávají v těle cyklu `Do...Loop`.

```
Dim Heslo As String, PočetZadání As Byte
Do
    Heslo = InputBox("Pro vstup do systému zadejte heslo:", "Zadání hesla")
    If Heslo = "" Then Exit Sub
    PočetZadání += CByte(1)
    If PočetZadání = 3 Then
        MessageBox.Show("Je mi líto, ovšem do systému vás " & _
        "nemůžu vpustit.", "Pokus o neautorizovaný vstup", _
        MessageBoxButtons.OK, MessageBoxIcon.Exclamation)
        Exit Do
    End If
    If Heslo <> "mars" Then
        Continue Do
    Else
        MessageBox.Show("Vítejte v systému, pane uživateli!", _
        "Vítejte", MessageBoxButtons.OK, MessageBoxIcon.Information)
        Exit Do
    End If
Loop
```

Virtuální hlídač zobrazuje okno pro zadání hesla (v tomto směru je pro nás velice užitečná funkce `InputBox`, která je populární již od dob šesté verze Visual Basicu). Jestliže uživatel nezadá žádné heslo,

anebo zadá třikrát po sobě špatné heslo, hlídač zobrazí hlášení o neautorizovaném vstupu do systému a celou proceduru ukončí. Zadá-li uživatel správné heslo („mars“), hlídač jej vpustí do systému. Povšimněte si zejména použití příkazu `Continue Do` v rozhodovacím příkazu `If...Then...Else`.

Příkaz `Continue` může být aplikován i ve spojitosti s vnořeným cyklem, tedy cyklem, který je umístěn v těle jiného cyklu. Je-li příkaz `Continue` přítomen ve vnořeném cyklu, jeho chování může být variabilní v závislosti od typu nadřazeného a vnořeného cyklu. Pokud oba cykly pocházejí ze stejné rodiny cyklů (například `For...Next`), lze pomocí příkazu `Continue` odstartovat novou iteraci pouze vnořeného cyklu (tedy cyklu, v němž se příkaz `Continue` aktuálně nachází).

```
For x = 1 To 2
    For y = 10 To 1 Step -1
        'Příkaz Continue For se vztahuje pouze na vnořený cyklus.
        If y <= 5 Then Continue For
        MessageBox.Show(CStr(y))
    Next y
Next x
```

Jestliže ovšem nejsou cykly stejného typu (kupříkladu `For Each...Next` a `While...End While`), příkaz `Continue` je možné použít ve dvou variantách (`Continue For` a `Continue While`) a iniciovat tak novou iteraci nadřazeného nebo vnořeného cyklu.

```
Dim Formulář As New Form
Dim Tlačítko1 As New Button
With Tlačítko1
    .Text = "Tlačítko1"
    .Location = New Point(10, 10)
    .Size = New Size(60, 30)
End With
Formulář.Controls.Add(Tlačítko1)
Formulář.Show()
For Each Prvek As Control In Formulář.Controls
    If TypeOf Prvek Is Button Then
        Dim Tlačítko_Odkaz As Button = CType(Prvek, Button)
        While Tlačítko_Odkaz.Top < 200
            Threading.Thread.Sleep(50)
            Tlačítko_Odkaz.Top += 10
            Formulář.Refresh()
            'Příkaz Continue While začíná další iteraci vnořeného cyklu.
            If Tlačítko_Odkaz.Top > 150 Then Continue While
        End While
    End If
Next
```

Tentokrát je výpis programového kódu složitější. Vytváříme v něm novou instanci třídy `Form` a novou instanci třídy `Button`. Možnost konstruovat instance specifických ovládacích prvků je vysoce ceněnou inovací jazyka Visual Basic 2005, která staví tento jazyk do zcela jiného světla. Poté, co jsou nastaveny vybrané vlastnosti tlačítka (`Text`, `Location` a `Size`), přidáváme tlačítko na formulář, který vzápětí zviditelňujeme voláním metody `Show`. V této chvíli do hry vstupuje iterační příkaz `For Each...Next`, jenž prochází kolekcí instancí ovládacích prvků formuláře. Pokud je nalezeno tlačítko, získanou objektovou referenci přetypujeme a uložíme do přichystané odkazové proměnné `Tlačítko_Odkaz`. V tomto bodě vstupuje kód do vnořeného cyklu `While...End While`, jehož úkolem je posouvat tlačítko směrem dolů po ploše formuláře. Posun tlačítka realizujeme inkrementací hodnoty vlastnosti `Top`. Abychom simulovali animaci postupné změny vlastnosti `Top` tlačítka, při každém průchodu cyklem uspíme aplikaci na 50 milisekund (uvedení primárního aplikačního vlákna do stavu spánku má na starosti sdílená metoda `Sleep` třídy `Thread` z jmenného prostoru `System.Threading`). Animace by

však nebyla zřetelná, kdybychom plochu formuláře po každé změně pozice tlačítka nepřekreslili. Za tímto účelem voláme instanční metodu formuláře s názvem `Refresh`. V rozhodovacím příkazu `If...Then` testujeme podmínku a aktivujeme příkaz `Continue While`, jenž zařídí start nového průchodu cyklem `While...End While`.

4.4.5 Automatické doplňování syntaktické konstrukce cyklů

Technologie IntelliSense, která je začleněna do integrovaného vývojového prostředí Visual Studia 2005, je mnohem vyspělejší než její kolegyně z roku 1998. IntelliSense bedlivě sleduje všechny vaše kroky, jež uskutečníte v editoru zdrojového kódu jazyka Visual Basic 2005. Pokud IntelliSense zjistí, že vám může pomoci, rychle tak udělá. Do portfolia služeb této senzitivní technologie patří i automatické doplňování syntaktické konstrukce cyklů `For...Next`, `For Each...Next`, `Do...Loop` (a jeho modifikací) a `While...End While`. Automatická kompletizace je obrovským přínosem, protože inteligentně doplňuje ty partie syntaktické stavby cyklů, které budete při své práci potřebovat. A kdy že se automatické doplňování dostává ke slovu? V těchto případech:

1. Jestliže programátor zapíše hlavičky cyklů `For...Next` a `For Each...Next` a stiskne klávesu ENTER, technologie IntelliSense vytvoří prázdná těla cyklů, přenese do nich zaměření (kurzor) a ohraničí je uzavíracími příkazy `Next`.

2. Zadá-li vývojář příkaz `Do` a aktivuje klávesu ENTER, Visual Basic 2005 vytvoří tělo cyklu `Do...Loop`, přesune do něj zaměření a přidá finalizační příkaz `Loop`. Tato pravidla platí také pro modifikace cyklu `Do...Loop` čili pro cykly `Do-While...Loop`, `Do-Until...Loop`, `Do...Loop-While` a `Do...Loop-Until`.

3. Poté, co byl zadán příkaz `While`, technologie IntelliSense iniciuje sestrojení těla budoucího cyklu, přesunutí kurzor myši na správné místo a přidání příkazu `End While`.

4.5 Pole

V této podkapitole rozebereme problematiku změn při programování s poli v jazyce Visual Basic 2005. Než se však pustíme do výkladu, měl bych vás upozornit, že v oblasti polí nezůstal kámen na kameni, takže novinek je opravdu hodně. No na druhou stranu, pole patří k základním datovým strukturám, které jsou při vývoji softwaru používány, a proto je jejich pochopení stěžejním krokem ke kariéře programátora ve Visual Basicu 2005.

První změnou, která vám ihned udře do očí, je zcela nová struktura polí. Pole již není chápáno pouze jako vyhrazená oblast operační paměti, kterou obsazují proměnné stejného datového typu. Nyní jsou pole plnohodnotnými objekty, přesněji řečeno instancemi třídy `Array` z jmenného prostoru `System`. Pole rovněž nabyly na inteligenci, neboť třída `Array` nabízí dostatečný počet metod pro vytváření, prohledávání a třídění jednotlivých polí. Nicméně programátoři nevytvářejí pole pomocí operátoru `New` a třídy `Array`. Tato třída je ve skutečnosti opatřena modifikátorem `MustInherit`, jenž říká, že třída musí sloužit jako bázová třída a nemůže být subjektem pro explicitní konstrukci instancí.

V jazyce Visual Basic 2005 mohou vývojáři pracovat s několika kategoriemi polí. První kriterium pro klasifikaci polí je jejich stálost v době kompilace programového kódu. Pokud již v tomto momentě víte, jak velké pole bude, mluvíme o statických polích. Naopak, není-li v době kompilace znám přesný počet prvků pole, jedná se o pole dynamické, které bude inicializováno teprve za běhu aplikace .NET. Pole lze dále členit podle počtu dimenzí, na jednorozměrná, dvourozměrná (2D) a trojrozměrná (3D) pole.

Největší frekvenci využití se těší jednorozměrná pole, která jsou někdy označována také termínem vektorová pole. Pole o jednom rozměru je vhodné použít pro zaznamenání řady navzájem souvisejících údajů, které jsou stejného datového typu. Aby ovšem nedošlo k nedorozumění: Ačkoliv všechny elementy pole musí disponovat totožným datovým typem, jejich obsah se může lišit. Jestliže bude datovým typem položek pole `Object`, pak každý element pole může obsahovat buď hodnotu hodnotového datového typu (například `Byte`, `Short`, `Integer` nebo `Boolean`), anebo objektovou referenci směřující na instanci odkazového typu uložené na řízené hromadě.

2D-pole nacházejí své upotřebení při manipulaci s tabulkou dat, přičemž jejich inicializace se nejčastěji provádí pomocí vnořeného cyklu. V kódu běžných softwarových aplikací neuvidíte trojrozměrná pole příliš často – tato se uplatňují spíše u složitějších vědeckých a finančních operací, ale také třeba při programování počítačové grafiky.

Pokud má pole obdélníkový tvar, jedná se o pole pravidelné. V jazyce Visual Basic 2005 můžete programovat i nepravidelná pole (někdy se jim také říká pole polí).

4.5.1 Deklarace polí

Pole lze deklarovat na mnoha místech: v lokálních procedurách, v modulech anebo ve třídách. Deklarace polí je svěřena do rukou příkazu `Dim`, podobně jak tomu bylo ve Visual Basicu 6.0. Za názvem pole přicházejí kulaté závorky s určením jeho horní meze. V této souvislosti je nutno upozornit, že dolní mez všech polí, které v prostředí jazyka Visual Basic 2005 vytvoříte, je vždy rovna nule, a tudíž ji nelze explicitně změnit. Stanovení nulté dolní hranice polí je počinem, jenž na platformě Microsoft .NET Framework 2.0 napomáhá lepší spolupráci aplikací napsaných v různých .NET-kompatibilních programovacích jazycích. S poli, jejichž dolní hranice je rovna 0, se proto setkáte také při vývoji aplikací v jazycích C# a C++/CLI.

```
'VB 2005: Deklarace jednorozměrného pole typu Byte.
Dim Pole(10) As Byte
```

Ukázkové statické pole má 11 prvků, přičemž jejich indexy začínají nulou a končí desítkou. Ačkoliv nultou dolní hranici pole nelze ve Visual Basicu 2005 měnit, můžeme její určení zahrnout (společně s horní hranicí) i do deklaračního příkazu pomocí klíčového slova `To`:

```
'VB 2005: Explicitní určení mezí pole v deklaračním příkazu.
Dim Pole(0 To 10) As Byte
```

I když vám Visual Basic 6.0 dovoloval podle potřeby měnit dolní hranici, níže uvedený kód již není ve verzi 2005 podporován.

```
'VB 6.0: Přímá změna dolní hranice pole. Pozor, VB 2005 tuto operaci
'nepovoluje realizovat.
Dim Pole(2 To 10) As Integer
```

UPOZORNĚNÍ: Příkaz **Option Base** a změna implicitní dolní hranice polí v jazyce Visual Basic 6.0

Příkaz `Option Base` jazyka Visual Basic 6.0 dovoloval programátorům pozměnit implicitní dolní hranici polí z nuly na jedničku. Vše, co bylo třeba udělat, bylo zapsat na první řádek modulu příkaz `Option Base 1`. Od této chvíle byla všem deklarovaným polím v tomto modulu přisouzena dolní hranice rovna 1. Jednotková dolní hranice s sebou nesla významné změny, které ovlivnily způsob manipulace s polem. Kupříkladu, při aktivním příkazu `Option Base 1` byla horní hranice pole rovna celkovému počtu prvků uložených v tomto poli.

```
'Příkaz Option Base 1 je uveden na začátku modulu.
Option Base 1
'Dolní hranice deklarovaného pole je rovna 1.
'Pole obsahuje 10 prvků a horní hranice pole je rovněž 10.
Dim Pole(10) As Double
```

Jazyk Visual Basic 2005 příkaz `Option Base` nedovoluje použít. Zde jsou všechna vytvářená pole implicitně opatřována nulovou dolní hranicí.

Je libo deklarovat dvourozměrné nebo dokonce trojrozměrné pole? Žádný problém, deklarační příkaz je velice podobný, pouze s tím rozdílem, že oproti vektorovému poli přidáváme jednu, respektive dvě dimenze navíc:

```
'VB 2005: Deklarace 2D-pole o 25 prvcích a 3D-pole o 27 prvcích.
Dim Pole2D(4, 4) As Double
Dim Pole3D(2, 2, 2) As Single
```

Kdybychom chtěli do uvedených deklaračních příkazů implementovat přímé zadání dolních hranic odpovídajících dimenzí, postupovali bychom takhle:

```
'VB 2005: Explicitní zadání dolní hranice při deklaraci 2D- a 3D-pole.
Dim Pole2D(0 To 4, 0 To 4) As Double
Dim Pole3D(0 To 2, 0 To 2, 0 To 2) As Single
```

POZNÁMKA: Deklarace pole s nulovou délkou

Některé z vás jistě napadne otázka, jak deklarovat pole o nulové délce. Kdybychom totiž zadali nulu do deklaračního příkazu, vytvořili bychom pole s jedním prvkem, což není to, co bychom chtěli. Když budete chtít vybudovat pole, jehož délka je nulová, použijte následující deklarační příkaz:

```
'VB 2005: Deklarace pole o nulové délce.
Dim Pole(-1) As Integer
```

Pole o délce 0 zhotovíme tak, že do kulatých závorek umístíme hodnotu -1.

Statická pole ve Visual Basicu 2005 nemají na rozdíl od svých protějšků v jazyce Visual Basic 6.0 fixní velikost. Poté, co je pole deklarováno, lze změnit jeho délku (neboli celkový počet prvků) pomocí příkazu `ReDim`.

```
'VB 2005: Změna délky jednorozměrného statického pole je realitou.
Dim Pole(4) As Integer
ReDim Pole(10)
MessageBox.Show(Pole.GetUpperBound(0).ToString)
```

V dialogu se objeví informace o horní hranici pole, která je po redeklaraci rovna 10. Pro zjištění horní hranice nepoužíváme funkci `UBound` známou z Visual Basicu 6.0, ale voláme metodu `GetUpperBound` pole, která nám vrátí délku pole. Metoda přijímá jeden argument označující dimenzi pole, jejíž délku

bychom rádi zjistili. Protože základem čítače dimenzí je 0, předáváme ji metodě `GetUpperBound`. Návratovou hodnotu metody převedeme do podoby textového řetězce pomocí metody `ToString`. Na této ukázce velice dobře vidíte, že pole se skutečně chová jako objekt: můžeme volat jeho metody a vykonávat tak užitečné činnosti. Objektový charakter polí se vám jistě zalíbí, neboť práce s poli se tak stává nejenom pohodlnější, ale také intuitivnější.

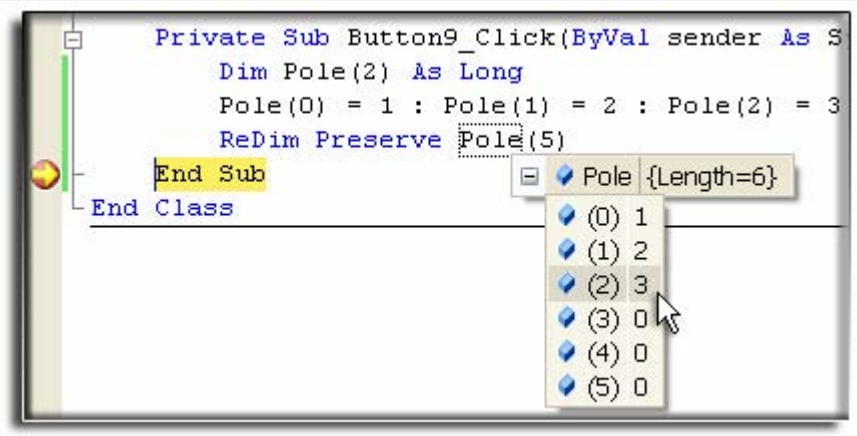
Je-li příkaz `ReDim` aplikován bez klíčového slova `Preserve`, vytvoří zcela nové prázdné pole a objektovou referenci na toto pole přiřadí do odkazové proměnné. Pokud budete chtít zachovat stávající obsah pole, můžete, podobně jako v jazyce Visual Basic 6.0, upotřebit klausuli `Preserve`.

```
'VB 2005: Příkaz ReDim Preserve a modifikace velikosti pole.
Dim Pole(2) As Long
Pole(0) = 1 : Pole(1) = 2 : Pole(2) = 3
ReDim Preserve Pole(5)
```

Poslední řádek kódu vytváří nové pole o šesti prvcích. Jakmile je pole vytvořeno, jsou do jeho elementů s indexy 0 až 2 překopírovány hodnoty z původního pole. Tím pádem je obsah pole zachován, což je hlavní přínos klíčového slova `Preserve`.

TIP: Vizualizátor dat a prohlížení obsahu pole

Při programování s poli vyvstane docela často potřeba rychle přezkoumat jeho obsah. Ačkoliv můžete napsat programovou smyčku, která si s tímto úkolem poradí, integrované vývojové prostředí má pro vás lepší řešení, kterým je vizualizátor dat. Když si budete chtít prohlédnout náplň pole, umístěte za předmětným kódem programovou zarážku (klávesa **F9**). Až se aplikace dostane do režimu přerušení, najeďte kurzorem myši na název pole. Přivítá vás vizualizátor dat, jehož pomocí můžete velice rychle analyzovat složení pole (obr. 4.16).



Obr. 4.16: Pomocí vizualizátoru dat můžeme snadno prohlížet obsah pole

Příkaz `ReDim` měl ve Visual Basicu 6.0 dva „mody" použití: první a jistě známější sloužil pro opětovné nastavení velikosti pole, zatímco druhý dokázal zastoupit deklarační příkaz `Dim` a uskutečnit deklaraci nového pole. Programátoři tudíž mohli standardní deklarační příkaz nahradit příkazem `ReDim`, třeba takto:

```
'VB 6.0: Deklaraci pole lze uskutečnit také příkazem ReDim.
ReDim Pole(4) As String
```

Visual Basic 2005 nepodporuje povýšení příkazu `ReDim` na deklarační příkaz. V tomto jazyce je zapotřebí postupovat jinak – nejprve se příkazem `Dim` deklaruje pole a až poté může být zavolán příkaz `ReDim`, který provede změnu velikosti tohoto pole.

Je-li příkaz `ReDim` zapsán v kódu jazyka Visual Basic 6.0, může být použit pro změnu hodnosti (počtu dimenzí) pole. Kupříkladu, dynamické jednorozměrné pole typu `Double` bylo v jazyce Visual Basic 6.0 možné nahradit 2D-polem stejného datového typu s velikostí 16 prvků:

```
'VB 6.0: Změna hodnosti pole pomocí příkazu ReDim.
Dim Pole() As Double
ReDim Pole(3, 3)
Dim i As Byte, j As Byte
For i = 0 To 3
    For j = 0 To 3
        Pole(i, j) = j + 2
    Next j
Next i
```

Ve Visual Basicu 2005 není modifikace počtu dimenzí pole prostřednictvím příkazu `ReDim` podporována.

4.5.2 Inicializace polí

Ve Visual Basicu 6.0 jste mohli pole inicializovat v zásadě dvěma způsoby: buď jste pole inicializovali manuálně prvek po prvku, anebo jste zodpovědnost za inicializaci přenesli na některý z programových cyklů. Visual Basic 2005 přichází s novými postupy pro inicializaci pole. Ano, i zde můžete využít postupnou nebo cyklickou inicializaci, ovšem k dispozici jsou vám i další alternativy, které si představíme pod drobnohledem.

4.5.2.1 Inicializace pole pomocí inicializačního seznamu

Jestliže hodláte vytvořit pole o několika málo prvcích, můžete ho explicitně inicializovat prostřednictvím inicializačního seznamu.

```
'Inicializace pole pomocí inicializačího seznamu.
Dim Pole() As Byte = {10, 20, 30, 40}
```

Inicializační seznam je tvořen inicializačními hodnotami, které jsou od sebe odděleny čárkami. Tyto hodnoty jsou uzavřeny ve složených závorkách ({}) a k deklaračnímu příkazu jsou připojeny pomocí přiřazovacího operátoru. Mějte prosím na paměti, že při použití inicializačního seznamu je nutno pole deklarovat jako dynamické, což znamená, že kulaté závorky za jménem pole musí být prázdné. Horní hranici pole není tedy zapotřebí uvádět. Je to proto, že Visual Basic 2005 si sám dokáže spočíst všechny inicializační hodnoty uvedené v inicializačním seznamu a podle vypočteného součtu je schopen determinovat správnou velikost pole. V našem případě má pole čtyři prvky a jeho horní hranice je proto rovna hodnotě 3.

4.5.2.2 Inicializace pole pomocí operátoru New a inicializačního seznamu

Vsuneme-li do předcházejícího inicializačního modelu operátor `New`, obdržíme nový postup, jehož dovednosti můžeme při inicializaci polí využít. Nejprve se podívejme na syntaktický obraz této inicializační metody, a poté si k ní řekneme pár slov.

```
'Inicializace pole pomocí operátoru New a inicializačního seznamu.
Dim Pole() As Byte = New Byte() {2, 4, 8, 10, 12}
```

První část deklaračního příkazu, která se nachází nalevo od operátoru pro přiřazení, je zcela shodná s předcházející ukázkou, v níž jsme hodnoty pole nastavovali pomocí inicializačního seznamu. Zásadní

změnou je použití operátoru `New`, který je následován názvem datového typu. Za specifikací datového typu se nacházejí kulaté závorky, mezera a inicializační seznam hodnot ve složených závorkách. I při použití tohoto způsobu explicitní inicializace je nutné deklarovat pole jako dynamické.

4.5.3 Nejčastější operace prováděné s poli

Deklarace a inicializace pole jsou činnosti, které stojí na začátku jeho životního cyklu. Během doby, kdy je pole „živé", s ním lze provádět různé činnosti. My si ukážeme, jak:

- přiřadit obsah jednoho pole do jiného pole,
- setřídit elementy pole podle abecedy,
- převrátit obsah pole.

4.5.3.1 Přiřazení obsahu jednoho pole do jiného pole

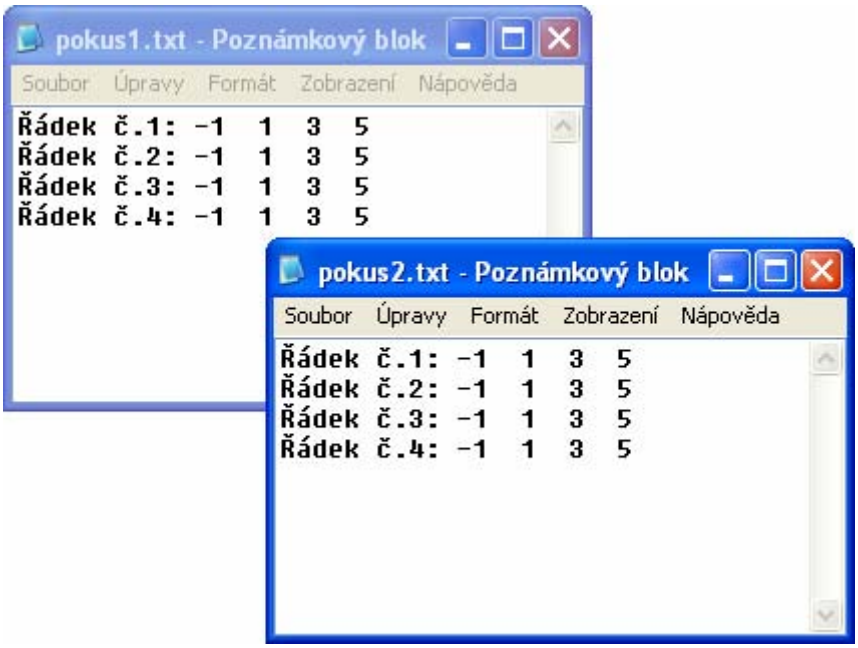
Pokud mají zdrojové a cílové pole shodný počet dimenzí a jejich prvky jsou stejných datových typů, můžeme obsah zdrojového pole překopírovat do pole cílového pomocí jednoduché přiřazovacího příkazu. Níže uvedený fragment kódu ukazuje charakterizovanou techniku v praxi.

```
Dim f1 As Integer = FreeFile()
Dim pole1(3, 3) As Integer
FileOpen(f1, "d:\pokus1.txt", OpenMode.Output)
For i As Byte = 0 To CByte(pole1.GetUpperBound(0))
    Print(f1, "Řádek č." & i + 1 & ": ")
    For j As Byte = 0 To CByte(pole1.GetUpperBound(1))
        pole1(i, j) = 2 * j - 1
        Print(f1, pole1(i, j))
    Next j
Print(f1, vbCrLf)
Next i
FileClose(f1)

Dim pole2(3, 3) As Integer
'Přiřazovacím příkazem převedeme obsah zdrojového pole do pole cílového.
pole2 = pole1

Dim f2 As Integer = FreeFile()
FileOpen(f2, "d:\pokus2.txt", OpenMode.Output)
For a As Byte = 0 To CByte(pole2.GetUpperBound(0))
    Print(f2, "Řádek č." & a + 1 & ": ")
    For b As Byte = 0 To CByte(pole2.GetUpperBound(1))
        Print(f2, pole2(a, b))
    Next b
    Print(f2, vbCrLf)
Next a
FileClose(f2)
```

V kódu vytváříme dvě pole (`pole1` a `pole2`). Zdrojové pole (`pole1`) inicializujeme pomocí vnořeného cyklu `For...Next`, přičemž obsah pole zapisujeme do textového souboru *pokus1.txt* uloženém na pevném disku. Cílové pole (`pole2`) má stejnou hodnost a specifikaci datového typu jako pole zdrojové. Pro umístění obsahu `pole1` do `pole2` použijeme jednoduchý přiřazovací příkaz. Abychom se přesvědčili o zdárném průběhu celé operace, ukládáme obsah `pole2` do textového souboru *pokus2.txt*. Posléze oba soubory porovnáme, čímž dojdeme k poznání, že obě pole jsou shodná (obr. 4.17).



Obr. 4.17: Komparace datového obsahu obou polí

4.5.3.2 Setřídění prvků pole podle abecedy

Rychlého seřazení všech elementů jednorozměrného pole typu `String` můžeme dosáhnout aktivací veřejně přístupné a sdílené metody `Sort` třídy `Array`.

```
Dim p1() As String = _
{"Microsoft", "Borland", "Dell", "Sun", "Corel", "Oracle"}
'Sdílená metoda Sort vykonává abecední klasifikaci prvků pole.
Array.Sort(p1)
'Nyní pole p1 vypadá následovně: „Borland", „Corel", „Dell",
'„Microsoft", „Oracle" a „Sun".
For k As Integer = 0 To UBound(p1)
    ListBox1.Items.Add(p1(k))
Next
```

V našem poli jsou seskupeny názvy populárních softwarových společností. Když sdílené metodě `Sort` třídy `Array` předáme objektovou referenci na pole, metoda se postará o abecední seřazení jednotlivých prvků pole. Obraz již setříděného vektorového pole je v další etapě přenesen do instance `ListBox1`.

4.5.3.3 Převrácení obsahu pole

Pod pojmem převrácení obsahu pole máme na mysli přeskupení prvků pole podle tohoto vzoru: poslední prvek pole se stane prvním prvkem, předposlední prvek pole zaujme místo druhého prvku, a tak dále až dokud není první prvek pole usazen na pozici s nejvyšším indexem. Povězme, že máme pole se třemi daty. Když zavoláme veřejnou sdílenou metodu `Reverse` třídy `Array`, získáme na počkání pole s novým rozvržením.

```
Dim Pole() As Date = {#6/6/1998#, #12/1/2000#, #4/8/2005#}
'Plánujete-li převrátit obsah pole, sdílená metoda Reverse
'se jeví jako dobrý pomocník.
Array.Reverse(Pole)
For m As Integer = 0 To Pole.GetUpperBound(0)
    MessageBox.Show("Datum na " & m & ". pozici pole je " & _
    Pole(m) & ".", "Převrácení obsahu pole", MessageBoxButtons.OK, _
    MessageBoxIcon.Information)
Next
```

4.6 Procedury

Procedury jsou základními pilíři, na kterých stojí vývoj softwaru ve vyšších programovacích jazycích, a tedy i ve Visual Basicu. Paletu procedur tvoří standardní procedury `Sub`, funkční procedury `Function`, procedury vlastností `Property` a zpracovatele událostí. Samozřejmě, všechny zmíněné programové elementy mohou programátoři využívat i v jazyce Visual Basic 2005, byť i tato sféra je poznačena několika zásadními změnami. A jakéže jsou? Změnil se implicitní mechanizmus předávání argumentů procedurám, dále došlo k modifikaci způsobu definování procedur s volitelnými parametry a jiný je také styl volání procedur. Procedury Sub a funkce mohou být obohaceny o nové modifikátory přístupu, které ovlivňují jejich viditelnost z klientského zdrojového kódu. Vylepšení se dočkal také příkaz `Return`, který lze nyní použít pro zaslání návratové hodnoty a ukončení exekuce procedury. Zcela přepracovány byly obslužní procedury Sub, které vystupují v roli zpracovatelů událostí. Signatura zpracovatelů událostí je doplněna o vesměs užitečné parametry, jejichž pomocí lze identifikovat původce události a další důležité informace, které se se vznikem události pojí. Inovací těžkého kalibru je přetěžování procedur, které nám umožňuje napsat více verzí jedné procedury (ty lze pak volat v závislosti od typu dodaných vstupních dat). Popsané změny si představíme podrobněji v následujících podkapitolách.

4.6.1 Změna implicitního mechanizmu předávání argumentů

Ve Visual Basicu verze 6 byly argumenty implicitně předávány cílovým procedurám odkazem (`ByRef`). Ve skutečnosti tak cílová procedura obdržela ukazatel, jehož pomocí mohla přímo přistupovat k dodanému argumentu a v případě potřeby jej také pozměnit. Přesně tuto situaci přibližuje níže zobrazený fragment kódu, v němž se nacházejí tři procedury: zpracovatel události `Click` tlačítka, procedura `ZjistitVelikostSouboru` a procedura s názvem `Procedura2`.

```
'VB 6.0: Ukázka potenciálních potíží, které s sebou nese implicitní
'předávání argumentů odkazem (ByRef).
Private Sub Command1_Click()
ZjistitVelikostSouboru "d:\test.doc"
End Sub

Private Sub ZjistitVelikostSouboru(ByVal JménoSouboru As String)
Dim VelikostSouboru As Long
VelikostSouboru = VBA.FileLen(JménoSouboru)
Procedura2 VelikostSouboru
MsgBox "Soubor " & JménoSouboru & " je veliký " & VelikostSouboru & _
" bajtů.", vbInformation, "Velikost souboru"
End Sub

'Tato procedura přijímá argumenty předávané odkazem. Klíčové slovo ByRef
'není nutno uvádět, nakolik jde o implicitní způsob předávání
'argumentů.
```

```
Private Sub Procedura2(Arg As Long)
Arg = Arg + 10000
End Sub
```

Procedura `ZjistitVelikostSouboru` se soustřeďuje na určení velikosti požadovaného souboru v bajtech. Alokační kapacitu souboru umístěného na pevném disku zjistíme poměrně snadno prostřednictvím funkce `FileLen` z knihovny `VBA`. Poté ovšem aktivujeme proceduru `Procedura2`, která přijímá argument odkazem. Této proceduře je poskytnuta paměťová adresa (v podobě interního ukazatele), na níž je hodnota argumentu uložena. Doteď je všechno v pořádku, ale co se stane, když v těle volané procedury dojde k nechtěné inkrementaci argumentu? Nuže, dojde ke změně původní hodnoty reprezentující velikost souboru, což je velice nepříjemné, neboť v tu ránu bude procedura `ZjistitVelikostSouboru` zobrazovat špatný výsledek, jenž neodpovídá realitě.

Ačkoliv je samozřejmě možné se podobným programátorským neduhům vyvarovat pečlivým a promyšleným psaním kódu, mnoho začínajících a středně pokročilých vývojářů padá do pasti způsobené mylným pochopením implicitního mechanizmu předávání argumentů. Jedním z fenoménů jazyka Visual Basic 2005 je dosažení větší bezpečnosti v procesu psaní programového kódu. Proto dochází ke změně ve způsobů předávání argumentů: ty jsou nyní standardně odevzdávány hodnotou (`ByVal`) a ne odkazem (`ByRef`). Cílové procedury tudíž obdrží vždy kopie původních argumentů, které mohou modifikovat podle své vůle, a to aniž by došlo k neželaným výsledkům. Kód Visual Basicu 2005, jenž mapuje předcházející ukázku napsanou v minulé verzi tohoto jazyka, může vypadat následovně:

```
Private Sub Button1_Click(ByVal sender As System.Object, _
ByVal e As System.EventArgs) Handles Button1.Click
        ZjistitVelikostSouboru("d:\test.doc")
End Sub
Private Sub ZjistitVelikostSouboru(ByVal JménoSouboru As String)
    Dim VelikostSouboru As Long
    Dim InfoOSouboru As New IO.FileInfo(JménoSouboru)
    VelikostSouboru = InfoOSouboru.Length()
    MessageBox.Show("Soubor " & JménoSouboru & " je veliký " & _
    VelikostSouboru & " bajtů.", "Velikost souboru", _
    MessageBoxButtons.OK, MessageBoxIcon.Information)
End Sub

Private Sub Procedura2(ByVal arg As Long)
    arg += 10000
End Sub
```

Zcela nejzásadnější změnou je nový styl předávání argumentů – ty jsou nyní předávány hodnotou (`ByVal`), což eliminuje riziko jejich neplánované modifikace. Všimněte si, že v signatuře procedury `Procedura2` je explicitně uvedeno klíčové slovo `ByVal`. Ačkoliv je v prostředí Visual Basicu 2005 předávání argumentů hodnotou implicitním mechanizmem, technologie IntelliSense doplňuje do kódu klíčové slovo `ByVal` i v případě, že jej nezadáte. Vizuální přítomnost formulky `ByVal` již na první pohled všem programátorům říká, jakým systémem jsou zpracovávány dodávané argumenty. V těle procedury `Procedura2` je hodnota argumentu zvětšena o 10000 jednotek. Tato operace ovšem nijak neovlivní správnou funkčnost kód, a proto bude v dialogu zobrazena skutečná velikost předmětného souboru.

Chtě nechtě musíme prozkoumat také další změny, jež kód uvádí. Syntaktická konstrukce zpracovatele události `Click` tlačítka je zcela jiná – je doplněna o dva parametry (`sender` typu `Object` a `e` typu `EventArgs`), zprostředkující dodatečné informace o vzniku události `Click`. Při

zpracovatelích událostí se přistavíme o něco později v této kapitole, takže v tuto chvíli je odložíme stranou.

Další do očí bijící novinku představuje postup, jehož pomocí vypočítáváme velikost souboru uloženého na pevném disku. Jak můžete vidět, již nevoláme funkci `FileLen` knihovny `VBA`. Proč bych to taky dělali, když nám Visual Basic 2005 na stříbrném podnosu nabízí daleko efektivnější cestu k cíli? Tou je třída `FileInfo` z jmenného prostoru `System.IO`. Tato třída definuje instanční metody pro vytváření, přemísťování, kopírování, otevírání a další operace, které lze se soubory provádět. Třída `FileInfo` obsahuje rovněž sadu užitečných vlastností, které nám mohou poskytnout relevantní informace o souborech. Naše ukázka sestrojuje novou instanci třídy `FileInfo` a poté volá vlastnost `Length`, která navrací velikost souboru v bajtech.

4.6.2 Procedury s volitelnými parametry

V signaturách procedur jazyka Visual Basic 6.0 se mohly vyskytovat volitelné parametry identifikované klíčovým slovem `Optional`. Volitelné parametry mohly, ale nemusely mít přiřazenu svoji implicitní inicializační hodnotu. Jestliže byl jeden parametr v signatuře procedury deklarován jako volitelný, musely být volitelné také všechny následující parametry. Zajímavá situace nastala, pokud byl přítomen volitelný parametr typu `Variant`, jemuž nebyla přiřazena žádná implicitní inicializační hodnota. V tomto případě mohl kód procedury zavolat funkci `IsMissing` a zjistit tak, zda kýžený volitelný parametr obsahuje nějakou inicializační hodnotu či nikoliv.

```
Private Sub Command1_Click()
Dim MůjIndexBMI As Single
MůjIndexBMI = VypočístIndexBMI()
MsgBox MůjIndexBMI
End Sub

Private Function VypočístIndexBMI(Optional ByVal Výška As Single = 1.75, _
Optional ByVal Váha As Variant) As Single
Dim IndexBMI As Single
If IsMissing(Váha) Then Váha = 70
IndexBMI = Váha / (Výška * Výška)
VypočístIndexBMI = IndexBMI
End Function
```

Jestliže byste rádi zjistili svůj index BMI (Body Mass Index), můžete aktivovat funkci `VypočístIndexBMI` a předat ji svou výšku a váhu v příslušných měřítcích.

POZNÁMKA: Index BMI a určení ideální váhy

Body Mass Index neboli zkráceně index BMI je v současné době nejpopulárnějším ukazovatelem pro posouzení ideální hmotnosti. Index se stal oblíbeným především pro snadnost svého výpočtu a relativní přesnost. Index BMI lze vypočítat podle tohoto matematického vzorce:

$$IndexBMI = \frac{Váha[kg]}{Výška^2[m]}$$

Nachází-li se váš index BMI v mezích <19, 25>, vaše váha je v pořádku. Hodnota indexu větší než 25 je již znakem nadváhy, při vyšších stupních dokonce obezity. Kupříkladu index BMI člověka s výškou 1,75 m a váhou 70 kg má hodnotu 22,86.

Funkce `VypočístIndexBMI` pracuje se dvěma parametry, přičemž oba jsou volitelné. Prvnímu parametru (`Výška` typu `Single`) přiřazujeme implicitní inicializační hodnotu (1.75), která bude použita v případě, že volání funkce se uskuteční bez dodání prvního argumentu. Druhý parametr (`Váha` typu `Variant`) svou inicializační hodnotou nedisponuje. Abychom předešli situaci, kdy programátor nezadá hodnotu pro druhý parametr v okamžiku volání funkce, aktivujeme funkci `IsMissing`, které dáváme za úkol zjistit, zda je druhý parametr naplněn smysluplnou hodnotou.

Signaturu funkce `VypočístIndexBMI` bychom v jazyce Visual Basic 2005 nemohli použít. Důvodem je skutečnost, že v tomto prostředí musí všechny volitelné parametry vlastnit své implicitní inicializační hodnoty. Ty se pak upotřebí tehdy, bude-li funkce volána bez přesného vymezení vstupních argumentů.

```
Private Sub Button1_Click(ByVal sender As System.Object, _
ByVal e As System.EventArgs) Handles Button1.Click
        MessageBox.Show(CStr(VypočístIndexBMI()))
End Sub

'VB 2005: Všechny volitelné parametry musí mít své implicitní
'inicializační hodnoty.
Private Function VypočístIndexBMI(Optional ByVal Výška As Single = 1.75, _
Optional ByVal Váha As Byte = 70) As Single
        Dim IndexBMI As Single = Váha / (Výška * Výška)
        VypočístIndexBMI = IndexBMI
End Function
```

4.6.3 Volání procedur

Znám mnoho začínajících a dokonce také pokročilejších programátorů v jazyce Visual Basic 6.0, kteří měli jisté potíže s určením správné volací sekvence procedur Sub a funkcí. Pravdou je, že Visual Basic verze 6 byl v tomto směru poněkud těžkopádný, což vyplývalo ze snad až příliš překomplikovaných syntaktických pravidel. Jestliže jste chtěli zavolat parametrickou funkci, mohli jste vstupní seznam argumentů nechat samostatně stojící, anebo jej uzavřít do kulatých závorek.

```
Private Sub Command1_Click()
Dim x As String, y As Long, z As Boolean
x = "Textový řetězec"
y = 1000
'Funkční argumenty stojí samostatně.
f x, y
x = "Další textový řetězec"
y = 2000
'Funkční argumenty jsou uzavřeny v kulatých závorkách.
z = f(x, y)
End Sub

Private Function f(ByVal ag1 As String, ByRef ag2 As Long) As Boolean
MsgBox "Zadané argumenty: " & vbCrLf & "ag1: " & ag1 & vbCrLf & _
"ag2: " & ag2, vbInformation, "Volání parametrické funkce"
End Function
```

Nemáme-li zájem zjišťovat návratovou hodnotu funkce, stačí, když funkční argumenty oddělíme čárkami, není nutné je ještě vkládat do kulatých závorek. Na druhou stranu, chceme-li z nějakého důvodu analyzovat hodnotu, kterou funkce vrátí, musíme seznam argumentů uzavřít do kulatých závorek.

Bezparametrickou funkci lze volat bez zapsání kulatých závorek za jejím jménem:

```
Private Sub Command1_Click()
f
End Sub

Private Function f() As Boolean
MsgBox "Byla aktivována funkce."
End Function
```

A to dokonce i tehdy, když nám jde o získání návratové hodnoty funkce:

```
Private Sub Command1_Click()
Dim b As Boolean
b = f
End Sub

Private Function f() As Boolean
MsgBox "Byla aktivována funkce."
f = True
End Function
```

Přesuňme se od funkcí ke standardním procedurám Sub. Parametrickou proceduru Sub můžeme volat pouze s volně stojícími argumenty:

```
Private Sub Command1_Click()
Dim m As Byte, n As Integer
m = 10: n = 100
p m, n
End Sub

Private Sub p(ByVal a As Byte, ByRef b As Integer)
MsgBox "Hodnota prvního parametru je " & a & "." & vbCrLf & _
"Hodnota druhého parametru je " & b & ".", vbInformation, _
"Volání parametrické procedury Sub"
End Sub
```

Jelikož procedury Sub nemohou nikdy vracet návratovou hodnotu, nemusíme se s ní zabývat. Bezparametrickou proceduru Sub aktivujeme bez uvedení kulatých závorek za jejím pojmenováním.

```
Private Sub Command1_Click()
p
End Sub
Private Sub p()
MsgBox "Byla aktivována procedura Sub."
End Sub
```

UPOZORNĚNÍ: Příkaz **Call** a volání funkcí a procedur Sub

Funkce a procedury Sub mohou být z prostředí jazyka Visual Basic 6.0 volány rovněž pomocí příkazu `Call`. Pokud je příkazem `Call` volána parametrická funkce nebo procedura Sub, musí být všechny argumenty, které mají být cílové funkce či proceduře předány, umístěny v kulatých závorkách. Při aktivaci funkce pomocí příkazu `Call` je její návratová hodnota zlikvidována, a proto ji nelze použít.

```
'Volání parametrické procedury Sub pomocí příkazu Call.
Call p(m, n)
'Příkaz Call můžeme použít také při aktivaci parametrické funkce.
Call f(x, y)
```

Kompilátor jazyka Visual Basic 2005 vyžaduje, aby součástí volání funkce či procedury Sub byly kulaté závorky, dokonce i v těch případech, kdy volaná funkce, respektive procedura Sub nedisponuje žádnými parametry. Nově zavedené pravidlo pro volání procedur odstraňuje dřívější syntaktickou schizofrenii.

```
Private Sub Button1_Click(ByVal sender As System.Object, _
ByVal e As System.EventArgs) Handles Button1.Click
'Příkaz pro volání bezparametrické procedury Sub musí obsahovat
'kulaté závorky za jménem procedury.
        p()
'Stejnými regulemi se řídí také aktivace bezparametrické funkce.
        f()
End Sub

Private Sub p()
'Kód těla procedury Sub byl vynechán.
End Sub

Private Function f() As Char
'Kód těla funkce byl vynechán.
End Function
```

Volitelně můžete před volání procedury Sub nebo funkce vložit příkaz **Call**:

```
Call p()
Call f()
```

4.6.4 Modifikátory přístupu procedur

Při psaní procedur v jazyce Visual Basic 6.0 jste jim mohli přisoudit modifikátory přístupu, které exaktně vymezovaly spektrum viditelnosti té-které procedury. Procedury mohly být definovány jako soukromé (`Private`) nebo veřejné (`Public`). K privátním procedurám mohly přistupovat pouze jiné procedury, jež byly umístěny ve stejném programovém modulu. O poznání větší volností disponovaly veřejné procedury, které bylo možné volat ze všech modulů tvořících aplikační projekt. Třetí modulátor přístupu měl název `Friend`, ovšem ve srovnání s modifikátory `Private` a `Public` se těšil daleko menší popularitě. (Někteří programátoři ve Visual Basicu 6.0 dokonce ani netuší, že něco takového jako modifikátor `Friend` vůbec existuje.) `Friend` je však poněkud specifický: lze ho použít pouze ve spojení s těmi procedurami, jejichž programový kód je uložen v modulu třídy nebo v modulu formuláře. Procedury opatřené modifikátorem `Friend` bylo možné s výhodou využít zejména při vytváření komponentů a dynamicky linkovaných knihoven standardu ActiveX. `Friend` procedury měly velice dobré předpoklady pro realizaci interní komunikace mezi objekty, protože Visual Basic 6.0 nezačleňoval odkazy na ně do rozhraní tříd, z nichž objekty vznikaly. (Tímto se „přátelské" procedury odlišovaly do svých veřejných protějšků.)

Stane-li se vaším dalším domovem jazyk Visual Basic 2005, můžete vytvářet následující typy procedur:

1. **Soukromé (`Private`) procedury.** Privátní procedury jsou dostupné pouze v tom kontextu, v němž byly deklarovány. Tímto kontextem může být třída, standardní modul nebo struktura, tedy uživatelsky definovaný hodnotový datový typ.

   ```
   'Hlavička soukromé parametrické procedury Sub.
    Private Sub SoukromáProcedura(ByRef a As Integer)
   ```

2. **Veřejné (`Public`) procedury.** Jestliže se v definici proceduru objeví klíčové slovo `Public`, jedná se o veřejně přístupnou programovou entitu. Ta je viditelná nejenom z celého aplikačního projektu, ale také z dalších projektů, které se na zdrojový aplikační projekt odkazují.

```
'Hlavička veřejné bezparametrické funkce.
 Public Function VeřejnáFunkce() As Date
```

3. **Přátelské (`Friend`) procedury.** Použití modifikátoru `Friend` je v jazyce Visual Basic 2005 chápáno jinak, než tomu bylo ve Visual Basicu 6.0. Oborem platnosti zdejších přátelských procedur je pouze to sestavení, ve kterém jsou tyto procedury definovány. Řečeno jinými slovy, k `Friend` procedurám nesmějí přistupovat žádná externí sestavení.

```
'Hlavička přátelské procedury s jedním volitelným parametrem.
Friend Sub PřátelskáProcedura(Optional ByVal b As Char = "0"c)
```

4. **Chráněné (`Protected`) procedury.** Použití chráněných procedur spadá do oblasti objektově orientovaného programování (OOP) v jazyce Visual Basic 2005. Procedury, v jejichž hlavičce svítí modifikátor přístupu `Protected`, mohou být volány buď ze třídy, v níž jsou definovány, nebo z jakékoliv jiné odvozené třídy. Jestliže tedy základní (bázová) třída obsahuje kód chráněné procedury, mohou k tomuto kódu přistupovat také procedury, které se nacházejí v odvozených třídách (ty vznikly v procesu dědění ze třídy bázové). Ano, Visual Basic 2005 je plně objektově orientovaným programovacím jazykem, což kromě jiného znamená, že umožňuje programátorům vytvářet nové třídy odvozováním od tříd již definovaných. Více si o implementaci principů OOP ve Visual Basicu 2005 povíme v 5. kapitole této příručky.

```
'Hlavička chráněné funkce s jedním parametrem.
Protected Function ChráněnáFunkce(ByRef s As String) As Boolean
```

5. **Chráněné přátelské (`Protected Friend`) procedury.** Zkombinujeme-li dovednosti modifikátorů `Protected` a `Friend`, získáme novou zbraň v podobě modifikátoru `Protected Friend`. Chráněná přátelská procedura je přístupná:

   - ze sestavení, ve kterém je definována,
   - z bázové třídy, v níž je uložen její zdrojový kód,
   - z odvozených tříd, které vznikly v procesu dědění z mateřské třídy,
   - ze všech uvedených oblastí.

```
'Hlavička chráněná přátelské jednoparametrické procedury.
Protected Friend Sub ChráněnáPřátelskáProcedura(ByVal x As Single)
```

4.6.5 Příkaz Return

V některých programovacích jazycích, jako je třeba C++, mohli programátoři vracet návratové hodnoty funkcí pomocí příkazu `return`. Visual Basic verze 6 sice příkaz `Return` obsahoval, ovšem tento nebyl určen pro zasílání návratových hodnot funkcí. Místo toho byl příkaz schopen přesunout exekuci z volané podprocedury zpátky do volající procedury. Abychom si použití příkazu `Return` ve Visual Basicu 6.0 přiblížili, povězme, že bychom chtěli sestavit kód, který bude určovat počet znaků, z nichž se skládá textový řetězec zadaný uživatelem. Stanovený úkol bychom mohli vyřešit následovně:

```
Private Sub Command1_Click()
Dim Řetězec As String, PočetZnaků As Integer
Řetězec = InputBox("Zadejte textový řetězec:", "Zadání řetězce")
If Řetězec <> "" Then GoSub ZjistitPočetZnaků
Exit Sub
ZjistitPočetZnaků:
PočetZnaků = Len(Řetězec)
MsgBox "Zapsaný řetězec tvoří " & PočetZnaků & _
" znaků.", vbInformation, "Počet znaků v řetězci"
Return
End Sub
```

V těle procedury `Command1_Click` se nachází podprocedura `ZjistitPočetZnaků`, která se rozprostírá od stejnojmenného návěstí až po příkaz `End Sub`. Tuto podproceduru budeme volat pomocí příkazu `GoSub`, jemuž předáme název cílového návěstí. Textový řetězec získáme od uživatele za asistence funkce `InputBox` – jde o rychlý a léty prověřený způsob zadávání a analýzy informací. Po krátkém testu, zda nebyl zadán prázdný řetězec, přesouváme zaměření programu na podproceduru `ZjistitPočetZnaků`. Ta spočítá znaky v řetězci, zobrazí informační zprávu a aktivuje příkaz `Return`, jehož přičiněním je exekuce transportována na řádek, jenž stojí za řádkem s příkazem `GoSub`. Kód podprocedury musíme od zbytku kódu událostní procedury oddělit příkazem `Exit Sub`, který zabraňuje nechtěnému zpracování podprocedury `ZjistitPočetZnaků`.

Příkaz `Return`, jenž je nedílnou součástí syntaktické skladby jazyka Visual Basic 2005, je zcela jiný a po pravdě řečeno, se svým předchůdcem nemá kromě názvu nic společného. Výše uvedený kód bychom ve verzi 2005 nedovedli přepsat, protože v tomto prostředí není příkaz `GoSub` podporován. Nový příkaz `Return` lze použít ve spojitosti s procedurami Sub a rovněž s funkcemi. Je-li příkaz `Return` zapsán do těla procedury, jeho účinek je stejný jako zpracování příkazu `Exit Sub`. To znamená, že příkaz `Return` ukončuje běh procedury, přičemž další exekuční cesta je vedena do volající procedury nebo funkce. Síla příkazu `Return` se však plně rozvine teprve v interakci s funkcemi: pomocí tohoto příkazu lze vracet návratové hodnoty funkcí volajícím procedurám! Jde o veliký pokrok, jemuž přijdete na chuť ihned, jak si jeho použití odzkoušíte v praxi. Již tedy není nutné odesílat návratové hodnoty funkcí prostřednictvím neobratného přiřazovacího příkazu s názvem funkce – nyní stačí zapsat příkaz `Return` a specifikovat hodnotu pro navrácení.

Další ukázka kódu demonstruje, jak použít příkaz `Return` z těla funkce pro navrácení její návratové hodnoty. Funkce `VypočístFaktoriál` dovede spočítat faktoriál čísla, které je jí předáno v podobě argumentu. Návratovou hodnotou funkce je hodnota faktoriálu, která je pro lepší použitelnost vrácena v podobě textového řetězce. Ten je v dalším kroku zobrazen v informačním dialogu.

```
Private Sub Button1_Click(ByVal sender As System.Object, _
ByVal e As System.EventArgs) Handles Button1.Click
    MessageBox.Show(VypočístFaktoriál(6))
End Sub

Private Function VypočístFaktoriál(ByVal n As Byte) As String
    Dim PoleFaktoriálů(n) As Integer
    PoleFaktoriálů(0) = 1
    PoleFaktoriálů(1) = 1
    For x As Byte = 2 To n
        PoleFaktoriálů(x) = x * PoleFaktoriálů(x - 1)
    Next
    'Pomocí příkazu Return vracíme návratovou hodnotu funkce.
    Return "Faktoriál č. " & n & " je " & PoleFaktoriálů(n) & "."
End Function
```

Pro úplnost výkladu si ještě musíme předvést aplikaci příkazu `Return` v těle procedury Sub. Jak již bylo vzpomenuto, v tomto případě příkaz `Return` ukončuje provádění kódu procedury a zaměření předává zpět volající proceduře.

```
Private Sub Button1_Click(ByVal sender As System.Object, _
ByVal e As System.EventArgs) Handles Button1.Click
        ZobrazitInfoOAplikaci()
End Sub

Private Sub ZobrazitInfoOAplikaci()
    'Nachází-li se příkaz Return v proceduře Sub, okamžitě
    'ukončuje její běh.
    Return
    MessageBox.Show( _
    "Jméno aplikace .NET: " & My.Application.Info.Title & _
    vbCrLf & "Název sestavení aplikace .NET: " & _
    My.Application.Info.AssemblyName & vbCrLf & _
    "Popis aplikace .NET: " & My.Application.Info.Description, _
    "Informace o aplikaci .NET", MessageBoxButtons.OK, _
    MessageBoxIcon.Information)
End Sub
```

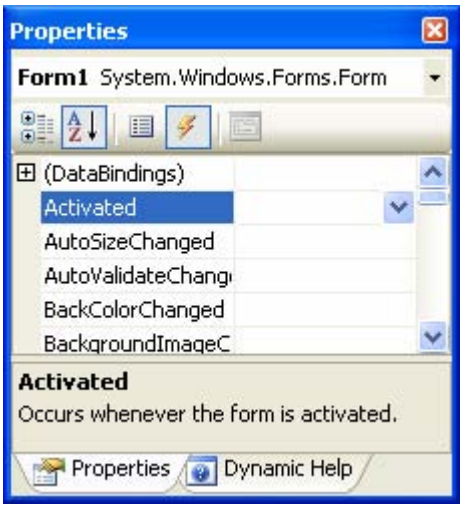
Příkaz `Return` je umístěn před voláním metody `Show` třídy `MessageBox`, a proto nebude informační dialog vůbec zobrazen. Po zpracování příkazu se totiž běh aplikace přesune zpátky do volající procedury.

4.6.6 Zpracovatele událostí

Zpracovatel události je obslužní procedura Sub, která je aktivována pokaždé, dojde-li ke generování jisté události. Jako pokročilí programátoři ve Visual Basicu 6.0 si s programováním založeným na událostech bezesporu dobře rozumíte. Visual Basic 2005 událostní programování samozřejmě podporuje také. Ovšem nejenom to, programování s událostmi je flexibilnější a nabízí více možností, jak události zpracovávat pomocí zdrojového kódu.

4.6.6.1 Vytváření zpracovatelů událostí pomocí vizuálního návrháře a editoru zdrojového kódu

Zpracovatele událostí můžete vytvářet podobně, jak jste to dělali ve Visual Basicu 6.0. Nejdříve na formulář přidáte instanci ovládacího prvku nebo komponenty, posléze zobrazíte editor zdrojového kódu, vyhledáte přidanou instanci, a poté vyberete požadovanou událost. Visual Basic 2005 automaticky sestaví syntaktickou kostru zpracovatele události, čímž získáte dobrou výchozí pozici pro psaní programových instrukcí. Zpracovatele událostí však můžete nyní přidávat i jinak. V okně **Properties** vyberte ze seznamu kýženou instanci ovládacího prvku nebo komponenty. Klepněte na tlačítko **Events** (⚡), na což se v okně **Properties** místo seznamu vlastností instance objeví seznam instančních událostí. Kdybychom popsanou proceduru aplikovali pro události formuláře **Form1**, okno **Properties** by mělo následující podobu (obr. 4.18).



Obr. 4.18: Seznam událostí instance **Form1** v okně **Properties**

Zpracovatele události vytvoříte velice jednoduše – poklepáním na název události. A je to! Visual Basic 2005 sestaví programovou kostru událostní procedury a přenese vás do editoru zdrojového kódu. Níže můžete vidět zkonstruovaného zpracovatele události `MouseMove` formuláře `Form1`.

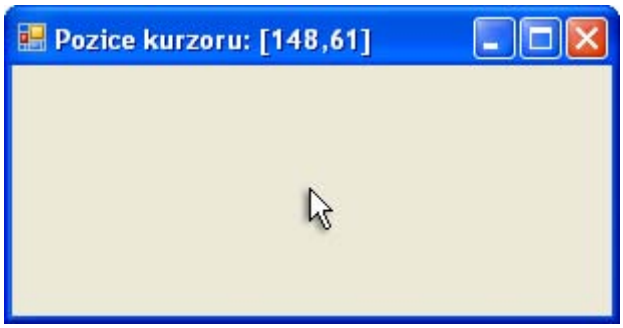
```
'Zpracovatel události MouseMove formuláře Form1.
Private Sub Form1_MouseMove(ByVal sender As System.Object, _
ByVal e As System.Windows.Forms.MouseEventArgs) Handles MyBase.MouseMove

End Sub
```

Zpracovatel události je úplně jiný než jeho protějšek v jazyce Visual Basic 6.0. Je zřejmé, že forma událostních procedur byla znatelně modifikována, takže bude vyžadovat vaši pozornost. Zpracovatel události `MouseMove` pracuje se dvěma parametry. Prvním je `sender` typu `Object`, který reprezentuje odesílatele události. Odesílatel události je ve skutečnosti zdrojem události – jako zdroje událostí působí objekty, u nichž jsou události vyvolávány. V našem případě je zdrojem neboli odesílatelem události objekt třídy `Form1`. Objektová reference mířící na tento objekt je přitom uložena do odkazové proměnné `sender` typu `Object`. Druhý parametr se tváří o poznání záhadněji. Jde o odkazovou proměnnou `e`, která je schopna uchovávat referenci na instanci třídy `MouseEventArgs` z jmenného prostoru `System.Windows.Forms`. Třída `MouseEventArgs` definuje několik prospěšných veřejně přístupných vlastností, jejichž schopnosti můžeme podle potřeby využít. Tak třeba pomocí vlastností `X` a `Y` můžeme určit přesnou pozici kurzoru myši kdykoliv se tento ocitne na ploše formuláře.

```
Private Sub Form1_MouseMove(ByVal sender As System.Object, _
ByVal e As System.Windows.Forms.MouseEventArgs) Handles MyBase.MouseMove
    Me.Text = "Pozice kurzoru: [" & e.X & "," & e.Y & "]"
End Sub
```

Jestliže chcete, můžete funkčnost tohoto kódu ihned prověřit. Spusťte aplikaci a kurzor myši přemístěte na plochu formuláře. Trajektorii kurzoru myši bude odrážet textová informace v titulkovém pruhu formuláře, udávající aktuální pozici kurzoru (obr. 4.19).



Obr. 4.19: Zobrazení aktuální pozice kurzoru myši na ploše formuláře

Za seznamem parametrů zpracovatele události se nachází klíčové slovo `Handles`, které říká, že tato procedura Sub je zpracovatelem události `MouseMove` mateřské třídy (klíčové slovo `MyBase`). Mateřkou třídou třídy `Form1` je třída `Form`. Když tedy dojde ke vzniku události v bázové třídě, tuto lze zpracovat ve třídě odvozené. O vztahu mezi mateřskými a odvozenými třídami se dozvíte více v následující podkapitole.

4.6.6.2 Vytváření zpracovatelů událostí pomocí příkazu AddHandler za běhu programu

Instance ovládacích prvků a komponent, které jsou vystavěné v soupravě nástrojů **Toolbox**, mohou reagovat na mnoho událostí. Není kupříkladu výjimkou, když jeden objekt dovede odpovídat až na několik desítek událostí. Dobrou zprávou je, že portfolio dostupných událostí je v jazyce Visual Basic 2005 mnohem, ale skutečně mnohem širší, než tomu bylo v dřívějších verzích tohoto programovacího nástroje. Za všechny příklady vzpomeňme alespoň jeden.

Povězme, že v aplikaci jazyka Visual Basic 6.0 jste chtěli vybudovat panel nabídek, který by byl naplněn několika nabídkami s příslušnými položkami. Nabídky jste udělali celkem snadno – jednoduše jste použili zabudovaný návrhář nabídek (**Menu Editor**) a bylo, jak se říká, vymalováno. Když jste pak na jistou položku nabídky poklepali, Visual Basic 6.0 sestavil zdrojový kód zpracovatele události `Click` této položky. Dobře, jestliže uživatel klepl na položku, mohli jste povést nějakou akci, kupříkladu zobrazit dialog, zavolat metodu či odeslat data do tiskárny. Bohužel, potíže se začali objevovat v okamžiku, kdy jste se rozhodli implementovat předem promyšlenou pokročilou funkcionalitu. Pokračujme v našich úvahách a předpokládejme, že jste zatoužili po tom, aby se v okamžiku umístění kurzoru uživatelovy myši na položku nabídky zobrazila její stručná charakteristika ve stavovém pruhu. Visual Basic 6.0 ovšem nedovedl zachytit tuto událost, a proto ani neumožňoval vytvořit jejího zpracovatele. Přestože se situace dala vyřešit více způsoby (aktivací nativních funkcí Win32 API, použitím ovládacího prvku nebo komponenty třetí strany), nešlo zrovna o procházku růžovým sadem.

Přejdete-li do prostředí Visual Basicu 2005, tento problém již můžete nadobro pustit z hlavy. Nové nabídky a jejich položky byly obdařeny vyšší inteligencí, čehož přímým důsledkem je, že nyní dovedou reagovat na přehršel rozmanitých událostí. Výše popsanou případovou studii bychom vyřešili velice rychle. Nejprve bychom vytvořili zpracovatele události `MouseHover` konkrétní položky nabídky, a poté bychom jej vyplnili adekvátním programovým kódem.

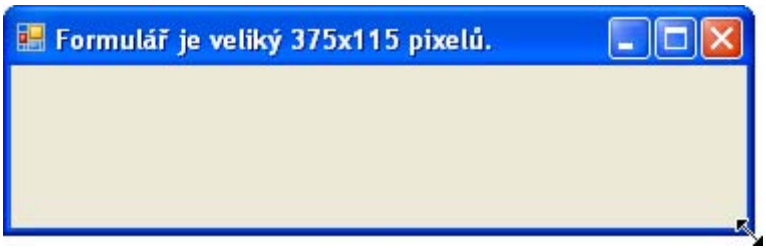
Vizuální návrhář působící v integrovaném vývojovém prostředí jazyka Visual Basic 2005 vám společně s editorem zdrojového kódu nabízí rychlou cestu pro vytvoření libovolného zpracovatele události. Nově však můžeme uskutečňovat propojení mezi událostí a jejím zpracovatelem také za běhu programu pomocí příkazu `AddHandler` a speciálního operátoru `AddressOf`. Na následujících řádcích si ukážeme, jak vytvořit virtuální most mezi událostí `Resize` formuláře `Form1` a uživatelsky definovanou procedurou Sub, která bude hrát roli jejího zpracovatele. Založte proto nový projekt standardní aplikace

pro systém Windows (projektová šablona **Windows Application**) a zobrazte programový kód třídy formuláře (klávesou **F7**). Posléze do třídy přidejte dvě procedury Sub tak, jak to uvádí níže umístěný fragment zdrojového kódu:

```
Public Class Form1
    Private Sub Form1_Load(ByVal sender As Object, _
    ByVal e As System.EventArgs) Handles Me.Load
    'Příkazem AddHandler uskutečňujeme asociaci mezi událostí
    'Resize formuláře a jejím zpracovatelem, jehož ztělesňuje
    'procedura Sub s názvem FormulářZměnilRozměry.
        AddHandler Me.Resize, AddressOf FormulářZměnilRozměry
    End Sub
    Private Sub FormulářZměnilRozměry(ByVal sender As Object, _
    ByVal e As EventArgs)
        Me.Text = "Formulář je veliký " & Me.Width & _
        "x" & Me.Height & " pixelů."
    End Sub
End Class
```

První procedura představuje zpracovatele události `Load` formuláře `Form1`. Událost `Load` je generována poté, co je třída formuláře zavedena do operační paměti počítače a jako taková nám nabízí skvělý prostor pro programové propojení události `Resize` s jejím zpracovatelem. Aby mohl příkaz `AddHandler` spolehlivě odvádět svoji práci, potřebuje znát dvě věci: událost, kterou je nutno sledovat a runtime adresu cílové procedury Sub, která bude danou událost obsluhovat. Zatímco jméno události určíme snadno, s paměťovou adresou cílové procedury je to o něco těžší. Když Just-In-Time kompilátor společného běhového prostředí CLR přeloží MSIL kód procedury Sub, uloží jej v podobě nativního kódu do paměti počítače. JIT kompilátor musí vědět, kde byla procedura uložena – za tímto účelem je sestaven speciální ukazatel, jenž je nasměrován na začátek oblasti, ve které se nachází zkompilovaný kód procedury. Operátor `AddressOf` vytváří systémového delegáta, kterému poskytuje ukazatel cílové procedury, respektive zpracovatele události. Systémový delegát je ve skutečnosti objektem neboli instancí třídy `EventHandler` z jmenného prostoru `System`. Jakmile je systémový delegát sestrojen, je mu předán ukazatel směřující na cílovou proceduru, která se v našem případě jmenuje `FormulářZměnilRozměry`. Signatura této procedury musí odpovídat potřebám systémového delegáta, což přesněji znamená, že musí deklarovat dva parametry, `sender` typu `Object` a `e` typu `EventArgs`. Do těla takto připravené procedury pak zapíšeme programový kód, který bude zpracován vždy, když dojde ke změně horizontálního nebo vertikálního rozměru aplikačního formuláře.

Po spuštění aplikace zkuste změnit rozměry formuláře. Když tak učiníte, v titulkovém pruhu se zobrazí jeho aktuální velikost (obr. 4.20).



Obr. 4.20: Vytvoření asociace mezi událostí a jejím zpracovatelem pomocí příkazu `AddHandler` za běhu aplikace .NET

4.6.6.3 Modifikátor WithEvents

V jazyce Visual Basic 2005 mohou vývojáři obohatit deklarovanou proměnnou na úrovni třídy nebo modulu o modifikátor `WithEvents`. Tím lze z řadové proměnné vytvořit odkazovou proměnnou schopnou uchovávat objektovou referenci ukazující na takovou instanci třídy, která umí generovat události. Ve verzi 2005 je navíc možné v deklaračním příkazu zkombinovat modifikátor `WithEvents` s operátorem `New`, což je z pohledu jazyka Visual Basic 6.0 zakázaná technika. Ukázku použití modifikátoru `WithEvents` předvádí další výpis programového kódu:

```
Public Class Form1
    WithEvents TextovéPole As New TextBox
    Private Sub Form1_Load(ByVal sender As Object, _
    ByVal e As System.EventArgs) Handles Me.Load
        With TextovéPole
            .Width = 120
            .Height = 60
            .Location = New Point(10, 10)
        End With
        Me.Controls.Add(TextovéPole)
    End Sub
    Private Sub TextovéPole_KeyPress(ByVal sender As Object, _
    ByVal e As System.Windows.Forms.KeyPressEventArgs) _
    Handles TextovéPole.KeyPress
        If e.KeyChar = "m"c Then
            Me.WindowState = FormWindowState.Minimized
        End If
    End Sub
End Class
```

Odkazová proměnná s modifikátorem `WithEvents` je deklarována na úrovni třídy formuláře. Typem této proměnné je třída `TextBox` z jmenného prostoru `System.Windows.Forms`. Jelikož v deklaračním příkazu používáme operátor `New`, provádíme ve skutečnosti dvě operace:

1. Vytváříme objektovou proměnnou, která dokáže obsloužit události své cílové instance.
2. Vytváříme novou instanci třídy `TextBox`.

V událostní proceduře `Form1_Load` nastavujeme některé vlastnosti textového pole a přidáváme jej do kolekce instancí formuláře. Protože je odkazová proměnná deklarována pomocí modifikátoru `WithEvents`, můžeme její název vybrat ze seznamu **Class Name**. Poté nám již nic nebrání v tom, abychom otevřeli seznam **Method Name** a specifikovali tu událost, kterou se chystáme ošetřit (v naší ukázce je to událost `KeyPress`). Vydáme-li uvedený pokyn, Visual Basic 2005 připraví programovou kostru zpracovatele události `KeyPress`. Ještě než si popíšeme význam kódu, který se v událostní proceduře `TextovéPole_KeyPress` nachází, přistavme se chvíli při samotné události `KeyPress`. Aby mohla být tato událost generována, musí být splněny dvě základní podmínky. Za prvé, textové pole musí disponovat zaměřením. A za druhé, uživatel musí aktivovat jistou klávesu. Pak dochází ke vzniku události `KeyPress`.

Co se děje uvnitř zpracovatele události `KeyPress`? Programový kód analyzuje textové znaky, které uživatel zapisuje. Jestliže je zadáno malé písmeno „m“, dochází k minimalizaci okna aplikace. Ačkoliv nejde o nijak zvlášť složitou operaci, povšimněte si použití odkazové proměnné `e`. Tato proměnná je pro nás nesmírně cenná, nakolik se s její pomocí můžeme dostat k instanci třídy `KeyPressEventArgs` z jmenného prostoru `System.Windows.Forms`. Instance této třídy definuje vlastnost `KeyChar`, která nám dovoluje pohodlným způsobem zjistit, jaký textový znak uživatel zadal. Máme-li jednou k dispozici hodnotu vlastnosti `KeyChar`, již velice snadno vypátráme, zda se jedná

o malé písmeno „m" či jiný textový znak. Ke změně stavu aplikačního formuláře používáme vlastnost `WindowState` formuláře, do níž ukládáme datovou hodnotu `Minimized` enumerace `FormWindowState`.

4.6.7 Přetěžování procedur

Přetěžování procedur je na první pohled poněkud tajemně vyhlížející slovní spojení, které může u vývojářů přicházejících z Visual Basicu 6.0 evokovat spíše příslušnost k letům nadzvukových stíhaček než k programování. Nemusíte však mít strach – pod pojmem přetížení procedury máme na mysli vytvoření několika variant jedné a tytéž procedury, které disponují stejným jménem, no liší se svými seznamy parametrů. Stěžejním prvkem techniky přetěžování procedur je sestrojení souboru verzí jedné procedury, které dovedou pracovat s rozmanitými typy vstupních hodnot. Přestože všechny definované verze přetížené procedury sdílejí stejné jméno, liší se svou signaturou. Ovšem jaká jsou kritéria pro tuto diferenciaci? Procedury, jež se nacházejí v přetížené sadě, se musí navzájem lišit počtem parametrů, jejich pořadím anebo použitými datovými typy parametrů. Na druhou stranu, procedury nelze odlišit třeba pomocí modifikátorů přístupu (`Private`, `Public`, `Friend` a další) či prostřednictvím modifikátorů, které určují styl práce parametrů (`ByVal`, `ByRef` nebo `Optional`).

Kupříkladu povězme, že se dostaneme do situace, kdy budeme chtít napsat proceduru, která by zobrazovala uživatelsky definovanou zprávu. Dejme tomu, že bychom pro proceduru vybrali jméno `ZobrazitZprávu` (máte-li chuť, můžete přijít i na originálnější název). Standardně bychom mohli chtít, aby procedura deklarovala jeden parametr typu `String`, jenž bude následně předán metodě `Show` třídy `MessageBox`.

```
Private Sub ZobrazitZprávu(ByVal Text As String)
    MessageBox.Show(Text, "Zobrazení zprávy", _
    MessageBoxButtons.OK, MessageBoxIcon.Information)
End Sub
```

Přestože je třída `MessageBox` pro zobrazování zpráv stavěná, neposkytuje nám žádný nekonvenční způsob pro sdělení vytoužených informací. Jste-li stálými uživateli operačního systému Microsoft Windows XP, jistě víte, že zprávy lze zobrazovat také v bublinovém okně oznamovací oblasti hlavního panelu. Co kdybychom tímto vizuálním stylem prezentovali i naši zprávu? Ano, takto bude zpráva určitě impresivnější. Vzhledem k tomu, že naše pozornost se upírá k přetěžování procedur, ukážeme si ji v praxi. Proto napíšeme ještě jednu verzi výše uvedené procedury `ZobrazitZprávu`, která bude vypadat takto:

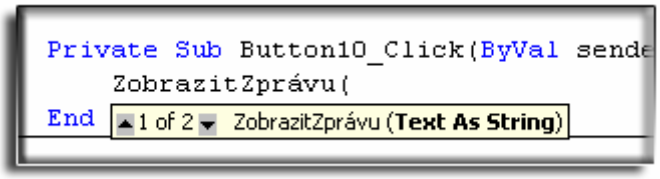
```
Private Sub ZobrazitZprávu(ByVal Text As String, _
  ByVal Titulek As String, ByVal DobaTrvání As Integer)
    Dim OznamovacíIkona As New NotifyIcon
    With OznamovacíIkona
        .BalloonTipIcon = ToolTipIcon.Info
        .BalloonTipTitle = Titulek
        .BalloonTipText = Text
        .Icon = Me.Icon
        .Visible = True
        .ShowBalloonTip(DobaTrvání)
    End With
End Sub
```

Přestože mají obě procedury shodný název, liší se svými signaturami. Zatímco první procedura přijímala pouze jeden textový argument, ta druhá si dovede poradit hned se třemi argumenty! První parametr je

stejný jako minule: přijímá textový řetězec, jenž tvoří jádro naší zprávy. Druhý parametr je připraven pro úschovu textového řetězce, který bude tvořit titulek bublinového okna. Konečně, třetí a poslední parametr reprezentuje dobu trvání v sekundách, po kterou bude bublinové okno zobrazeno v oznamovací oblasti hlavního panelu operačního systému.

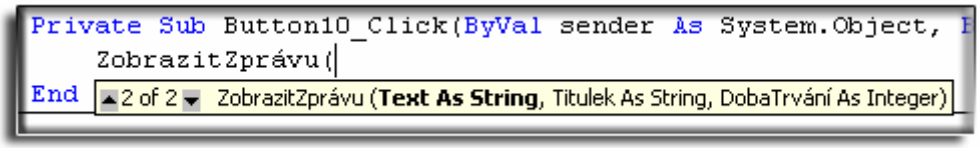
Pro zobrazení bublinového okna používáme instanci třídy `NotifyIcon` z jmenného prostoru `System.Windows.Forms`. Avšak předtím, než můžeme bublinové okno zobrazit, musíme do oznamovací oblasti přidat ikonu a zviditelnit ji. To ovšem nečiní žádné potíže, neboť instance třídy `NotifyIcon` obsahuje vlastnosti `Icon` a `Visible`, které svému účelu důstojně slouží. Z grafické stránky se bublinové okno skládá ze tří nejvýznamnějších součástí: informační ikony (vlastnost `BalloonTipIcon`), textu v záhlaví (vlastnost `BalloonTipTitle`) a hlavního textu (vlastnost `BalloonTipText`). Všechny vlastnosti můžeme nastavit podle svých požadavků. Nakonec bublinové okno zobrazíme pomocí metody `ShowBalloonTip`.

V tuto chvíli jsme vytvořili dvě varianty procedury `ZobrazitZprávu`, takže můžeme říkat, že procedura je přetížená. Když budete chtít přetíženou proceduru zavolat, zapíšete její jméno a vyberete si tu verzi, kterou budete chtít použít. Jakmile zadáte název procedury společně se symbolem otevírací kulaté závorky, technologie IntelliSense zjistí, že tato procedura existuje ve více vyhotoveních (obr. 4.21).



Obr. 4.21: Technologie IntelliSense správně rozpoznala, že pracujeme s přetíženou procedurou

Implicitně vám IntelliSense nabízí první přetíženou variantu, která přebírá jediný argument typu `String`. Jestliže nechcete použít tuto verzi přetížené procedury, můžete klepnout na symbol šipky směřující dolů (▼), čímž technologii IntelliSense vybídnete k tomu, aby vám ukázala signaturu druhé varianty přetížené procedury (obr. 4.22).



Obr. 4.22: Technologie IntelliSense zobrazuje druhou verzi přetížené procedury

TIP: Rychlé zobrazení signatur jednotlivých variant přetížené metody

Pokud byste rádi spatřili podobu další signatury přetížené procedury, nemusíte nevyhnutně používat levé tlačítko myši a klepat na symbol šipky dolů. Stejnou metu totiž dosáhnete i rychleji – pomocí příslušné kurzorové klávesy (↓). Vrátit na předcházející signaturu se můžete buďto klepnutím na symbol šipky nahoru (▲), anebo opět pomocí adekvátní kurzorové klávesy (↑).

Procedura `ZobrazitZprávu`, která bude naše informační oznámení zobrazovat v bublinovém okně je přece jenom lákavější, a proto ji ihned vyzkoušíme. Podle přítomné nápovědy dovedeme vyplnit všechny parametry velice rychle:

```
ZobrazitZprávu("Přetěžování procedur je super!", _
"Programujeme v jazyce Visual Basic 2005", 5)
```

Výsledek práce přetížené procedury `ZobrazitZprávu` můžete vidět na obr. 4.23.

Obr. 4.23: Přetížená procedura zvládá zobrazení bublinového okna v oznamovací oblasti na jedničku

Vždycky když napíšete více než jeden exemplář stejnojmenné procedury, přičemž jednotlivé verze procedur se budou lišit svou signaturou, přetížíte první variantu procedury. Když pak jistou přetíženou verzi procedury zavoláte, Visual Basic 2005 na základě datových typů dodaných argumentů prohledá všechny definované verze přetížené procedury a vybere tu, která bude svou signaturou odpovídat vstupním argumentům. V další etapě jsou argumenty poskytnuty parametrům cílové přetížené procedury a procedura již sama pokračuje ve svém pracovním algoritmu.

Syntaktická výbava jazyka Visual Basic 2005 je rozšířena o nové klíčové slovo `Overloads`, které lze použít ve spojení s přetíženými procedurami. Ačkoliv vsunutí klíčového slova `Overloads` do hlavičky definované přetížené procedury není povinné, je ke prospěchu věci, protože vývojáři nabízí vizuální informaci o tom, že má do činění s přetíženou procedurou. Kdybychom chtěli implementovat klíčové slovo `Overloads` také do přetížené procedury `ZobrazitZprávu`, změnili bychom programový kód obou jejich verzí podle tohoto vzoru:

```
Private Overloads Sub ZobrazitZprávu(ByVal Text As String)
    MessageBox.Show(Text, "Zobrazení zprávy", _
    MessageBoxButtons.OK, MessageBoxIcon.Information)
End Sub

Private Overloads Sub ZobrazitZprávu(ByVal Text As String, _
  ByVal Titulek As String, ByVal DobaTrvání As Integer)
    Dim OznamovacíIkona As New NotifyIcon
    With OznamovacíIkona
        .BalloonTipIcon = ToolTipIcon.Info
        .BalloonTipTitle = Titulek
        .BalloonTipText = Text
        .Icon = Me.Icon
        .Visible = True
        .ShowBalloonTip(DobaTrvání)
    End With
End Sub
```

Jestliže se klíčové slovo `Overloads` vyskytuje v hlavičce jedné varianty přetížené procedury, musí být uvedeno rovněž ve všech ostatních přetížených verzích této procedury. Čili není možné, aby jedna varianta přetížené procedury klíčové slovo `Overloads` používala a jiná ne. Správné pochopení principů přetěžování procedur je pro vývoj nových aplikací .NET více než důležité. Jedním z důvodů, proč je význam přetížených procedur poznačen strmou růstovou křivkou je, že tyto programové entity obsahuje v ohromném počtu i samotná bázová knihovna tříd FCL vývojově-exekuční platformy Microsoft .NET Framework 2.0 (jsou jich stovky a stovky).

Kapitola 5.: Objektově orientované programování v jazyce Visual Basic 2005

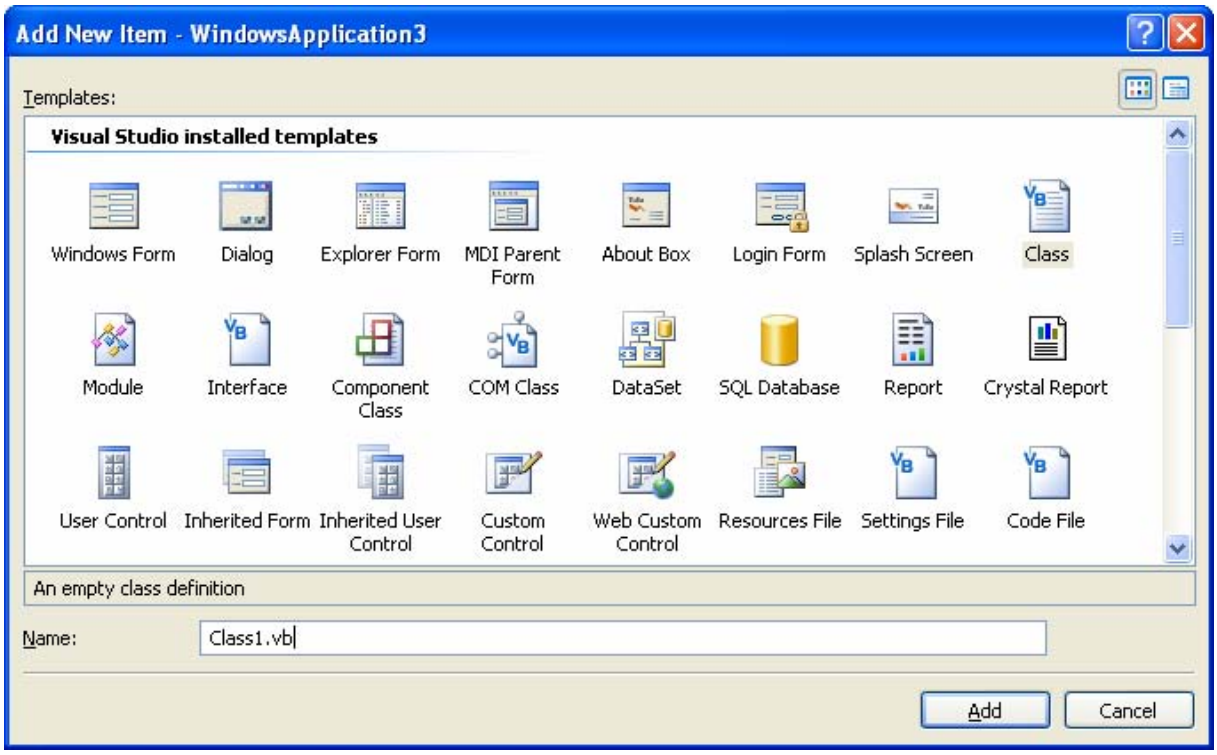
Objektově orientované programování (OOP) představuje zánovní programátorský styl, jenž počátkem devadesátých let minulého století zasedl na programátorský trůn a úspěšně tak nahradil léty používanou koncepci procedurálního programování. OOP je založené na několika základních pilířích, ke kterým patří abstrakce, zapouzdření, skrývání dat, dědičnost, polymorfizmus a opětovná použitelnost programového kódu. Ačkoliv objektově orientovaná příchuť byla cítit již u Visual Basicu verze 4, která byla na trh uvedena v roce 1995, nebyla podpora OOP implementována v celé své šíři. Toto tvrzení dostatečně vystihuje i další dvě verze Visual Basicu s pořadovnými čísly 5 a 6. Jistě, Visual Basic 6.0 dovoloval vývojářům pracovat s objekty, volat jejich metody či aktivovat jejich vlastnosti. S vestavěnými objekty bylo možné pracovat opravdu rychle a intuitivně, čímž se naplňovaly požadavky kladené na systém rychlého vývoje aplikací (RAD). Bohužel, jazyk Visual Basic 6.0 postrádal přímé použití některých prvků objektově orientovaného programování. Snad největší minusový bod představovala absence dědičnosti. Programátoři ve Visual Basicu 6.0 tak nebyli schopni odvozovat od mateřských tříd třídy nové a bleskurychle upotřebovat jednou navrženou a napsanou programovou funkcionalitu. Objektově orientovaná skládanka jazyka Visual Basic 6.0 byla prosta dědičnosti, což mělo další neblahé následky. Ptáte se jaké? Nuže, pokud nebylo možné použít dědičnost v procesu odvozování nových tříd, nedala se pochopitelně aplikovat ani koncepce polymorfizmu realizovaného pomocí dědičnosti. A ačkoliv Visual Basic 6.0 nabízel možnost definovat rozhraní a uplatňovat tak koncepci polymorfizmu realizovaného přes rozhraní, tento přístup byl ve specifických případech ne zrovna vítanou oklikou, kterou museli programátoři absolvovat.

Programovací jazyk Visual Basic 2005 prošel na své stezce k objektové orientované dokonalosti již hodně dlouhý kus. Nejnovější Visual Basic je vpravdě objektově orientovaným vývojovým nástrojem, který nejenomže nabízí hlubokou podporu dědičnosti, ale potěší také implementací dalších objektově orientovaných rysů: nyní lze pracovat s konstrukty a finalizačními metodami, překrývat metody a vlastnosti v odvozených třídách (čili uskutečňovat polymorfizmus pomocí dědičnosti), definovat sdílené (`Shared`) datové členy tříd či volat metody bázových tříd pomocí klíčového slova `MyBase`. Visual Basic 2005 má pro vás připravena další milá překvapení. Jestliže vás v minulé verzi jazyka obtěžovalo, že do jednoho modulu třídy bylo možné umístit zdrojový kód pouhé jedné třídy, budete jistě potěšeni, když prohlásím, že na toto omezení již můžete zapomenout. V jazyce Visual Basic 2005 můžete do jednoho souboru třídy vložit definice tolika tříd, kolik budete potřebovat. Nedá mi, abych se ještě na chvíli nepřistavil u dědičnosti. Výborné je, že s pomocí dědičnosti můžete odvozovat nové třídy, a to tak od vestavěných jako i od uživatelsky definovaných tříd. Když si uvědomíte, že společná knihovna tříd vývojově-exekuční platformy Microsoft .NET Framework 2.0 je doslova přeplněna třídami různého druhu a povahy, otevírá se vám neuvěřitelný prostor pro plný rozvoj vaší kreativity. Uveďme si malou ukázku. Myslíte si, že tlačítka, která vytváří třída `Button` z jmenného prostoru `System.Windows.Forms` nejsou dost atraktivní? Rádi byste je obohatili o nové vizuální styly, nebo dokonce o novou funkcionalitu? Je-li vaším společníkem Visual Basic 2005, tohle všechno můžete udělat! Stačí, když od třídy `Button` odvodíte novou, svou vlastní třídu, kterou podle svých potřeb rozšíříte. Budete-li chtít, můžete třídu zapouzdřit do uživatelského ovládacího prvku. Ten pak budete moci přidat do soupravy nástrojů **Toolbox**, což znamená, že vytváření moderních tlačítek bude otázkou několika klepnutí myší. Navíc, ovládací prvek, který si připravíte, můžete zaslat také svým spolupracovníkům, kamarádům či zákazníkům. Vaše know-how tak může pomoci velké skupině lidí.

5.1 Definice třídy, datových položek, vlastností a metod

Podobně jako v jazyce Visual Basic 6.0, také ve Visual Basicu 2005 je třída základní šablonou pro generování objektů jistého typu. Třída je tedy jakýmsi modrotiskem, který musí být správně navržený a definovaný tak, aby jej bylo možné použít pro vytváření instancí čili zapouzdřených obrazů třídy v operační paměti počítače. Nacházíte-li se v integrovaném vývojovém prostředí jazyka Visual Basic 2005, můžete novou třídu přidat podle následující postupu:

1. Otevřete nabídku **Project** a klepněte na položku **Add Class...**.
2. Zobrazí se dialogové okno **Add New Item** se seznamem instalovaných šablon. Ujistěte se, že je vybrána šablona **Class** – ta reprezentuje standardní třídu jazyka Visual Basic 2005 (obr. 5.1).



Obr. 5.1: Přidání nové třídy do projektu jazyka Visual Basic 2005

3. Programový kód naší nově vytvářené třídy bude uložen do souboru s implicitním názvem Class1.vb. Pokud chcete, můžete pojmenování souboru změnit zadáním vašeho názvu do textového pole **Name**.
4. Jste-li spokojeni, klepněte na tlačítko **Add**.

Za předpokladu, že jste neměnili název souboru s třídou, Visual Basic 2005 vytvoří soubor Class1.vb a umístí jej do kořenové složky, v níž se nacházejí všechny ostatní projektové soubory aplikace .NET. Odkaz na sestavený soubor bude uložen také do stromové struktury projektových součástí, jež jsou seskupeny v okně **Solution Explorer**. Visual Basic 2005 ovšem neponechá obsah vytvořeného souboru prázdný, ale vloží do něj programovou kostru nové třídy s názvem `Class1`. Poté soubor otevře v editoru zdrojového kódu, čímž připraví vše potřebné pro to, abyste mohli začít se svou prací.

Obraz třídy `Class1` je velice jednoduchý:

```
Public Class Class1

End Class
```

Tělo nové třídy je ohraničeno příkazy `Class` a `End Class`. Jelikož v jednom souboru se může nyní nacházet i více tříd, je toto explicitní vymezení velice důležité. Prostřednictvím něj totiž může Visual Basic 2005 vždy přesně určit, kde kód jedné třídy končí, anebo naopak, kde kód jiné třídy začíná. Před příkazem `Class` stojí modifikátor přístupu. Jazyk Visual Basic 2005 standardně přiřazuje všem nově vygenerovaným třídám možnost veřejného přístupu, což dává trefně na známost přístupový modifikátor `Public`. Třída však nemusí nevyhnutně disponovat veřejným přístupem. Budete-li chtít viditelnost třídy omezit, můžete použít vhodnějšího zástupce z plejády přístupových modifikátorů (například `Private`, `Friend`, `Protected` nebo `Protected Friend`). V naprosté většině případů se však v hlavičce třídy nachází klíčové slovo `Public`. Pokud modifikátor přístupu není uveden, třída je implicitně deklarována jako přátelská (`Friend`).

V těle třídy se mohou vyskytovat rozmanité datové členy, z nichž nejpoužívanější jsou datové položky, vlastnosti a metody. Jejich stručná charakteristika je uvedena níže.

1. **Datové položky.** Datové položky představují proměnné a konstanty deklarované uvnitř třídy. Proměnné reprezentují zdroje uchovávání dat, které jsou potřebné pro správné fungování instancí třídy, tedy objektů. Konstanty zase dobře poslouží v případě, kdy je nutné používat jednu pevně inicializovanou hodnotu na více místech v těle třídy. Tak proměnné jako i konstanty mohou být deklarovány pomocí přístupových modifikátorů. Není-li ovšem modifikátor zadán přímo, datové položky jsou implicitně soukromé, a tudíž k nim klientský kód nemůže přistupovat. Ačkoliv je možné datové položky deklarovat s příznakem `Public` a vystavit je tak navenek, není tato technika doporučována. Příčinou je, že takto bychom porušili jeden z principů objektově orientovaného programování, kterým je skrývání dat. Veškerá data, která objekt vlastí, mají být pouze jeho – nelze dovolit, aby bylo k těmto datům přistupováno dle libovůle zvenčí. Datové položky mohou být instanční (přináleží instanci třídy a pro každou instanci jsou jedinečné) nebo sdílené (přináleží samotné třídě a jsou sdíleny všemi jejími instancemi). Deklarujete-li typickou proměnnou v těle třídy, vytváříte instanční datovou položkou. Pro tento druh datových položek je charakteristické, že každá instance obsahuje svou vlastní sadu položek. Když ve třídě `A` deklarujeme jednu proměnnou `X` typu `Integer` a posléze vytvoříme dva objekty této třídy `Obj_A` a `Obj_B`, každý z těchto objektů bude mít k dispozici svoji datovou položku `X`. Obě „kopie" datové položky `X` budou na sobě zcela nezávislé, a nebudou se tudíž nijak ovlivňovat. Na druhou stranu, sdílené datové položky patří třídě a jako takové jsou společné všem instancím dané třídy. Nadvážeme-li na zmíněnou třídu `A`, můžeme pomocí modifikátoru `Shared` deklarovat sdílenou datovou položku `Y`. Sdílená datová položka se bude vyskytovat pouze v jednom vyhotovení, což si můžeme ověřit poté, co uskutečníme instanciaci třídy a založení dvou objektů (`Obj_A` a `Obj_B`).

2. **Vlastnosti.** Aby bylo možné zjišťovat informace o stavových veličinách, respektive datových položkách objektu, je zapotřebí implementovat mechanizmus vlastností. Vlastnosti působí jako přestrojené procedury Sub, které se deklarují pomocí příkazu `Property`. V jazyce Visual Basic 2005 vypadá definice vlastností jinak, než tomu bylo ve Visual Basicu 6.0. Zatímco dříve byly bloky `Get` a `Set` vlastností odděleny, nyní jsou syntakticky spojeny do jednoho souvislého segmentu, jenž tvoří tělo vlastnosti. Podobně jako datové položky, také vlastnosti mohou mít své modifikátory přístupu. Přestože k dispozici je všech pět známých modifikátorů (`Private`, `Public`, `Friend`, `Protected` a `Protected Friend`), první z nich se využívá velice málo, ne-li vůbec. Vzhledem k tomu, že soukromé vlastnosti nemohou být volány z vnějšího kódu, jejich míra využití je minimální, přičemž se omezuje pouze na interní komunikaci s jinými datovými členy deklarovanými v těle jedné a tytéž třídy. Jestliže budete pracovat pouze s instančními datovými položkami, ve vaší třídě najdou své uplatnění nejspíše pouze instanční vlastnosti.

Také vlastnosti však mohou být sdílené – jejich proměnu lze uskutečnit pomocí klíčového slova `Shared`. Sdílené vlastnosti jsou využívány zejména při potřebě získávat nebo modifikovat hodnoty sdílených datových položek.

3. **Metody.** Během svého životního cyklu objekty reagují na události a provádějí příslušné akce, které jsou ze syntaktické stránky reprezentovány metodami. Roli metod sehrávají procedury Sub a funkce, podle toho, zda je pro programátora důležité pracovat také s návratovými hodnotami či nikoliv. Metody mohou přebírat argumenty, přistupovat ke třídám společné knihovny tříd platformy Microsoft .NET Framework 2.0 a dělat spoustu jiných činností. Není-li v deklaraci metody uveden modifikátor přístupu, je takováto metoda považována za veřejnou. Programátor však samozřejmě může rozsah viditelnosti metody omezit pomocí vhodně zapsaného přístupového modifikátoru. Zatímco instanční metody mohou pracovat pouze ve spojení s jistou instancí třídy, sdílené metody tímto omezením netrpí. Lze je proto aktivovat přímo, aniž by byla vyžadována instanciace třídy. Sdílené metody deklarujeme pomocí modifikátoru `Shared`.

Abychom však nezůstali pouze u teorie, představíme si třídu, která bude definovat datovou položku, vlastnost a metodu. Třída se jmenuje `Nápověda` a její zdrojový kód vypadá následovně:

```
'Definice třídy Nápověda.
Public Class Nápověda
    'Deklarace privátní datové položky třídy.
    Dim Soubor As String
    'Definice instanční vlastnosti SouborSNápovědou.
    Public Property SouborSNápovědou() As String
        Get
            Return Soubor
        End Get
        Set(ByVal value As String)
            Soubor = value
        End Set
    End Property
    'Definice instanční metody ZobrazitNápovědu.
    Public Sub ZobrazitNápovědu()
        Dim AktivníFormulář As Form = Form.ActiveForm
        Help.ShowHelp(AktivníFormulář, Soubor)
    End Sub
End Class
```

Instance třídy `Nápověda` budou umět zobrazovat obsahy komprimovaných souborů s nápovědou HTML (extenze .chm). Nápověda ve stylu HTML je nástupkyní o něco starší technologie elektronické dokumentace, která byla poháněna systémem WinHelp 4.0. Uvnitř naší třídy je deklarována jedna odkazová proměnná s názvem `Soubor` a datovým typem `String`. Tato datová položka bude sloužit pro ukládání nebo získávání jména souboru s nápovědou HTML. Všimněte si, že datová položka `Soubor` je deklarována pomocí příkazu `Dim`, aniž by byl výslovně určen její modifikátor přístupu. Visual Basic 2005 takto deklarované proměnné implicitně přiřadí přístupový modifikátor `Private`, takže proměnná bude dostupná pouze pro ostatní datové členy umístěné v těle třídy.

Popojedeme-li dále, objevíme kód instanční vlastnosti `SouborSNápovědou`. Uvedená vlastnost je určená pro čtení i zápis, což znamená, že hodnotu cílové datové položky (`Soubor` typu `String`) bude pomocí této vlastnosti možné nejenom číst, ale i modifikovat. Již jsme si řekli, že syntaktická struktura vlastností je v jazyce Visual Basic 2005 jiná. Důkazem tohoto konstatování je samotný kód, který názorně demonstruje, že použití dvou samostatných příkazů `Property` pro definici bloků `Get` a `Set` je již minulostí. Nové rozvržení je kompaktnější a efektivnější: definice vlastnosti pozůstává ze

dvou samostatně stojících bloků neboli procedur, které jsou vymezeny příkazy `Get-End Get` a `Set-End Set`. V bloku `Get` je volán příkaz `Return`, jenž zabezpečuje vrácení návratové hodnoty privátní datové položky `Soubor` zpět do klientského kódu. Není žádným překvapením, že za inicializaci soukromé datové položky `Soubor` je zodpovědný blok `Set`. Všimněte si, že proceduře `Set` je nutno předat argument v podobě textového řetězce (typ `String`), jenž je uložen do parametru s názvem `value`. Jakmile je parametr `value` inicializován, je jeho hodnota v těle procedury `Set` přiřazena hodnotové proměnné `Soubor`. V jazyce Visual Basic 6.0 jste mohli, kromě procedur `Get` a `Set`, definovat také proceduru `Let`, účelem které bylo přiřazovat hodnotu (ne objektovou referenci) do datové položky. Visual Basic 2005 neprovádí diferenciaci mezi procedurami `Set` a `Let` na základě typu hodnot, s nimiž pracují. Proto došlo k vyloučení procedury `Let`, takže ta se již nepoužívá. K inicializaci privátních datových položek nyní postačuje pouze procedura `Set`.

Instanční metoda `ZobrazitNápovědu` je vskutku jednoduchá, neboť vše se točí kolem třídy `Help` a její sdílené metody `ShowHelp`. Třída `Help` zapouzdřuje funkcionalitu stroje pro práci s nápovědou HTML. Můžete ji najít v jmenném prostoru `System.Windows.Forms`. Pokud sdílené metodě `ShowHelp` předáte odkaz na aktivní formulář a cestu k souboru s nápovědou, metoda zobrazí obsah nápovědy v dialogovém okně.

5.2 Vytváření instancí třídy

Třídy jsou uživatelsky definované odkazové datové typy. Jejich instance se vytvářejí pomocí operátoru `New`. Založení instance třídy se v jazyce Visual Basic 2005 nejčastěji uskutečňuje třemi níže uvedenými způsoby.

```
'1. způsob založení instace třídy: Použití operátoru New
'v deklaračním příkazu Dim.
Dim obj As New Nápověda

'2. způsob založení instance třídy: Explicitní inicializace odkazové
'proměnné v deklaračním příkazu Dim pomocí operátoru New.
Dim obj As Nápověda = New Nápověda

'3. způsob založení instance třídy: Použití operátoru New v samostatném
'přiřazovacím příkazu.
Dim obj As Nápověda
obj = New Nápověda
```

Všechny představené varianty vytvoření instance ukázkové třídy `Nápověda` jsou funkčně ekvivalentní. Použijete-li kterýkoliv z nich, výsledkem bude založení nové instance třídy `Nápověda`, která bude po svém zrodu uložena na řízenou hromadu.

UPOZORNĚNÍ: Příkaz **Set** a zakládaní instancí tříd

Jestliže nebyla instance třídy v jazyce Visual Basic 6.0 vytvořena pomocí operátoru `New` v rámci deklaračního příkazu `Dim`, bylo možné ji vytvořit později použitím příkazu `Set`.

```
Dim obj As SpouštěčAplikací
Set obj = New SpouštěčAplikací
```

Visual Basic 2005 příkaz `Set` v tomto kontextu již nepodporuje. I kdybyste však tento příkaz použili, k chybě nedojde. Visual Basic 2005 je natolik chytrý, že si použití zastaralého příkazu všimne a pohotově jej odstraní.

Řízená hromada je část operační paměti aplikačního procesu, která je vyhrazena pro úschovu objektů řízených odkazových datových typů. Řízenou hromadu kontroluje společné běhové prostředí CLR prostřednictvím automatického správce paměti. Správce paměti sleduje stupeň vytížení řízené hromady a v případě potřeby z ní automaticky odstraňuje nepotřebné a z programového kódu nedosažitelné objekty. Přínosem automatického správce paměti je skutečnost, že programátoři nemusí explicitně uvolňovat své objekty poté, co s nimi skončí. Ačkoliv je automatická správa paměti jedním z hlavních pilířů vývojově-exekuční platformy Microsoft .NET Framework 2.0, pro vývojáře zvyklé na Visual Basic nejde zas až o tak zásadní změnu. Je to proto, že Visual Basic nikdy programátory nenutil, aby jednou alokované objekty někdy uvolňovali. Nicméně je nutné poukázat na to, že paměťový management se od dob Visual Basicu 6.0 hodně změnil. V minulosti se doba životnosti objektu řídila mechanizmem počítání referencí (šlo o důsledek aplikované technologie COM). Každý objekt disponoval svým vlastním interním počítadlem referencí, jehož hodnota se zvětšovala nebo naopak zmenšovala podle toho, zda vznikaly anebo zanikaly objektové reference nasměrované na tento objekt. Když hodnota počítadla referencí klesla na nulu, neexistovaly žádné platné objektové reference a objekt se tudíž sám zlikvidoval. Na platformě .NET je situace jiná – životní cykly objektů sleduje automatický správce paměti na základě stromu referencí. Každý objekt má vlastní strom, který odráží míru jeho využití. Pomocí referenčního stromu je automatický správce paměti schopný určit okamžik, kdy již objekt není nedosažitelný ze zdrojového kódu. Nedosažitelný objekt je nepotřebný, a proto může být bez jakýchkoliv potíží podroben destrukci.

Operátor `New` vytváří objekty na řízené hromadě. Ačkoliv je tato operace hlavní, není jediná, kterou tento operátor iniciuje. Dále totiž dochází k vytvoření odkazové proměnné, do níž je uložena typově silná objektová reference směřující na objekt nacházející se na řízené hromadě. Pouto mezi odkazovou proměnnou a cílovým objektem je velice silné, neboť je to právě tato proměnná, která dovoluje získat přímý přístup k požadovanému objektu.

Je-li objekt sestrojen, můžeme používat jeho vlastnosti a metody:

```
'Do vlastnosti SouborSNápovědou uložíme textový řetězec, jenž
'popisuje cestu k souboru s nápovědou HTML.
obj.SouborSNápovědou = "c:\windows\help\windows.chm"
'Obsah zvoleného souboru s nápovědou HTML zobrazíme pomocí
'metody ZobrazitNápovědu.
obj.ZobrazitNápovědu()
```

Jakmile je objekt zrozen, můžete s ním provádět naprogramované operace. Kupříkladu výsledkem výše zapsaného zdrojového kódu bude spuštění stroje nápovědy HTML (hh.exe) a zobrazení hlavní nabídky souboru *windows.chm*.

5.3 Konstruktory a destruktory

Když jsme si povídali o datových členech třídy, vzpomenuli jsme datové položky, vlastnosti a metody. Přestože je tato trojice datových členů nejčastějším „obyvatelem" třídy, existují také další programové elementy, které se mohou uvnitř třídy vyskytnout. V této podkapitole probádáme konstruktory a destruktory. V obou případech se jedná o nové datové členy jazyka Visual Basic 2005, takže bude jistě přínosné, když se u nich na chvíli přistavíme.

Konstruktor je první metoda, která je automaticky aktivována poté, co dojde k vytvoření instance řízené třídy. Z přísně formálního hlediska je konstruktor chápán jako čistě inicializační metoda, jejíž úkolem je uvést datové položky zrozené instance do použitelného, respektive výchozího stavu. Na rozdíl od jiných programovacích jazyků .NET, jakými jsou třeba C# nebo C++/CLI, konstruktor Visual Basicu 2005 je reprezentován speciální procedurou Sub s názvem `New`.

```
'Konstruktor jazyka Visual Basic 2005.
Public Sub New()
'Tělo konstruktoru je implicitně prázdné.
End Sub
```

Tento konstruktor je veřejně přístupný, bezparametrický a ve svém těle nemá žádné programové instrukce. Konstruktor nevrací žádnou hodnotu, což lze snadno vyčíst, neboť je implementován jako procedura Sub a nikoliv jako funkce. Syntaktická přítomnost konstruktoru uvnitř třídy není povinná, z čehož bychom mohli usuzovat, že s konstruktorem nemusí třída vůbec pracovat. To ovšem není tak docela pravda: Jestliže se ve třídě nenachází explicitně definovaný konstruktor, Visual Basic 2005 vygeneruje přesně takový konstruktor, jaký je uveden výše (čili veřejně přístupný, bezparametrický a s prázdným tělem). Takto připravený konstruktor je volán pokaždé, když je na řízené hromadě alokována nová instance třídy. Aby mohla být instance vůbec vytvořena, musí být konstruktor deklarován jako veřejně přístupný. Kdybychom totiž přístupový modifikátor konstruktoru `Public` zaměnili za `Private`, konstruktor by nemohl být volán, a proto by ani nebylo možné vytvářet instance dané třídy.

Programátoři v jazyce Visual Basic 2005 mohou konstruktor obohatit o parametry, čímž z něj udělají parametrický konstruktor.

```
'Signatura konstruktoru může být doplněna o parametry.
Public Sub New(ByVal x As Byte, ByVal y As Long)

End Sub
```

Nic jim dokonce nebrání ani v tom, aby napsali více variant konstruktoru, a ten tak přetížili. Výsledkem je přetížený konstruktor, jenž si dovede poradit se zpracováním různých typů dat.

```
'Ukázka techniky přetížení konstruktoru.
Public Sub New(ByVal a As Boolean)

End Sub

Public Sub New(ByVal x As Byte, ByVal y As Long)

End Sub

Public Sub New(ByRef s As String, ByVal o As Object)

End Sub
```

Co myslíte, mohl by nám být konstruktor užitečný v naší ukázkové třídě `Nápověda`? Odpověď zní ano, protože pomocí konstruktoru bychom mohli inicializovat soukromou datovou položku tak, aby již od vytvoření instance třídy obsahovala smysluplnou hodnotu (v našem případě jde o textový řetězec reprezentující cestu k souboru s nápovědou HTML). Zkusme tedy kód třídy `Nápověda` modifikovat.

```
Public Class Nápověda
    Dim Soubor As String
    'Bezparametrický konstruktor uskutečňuje inicializaci
    'soukromé datové položky třídy Nápověda.
    Public Sub New()
        Soubor = "d:\Nápověda.chm"
    End Sub
    Public Property SouborSNápovědou() As String
        Get
            Return Soubor
        End Get
        Set(ByVal value As String)
            Soubor = value
        End Set
    End Property
    Public Sub ZobrazitNápovědu()
        Dim AktivníFormulář As Form = Form.ActiveForm
        Help.ShowHelp(AktivníFormulář, Soubor)
    End Sub
End Class
```

Pakliže inicializujeme hodnotovou proměnnou `Soubor` typu `String` již v konstruktoru, nemusíme později volat vlastnost `SouborSNápovědou`, ale ihned můžeme obsah implicitně určeného souboru zobrazit pomocí metody `ZobrazitNápovědu`.

```
Dim obj As New Nápověda
obj.ZobrazitNápovědu()
```

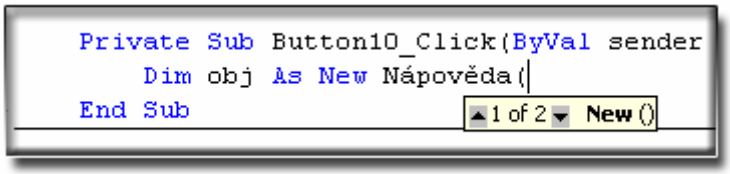
Dobrá, kód funguje spolehlivě, ovšem co byste řekli tomu, kdybychom chtěli zadat jméno souboru s nápovědou HTML již v dobře zakládání instance pomocí operátoru `New`? Je to vůbec možné? Ano, v jazyce Visual Basic 2005 to možné je. Nastíněný problém vyřešíme vložením dalšího konstruktoru, tentokrát parametrického. Parametr bude přijímat textovou identifikaci kýženého souboru s nápovědou stylu HTML. Jelikož se budou v naší třídě nacházet dvě varianty procedury Sub s názvem `New`, původní konstruktor bude přetížen.

```
'Ukázka přetížení konstruktoru třídy Nápověda.
'-----------------------------------------------

'Původní bezparametrický konstruktor.
Public Sub New()
    Soubor = "d:\Nápověda.chm"
End Sub

'Nový parametrický konstruktor.
Public Sub New(ByVal SouborSNápovědouHTML As String)
    Soubor = SouborSNápovědouHTML
End Sub
```

Když nyní v deklaračním příkazu zadáme za názvem třídy `Nápověda` otevírací kulatou závorku, technologie IntelliSense zjistí, že konstruktor této třídy je přetížen. Proto pro nás zobrazí nabídku s otázkou, kterou verzi konstruktoru si přejeme použít (obr. 5.2).



Obr. 5.2: Technologie IntelliSense nám nabízí dvě verze přetíženého konstruktoru

Pokud bychom chtěli použít výchozí inicializaci soukromé datové položky, mohli bychom zavolat explicitně definovaný bezparametrický konstruktor. Náš konstruktor je ovšem inteligentnější, neboť nám umožňuje přímo zadat cestu k požadovanému souboru s HTML nápovědou. Textový řetězec s cestou předáme přetíženému konstruktoru v podobě argumentu takto:

```
'Využití parametrického konstruktoru při explicitní inicializaci
'soukromé datové položky instance třídy Nápověda.
Dim obj As New Nápověda("d:\Nápověda.chm")
obj.ZobrazitNápovědu()
```

Na základě dodaného argumentu bude aktivována odpovídající verze přetíženého konstruktoru, která zabezpečí inicializaci privátní datové položky. Pro spuštění nápovědy již proto nemusíme používat vlastnost `SouborSNápovědou`, ale ihned můžeme zavolat metodu `ZobrazitNápovědu`.

Jazyk Visual Basic 2005 uvádí také protějšky konstruktorů, které jsou známé jako destruktory. Jak jsme si již řekli, konstruktor je první procedura, která je volána okamžitě po vytvoření instance řízené třídy. Ačkoliv je primárním úkolem konstruktoru inicializace datových položek, spektrum činností realizovaných konstruktorem může být daleko širší. Konstruktor může volat další metody, domlouvat se s vlastnostmi anebo aktivovat vestavěné datové členy společné knihovny tříd platformy Microsoft .NET Framework 2.0. Opakem konstruktoru je destruktor. Destruktor je obvykle poslední metoda, která se dostává ke slovu těsně před koncem doby životnosti objektu. Slovíčko „těsně" je v předcházející větě použito záměrně: likvidace objektů je totiž v rukou automatického správce paměti, a tudíž přesný okamžik aktivace destruktoru je určen běhovým prostředím a nikoliv programátorem. Tomuto mechanizmu likvidace objektů se říká nedeterministická finalizace. Tento typ finalizace se zásadně odlišuje od způsobu, jakým byly objekty likvidovány v nativním prostředí (například v jazyku C++).

Destruktor je v jazyce Visual Basic 2005 implementován jako speciální finalizační metoda neboli finalizér. Pokud se tato metoda nachází v těle třídy, automatický správce paměti ji v příhodné chvíli zavolá. Ještě než se podíváme na zdrojový kód finalizéru, chtěl bych vás upozornit, že ji nemusíte definovat. Řízené zdroje, které vaše aplikace .NET během svého života alokuje, budou zcela automaticky zlikvidovány pomocí správce paměti. Není proto nutné, abyste prováděli explicitní dealokaci těchto objektů. Jiné je to však tehdy, kdy budete z řízené aplikace .NET volat nativní funkce (ať už z vlastní dynamicky linkované knihovny nebo komponenty třetí strany). Jakékoliv nativní zdroje, které takto vytvoříte (typicky se jedná například o manipulátory oken či souborů), budete muset posléze manuálně dealokovat. V tomto případě vám může být finalizační metoda užitečná.

Finalizér sestrojíte následovně:

1. Otevřete editor zdrojového kódu.
2. Ze seznamu **Class Name** vyberte jméno vaší třídy (v této ukázce budeme používat pokusnou třídu `Nápověda`).
3. Ze seznamu **Method Name** vyberte položku s názvem `Finalize()`.

Visual Basic 2005 přidá do editoru zdrojového kódu instrukce definující finalizační metodu `Finalize`:

```
'Programová kostra finalizační metody Finalize.
Protected Overrides Sub Finalize()
    MyBase.Finalize()
End Sub
```

Finalizační metoda je představována chráněnou (modifikátor `Protected`) procedurou Sub s názvem `Finalize`, která překrývá (klíčové slovo `Overrides`) stejnojmennou proceduru bázové třídy `System.Object`. Ano, vím, že o překrývání metod, respektive implementaci polymorfizmu jsme ještě nemluvili, ovšem definice finalizační metody si využití této techniky vyžaduje. Finalizér je chráněný, což znamená, že jeho obor platnosti je omezen pouze na bázovou třídu (`System.Object`) a třídy odvozené od této bázové třídy.

V těle metody `Finalize` je volána finalizační metoda mateřské třídy `System.Object` pomocí klíčového slova `MyBase`. Tento mechanizmus je nutno dodržovat – finalizační metoda v podtřídě by měla aktivovat svůj protějšek, jehož kód je uložen v bázové třídě.

Abychom poukázali na aktivaci finalizéru, umístíme do něj následující programový kód:

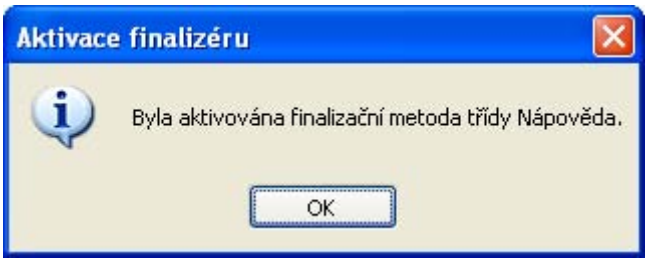
```
Protected Overrides Sub Finalize()
    MyBase.Finalize()
    MessageBox.Show("Byla aktivována finalizační metoda třídy Nápověda.", _
    "Aktivace finalizéru", MessageBoxButtons.OK, _
    MessageBoxIcon.Information)
End Sub
```

Kdy však bude finalizační metoda ve skutečnosti zavolána? Zde narážíme na problém, protože přesný okamžik aktivace finalizéru nedovedeme určit. Když vytvoříme pouze jednu instanci třídy `Nápověda`, budeme mít k dispozici jednu odkazovou proměnnou, v níž je uložena objektová reference směřující na tuto instanci. Ve chvíli, kdy se odkazová proměnná dostane mimo svůj obor platnosti, dojde k destrukci v ní uložené objektové reference. Neexistuje-li žádná jiná objektová reference směřující na instanci třídy `Nápověda`, automatický správce paměti zjistí, že naše instance již není dosažitelná z programového kódu. Přestože jsme si říkali, že nedosažitelná instance může být odstraněna, v tomto případě tomu tak není. Instance nemůže být zlikvidována, protože definuje finalizační metodu a ta musí být zavolána ještě před samotnou destrukcí instance. Ovšem čas aktivace finalizéru je plně v rukou automatického správce paměti. Správa paměti přitom nebude uskutečněna dříve, než bude doručen požadavek na nedostatek alokovatelné paměti. Tehdy se iniciativy chopí automatický správce paměti, jenž prozkoumá řízenou hromadu a označí nedosažitelné objekty. Ty objekty, které vyžadují explicitní finalizaci (jako ten náš), budou identifikovány a odkazy na ně budou přidány do speciálního seznamu. Automatický správce poté vytvoří nové, takzvané finalizační programové vlákno, na němž bude uskutečňována finalizace.

Automatická správa paměti se spustí za výše uvedených podmínek, a také v těchto případech:

1. Při ukončení běhu aplikace .NET.
2. Při volání sdílené metody `Collect` třídy `GC` z jmenného prostoru `System`.

Zkusme vyzkoušet první případ: spusťme aplikaci, vytvořme instanci třídy `Nápověda` a ukončeme aplikaci. Jestliže vše proběhlo tak jak mělo, na obrazovce se na okamžik objeví toto dialogové okno:



Obr. 5.3: Informační okno dokazující aktivaci finalizační metody `Finalize`

Stejného cíle dosáhneme voláním sdílené metody `Collect` třídy `GC`. Tato metoda nařídí ihned vykonat automatickou správu paměti:

```
'Sílená metoda Collect třídy GC garantuje neprodlenou aktivaci
'automatické správy paměti.
GC.Collect()
```

5.4 Sdílené datové členy tříd

Třídy jazyka Visual Basic 2005 mohou obsahovat sdílené datové členy. Pro všechny sdílené členy třídy platí stejná pravidla:

- Ke sdílenému datovému členu třídy mají přístup všechny instance třídy. Datové členy jsou tedy společným vlastnictvím všech instancí, které jsou ze třídy vytvořeny.
- K přístupu ke sdíleným datovým členům není nutné vytvářet instanci třídy. Na rozdíl od instančních datových členů (instanční datové položky, vlastnosti a metody), lze se sdílenými datovými členy pracovat, aniž by bylo nutné sestrojovat novou instanci třídy.

Sdílené datové členy jsou novinkou Visual Basicu 2005, takže v jazyce Visual Basic 6.0 je nenajdete. Sdílené datové členy si velice rychle oblíbíte – s jejich pomocí totiž můžete vyřešit mnoho programátorských úkolů bez nutnosti instanciace třídy. Také naše ukázková třída `Nápověda` se dá upravit tak, aby byla veškerá funkčnost shromážděna do jedné sdílené metody `ZobrazitNápovědu`.

```
Public Class Nápověda
'Metoda ZobrazitNápověda je deklarována jako sdílená (Shared).
    Public Shared Sub ZobrazitNápovědu _
    (ByVal SouborSNápovědouHTML As String)
        Dim AktivníFormulář As Form = Form.ActiveForm
        Help.ShowHelp(AktivníFormulář, SouborSNápovědouHTML)
    End Sub
End Class
```

Je to docela veliká změna, že? V hlavičce sdílené metody `ZobrazitNápovědu` se nachází klíčové slovo `Shared`, které říká, že metoda není instanční, nýbrž sdílená. V těle metody je pak uložen již známý kód pro spuštění stroje nápovědy HTML a zobrazení obsahu zadaného souboru s extenzí .chm. Použití sdílené metody nevyžaduje sestrojení instance. Řečeno jinými slovy, nemusíte deklarovat žádnou odkazovou proměnnou, ani používat operátor `New`. Vše, co musíte udělat, je zapsat jméno třídy, které je následováno tečkovým operátorem (`.`) a názvem sdílené metody se specifikací vstupního argumentu:

```
'Volání sdílené metody ZobrazitNápovědu třídy Nápověda.
Nápověda.ZobrazitNápovědu("d:\Nápověda.chm")
```

TIP: Optimalizace kódu sdílené metody **ZobrazitNápovědu**

Programovou podobu sdílené metody `ZobrazitNápovědu` třídy `Nápověda` můžeme optimalizovat. Jednoduše vynecháme deklaraci a explicitní inicializaci odkazové proměnné `AktivníFormulář`. Sdílenou vlastnost `ActiveForm` třídy `Form` pak můžeme zavolat přímo ze signatury sdílené metody `ZobrazitNápovědu`.

```
Public Class Nápověda
    Public Shared Sub ZobrazitNápovědu _
    (ByVal SouborSNápovědouHTML As String)
        Help.ShowHelp(Form.ActiveForm, SouborSNápovědouHTML)
    End Sub
End Class
```

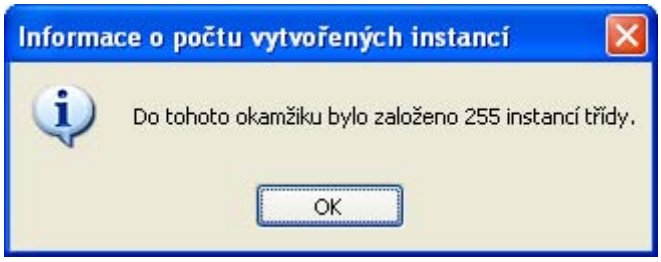
Možnost použít sdílenou datovou položku nám přijde vhod kupříkladu tehdy, když budeme chtít zjistit, kolik instancí třídy již bylo založeno. Podívejte se na níže uvedený výpis zdrojového kódu:

```
'Třída obsahuje sdílenou datovou položku a sdílenou metodu.
Public Class Třída_A
    Shared PočetInstancí As Integer
    Public Sub New()
        PočetInstancí += 1
    End Sub
    Public Shared Function ZjistitPočetInstancí() As Integer
        Return PočetInstancí
    End Function
End Class
```

Předestřená třída deklaruje jednu sdílenou datovou položkou `PočetInstancí`, do které budeme ukládat číselnou reprezentaci množství zrozených instancí. Abychom mohli proces zakládání instancí sledovat, definujeme bezparametrický veřejně přístupný konstruktor, v jehož těle provádíme inkrementaci hodnoty soukromé datové položky třídy. Informaci o skutečném počtu vytvořených objektů nám na požádání poskytne sdílená metoda `ZjistitPočetInstancí`. Pro použití třídy zapíšeme následující programový kód:

```
For x As Byte = 0 To Byte.MaxValue - 1
    Dim obj As New Třída_A
Next
MessageBox.Show("Do tohoto okamžiku bylo založeno " & _
Třída_A.ZjistitPočetInstancí() & " instancí třídy.", _
"Informace o počtu vytvořených instancí", MessageBoxButtons.OK, _
MessageBoxIcon.Information)
```

V cyklu `For...Next` vytváříme 255 instancí třídy `Třída_A`, přičemž o této skutečnosti nás záhy informuje objevivší se dialogové okno (obr. 5.4).



Obr. 5.4: Pomocí sdílených datových členů lze analyzovat počet sestrojených instancí třídy

5.5 Dědičnost – vytváření odvozených tříd

Programovací jazyk Visual Basic 2005 je plně objektově orientován, takže implementuje také jeden ze stěžejních prvků objektově orientované koncepce programování, kterým je dědičnost. Podpora dědičnosti znamená pro vývojáře to, že od jednou definovaných tříd mohou v procesu dědění odvozovat třídy nové. Tyto odvozené třídy, jež jsou nazývány také jako podtřídy, disponují veškerou funkcionalitou své mateřské třídy. Jestliže mateřská třída vytváří základní funkcionalitu, podtřída „zděděnými" dovednostmi disponuje také. Ba co víc, podtřída může stávající konstrukci bázové třídy dále rozšiřovat či modifikovat.

Vývojově-exekuční platforma Microsoft .NET Framework 2.0 dovoluje aplikovat pouze jednoduchou dědičnost. To znamená, že od jedné bázové třídy je možné vždy odvodit právě jednu podtřídu. Společná jazyková specifikace CLS neumožňuje v .NET-programovacích jazycích používat vícenásobnou dědičnost (kdy by jedna podtřída mohla dědit své charakteristiky od většího počtu mateřských tříd). Po pravdě řečeno, vícenásobnou dědičnost lze využít jenom při vývoji nativních aplikací pomocí softwarového produktu Visual C++ 2005. Jazyky Visual Basic 2005, C# 2.0 a ani další nástroje přímé použití vícenásobné dědičnosti nedovolují. V dalším textu proto budeme pojem „dědičnost" chápat jako synonymum „jednoduché dědičnosti".

Dědění se v jazyce Visual Basic 2005 uskutečňuje vytvořením nové třídy, v jejímž těle se nachází příkaz `Inherits` se specifikací mateřské třídy. Kdybychom chtěli od naší třídy `Nápověda` odvodit další třídu, postupovali bychom takhle:

```
'Třída Nápověda sehrává roli mateřské třídy.
Public Class Nápověda
    Dim Soubor As String
    Public Sub New()
        Soubor = "d:\Nápověda.chm"
    End Sub
    Public Sub New(ByVal SouborSNápovědouHTML As String)
        Soubor = SouborSNápovědouHTML
    End Sub
    Public Property SouborSNápovědou() As String
        Get
            Return Soubor
        End Get
        Set(ByVal value As String)
            Soubor = value
        End Set
    End Property
    Public Sub ZobrazitNápovědu()
        Dim AktivníFormulář As Form = Form.ActiveForm
        Help.ShowHelp(AktivníFormulář, Soubor)
    End Sub
End Class

'Třída NováNápověda je odvozena od třídy Nápověda.
'Proces dědění je iniciován příkazem Inherits.
Public Class NováNápověda
    Inherits Nápověda
End Class
```

POZNÁMKA: Visual Basic 2005 umožňuje umístit definice více tříd do jednoho souboru

Jazyk Visual Basic 2005 nabízí programátorům možnost vkládat do jednoho souboru s příponou .vb větší množství tříd. Již tedy neplatí zažité paradigma z Visual Basicu 6.0, které bychom mohli charakterizovat jako „jeden soubor jedna třída". Mít více tříd v jednom souboru je praktické řešení, které nejenomže minimalizuje počet „třídních" souborů, ale také napomáhá lepší kompaktnosti a strukturovanosti vyvíjené softwarové aplikace.

Třída `NováNápověda` je podtřídou třídy `Nápověda`, z čehož plyne, že nová třída dědí všechny datové členy třídy mateřské kromě konstruktorů. Pro nás to znamená, že instance třídy `NováNápověda` může volat vlastnost `SouborSNápovědou` a metodu `ZobrazitNápovědu` (obě dvě jsou veřejně přístupné). Konstruktory nejsou předmětem dědění, takže ani námi definované dvě verze přetíženého konstruktoru nebudou zděděné. Ve skutečnosti je v našem případě po zrození instance odvozené třídy volán veřejně přístupný bezparametrický konstruktor bázové třídy. O této skutečnosti se můžeme přesvědčit takto:

```
Dim obj As New NováNápověda
obj.ZobrazitNápovědu()
```

Instanční metoda `ZobrazitNápovědu` třídy `NováNápověda` zobrazí obsah souboru *Nápověda.chm*.

Jestliže v odvozené třídě nedefinujeme vlastní konstruktor, bude implicitně zavolán veřejně přístupný bezparametrický konstruktor bázové třídy.

Koncepce objektově orientovaného programování uplatňuje následující schéma: Konstruktor odvozené třídy by měl volat konstruktor bázové třídy, který je zodpovědný za patřičnou inicializaci datových položek mateřské třídy. Když proto budeme chtít v podtřídě explicitně deklarovat konstruktor, na prvním řádku v jeho těle budeme volat konstruktor bázové třídy. Vzhledem k tomu, že konstruktor bázové třídy je přetížen, můžeme se rozhodnout, kterou verzi tohoto konstruktoru aktivujeme. Víc již prozrazuje další fragment zdrojového kódu:

```
Public Class Nápověda
    Dim Soubor As String
    Public Sub New()
        MessageBox.Show("Bezparametrický konstruktor třídy Nápověda.")
        Soubor = "d:\Nápověda.chm"
    End Sub
    Public Sub New(ByVal SouborSNápovědouHTML As String)
        MessageBox.Show("Parametrický konstruktor třídy Nápověda.")
        Soubor = SouborSNápovědouHTML
    End Sub
    Public Property SouborSNápovědou() As String
        Get
            Return Soubor
        End Get
        Set(ByVal value As String)
            Soubor = value
        End Set
    End Property
    Public Sub ZobrazitNápovědu()
        Dim AktivníFormulář As Form = Form.ActiveForm
        Help.ShowHelp(AktivníFormulář, Soubor)
    End Sub
End Class
```

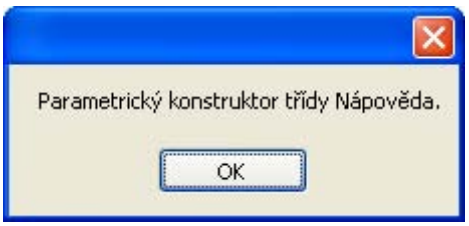
```
Public Class NováNápověda
    Inherits Nápověda
    Public Sub New()
        MyBase.New("d:\JináNápověda.chm")
        MessageBox.Show("Konstruktor třídy NováNápověda.")
    End Sub
End Class
```

Abychom mohli pečlivěji sledovat řetězec aktivací konstruktorů, obohatili jsme je o volání metody `Show` třídy `MessageBox` s informačním textem. V podtřídě `NováNápověda` definujeme bezparametrický konstruktor, ze kterého voláme přetíženou verzi parametrického konstruktoru bázové třídy. Když vytvoříme instanci odvozené třídy, bude nejprve zavolán její konstruktor. Dále bude aktivován parametrický konstruktor mateřské třídy, kterému bude předána textová reprezentace popisující umístění nového souboru s nápovědou HTML. Posléze bude zpracováván další kód konstruktoru odvozené třídy. Když budeme chtít zobrazit obsah souboru *JináNápověda.chm*, použijeme následující programový kód:

```
Dim obj As New NováNápověda
obj.ZobrazitNápovědu()
```

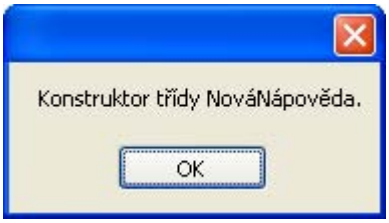
Exekuce těchto dvou řádků zdrojového kódu bude doprovázena třemi vizuálními atributy:

1. Nejprve bude zobrazen dialog, jenž bude říkat, že byl spuštěn parametrický konstruktor bázové třídy (obr. 5.5).



Obr. 5.5: Aktivace parametrického konstruktoru mateřské třídy

2. Poté se objeví dialog, který bude dokumentovat činnost bezparametrického konstruktoru odvozené třídy (obr. 5.6).



Obr. 5.6: Zavolání bezparametrického konstruktoru odvozené třídy.

3. Nakonec bude vyvolána instanční metoda `ZobrazitNápovědu`, která otevře soubor s nápovědou HTML.

5.6 Polymorfizmus realizovaný pomocí dědičnosti

Jako ostřílení vývojáři v jazyce Visual Basic 6.0 víte, že v tomto prostředí lze implementovat polymorfizmus pomocí rozhraní. Visual Basic 2005 jde dále: Ruku v ruce s dědičností se vám nabízí také možnost vytváření polymorfně se chovajících metod a vlastností napříč řetězcem odvozených tříd.

Polymorfizmus realizovaný pomocí dědičnosti je založen na překrytí metody nebo vlastnosti bázové třídy ve třídě odvozené. Aby bylo možné v odvozené třídě překrýt metodu nebo vlastnost mateřské třídy, musí být její deklarace v bázové třídě rozšířena o klíčové slovo `Overridable`. To říká, že metoda nebo vlastnost je virtuální, a tudíž ji lze překrýt v odvozené třídě. Na druhou stranu, metoda nebo vlastnost, která hodlá v podtřídě překrýt svůj protějšek z mateřské třídy, musí být deklarována pomocí klíčového slova `Overrides`. Modifikátor `Overrides` způsobí, že metoda deklarovaná v odvozené třídě bude překrývat stejnojmennou metodu bázové třídy. Kromě shodného jména však musí překrývající metoda disponovat i totožnou signaturou. Řečeno jinými slovy, metoda podtřídy deklarována jako `Overrides` musí mít stejné parametry jako její protějšek v bázové třídě. Parametry se přitom musí shodovat nejenom ve svém počtu či pořadí, ale také v použitých datových typech. Navíc, shoda musí panovat také v dalších atributech signatury, jimiž jsou: modifikátory přístupu, datový typ návratové hodnoty (v případě, že jde o funkci a ne o proceduru Sub) a způsob předávání argumentů parametrům metody (`ByVal` nebo `ByRef`).

Na následujících řádcích si předvedeme, jak překrýt metodu `ZobrazitNápovědu` třídy `Nápověda`. Začneme tím, že hlavičku metody rozšíříme o klíčové slovo `Overridable`, čímž metodu povyšujeme do „virtuálního stavu".

```
Public Overridable Sub ZobrazitNápovědu()
    Dim AktivníFormulář As Form = Form.ActiveForm
    Help.ShowHelp(AktivníFormulář, Soubor)
End Sub
```

Nyní se přesuneme do odvozené třídy `NováNápověda` a zde zapíšeme programový kód metody `ZobrazitNápovědu`, která bude překrývat stejně znějící metodu mateřské třídy.

```
Public Overrides Sub ZobrazitNápovědu()
    Dim AktivníFormulář As Form = Form.ActiveForm
    Dim PoziceX As Integer = Screen.PrimaryScreen.Bounds.Width \ 2
    Dim PoziceY As Integer = Screen.PrimaryScreen.Bounds.Height \ 2
    Help.ShowPopup(AktivníFormulář, "Ukázka techniky překrývání metod " & _
    "v jazyce Visual Basic 2005.", New Point(PoziceX, PoziceY))
End Sub
```

Je-li původní metoda překryta, znamená to, že po vytvoření instance odvozené třídy a zavolání metody s názvem `ZobrazitNápovědu` bude aktivována nově definovaná metoda podtřídy. Abychom nezapomněli, překrývající metoda odvozené třídy se nezaměřuje na zobrazení obsahu nápovědy HTML v hlavním dialogu, ale vykresluje předem určený textový řetězec v bublinovém okně na jisté pozici obrazovky. Pro dosažení stanoveného cíle je volána sdílená metoda `ShowPopup` třídy `Help`, které jsou předány všechny nezbytné parametry (objektová reference na aktivní formulář, uživatelsky definovaný textový řetězec a instance struktury `Point` determinující pozici bublinového okna).

Po svém překrytí se metoda `ZobrazitNápovědu` chová polymorfně. Její styl práce se liší podle charakteru vytvořené instance. Pokud je založena instance bázové třídy a je-li zavolána metoda `ZobrazitNápovědu`, dojde k prezentaci hlavní nabídky souboru s HTML nápovědou. Jestliže je ovšem tato metoda zavolána ve spojení s instancí podtřídy, bude zobrazen textový řetězec v bublinovém okně.

```
'Zde je volána metoda ZobrazitNápovědu bázové třídy.
Dim obj_1 As New Nápověda
obj_1.ZobrazitNápovědu()

'Zde je volána metoda ZobrazitNápovědu odvozené třídy.
Dim obj_2 As New NováNápověda
obj_2.ZobrazitNápovědu()
```

5.7 Rozhraní

V objektově orientované terminologii se pod pojmem rozhraní rozumí uzavřený seznam deklarací metod, vlastností a událostí, které je nutno přesně definovat ve třídách, jež dané rozhraní implementují. Rozhraní je tedy jakýmsi předpisem, který sice říká, co má být uděláno, ovšem již se nijak nezmiňuje o tom, jak to má být uděláno. Podpora rozhraní má na vývojově-exekuční platformě Microsoft .NET Framework 2.0 své nezastupitelné místo, neboť dovoluje svým způsobem vhodně substituovat absenci vícenásobné dědičnosti. Třídy, které v jazyce Visual Basic 2005 připravíte, totiž mohou implementovat libovolný počet rozhraní, čímž se v notné míře minimalizuje potřeba aplikace vícenásobné dědičnosti. Rozhraní je definováno příkazy `Interface` a `End Interface`, které současně vymezují jeho tělo. V něm se mohou nacházet deklarace metod, vlastností a událostí podle potřeb vývojáře. Rozhraní se umísťují do modulů a mohou být opatřena svými přístupovými modifikátory. Implicitně je každé rozhraní přátelské (`Friend`) – je tedy viditelné pouze z toho sestavení, v němž je jeho zdrojový kód uložen. V případě potřeby je však samozřejmě možné viditelnost rozhraní vhodně adjustovat. Členy rozhraní, ať už se jedná o metody, vlastnosti nebo události, jsou přítomné pouze prostřednictvím svých deklaračních příkazů. Ty charakterizují typ daného členu, jeho pojmenování a signaturu. Důležité je mít na paměti skutečnost, že samostatnou definici členů rozhraní dodávají až cílové třídy, které toto rozhraní implementují.

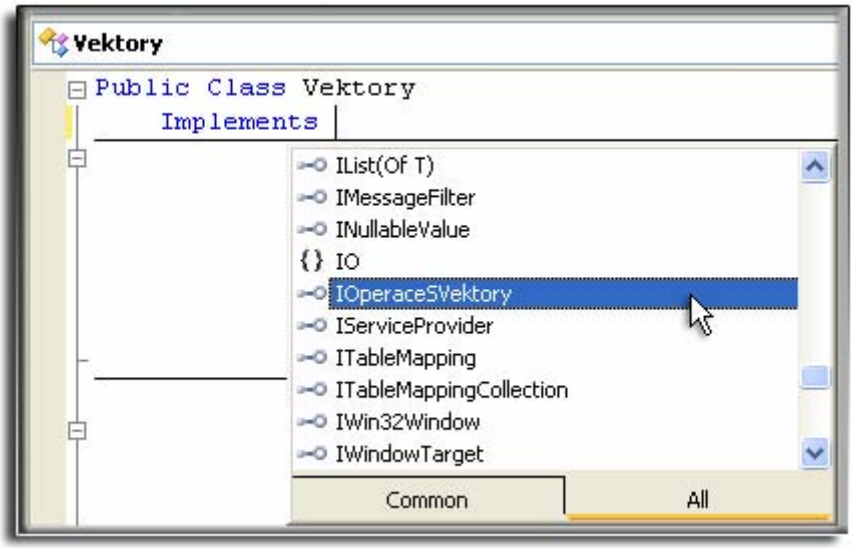
Povězme, že bychom rádi napsali jednoduché rozhraní s názvem `IOperaceSVektory`, které bude deklarovat tři matematické metody pro práci s vektory: `SečístVektory`, `OdečístVektory` a `VynásobitVektorSkalárem`. Rozhraní bude veřejně přístupné, přičemž jeho programový kód se bude nacházet v modulu. Pro potřeby pozdější implementace rozhraní vytvoříme uživatelsky definovanou strukturu `Vektor`, která bude opatřena třemi členy reprezentující souřadnice vektoru (`X`, `Y` a `Z`). Struktura `Vektor` bude uložena v modulu a bude podobně jako rozhraní veřejně přístupná.

```
Public Module Module1
'Definice struktury Vektor.
    Public Structure Vektor
        Dim X As Integer
        Dim Y As Integer
        Dim Z As Integer
    End Structure
'Deklarace rozhraní IOperaceSVektory.
    Public Interface IOperaceSVektory
        Function SečístVektory _
        (ByVal V1 As Vektor, ByVal V2 As Vektor) As Vektor
        Function OdečístVektory _
        (ByVal V1 As Vektor, ByVal V2 As Vektor) As Vektor
        Function VynásobitVektorSkalárem _
        (ByVal Vektor As Vektor, ByVal Skalár As Integer) As Vektor
    End Interface
End Module
```

Název rozhraní začíná písmenem „I“, čímž dodržujeme konvenci pro pojmenovávání rozhraní. V těle rozhraní jsou zapsány deklarace třech členů – ve všech případech se jedná o funkce, jejichž návratovou hodnotou je instance struktury `Vektor`. Tento návrh vychází z předpokladu pozdějšího použití funkcí: každá funkce uskuteční nad vektory jistou operaci a posléze vrátí nový vektor coby získaný produkt.

Rozhraní je sice pěkné, ovšem jeho praktická využitelnost je prozatím nulová. Proto vytvoříme třídu `Vektory`, která bude rozhraní `IOperaceSVektory` implementovat. V procesu implementace se mezi třídou a rozhraním zřizuje vzájemný vztah, na základě kterého je třída povinná uskutečnit definici všech členů, které rozhraní deklaruje. Zmíněnou asociaci mezi třídou a rozhraním v jazyce Visual Basic 2005

vytvoříme pomocí příkazu `Implements`. Tento příkaz se musí nacházet jako první v těle třídy, před veškerým dalším programovým kódem. Jakmile zadáte klíčové slovo `Implements`, technologie IntelliSense vám nabídne seznam dostupných rozhraní, které můžete použít (obr. 5.7).



Obr. 5.7: Příkaz `Implements` a seznam dosažitelných rozhraní, které nabízí technologie IntelliSense

Když provedete výběr požadovaného rozhraní a stiskněte klávesu ENTER, Visual Basic 2005 vygeneruje implementační programové kostry všech členů, které rozhraní deklaruje.

```
Public Class Vektory
    Implements IOperaceSVektory

    Public Function OdečístVektory(ByVal V1 As Module1.Vektor, _
    ByVal V2 As Module1.Vektor) As Module1.Vektor _
    Implements Module1.IOperaceSVektory.OdečístVektory

    End Function

    Public Function SečístVektory(ByVal V1 As Module1.Vektor, _
    ByVal V2 As Module1.Vektor) As Module1.Vektor _
    Implements Module1.IOperaceSVektory.SečístVektory

    End Function

    Public Function VynásobitVektorSkalárem _
    (ByVal Vektor As Module1.Vektor, _
    ByVal Skalár As Integer) As Module1.Vektor _
    Implements Module1.IOperaceSVektory.VynásobitVektorSkalárem

    End Function
End Class
```

Jak můžete vidět, sestavené metody mají stejné signatury jako členy rozhraní. Součástí jednotlivých signatur jsou také příkazy `Implements`, které vizuálně naznačují spojitost mezi vygenerovanými metodami a členy rozhraní. Skvělé je, že Visual Basic 2005 nám dovede ušetřit docela velkou porci času, kterou bychom jinak museli věnovat psaní kódu. V okamžiku, kdy jsou na světě implementační kostry, můžeme je doplnit o funkční zdrojový kód, jenž v naší ukázce vykonává sčítávání, odčítávání a násobení vektorů.

```
Public Class Vektory
    Implements IOperaceSVektory
    Public Function OdečístVektory(ByVal V1 As Module1.Vektor, _
    ByVal V2 As Module1.Vektor) As Module1.Vektor _
    Implements Module1.IOperaceSVektory.OdečístVektory
        Dim V3 As Vektor
        V3.X = V1.X - V2.X
        V3.Y = V1.Y - V2.Y
        V3.Z = V1.Z - V2.Z
        Return V3
    End Function

    Public Function SečístVektory(ByVal V1 As Module1.Vektor, _
    ByVal V2 As Module1.Vektor) As Module1.Vektor _
    Implements Module1.IOperaceSVektory.SečístVektory
        Dim V3 As Vektor
        V3.X = V1.X + V2.X
        V3.Y = V1.Y + V2.Y
        V3.Z = V1.Z + V2.Z
        Return V3
    End Function

    Public Function VynásobitVektorSkalárem _
    (ByVal Vektor As Module1.Vektor, _
    ByVal Skalár As Integer) As Module1.Vektor _
    Implements Module1.IOperaceSVektory.VynásobitVektorSkalárem
        Dim V3 As Vektor
        V3.X = Skalár * Vektor.X
        V3.Y = Skalár * Vektor.Y
        V3.Z = Skalár * Vektor.Z
        Return V3
    End Function
End Class
```

Použití třídy `Vektory` je pak již velice snadné:

```
Dim v1, v2 As Vektor
v1.X = 10 : v1.Y = 20 : v1.Z = 30
v2.X = 30 : v2.Y = 20 : v2.Z = 10
Dim obj As New Vektory
Dim v3 As Vektor = obj.SečístVektory(v1, v2)
MessageBox.Show("Výsledný vektor: " & "V[" & v3.X & _
"," & v3.Y & "," & v3.Z & "].", "Součet vektorů", _
MessageBoxButtons.OK, MessageBoxIcon.Information)
```

Na začátku našeho pojednání o rozhraních jsme si řekli, že jedna třída v jazyce Visual Basic 2005 může využívat schopností většího počtu rozhraní. Budete-li chtít, aby vaše třída implementovala více rozhraní, uveďte je za klíčovým slovem `Implements` (názvy jednotlivých rozhraní přitom oddělte čárkou).

```
'Třída A implementuje dvě rozhraní.
Public Class A
    Implements IRozhraní1, IRozhraní2
'Zbytek kódu třídy byl vynechán.
```

5.8 Polymorfizmus realizovaný pomocí rozhraní

Jestliže dvě různé třídy uskutečňují rozdílnou implementaci jednoho rozhraní, můžeme mluvit o realizaci polymorfizmu založeného na rozhraní. Tato varianta polymorfizmu představuje zpravidla generické rozhraní, které vystavuje všeobecně platné členy. Tyto členy mohou být v různých třídách definovány různě. Tak třeba předpokládejme, že máme rozhraní `IGrafika`, které deklaruje prototyp jedné metody s názvem `VykreslitGrafiku`. Pojmenování této metody je dost generické pro to, aby ji bylo možné implementovat polymorfně. Jedna třída by kupříkladu mohla metodu `VykreslitGrafiku` definovat tak, že po jejím zavolání by byl na plochu formuláře vykreslen barevný přechod. V těle jiné třídy by se zase mohl nacházet kód stejnojmenné metody, který by na určenou pozici formuláře vykreslil obraz uživatelsky zadaného grafického souboru s extenzí .png.

```
'Podoba rozhraní IGrafika.
Public Interface IGrafika
    Sub VykreslitGrafiku()
End Interface

'Třída A implementuje metodu VykreslitGrafiku rozhraní IGrafika.
Public Class A
    Implements IGrafika
    Public Sub VykreslitGrafiku() _
    Implements Module1.IGrafika.VykreslitGrafiku
        Dim NovýFormulář As New Form
        NovýFormulář.Text = "Polymorfizmus pomocí rozhraní"
        Dim Graf_Objekt As Graphics = NovýFormulář.CreateGraphics
        Dim Graf_Štětěc As New _
        System.Drawing.Drawing2D.LinearGradientBrush _
        (Graf_Objekt.VisibleClipBounds, Color.Red, Color.Blue, _
        Drawing2D.LinearGradientMode.ForwardDiagonal)
        NovýFormulář.Show()
        Graf_Objekt.FillRectangle(Graf_Štětěc, _
        Graf_Objekt.VisibleClipBounds)
        Graf_Objekt.Dispose()
        Graf_Štětěc.Dispose()
    End Sub
End Class
```

Metoda `VykreslitGrafiku` třídy `A` provádí tyto hlavní činnosti:

1. Vytváří nový formulář (objekt třídy `Form`).
2. Volá metodu `CreateGraphics` formuláře, která vzápětí sestrojuje nový grafický objekt (objekt třídy `Graphics` z jmenného prostoru `System.Drawing`). Grafický objekt je spjat s formulářem a dovoluje na jeho ploše realizovat rozmanité grafické operace.
3. Vytváří nový grafický štětec pro vykreslení barevného přechodu pomocí lineárního gradientu (objekt třídy `LinearGradientBrush` z jmenného prostoru `System.Drawing.Drawing2D`). Přetížený konstruktor třídy přijímá v naší ukázce čtyři argumenty:
    - obdélníkový region plochy, na který se bude kreslit,
    - počáteční barvu lineárního gradientu,
    - koncovou barvu lineárního gradientu,
    - orientaci gradientu (od této orientace závisí poloha počátečního a koncového bodu gradientu).
4. Zobrazuje nový formulář nemodálně pomocí instanční metody `Show`.

5. Používá grafický objekt a grafický štětec pro vykreslení lineárního gradientu.
6. Uvolňuje alokované zdroje grafického objektu a grafického štětce voláním jejich metod `Dispose`.

Jakmile je třída `A` implementující rozhraní `IGrafika` zhotovena, můžeme založit její instanci a aktivovat polymorfní metodu `VykreslitGrafiku`.

```
Dim obj As New A
obj.VykreslitGrafiku()
```

Výsledek je znázorněn na obr. 5.8.

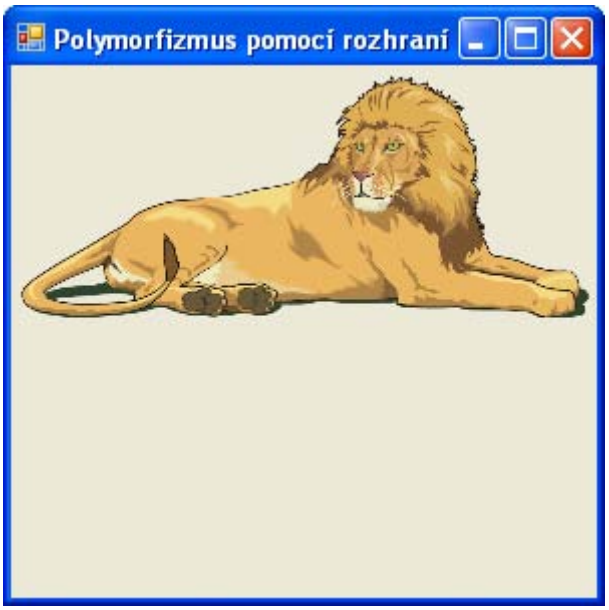
Obr. 5.8: Výsledek volání polymorfní metody `VykreslitGrafiku` rozhraní `IGrafika`

Abychom demonstrovali možnosti aplikace polymorfizmu prostřednictvím rozhraní, zkonstruujeme další třídu, tentokrát třídu `B`. Rovněž třída `B` bude implementovat rozhraní `IGrafika`, z čehož pro ni plyne povinnost sestavit definici metody `VykreslitGrafiku`. Samotné implementační detaily se ovšem budou lišit: metoda `VykreslitGrafiku` třídy `B` bude na předem dané pozici formuláře zobrazovat obsah uživatelsky zadaného grafického souboru .png.

```
'Třída B také implementuje metodu VykreslitGrafiku rozhraní IGrafika.
Public Class B
    Implements IGrafika
    Public Sub VykreslitGrafiku() _
    Implements Module1.IGrafika.VykreslitGrafiku
        Dim DialogOtevřít As New OpenFileDialog
        DialogOtevřít.Filter = "Grafické soubory PNG (*.png)|*.png"
        If DialogOtevřít.ShowDialog() = DialogResult.OK Then
            If DialogOtevřít.FileName <> "" Then
                Dim Bitmapa As New Bitmap(DialogOtevřít.FileName)
                Dim NovýFormulář As New Form
                NovýFormulář.Text = "Polymorfizmus pomocí rozhraní"
                Dim Graf_Objekt As Graphics = NovýFormulář.CreateGraphics
                NovýFormulář.Show()
                Graf_Objekt.DrawImage(Bitmapa, New Point(5, 5))
                Graf_Objekt.Dispose()
                Bitmapa.Dispose()
            Else
```

```
                Exit Sub
            End If
        End If
    End Sub
End Class
```

Programový kód metody `VykreslitGrafiku` třídy `B` vám bude až na několik málo drobností povědomý. Je to způsobeno tím, že také tato metoda pro realizaci grafických operací využíva objekt třídy `Graphics`. Nejprve je ale nutno získat od uživatele specifikaci grafického souboru. Jelikož chceme, aby byl program uživatelsky přívětivý, používáme standardní dialog pro otevření souboru, který je vám jistě dobře známý. V jazyce Visual Basic 6.0 jste mohli pracovat s komponentou `CommonDialog`. Jejím prostřednictvím bylo možné zobrazovat a samozřejmě také programovat standardní dialogová okna systému Microsoft Windows pro otevírání a ukládání souborů (ale také pro zobrazování nápovědy, realizaci tiskových operací či provádění výběru barvy z barevné palety). Novinkou Visual Basicu 2005 je, že v soupravě nástrojů **Toolbox** se nyní nenachází jenom jedna, ale hned několik komponent pro vyvolávání standardních dialogových oken. Monolitní funkcionalita komponenty `CommonDialog` tak byla „roztříštěna" do několika samostatně působících komponent, mezi něž patří `OpenFileDialog`, `SaveFileDialog`, `ColorDialog` či `PrintDialog`. Naše ukázka zakládá novou instanci třídy `OpenFileDialog` z jmenného prostoru `System.Windows.Forms`. Tato instance představuje standardní dialog pro otevření souboru. Poněvadž chceme, aby se v dialogu objevovaly pouze grafické soubory s příponou .png, používáme vlastnost `Filter`. Dialog zobrazíme zavoláním metody `ShowDialog`. Návratovou hodnotou této metody je člen enumeračního typu `DialogResult`, jenž označuje tlačítko, které uživatel stiskl. Nás zajímá aktivace tlačítka **OK**: je-li toto tlačítko stisknuto, testujeme hodnotu vlastnosti `FileName`, která uchovává jméno, respektive cestu k zvolenému souboru s grafickým obsahem. Pokud byl vybrán nějaký soubor, vytváříme novou instanci třídy `Bitmap` reprezentující paměťový obraz grafického souboru. S vykreslením grafických dat na plochu formuláře nám pomůže metoda `DrawImage` grafického objektu. Metoda `DrawImage` je přetížena, ovšem nejde o ledajaké přetížení. Tato metoda se totiž vyskytuje v celkem třiceti exemplářích! Náš fragment kódu volá tu přetíženou verzi, která pracuje se dvěma parametry. Prvním je odkaz na instanci třídy `Image`, zatímco druhým je instance struktury `Point` udávající přesnou pozici levého horního rohu obrázku na ploše formuláře. Jestliže se lépe podíváte na programový kód, zjistíte, že metodě `DrawImage` ve skutečnosti předáváme odkaz na instanci třídy `Bitmap` a ne `Image`. Tato implikace je povolena, protože třída `Bitmap` je přímým potomkem třídy `Image`. Když jsou kvanta grafických dat přetransformována do obrazovkových pixelů, dochází k uvolnění alokovaných zdrojů pomocí metody `Dispose`. Ilustraci vykreslení bitové mapy přináší obr. 5.9.



Obr. 5.9: Polymorfní metoda `VykreslitGrafiku` třídy `B` si dokáže poradit se zobrazním obsahu grafického souboru .png

Kapitola 6.: Visual Basic 2005 Express

Rok 2005 je pro společnost Microsoft výjimečný hned dvakrát – vedle inovovaného vývojářského nástroje Visual Basic 2005 se na světlo světa dostává také jeho „mladší bráška", kterému sudičky do názvu přidali slovíčko „Express". Softwarový produkt Visual Basic 2005 Express je zcela novým počinem v marketingové strategii společnosti Microsoft. Expresní vydání Visual Basicu 2005 představuje „odlehčenou" verzi tohoto programovacího jazyka, která je v první řadě zaměřena na neprofesionální programátory, hobby vývojáře, studenty informačních technologií a všechny další uživatele, kterým dělá programování radost. Visual Basic 2005 Express je tedy jakýmsi doplňkem „velikého" Visual Basicu. Visual Basic 2005 Express ovšem nepůsobí osamoceně. Ve skutečnosti je součástí nové produktové řady, kterou kromě něho tvoří ještě další Express nástroje. Jedná se o Visual C# 2005 Express, Visual C++ 2005 Express, Visual J# 2005 Express a Visual Web Developer 2005 Express. Všechny Express nástroje mají několik společných vlastností:

1. Nabízejí jednoduchou a rychlou cestu pro vytváření nejběžnějších softwarových aplikací běžících v prostředí vývojově-exekuční platformy Microsoft .NET Framework 2.0.

2. Obsahují úplnou implementaci syntaktické specifikace příslušných programovacích jazyků. To znamená, že programátoři používající produkt Visual Basic 2005 Express mohou využívat všechny syntaktické novinky a sémantické inovace programovacího jazyka Visual Basic 2005.

3. Dovolují programátorům zcela neomezeně přistupovat do společné knihovny tříd platformy .NET Framework 2.0. Díky tomu mohou vývojáři využívat dovedností několika tisíc řízených tříd, které zapouzdřují programovou funkcionalitu na vysoké úrovni.

4. Součástí dodávky Express nástrojů tvoří sada ukázkových projektových šablon, respektive řešení, jejichž pomocí se začátečníci dovedou snadněji seznámit s novým programovacím jazykem a integrovaným vývojovým prostředím. Tyto takzvané „Starter Kits" jsou velice vítaným zpestřením, neboť jsou zaměřeny vysloveně prakticky a důsledně tak naplňují koncepci „školy hrou". Visual Basic 2005 Express přichází se dvěma starter kity: zatímco jeden předvádí postup, jak si vytvořit program pro správu domácích DVD disků, další vás naučí sestrojit plnohodnotný spořič obrazovky. Oba ukázkové projekty jsou doplněny přehlednou dokumentací, která v graficky přívětivé podobě vysvětluje všechny důležité principy, na kterých ta-která aplikace pracuje.

5. Jelikož primárním cílovým publikem nástrojů patřících do produktové řady Express jsou počítačoví nadšenci a IT fandové, je jejich celkové pojetí výrazně jednodušší, přehlednější a přímočařejší. Nestřetnete se tedy s žádnými komplikovanými nástroji, složitými konfiguračními volbami či jinými součástmi, nad nimiž sice zaplesá srdce profesionála, ovšem v očích běžného uživatele jde zpravidla o „španělskou vesnici". Visual Basic 2005 Express se snaží oprostit od všeho, co by mohlo začátečníkům nebo migrujícím vývojářům způsobovat potíže. Samozřejmě, ne všechny programové prvky mohou být vypuštěny nebo zjednodušeny. Visual Basic 2005 s přídomkem Express je pořád vývojářský produkt a ne aplikace typu Microsoft Word.

6. Na závěr snad nejlepší zpráva: podle plánů společnosti Microsoft budou všechny Express nástroje poskytovány zcela zdarma. Ve formě kapacitně nepříliš náročných instalačních balíčků (řádově pár desítek MB) bude produkty s logem Express možné bezplatně stáhnout z webových stránek Microsoftu.

Jelikož je tato vývojářská příručka zacílena primárně na profesionální vývojáře migrující z jazyka Visual Basic 6.0, mohli byste se ptát, proč zde mluvíme o Visual Basicu 2005 Express, když jde evidentně o jednodušší verzi Visual Basicu 2005. Odpověď zní: právě proto. Domnívám se, že expresní edice Visual Basicu 2005 dokáže být velikým pomocníkem vývojáře, který projeví zájem a rozhodne se přiučit novému stylu vývoje moderních aplikací .NET. Důvodů, proč sáhnout po produktu Visual Basic 2005 Express je hned několik:

1. **Minimální pořizovací náklady.** Pokud byste se rádi seznámili s filozofií návrhu a vývoje aplikací v prozatím nejaktuálnější verzi programovacího jazyka Visual Basic, produkt Visual Basic 2005 Express je pro vás to pravé. Náklady na jeho pořízení jsou opravdu nízké: můžete jej získat buď bezplatně z webových stránek společnosti Microsoft, anebo za malý poplatek na kompaktním disku. Jistě uznáte, že za těchto podmínek stojí Visual Basic 2005 Express bezesporu přinejmenším alespoň za vyzkoušení.

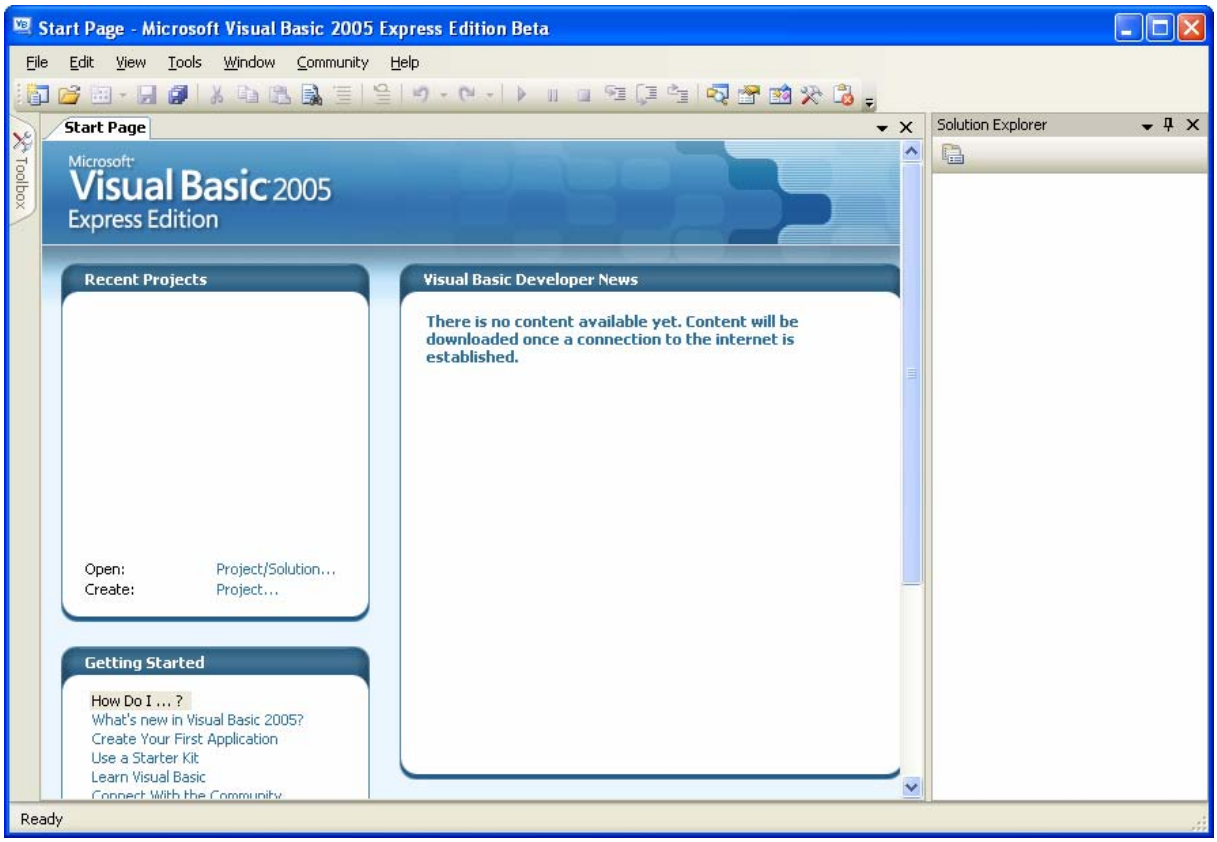
2. **Plná syntaktická podpora jazyka Visual Basic 2005.** Již jsme o ní mluvili, ovšem jedná se o tak významnou vlastnost, že stojí za to ji rozvést. Visual Basic 2005 Express v celé šíři podporuje novou jazykovou specifikaci Visual Basicu 2005. V tomto směru jde o ryzí Visual Basic 2005: všechny relevantní programovací příkazy, struktury, konstrukce, příkazy a klíčová slova máte neustále na dosah ruky. Vše si tak můžete pohodlně vyzkoušet.

3. **Přímá cesta k Visual Basicu 2005.** Poté, co zvládnete Visual Basic 2005 Express, můžete bez jakýchkoliv potíží migrovat do sofistikovanějšího prostředí produktu Visual Basic 2005. Tento již pochopitelně není poskytován zdarma, ovšem na druhou stranu, jeho součástí jsou profesionální nástroje, bez nichž se skutečný vývojář při své náročné práci neobejde. Visual Basic 2005 není samostatně prodávaným produktem, ale je integrován v komplexní sadě Visual Studio 2005.

6.1 Visual Basic 2005 Express – Instalace a první spuštění

Instalace produktu Visual Basic 2005 Express je velice jednoduchá. Jakmile je instalační balíček stažen z celosvětové sítě, můžete zahájit instalační proces. Ten je ve srovnání s instalací Visual Basicu 2005 daleko kratší a přehlednější. Poté, co je instalace úspěšně ukončena, můžete Visual Basic 2005 po prvýkrát spustit. Při prvním spuštění budete informováni o tom, že nainstalovaný produkt je nutno během 30 dnů aktivovat. Aktivace je elektronická a realizuje se připojením na server společnosti Microsoft, ověřením produktu a přidělením uživatelského aktivačního klíče. Je-li produkt Visual Basic 2005 Express aktivován, lze jej začít používat.

Když Visual Basic 2005 Express spustíte prvně, přivítá vás úvodní stránka **Start Page** (obr. 6.1).

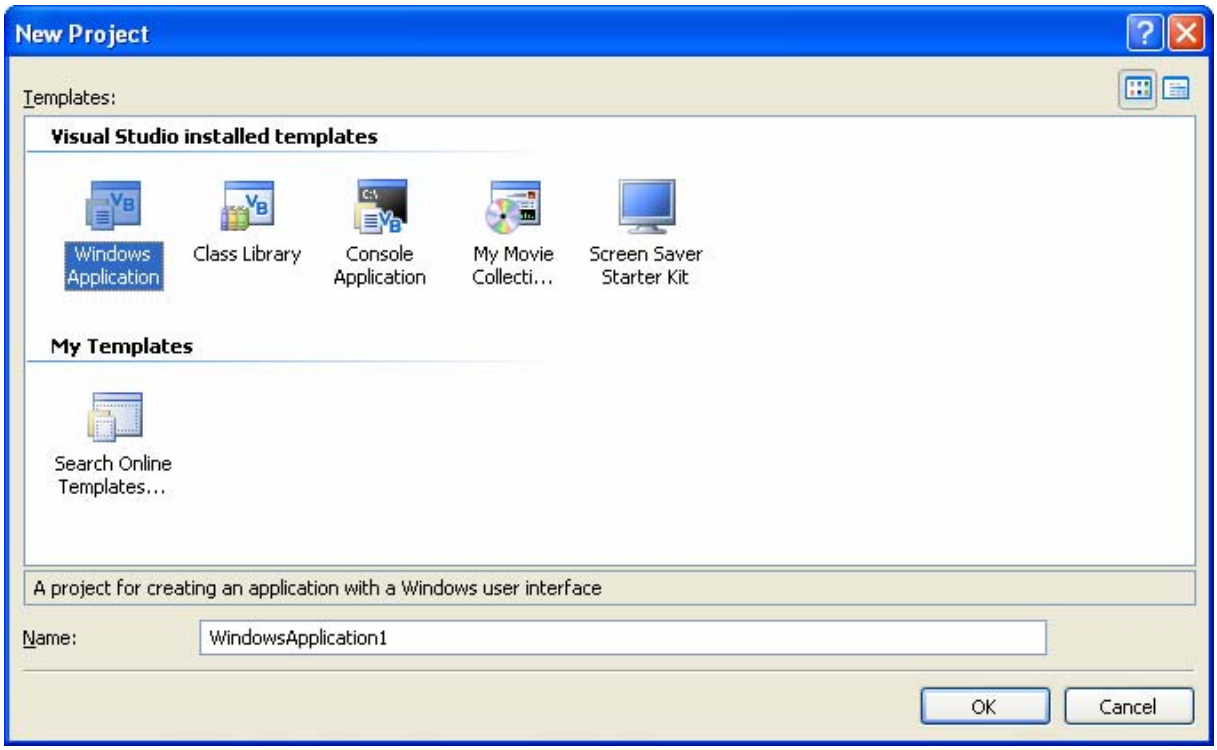


Obr. 6.1: Visual Basic 2005 Express a úvodní stránka **Start Page**

Úvodní obrazovka produktu Visual Basic 2005 Express je stejná jako její protějšek z Visual Basicu 2005. V podokně **Recent Project** se zobrazují názvy posledně otevřených aplikačních projektů, a také se zde nacházejí hypertextové odkazy pro otevření již existujícího projektu či vytvoření projektu zcela nového. Informace o jazyce Visual Basic 2005 Express můžete najít ve spodním podokně **Getting Started**. Nepřehlédnutelnou část úvodní obrazovky vyplňuje podokno **Visual Basic Developer News**, které slouží pro zobrazování zpráv a novinek z oblasti programování aplikací a vývoje softwaru v jazyce Visual Basic 2005 Express. Jste-li připojeni k celosvětové síti, můžete v tomto podokně nalézt mnoho zajímavých informací.

6.2 Spektrum projektů Visual Basicu 2005 Express

Visual Basic 2005 Express nenabízí tolik aplikačních projektů jako jeho „velký bratr". O této skutečnosti se přesvědčíte krátce poté, co otevřete nabídku **File** a klepnete na položku **New Project** (obr. 6.2).

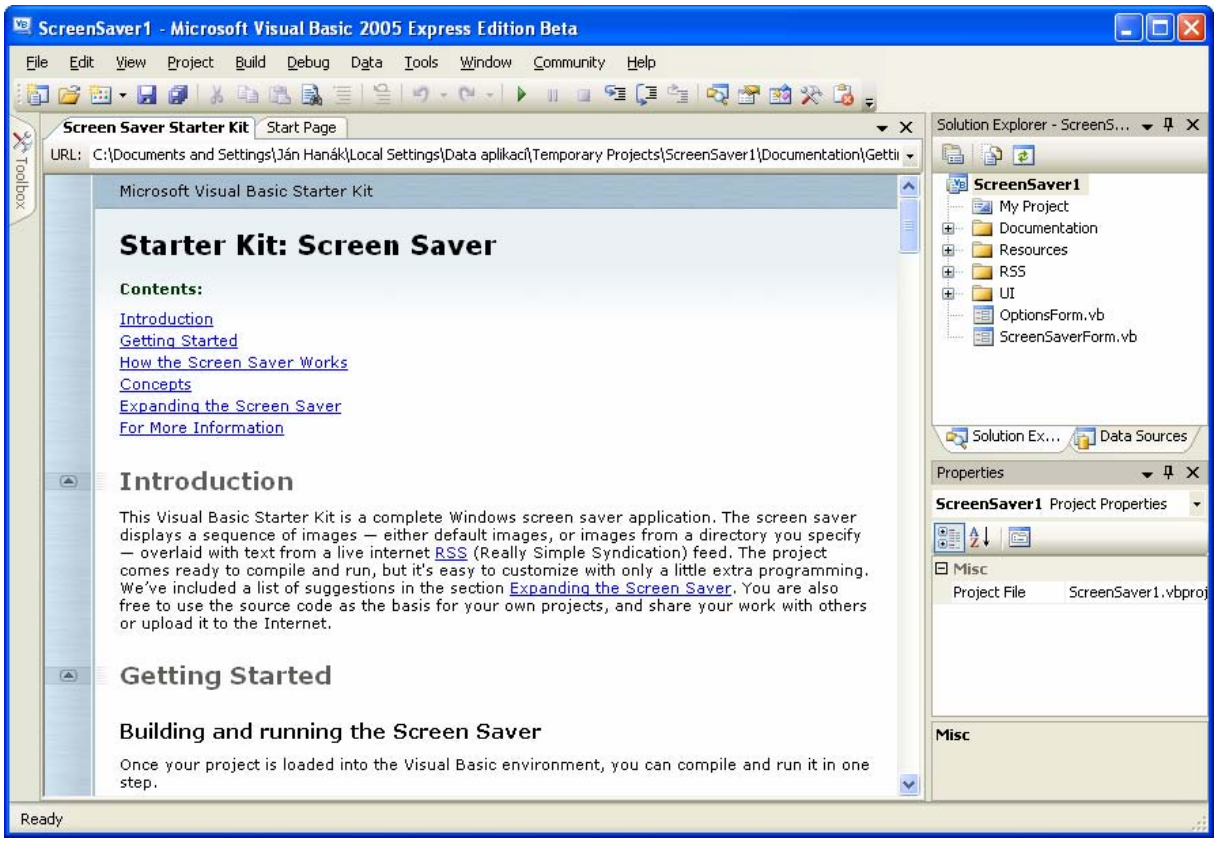


Obr. 6.2: Přehled aplikačních projektů ve Visual Basicu 2005 Express

S pomocí Visual Basicu 2005 Express mohou programátoři vyvíjet pouze tři typy projektů:

Windows Application	Standardní aplikace pro systém Windows (šablona **Windows Application**)
Class Library	Knihovny tříd (šablona **Class Library**)
Console Application	Konzolové aplikace (šablona **Console Application**)

Vyjma uvedených projektových šablon se v dialogovém okně **New Project** nacházejí také dvě šablony starter kitů s názvy **My Collection Starter Kit** a **Screen Saver Starter Kit**. Jelikož právě ukázkové projektové šablony jsou novinkou expresního Visual Basicu 2005, podívejme se na ně blíže. Zvolme tedy šablonu **Screen Saver Starter Kit** a klepněme na tlačítko **OK**. Visual Basic 2005 Express vygeneruje nový projekt, jenž implicitně pojmenuje jako **ScreenSaver1**. Do nově sestrojeného projektu posléze doplní všechny nezbytné součásti, které tvoří plně funkční jádro aplikace .NET. Rovněž otevře informační dialog (s textem **Screen Saver Starter Kit**), v němž jsou shrnuty informace týkající se základní koncepce projektu (obr. 6.3).



Obr. 6.3: Informační dialog popisující ukázkový projekt **Screen Saver Starter Kit**

Skvělé je, že programový kód ukázkového projektu s obrazovkovým spořičem můžete ihned zkompilovat a spustit. Když tak učiníte, spustí se spořič, který bude zobrazovat sérii obrázků (obr. 6.4).

Obr. 6.4: Spořič obrazovky v akci

6.3 Porovnání produktů Visual Basic 2005 Express a Visual Basic 2005

Ačkoliv Visual Basic 2005 kráčí po stopách svéh
o robustnějšího kolegy, v mnoha ohledech se tyto dva vývojářské nástroje od sebe zásadně liší. V čem tkví nejmarkantnější rozdíly?

V čem se Visual Basic 2005 Express liší od Visual Basicu 2005:

Absence některých projektových šablon.

Visual Basic 2005 Express dovoluje vývojářům pracovat pouze se třemi základními projektovými šablonami, jejichž pomocí lze vytvářet standardní aplikace pro systém Windows, knihovny tříd a konzolové aplikace. Ty sice dokáží uspokojit počáteční potřeby migrujících programátorů, ovšem omezenost v počtu projektových šablon se dříve nebo později začne jevit jako velice svazující prvek. Jestliže budete chtít vytvářet služby pro systém Windows, knihovny ovládacích prvků, mobilní aplikace či řešení pro platformu Microsoft Office System, budete muset sáhnout po Visual Basicu 2005. Visual Basic 2005 Express budete nuceni opustit také tehdy, kdy budete chtít vyvinout svou vlastní webovou aplikaci pracující pod křídly technologie ASP.NET 2.0.
V tomto směru však máte na výběr: buď vsadíte na Visual Basic 2005, anebo vyzkoušíte další produkt z řady Express – Visual Web Developer 2005.

Absence vizuálního návrháře tříd (Class Designer).

V prostředí Visual Basicu 2005 Express není implementován nástroj **Class Designer**, který hraje roli užitečného pomocníka v procesu návrhu tříd a designování vzájemných vztahů mezi nimi. Vývojáři v jazyce Visual Basic 2005 Express proto nemají k dispozici nástroj, jehož prostřednictvím by mohli vizuálním způsobem navrhovat architektonickou strukturu tříd. Nicméně nepřítomnost vizuálního návrháře tříd není zas až tak tristní – nelze totiž předpokládat, že začínající programátoři v jazyce Visual Basic 2005 budou pro svou práci potřebovat navrhovat třídy.

Absence okna Class View a nástroje Object Test Bench pro vizuální práci s objekty.

Okno **Class View** dovoluje prohledávat jmenné prostory a třídy aplikačního projektu. Pomocí nástroje **Object Test Bench** je zase možné vizuálně testovat objekty tříd. Ani jeden z těchto důmyslných elementů není začleněn do integrovaného vývojového prostředí produktu Visual Basic 2005 Express.

Nemožnost zaznamenávat a psát makra pro IDE.

Makra jsou inteligentní fragmenty programového kódu, které dovolují automatizovat rutinně prováděné činnosti uvnitř integrovaného vývojového prostředí Visual Studia 2005. Makra se píší v jazyce Visual Basic 2005 a jejich spojení s IDE je až neskutečně silné: pomocí maker lze ovládat takřka všechny součásti vývojového prostředí. Visual Basic 2005 Express vytváření a programování maker nepodporuje.

Nemožnost automatizovaného testování.

Visual Basic 2005 Express neumožňuje uskutečňovat automatizované testovaní funkčních aplikačních bloků.

V čem je Visual Basic 2005 Express stejný jako Visual Basic 2005:

1	**Implementovaná jazyková specifikace**
2	**Návaznost na společnou knihovnu tříd vývojově-exekuční platformy Microsoft .NET Framework 2.0**
3	**Integrované vývojové prostředí (IDE)**
4	**Vestavěné ovládací prvky a komponenty**
5	**Distribuce aplikací .NET pomocí technologie ClickOnce**

Závěr

Vážení vývojáři, programátoři a IT specialisté,

právě jste dočetli migrační vývojářskou příručku *VB 6.0 → VB 2005: Přecházíme z jazyka Visual Basic 6.0 na jazyk Visual Basic 2005*. Cílem této publikace bylo obeznámit vás s novou verzí programovacího jazyka Visual Basic 2005 a vývojově-exekuční platformou Microsoft .NET Framework 2.0. Ačkoliv příručka není zrovna nejútlejší, o migračním procesu a jeho jemných aspektech bychom mohli napsat ještě mnoho desítek ne-li stovek stránek. Přechod z jedné verze vývojářského nástroje na jinou není nikdy docela snadný, zejména tehdy, je-li aktuálnější verze zcela přepracovaná a plná inovací, s jimiž je nutno se podrobně obeznámit. Visual Basic 2005 je právě takový: moderní programovací jazyk, v němž mohou programátoři psát moderní aplikace .NET pro 21. století. Samozřejmě, zvládnout nový jazyk a jeho zákoutí si vyžaduje mnoho energie a především času. Avšak jsem přesvědčen o tom, že jakmile vplujete do vod nového Visual Basicu, již se vám nebude chtít jít zpátky. Přeji vám proto mnoho úspěchů a hodně skvělých softwarových aplikací napsaných v nejpopulárnějším programovacím jazyce v naší sluneční soustavě.

O autorovi

Ing. Ján Hanák vystudoval Obchodní fakultu Ekonomické univerzity v Bratislavě. Působí jako programátor, vývojář, technický konzultant a spisovatel, přičemž se specializuje na vývoj širokého spektra počítačových aplikací. Při své práci využívá zejména tři programovací jazyky, a sice Visual Basic, C# a C++. V roce 2004 napsal publikaci *Visual Basic .NET 2003 – Začínáme programovat*, kterou vydalo nakladatelství Grada Publishing. Je také autorem překladu vývojářské brožury „*Přecházíme na platformu Microsoft .NET*" pro společnost Microsoft ČR. Svá důmyslná řešení publikoval také v časopisech *CHIP*, *PC REVUE a INFOWARE*. Rovněž je stálým přispěvatelem počítačové přílohy deníku SME s názvem „*SME v počítačích*".

Texty neprošly jazykovou úpravou.

Vydal:
Microsoft s.r.o., BB Centrum, budova Alpha, Vyskočilova 1461/2a, 140 00 Praha 4
tel.: +420-261 197 111 , fax: +420-261 197 100, http://www.microsoft.com/cze