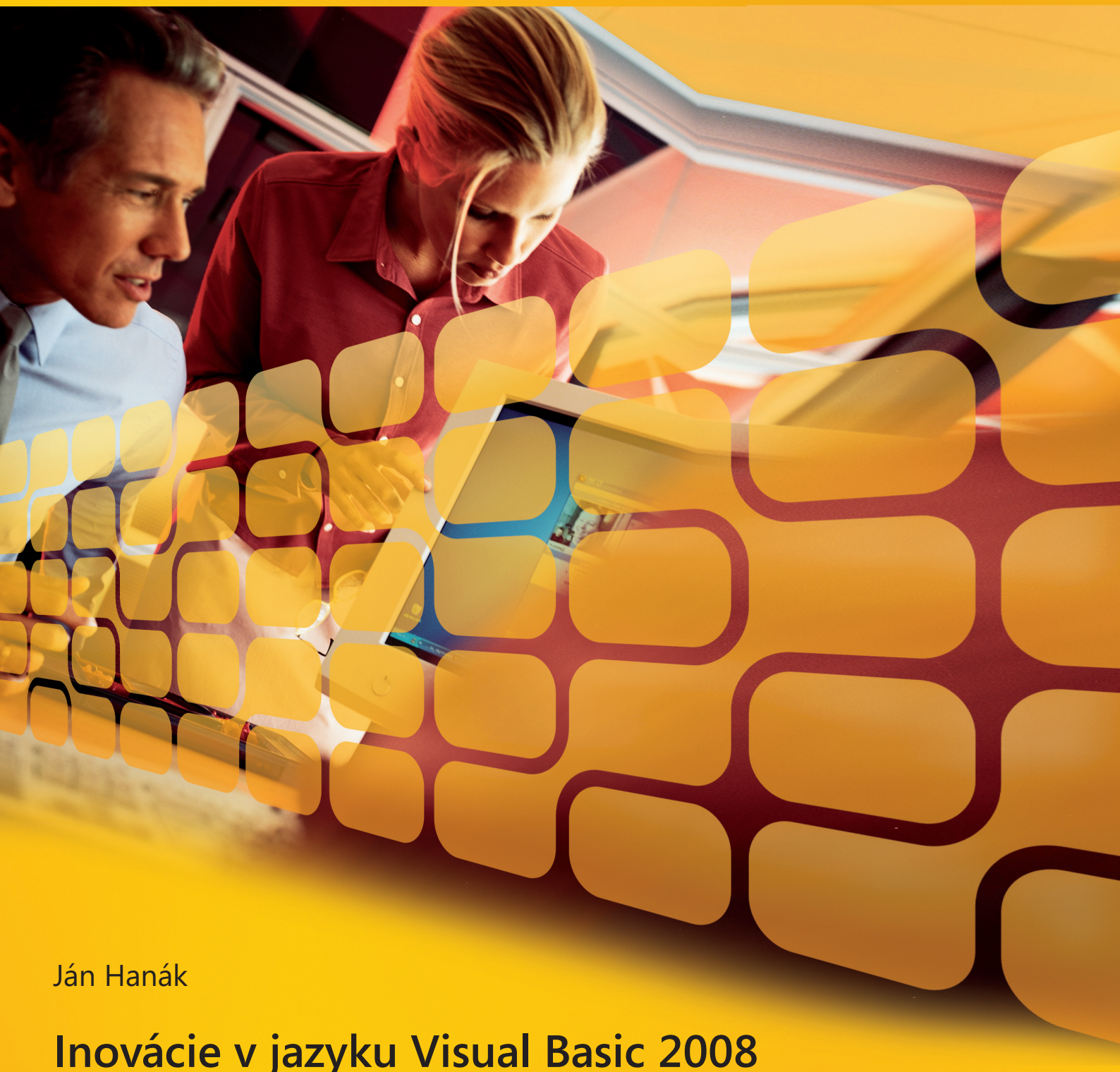
Microsoft



Ján Hanák

Inovácie v jazyku Visual Basic 2008

Príručka pre vývojárov, programátorov a softvérových expertov

Microsoft® Visual Studio® 2008

Ján Hanák

Inovácie v jazyku Visual Basic 2008

(Príručka pre vývojárov, programátorov
a softvérových expertov)

Microsoft
2008

Obsah

Úvod

Vážení priatelia,

dovoľte nám, aby sme vás s radosťou a nadšením privítali vo vzrušujúcom svete jazyka Visual Basic! Možno to bude znieť neuveriteľne, no vôbec prvá verzia tohto programovacieho jazyka sa medzi programátorov, vývojárov a softvérových odborníkov dostala už v roku 1991. Odvtedy ušiel náš obľúbený „bejzik" naozaj úctyhodnú cestu. Ak sa ohliadneme späť, v mysliach sa nám začnú okamžite vynárať všetky tie nezabudnuteľné momenty, ktoré sme s jazykom Visual Basic prežili. Vedno s ním sme objavili pôvab tvorby softvérových aplikácií s bohatým grafickým používateľským rozhraním, ktoré boli projektované presne podľa smerníc daných metodológiou rýchleho vývoja softvéru (RAD, Rapid Application Development). Hoci sme najskôr začínali programovaním rýdzo 16-bitových aplikácií, počnúc verziou 4.0 sme dostali možnosť prvýkrát nahliadnuť do plne 32-bitového prostredia. Počas nášho odborného a profesionálneho rastu sme sa zoznámili s množstvom rozmanitých technológií. Ako šiel čas, postupne sme sa naučili, čo je to Win32 API, zistili sme, ako fungujú ovládacie prvky a komponenty, dozvedeli sme sa, aké konkurenčné výhody prináša priemyselná produkcia softvérových jednotiek (COM, COM+ a ActiveX). Mimo náš záujem nezostala ani práca s databázami (DAO/ADO), automatizácia externých aplikácií či vytváranie programov pre nasadenie v distribuovanom prostredí internetu.

Napriek tomu, že aj v súčasnej dobe existuje nemalá skupina vývojárov, ktorí pri svojej tvorivej činnosti používajú Visual Basic 6.0, musíme podotknúť, že pre ďalšie smerovanie tohto programovacieho prostriedku bol najvýznamnejší rok 2002. Vtedy sa totiž Visual Basic prehupol do svojej „.NET" verzie. Áno, aj my sme sa stretli s názormi, ktoré hlásali, že po príchode vývojovo-exekučnej platformy Microsoft .NET Framework sa Visual Basic zmenil na nepoznanie a že to už nikdy nebude ten „starý a dobrý" Visual Basic. Len čo sa však autori spomenutých vyhlásení bližšie oboznámili s novinkami zapracovanými do programovacieho jazyka Visual Basic .NET a integrovaného vývojového prostredia produktu Visual Studio .NET, zrazu aj oni boli nútení modifikovať svoje pôvodné tvrdenia. Veľmi rýchlo totiž zistili, že „dotnetový" Visual Basic je majstrovsky skonštruovaným hybridným programovacím jazykom, ktorý do posledných detailov implementuje všetky hlavné piliere štruktúrovaného, objektovo orientovaného a komponentového programovania. Aj keď proces prechodu na verziu .NET nebol najmä z nižších edícií jazyka Visual Basic dokonale hladký a priamočiary, vynaložené úsilie sa nám vrátilo aj s pomyselnými úrokmi. Vo chvíli, keď sme si zvykli na všetky zapracované vylepšenia a inovácie, ktoré znásobili našu pracovnú produktivitu, ani vo sne nám nenapadlo, že by sme sa mali vrátiť k tomu „starému a dobrému" jazyku Visual Basic.

Technologický rozmach v oblasti tvorby počítačového softvéru a v informatike všeobecne sa pochopiteľne nezastavil s uvedením jazyka Visual Basic .NET. V ďalšom evolučnom reťazci môžeme identifikovať edíciu s prívlastkom .NET 2003, ktorá okrem minoritných syntakticko-sémantických doplnkov priniesla zvlášť cenenú integrovanú podporu pre programovanie aplikácií .NET zacielených na inteligentné mobilné zariadenia. Istotne rigoróznejší inovatívny progres sme zaznamenali koncom roka 2005, kedy sa naším novým favoritom stal Visual Basic 2005. Generické dátové typy, neznamienkové a nulovateľné dátové typy, neúplné (parciálne)

dátové typy, preťažovanie operátorov, používateľské udalosti, príkaz Using a vlastnosti s odlišnou úrovňou prístupu – to je zoznam primárnych inovácií vyššieho rádu, ktoré boli starostlivo zabudované do jazyka špecifikácie Visual Basic 2005.

V dobách, keď sme vytvárali programy v jazyku Visual Basic 2005, panovalo u nás jasné presvedčenie o tom, že lepší snáď už ani ten Basic nemôže byť. Veď sme mali všetko, čo sme potrebovali, ba tu a tam, ešte aj niečo navyše. Keď píšeme v textovom procesore tieto riadky, v pozadí operačného systému tichučko beží relácia Visual Basic 2008 a my sa opäť nemôžeme zbaviť dojmu, že novému jazyku Visual Basic sa zase podarilo predbehnúť svojich predchodcov prinajmenšom o niekoľko svetelných rokov!

Visual Basic 2008 už nie je len programovacím jazykom, stáva sa integrálnou súčasťou životného štýlu moderných a progresívnych ľudí, ktorí tvoria prvotriedne softvérové aplikácie schopné zmeniť tento svet k lepšiemu. Budeme radi, ak sa k nám pridáte.

Autor
Praha, jún 2008

Pre koho je táto kniha určená

Táto kniha sa zameriava na segment pokročilých a profesionálnych vývojárov, ktorí by sa radi zoznámili so syntakticko-sémantickými inováciami, vylepšeniami a modifikáciami, ktoré prináša programovací jazyk Visual Basic 2008. Publikácia predpokladá, že doposiaľ ste pracovali s niektorou z nasledujúcich verzií jazyka Visual Basic: Visual Basic .NET, Visual Basic .NET 2003 a Visual Basic 2005. Ak patríte k programátorom používajúcim jazyk Visual Basic 6.0, odporúčame vám, aby ste pred štúdiom tejto knihy siahli po inej publikácii, ktorá sa venuje základom programovania v niektorej z vyššie uvedených .NET-kompatibilných verzií jazyka Visual Basic. Vezmite prosím na vedomie, že cieľom tejto knihy nie je vysvetľovať základné princípy objektovo orientovaného programovania v jazyku Visual Basic 2008. Jednotlivé kapitoly obsahovej štruktúry diela sú koncipované ako prehľad hlavných technologických inovácií zapracovaných do jazyka Visual Basic 2008. Výklad kladie mimoriadny dôraz na vysokú technickú kvalitu, terminologickú exaktnosť a vedecké spracovanie problematiky, čím napĺňa koncepciu primeraného študijného zaťaženia čitateľov.

Obsahová štruktúra knihy

Kniha **Inovácie v jazyku Visual Basic 2008** je zložená z nasledujúcej kolekcie kapitol:

- Kapitola 1: Typová inferencia lokálnych entít.
- Kapitola 2: Inštanciácia pomenovaných a anonymných tried pomocou inicializátorov uvádzaných v inicializačných zoznamoch.
- Kapitola 3: Rozširujúce metódy.
- Kapitola 4: λ-výrazy a λ-kalkul.
- Kapitola 5: Kovariancia a kontravariancia delegátov.
- Kapitola 6: XML konštanty a XML konštanty s pridruženými programovými výrazmi.
- Kapitola 7: LINQ (Language Integrated Query): Unifikovaný dopytovací jazyk.

Čitateľom, ktorí nemajú doposiaľ žiadne skúsenosti s novými programovými rysmi jazyka Visual Basic 2008, odporúčame, aby aplikovali tzv. inkrementálny režim štúdia. To znamená postupné zvládanie kapitol v stanovenom poradí, od úvodnej až po záverečnú. Postupnosť kapitol totiž reflektuje našu snahu, ktorej cieľom je segmentácia výkladu v adekvátnych dávkach garantujúcich generovanie optimálneho grafu krivky predstavujúcej dobu osvojenia istej syntakticko-sémantickej inovácie jazyka Visual Basic 2008.

Okrem úlohy sprievodcu po hlavných inováciách jazyka Visual Basic 2008 dokáže kniha vystupovať aj ako referenčná príručka. Tento štýl práce s knihou bude vyhovovať najmä čitateľom, ktorí uprednostňujú tzv. selektívny režim štúdia. Pri ňom je na čitateľovi, aby si vybral kapitoly, ktoré sa stanú centrom jeho záujmu. Selektívny režim štúdia možno uplatniť v prípade, ak čitateľ nie je úplným nováčikom v oblasti inovácií jazyka Visual Basic 2008. V tejto súvislosti si dovoľujeme pripomenúť, že abstraktná náročnosť výkladu narastá spoločne s ordinálnym číslovaním kapitol.

Typografické konvencie

Aby sme vám čítanie tejto knihy spríjemnili v čo možno najväčšej miere, bol prijatý kódex typografických konvencií, pomocou ktorých prišlo k štandardizácii a unifikácii použitých textových štýlov a grafických symbolov. Veríme, že prijaté konvencie napomôžu zvýšeniu prehľadnosti a používateľskej prívetivosti výkladu. Prehľad použitých typografických konvencií uvádzame v tab. A.

Tab. A: Prehľad použitých typografických konvencií

Typografická konvencia	Ukážka použitia typografickej konvencie
Štandardný text výkladu, ktorý neoznačuje zdrojový kód, identifikátory, modifikátory a kľúčové slová jazyka Visual Basic 2008, ani názvy iných syntaktických elementov a entít, je formátovaný týmto typom písma.	Vývojovo-exekučná platforma Microsoft .NET Framework 3.5 vytvára spoločne s jazykom Visual Basic 2008 jednotnú technologickú bázu na vytváranie moderných riadených aplikácií pre Windows, web a inteligentné mobilné zariadenia.
Názvy ponúk, položiek ponúk, ovládacích prvkov, komponentov, dialógových okien, podporných softvérových nástrojov, typov projektov ako aj názvy ďalších súčastí grafického používateľského rozhrania sú formátované **tučným** písmom.	V záujme založenia nového projektu štandardnej aplikácie pre systém Windows (**Windows Forms Application**) v prostredí jazyka Visual Basic 2008 postupujeme takto: 1. Otvoríme ponuku **File** a klikneme na položku **New Project**. 2. V dialógovom okne **New Project** klikneme v stromovej štruktúre **Project Types** na položku **Visual Basic**. 3. Zo súpravy projektových šablón (**Templates**) vyberieme ikonu šablóny **Windows Forms Application**. 4. Do textového poľa **Name** zapíšeme názov pre novú aplikáciu a stlačíme tlačidlo **OK**.
Symboly klávesov, klávesových skratiek a ich kombinácií sú uvádzané VEĽKÝMI PÍSMENAMI.	Ak chceme otvoriť už existujúci projekt jazyka Visual Basic 2008, použijeme klávesovú skratku CTRL+O.

Fragmenty zdrojového kódu jazyka Visual Basic 2008, prípadne akýchkoľvek iných programovacích jazykov, sú formátované neproporcionálnym písmom `Courier New`.	`'Definícia premennej typu String.`
	`Dim Správa As String`
	`'Inicializácia premennej.`
	`Správa = "Vitajte v jazyku " & _`
	`"Visual Basic 2008!"`
	`'Zobrazenie okna so správou`
	`'pomocou metódy Show triedy`
	`'MessageBox.`
	`MessageBox.Show(Správa)`

Okrem typografických konvencií predstavených v tab. A sa môžeme na stránkach knihy stretnúť aj s informačnými ostrovčekmi, ktoré ponúkajú hodnotné informácie súvisiace s práve preberanou problematikou. Zoznam použitých informačných ostrovčekov a s nimi asociovaných grafických symbolov je zobrazený v tab. B.

Tab. B: Prehľad použitých informačných ostrovčekov

Grafický symbol informačného ostrovčeka	Názov informačného ostrovčeka	Charakteristika
	**Upozornenie**	Upozorňuje čitateľov na dôležité skutočnosti, ktoré by mali mať v každom prípade na pamäti, pretože od nich môže závisieť pochopenie ďalších súvislostí alebo úspešné uskutočnenie postupu či pracovného algoritmu.
	**Poznámka**	Oboznamuje čitateľov s podrobnejšími informáciami, ktoré sa spájajú s vysvetľovanou problematikou. Hoci je miera dôležitosti tohto informačného ostrovčeka menšia ako u jeho predchodcu, vo všeobecnosti sa odporúča, aby čitatelia venovali doplňujúcim informačným vyhláseniam svoju pozornosť. Môžu sa tak dozvedieť nové fakty, alebo nájsť skryté súvislosti medzi už známymi poznatkami.
	**Tip**	Poukazuje na lepšie, rýchlejšie a efektívnejšie splnenie úlohy alebo postupu. Keď čitatelia uvidia v texte knihy tento informačný ostrovček, môžu si byť istí, že nájdu spôsob, ako produktívne dosiahnuť požadovaný cieľ.

Poďakovanie

Na tomto mieste by autor knihy rád vyjadril svoje poďakovanie pánovi Jiřímu Burianovi, ktorý zastáva pozíciu produktového manažéra vo vývojárskej divízii českej pobočky spoločnosti Microsoft za výbornú niekoľkoročnú spoluprácu, ktorá priniesla vývojárskej komunite mnoho hodnotných produktov. Rovnako srdečne ďakuje autor za skvelú spoluprácu, ochotu a podporu aj pánovi Miroslavovi Kubovčíkovi, ktorý je kmeňovým členom vývojárskeho tímu slovenskej pobočky spoločnosti Microsoft. Bez ich podpory by nemohla vzniknúť nielen táto kniha, ale ani mnohé ďalšie diela, ktoré permanentne zlepšujú stav vývojárskeho ekosystému. Takže páni, ešte raz úprimná vďaka!

Kapitola 1

Typová inferencia lokálnych entít

Microsoft® Visual Studio® 2008

Typová inferencia lokálnych entít

Prvou zo syntakticko-sémantických noviniek, ktoré jazyk Visual Basic 2008 prináša, je typová inferencia lokálnych entít. Termín „inferencia“ vo všeobecnom informatickom ponímaní znamená strojové odvodzovanie. Ak máme hovoriť o typovej inferencii, pôjde o strojové odvodzovanie dátových typov. Kompilátor jazyka Visual Basic 2008 totiž dokáže automaticky určiť dátový typ lokálnych premenných aj vtedy, ak pri definičných inicializáciách týchto premenných nebudeme príslušný dátový typ explicitne špecifikovať. Ak v definičnom príkaze lokálnej premennej vynecháme klauzulu As s určením typu, Visual Basic 2008 bude vhodný dátový typ inferovať na základe inicializačnej hodnoty premennej. Predpokladajme, že do editora zdrojového kódu jazyka Visual Basic 2008 zapíšeme nasledujúci definičný príkaz:

```
Option Strict On
Module Module1

    Sub Main()
        Dim x = 10
    End Sub

End Module
```

V tele hlavnej metódy Main je definovaná premenná s identifikátorom x, avšak bez priameho určenia svojho dátového typu. Pokiaľ neurčíme typ premennej, no determinujeme buď diskrétnu inicializačnú hodnotu, alebo výraz, ktorého hodnotu vie kompilátor určiť, mechanizmus typovej inferencie zabezpečí, že náležitá lokálna premenná bude definovaná ako typovo silná. Typovo silná premenná je premenná s konkrétnym dátovým typom, ktorý bol diagnostikovaný ešte v čase prekladu. Kompilátor teda premennej prisúdi najvhodnejší možný dátový typ, a to podľa charakteru entity nachádzajúcej sa na pravej strane priraďovacieho príkazu. V našom prípade sa snažíme do lokálnej premennej priradiť diskrétnu celočíselnú hodnotu (konštantu) 10, ktorej dátovým typom je implicitne Integer. Preto bude rovnaký typ pričlenený taktiež k samotnej premennej. To znamená, že po realizácii typovej inferencie skonštruuje kompilátor takýto zdrojový kód:

```
Option Strict On
Module Module1

    Sub Main()
        Dim x As Integer = 10
    End Sub

End Module
```

Ak by sme pôvodnú definičnú inicializáciu lokálnej premennej x zapísali v jazyku Visual Basic 2005, kompilátor by nás pri preklade kódu zastavil s chybovým hlásením. Keďže je aktívna voľba Option Strict, každý definičný príkaz lokálnej premennej musí obsahovať klauzulu As s určením dátového typu premennej. Ak by sme voľbu Option Strict v jazyku Visual Basic 2005 deaktivovali (jednoduchým prepnutím na Off), prekladač by sa síce utíšil, no premennej by bol priradený generický primitívny odkazový dátový typ Object. Túto skutočnosť môžeme verifikovať pri pohľade na vygenerovaný MSIL kód:

```
.method public static void  Main() cil managed
{
  .entrypoint
  .custom instance void [mscorlib]System.STAThreadAttribute::.ctor() =
   ( 01 00 00 00 )
  // Code size       11 (0xb)
  .maxstack  1
  .locals init ([0] object x)
  IL_0000:  nop
  IL_0001:  ldc.i4.s   10
  IL_0003:  box        [mscorlib]System.Int32
  IL_0008:  stloc.0
  IL_0009:  nop
  IL_000a:  ret
} // end of method Module1::Main
```

Vypnutie voľby Option Strict a predovšetkým absencia dátového typu sú udalosti, ktoré majú ďalekosiahle praktické implikácie. Za prvé, vzhľadom na to, že je definovanej premennej x automaticky vybratý primitívny odkazový typ Object, definovaná premenná je odkazovou a nie hodnotovou premennou. Za druhé, nakoľko v definičnej inicializácii je do implicitne odkazovej premennej x priradená celočíselná konštanta (ktorej typom je implicitne Integer), dochádza k spusteniu mechanizmu zjednotenia dátových typov. Tento proces, v origináli známy ako boxing, generuje značnú vedľajšiu réžiu, pretože musia byť uskutočnené tieto operácie:

1. Na riadenej halde je vytvorená objektová skrinka (presnejšie inštancia podtriedy odvodenej z bázovej triedy System.ValueType).

2. Do alokovanej objektovej skrinky je nakopírovaná pôvodná inicializačná hodnota lokálnej premennej.

3. Objektová referencia, ktorá jednoznačne identifikuje umiestnenie objektovej skrinky na riadenej halde, je uložená do lokálnej premennej s implicitným odkazovým typom Object.

Keď sa pozrieme na emitovaný MSIL kód, tak zreteľne postrehneme návestidlo IL_003 s inštrukciou box [mscorlib]System.Int32, ktoré verifikuje tvrdenia o spustení mechanizmu zjednotenia typov.

V jazyku Visual Basic 2008 pracuje typová inferencia aj vtedy, keď je voľba Option Strict vypnutá. Ak neuvedieme dátový typ lokálnej premennej, kompilátor požadovaný typ strojovo odvodí. Pozrime sa na ďalšie príklady typovej inferencie v súvislosti s lokálnymi premennými:

```
Option Strict On
Module Module1

    Sub Main()
        Dim a = 3.14 'Inferovaný typ: Double.
        Dim b = a * 2 'Inferovaný typ: Double.
        Dim c = CLng(b / a) \ 2 'Inferovaný typ: Long
        Dim d = "Visual Basic 2008" 'Inferovaný typ: String
    End Sub

End Module
```

Typová inferencia si poradí aj s ďalšími lokálnymi entitami, nemusí ísť vždy len o premenné primitívnych dátových typov. Napríklad, ďalej uvádzame typovú inferenciu, ktorú kompilátor uskutočňuje pri poliach:

```
Option Strict On
Module Module1

    Sub Main()
        Dim pole = New Integer() {1, 2, 3, 4, 5, 6}
        For i = 0 To pole.Length - 1
            Console.WriteLine((i + 1).ToString() & " prvok poľa " & _
                "má hodnotu " & pole(i) & ".")
        Next
        Console.Read()
    End Sub

End Module
```

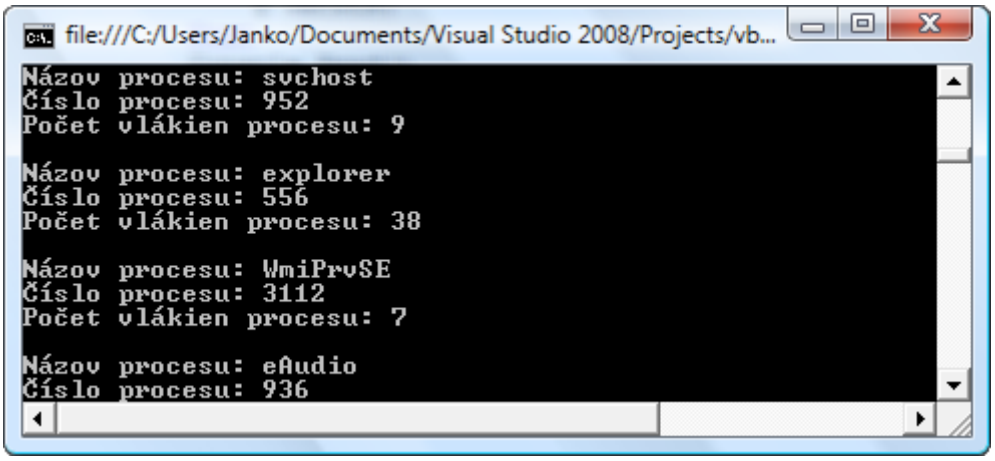
Vo fragmente kódu najskôr definujeme pole celých čísel. Všimnime si, že za identifikátorom poľa chýba dátový typ poľa (a teda dátový typ všetkých prvkov poľa). Pretože sme typ poľa sami neurčili, urobí to kompilátor za nás. Kompilátor sa pozrie na pravú stranu priraďovacieho príkazu a tu uvidí operátor New, ktorý zakladá nové jednorozmerné celočíselné pole. Pripomeňme, že pole patrí k používateľsky deklarovaným odkazovým dátovým typom. Pole je inštanciované na riadenej halde a v skutočnosti je reprezentované objektom zapuzdrujúcim kolekciu svojich prvkov. Všetky prvky definovaného poľa budú celé čísla, čo kompilátor zistí po analýze konštánt zoskupených v inicializačnom zozname.

Typová inferencia sa dostáva k slovu rovnako pri stanovení dátového typu riadiacej premennej cyklu For...Next. Premenná s identifikátorom i bude celočíselná, s implicitne určeným typom Integer.

Kompilátor dokáže inferovať dátový typ riadiacej premennej aj pri cykle For Each...Next. Uvádzaný výpis zdrojového kódu jazyka Visual Basic 2008 predvádza, ako možno s elegantnou jednoduchosťou „preskenovať“ procesy bežiace v operačnom systéme.

```
Option Strict On
Module Module1

    Sub Main()
        For Each Proces In Process.GetProcesses()
            Console.WriteLine("Názov procesu: " & Proces.ProcessName _
                & vbCrLf & "Číslo procesu: " & Proces.Id & _
            vbCrLf & "Počet vlákien procesu: " & Proces.Threads.Count _
            & vbCrLf)
        Next
        Console.Read()
    End Sub

End Module
```

Výstup programu je zobrazený na obr. 1.



Obr. 1: Informácie o aktívnych procesoch v operačnom systéme

Analogicky môžeme v jazyku Visual Basic 2008 zapísať inštanciačný príkaz štruktúry:

```
Option Strict On
Module Module1

    Structure Auto
        Dim Model As String
        Dim Cena As UInteger
        Dim VýkonVKW As UInteger
        Public Sub New(ByVal model As String, _
            ByVal cena As UInteger, ByVal výkon As UInteger)
            Me.Model = model
            Me.Cena = cena
            Me.VýkonVKW = výkon
        End Sub
    End Structure

    Sub Main()
        Dim mojeAuto = New Auto("Suzuki Ignis", _
        420000, 69)
        Console.WriteLine("Model auta: " & mojeAuto.Model & vbCrLf & _
            "Cena auta: " & mojeAuto.Cena & vbCrLf & _
            "Výkon auta (v KW): " & mojeAuto.VýkonVKW)
        Console.Read()
    End Sub

End Module
```

Štruktúra je používateľsky deklarovaný hodnotový dátový typ, ktorého inštancie sú alokované na zásobníku programového vlákna (a nie na riadenej halde, ako je to pri inštanciách odkazových dátových typov). Dátové členy deklarované v tele štruktúry majú implicitne verejnú viditeľnosť, čo platí aj v prípade našej štruktúry. Samozrejme, v záujme zachovania princípu ukrývania dát (ako jedného zo základných princípov objektovo orientovaného programovania), by sme mali dátové členy definovať ako súkromné. V ukážke sme však dali prednosť prehľadnosti a okamžitej dostupnosti dátových členov inštancie štruktúry bez nutnosti zavádzania definícií pre adekvátne prístupové metódy. Z pohľadu implicitného určenia dátového typu je významný inštanciačný príkaz:

```
Dim mojeAuto = New Auto("Suzuki Ignis", 420000, 69)
```

Reštrikcie v použití typovej inferencie

Mechanizmus typovej inferencie nemôže byť aplikovaný v týchto prípadoch:

1. **Pri definícii dátových členov používateľsky deklarovaných hodnotových a odkazových dátových typov.** Uvažujme deklaráciu triedy Kocka, ktorej inštancia bude pôsobiť ako virtuálna hracia kocka.

```
Option Strict On
Module Module1

    Class Kocka
        Private Číslo, Šestka As UInteger
        Private Generátor As Random
        Public Sub New()
            Generátor = New Random()
        End Sub
        Public Function Hodiť(Optional ByVal PočetHodov As Integer = 1) _
        As Integer
            For i = 1 To PočetHodov
                Číslo = CUInt(Generátor.Next(1, 7))
                Console.WriteLine("Bolo hodené číslo " & _
                    Číslo & ".")
                If Číslo = 6 Then
                    Šestka += CUInt(1)
                End If
            Next
            If Šestka = 0 Then
                Console.WriteLine("Číslo 6 nepadlo ani jedenkrát.")
            Else
                Console.WriteLine("Číslo 6 padlo " & Šestka & "krát.")
                Console.WriteLine("Vaša úspešnosť bola " & _
                (CType(Šestka, Single) / PočetHodov) * 100 & " %.")
            End If
        End Function
    End Class

    Sub Main()
        Dim kocka As New Kocka()
        kocka.Hodiť(10)
        Console.Read()
    End Sub

End Module
```

V tele triedy Kocka sú definované tri dátové položky, z ktorých prvé dve sú premennými primitívneho hodnotového dátového typu UInteger. Premenným Číslo a Šestka sme priradili konkrétny typ, pretože keby sme tak neurobili, kompilátor by nedokázal ich typ implicitne určiť. Aktívna voľba Option Strict by nám napokon ani neumožnila vynechať dátové typy v definičných príkazoch spomenutých premenných. Ak by sme voľbu Option Strict nastavili na hodnotu Off, potom by kompilátor premenným Číslo a Šestka prisúdil generický typ Object, čo nie je, ako sme si už vysvetlili, optimálne riešenie.

2. Pri definícii formálnych parametrov metód.

Trieda Kocka obsahuje verejne prístupný konštruktor, v tele ktorého dochádza k inicializácii generátora pseudonáhodných celých čísel (a samozrejme aj k inicializácii odkazovej premennej Generátor). „Hádzanie“ virtuálnou hracou kockou je v kompetencii metódy Hodiť, ktorá je verejne prístupná a inštančná (teda nezdieľaná). Zo signatúry tejto metódy vieme určiť, že pracuje s jedným implicitným (voliteľným) formálnym parametrom. Implicitný formálny parameter disponuje implicitnou inicializačnou hodnotou, ktorou je v našom prípade celočíselná konštanta 1. Na základe tejto inicializačnej hodnoty vieme stanoviť, že po zavolaní metódy Hodiť bude kocka hodená najmenej jedenkrát. Napriek tomu, že dátový typ formálneho parametra s implicitnou inicializačnou hodnotou by kompilátor dokázal inferovať, nie je to v jazyku Visual Basic 2008 možné. Ak by sme signatúru metódy Hodiť zmenili takto:

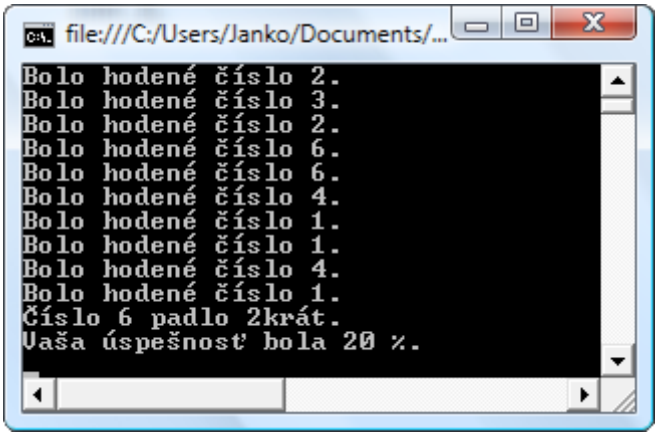
```
Public Function Hodiť(Optional ByVal PočetHodov = 1) _
```

kompilátor by sa pri aktívnej voľbe Option Strict sťažoval na absenciu klauzule As. Keď voľbu Option Strict vypneme, zdrojový kód bude síce preložený bez ďalších námietok, no dátovým typom formálneho parametra bude Object.

Inštanciu hracej kocky zostrojíme pomocou operátora New:

```
Sub Main()
    Dim kocka As New Kocka()
    kocka.Hodiť(10)
    Console.Read()
End Sub
```

Keď je kocka na svete, môžeme ňou hádzať, čo samozrejme robíme. Vyššie uvedený výpis kódu hodí virtuálnou hracou kockou desaťkrát, a potom vypíše informácie o hodoch a hráčovej úspešnosti (obr. 2).



Obr. 2: Hra s virtuálnou kockou

Poznámka: Na druhú stranu, riadiacu premennú cyklu For...Next nie je nutné explicitne opatriť dátovým typom, pretože v súvislosti s ňou sa typová inferencia postará o správnu voľbu dátového typu (pôjde o typ Integer).

```
'Typ riadiacej premennej cyklu je inferovaný.
For i = 1 To PočetHodov
 'Príkazy v tele cyklu boli vynechané.
Next
```

Inferencia by však nebola možná napríklad vtedy, keď by sa zmenil typ formálneho parametra PočetHodov na typ s užším rozsahom (povedzme na UInteger). Naopak, ak by sa typ formálneho parametra zmenil na typ so širším rozsahom (napríklad Long), typová inferencia by sa opäť uskutočnila.

3. **Pri definícii zdieľaných dátových atribútov tried.** Skúsme najskôr vysvetliť základný rozdiel v použití bežných (inštančných) a zdieľaných dátových atribútov. Zdieľané dátové atribúty (v origináli nazývané ako Shared) sú atribúty, ktoré sú asociované s triedou samotnou a nie s inštanciami triedy. Každá inštancia triedy obsahuje „svoju" množinu inštančných dátových atribútov, no na druhej strane, vždy existuje práve jedna kolekcia zdieľaných dátových atribútov, a to bez ohľadu na počet zrodených inštancií triedy. K zdieľaným dátovým atribútom smú pristupovať len zdieľané metódy. Podobne ako atribúty, i zdieľané metódy nie sú závislé od inštancií tried, a teda smú byť aktivované kedykoľvek.

   Aby sme si ukázali použitie zdieľaných dátových atribútov, upravili sme zdrojový kód triedy Kocka takto:

```
Class Kocka
    Private Číslo, Šestka As UInteger
    Private Generátor As Random

    Shared PočetKociek As Byte

    Public Sub New()
        Generátor = New Random()
        PočetKociek += CByte(1)
    End Sub

    Protected Overrides Sub Finalize()
        PočetKociek -= CByte(1)
    End Sub

    Public Function Hodiť(Optional ByVal PočetHodov As Integer = 1) _
    As Integer
        For i = 1 To PočetHodov
            Číslo = CUInt(Generátor.Next(1, 7))
            Console.WriteLine("Bolo hodené číslo " & _
                Číslo & ".")
            If Číslo = 6 Then
                Šestka += CUInt(1)
            End If
        Next
        If Šestka = 0 Then
```

```
            Console.WriteLine("Číslo 6 nepadlo ani jedenkrát.")
        Else
            Console.WriteLine("Číslo 6 padlo " & Šestka & _
                              "krát.")
            Console.WriteLine("Vaša úspešnosť bola " & _
            (CType(Šestka, Single) / PočetHodov) * 100 & " %.")
        End If
    End Function

    Shared Function ZistiťPočetKociek() As Byte
        Return PočetKociek
    End Function
End Class
```

Zmeny, ktoré sme uskutočnili, sú zvýraznené sivou podkladovou farbou. Nuž a o aké zmeny ide? Triedu sme modifikovali tak, aby si dokázala „pamätať“ počet svojich inštancií. Povedané inak, trieda nám bude kedykoľvek schopná poskytnúť informáciu o počte svojich inštancií, teda existujúcich hracích kociek. Za týmto účelom definujeme zdieľaný dátový atribút PočetKociek s konkrétnym typom Byte. Ďalej pridávame verejne prístupný bezparametrický konštruktor a finalizér (metódu Finalize, ktorá prekrýva pôvodne zdedenú finalizačnú metódu primárnej bázovej triedy System.Object). Keď bude založená nová inštancia triedy Kocka, prostredie CLR automaticky zavolá konštruktor, aby zabezpečil inicializáciu zrodenej inštancie. V tomto okamihu vieme, že inštancia existuje, a preto inkrementujeme hodnotu zdieľaného dátového atribútu. Naopak, keď sa inštancia triedy Kocka stane nepotrebnou, automatický správca pamäte (Garbage Collector) uskutoční jej explicitnú finalizáciu. Predtým, ako bude uvedená inštancia uvoľnená z riadenej haldy, GC zavolá jej finalizér. Keď bude vyvolaný finalizér, je zrejmé, že inštancia sa nachádza v poslednej etape svojho životného cyklu, a preto znižujeme (dekrementujeme) hodnotu udávajúcu počet „žijúcich“ inštancií triedy Kocka.

Keďže sme vyhlásili, že k zdieľanému dátovému atribútu môžu pristupovať len zdieľané metódy, jednu takú potrebujeme. Počet hracích kociek nám rada poskytne zdieľaná metóda ZistiťPočetKociek. Všimnime si, že dátový typ návratovej hodnoty tejto metódy sme opäť explicitne determinovali. Inými slovami, typová inferencia nie je schopná odvodiť si sama typ návratovej hodnoty zdieľanej metódy. Toto obmedzenie sa však nevzťahuje len na zdieľané metódy, ale na metódy vo všeobecnosti. Ak by metóda nešpecifikovala dátový typ svojej návratovej hodnoty, existujú dve cesty ďalšieho postupu:

1. Ak je aktívna voľba Option Strict, kompilátor zahlási pri preklade kódu chybu (absentujúca klauzula As v signatúre metódy).

2. Ak je voľba Option Strict deaktivovaná, kompilátor zariadi, aby metóda vracala hodnotu typu Object (metóda bude v skutočnosti vracať objektovú referenciu na objektovú skrinku alokovanú na riadenej halde, ktorej vznik bol podmienený aktiváciou mechanizmu zjednotenia typov).

Pozrime sa na ukážku, ktorá predvádza „žonglovanie“ s tromi hracími kockami:

```
Sub Main()
    'Vytvorenie poľa s tromi kockami. Typová inferencia je aktívna.
    Dim Kocky() = New Kocka() { _
    New Kocka(), New Kocka(), New Kocka()}

    'S každou kockou hodíme jedenkrát.
    For Each Kocka In Kocky
        Kocka.Hodiť()
    Next

    'Napokon zistíme, koľko kociek máme.
    Console.WriteLine("Počet kociek: " & _
        Kocka.ZistiťPočetKociek() & ".")
    Console.Read()
End Sub
```

4. **Pri definícii lokálnych entít, ktoré nie sú v rámci definičného príkazu inicializované.** Typicky ide o tzv. „čistú“ definíciu, napríklad:

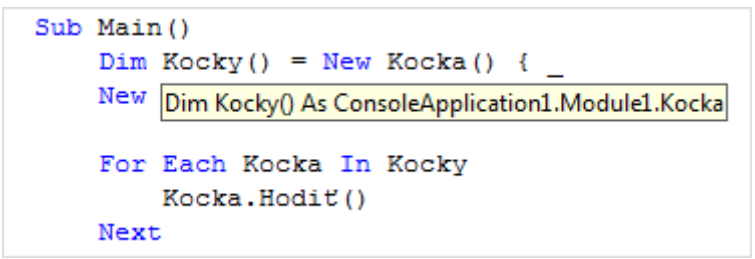
```
Dim m
```

V tomto prípade je definovaná lokálna premenná m, ktorej dátový typ nevie kompilátor určiť, pretože nepozná hodnotu, ktorá bude do tejto premennej priradená. Ak je aktívna voľba Option Strict, kompilátor preruší preklad s generovaním chybového hlásenia. Premennú bude potrebné definovať s určením konkrétneho dátového typu, alebo nastaviť voľbu Option Strict na hodnotu Off (to samozrejme neodporúčame).

Voľba Option Infer

Kolekcia konfiguračných nastavení s prívlastkom „Option“, ktoré usmerňujú prácu kompilátora, sa v jazyku Visual Basic 2008 rozrástla o nového zástupcu: Option Infer. V zásade platí, že po založení nového aplikačného projektu jazyka Visual Basic 2008 je voľba Option Infer implicitne aktívna (nastavená na hodnotu On). Tým pádom dokáže kompilátor inferovať dátové typy lokálnych premenných na základe hodnôt do nich ukladaných. V situáciách, keď nie je realizácia typovej inferencie žiaduca, môže byť jej mechanizmus deaktivovaný „prepnutím“ voľby Option Infer do pozície Off. Typová inferencia je štandardne vypnutá pri inováciách projektov, ktoré boli vytvorené v predchádzajúcich verziách jazyka Visual Basic.

Stav voľby Option Infer reflektuje senzitívna technológia IntelliSense. Tá sleduje naše kroky v prostredí editora zdrojového kódu a pomáha nám všade, kde sa len dá. Keď je typová inferencia zapnutá, IntelliSense nám ukazuje inferované dátové typy lokálnych entít (obr. 3).



Obr. 3: Senzitívna technológia IntelliSense a typová inferencia

Je rovnako zaujímavé sledovať vzťah medzi voľbami Option Strict a Option Infer. Skúsme uskutočniť takýto experiment: Voľbu Option Strict vypneme (Off), no inferenciu ponecháme aktivovanú (On) a vzápätí vložíme tento zdrojový kód:

```
'Kód jazyka Visual Basic 2008.
Option Strict Off
Option Infer On

Module Module1

    Sub Main()
        'Dátovým typom premennej x je Integer.
        Dim x = 10
        'Tento príkaz spôsobí generovanie chybovej výnimky
        'počas behu programu.
        x = "Desať"
        Console.WriteLine(x)
        Console.Read()
    End Sub

End Module
```

Typom lokálnej premennej x bude Integer, pretože do nej ukladáme celočíselnú konštantu (ktorej implicitným typom je Integer). V druhom príkaze sa snažíme vložiť do premennej x typu Integer objektovú referenciu na inštanciu triedy System.String s reťazcovým literálom. Hoci kompilátor by mal namietať, že takéto priradenie nie je možné, kód bude preložený bez problémov. Katastrofa však na seba nenechá dlho čakať: len čo program spustíme, vznikne chybová výnimka, ktorá predčasne preruší ďalšie spracovanie programu. Pôvod chybového stavu je pochopiteľný, pretože do premennej primitívneho hodnotového dátového typu nemôže byť uložená referencia na inštanciu istej triedy.

Ak rovnaký experiment zrealizujeme v jazyku Visual Basic 2005, zistíme dve veci:

1. Zdrojový kód bude preložený v poriadku, nebudú hlásené žiadne chyby, ani varovania.
2. Beh programu je takisto hladký, nenarazíme na žiadnu chybovú výnimku.

Pozrime sa na zdrojový kód:

```
'Kód jazyka Visual Basic 2005.
Option Strict Off
Module Module1

    Sub Main()
        Dim x = 10
        x = "Desať"
        Console.WriteLine(x)
        Console.Read()
    End Sub

End Module
```

Keďže predchádzajúca verzia jazyka Visual Basic nepracuje s typovou inferenciou, kompilátor bude na lokálnu premennú x nahliadať ako na odkazovú premennú typu Object. Definičná inicializácia (Dim x = 10) spôsobí aktiváciu mechanizmu zjednotenia typov.

Druhé priradenie (x = "Desať") potom umiestňuje do odkazovej premennej referenciu na inštanciu triedy System.String s textovým reťazcom.

Význam typovej inferencie

Na záver našej rozpravy o typovej inferencii sme si nechali niekoľko postrehov, ktoré sa spájajú s praktickým nasadením mechanizmu typovej inferencie v jazyku Visual Basic 2008. Existuje totiž viacero chybných či nepresných, v každom prípade zavádzajúcich premís, s ktorými sa ako vývojári môžeme stretnúť. Cieľom tejto podkapitoly je tieto nekorektné tvrdenia uviesť na pravú mieru a poukázať na skutočnosť, že typová inferencia je významnou syntakticko-sémantickou inováciou s rozsiahlym praktickým uplatnením.

Mýtus č. 1: Lokálna entita, ktorej dátový typ nie je explicitne určený v rámci definičnej inicializácie, má priradený typ Object.

Demystifikácia: Toto konštatovanie sa nezakladá na pravde, pretože kompilátor automaticky odvodzuje dátový typ lokálnej entity na základe hodnoty, ktorá je do tejto entity priradená. To môže byť buď diskrétna hodnota (konštanta) primitívneho hodnotového alebo odkazového dátového typu, alebo hodnota istého výrazu. Samozrejme, nie je vylúčené, že mechanizmus typovej inferencie rozhodne o priradení typu Object. To sa deje napríklad v nasledujúcom výpise zdrojového kódu jazyka Visual Basic 2008:

```
Sub Main()
    'Typom lokálnej premennej o je Object.
    Dim o = New Object()
End Sub
```

Mýtus č. 2: Typová inferencia determinuje dátový typ lokálnej entity až v čase exekúcie programu.

Demystifikácia: Typovú inferenciu realizuje kompilátor pri preklade zdrojového kódu jazyka Visual Basic 2008. Typová inferencia neznamená vytváranie neskorých väzieb medzi dátovými objektmi a ich typmi. Túto skutočnosť môžeme verifikovať nahliadnutím do vygenerovaného MSIL kódu, ktorý na nízkej úrovni reflektuje kód jazyka Visual Basic 2008, v ktorom je uskutočňovaná typová inferencia.

Mýtus č. 3: Zapracovaním typovej inferencie sa jazyk Visual Basic 2008 vracia k prekonaným praktikám, ku ktorým patrilo používanie „typovo prostých" entít.

Demystifikácia: Tento mýtus sme objasnili už v predchádzajúcej odpovedi. Povedané stručne a jasne, Visual Basic 2008 sa v žiadnom prípade nevracia do „praveku", do historických čias, keď vládol jazyk BASIC. Vždy, keď je to možné, kompilátor inferuje správne dátové typy lokálnych entít.

Mýtus č. 4: Typová inferencia nie je vôbec podstatná, pretože vývojár predsa vždy dokáže určiť dátový typ lokálnej entity.

Demystifikácia: Áno, na prvý pohľad to vyzerá, že tento mýtus je predsa len pravdivý. Veď keď programátor zapisuje zdrojový kód lokálnych entít, musí poznať ich typy. To je pravda, hoci tento efekt je zreteľný najmä v situáciách, kedy je stanovenie dátového typu pre programátora prirodzené. No niekedy si okolnosti vynútia generovanie anonymného typu. Anonymné typy sú ďalšou zo signifikantných syntakticko-sémantických inovácií jazyka Visual Basic 2008. Adjektívum „anonymný“ vraví, že zostavenie deklarácie tohto typu je plne v réžii kompilátora a nie vývojára. Vývojár teda nedokáže lokálnu entitu anonymného typu opatriť vhodným typom práve preto, že daný typ je pre neho skrytý, a teda anonymný. Len kompilátor vie, ako presne anonymný typ vyzerá (a aký má identifikátor). Vďaka typovej inferencii je možné, aby aj lokálna entita disponovala v čase prekladu konkrétnym (hoci z pohľadu programátora anonymným) dátovým typom.

Kapitola 2

Inštanciácia pomenovaných a anonymných tried pomocou inicializátorov uvádzaných v inicializačných zoznamoch

Microsoft® Visual Studio® 2008

Inštanciácia pomenovaných a anonymných tried pomocou inicializátorov uvádzaných v inicializačných zoznamoch

Visual Basic 2008 uvádza možnosť inštanciovať pomenovaný alebo anonymný používateľsky deklarovaný odkazový dátový typ pomocou inicializátorov, ktoré sú zoskupené do inicializačného zoznamu. Inicializačný zoznam predstavuje množinu inicializátorov, ktoré sú umiestnené v bloku ohraničenom zloženými zátvorkami ({}). K inštanciačnému výrazu s operátorom New je inicializačný zoznam pripojený kľúčovým slovom With. V inicializačnom zozname sú pomocou bodkovej (.) notácie špecifikované vlastnosti, do ktorých majú byť uložené inicializátory, teda inicializačné hodnoty. Pochopiteľne, z technického hľadiska sú inicializátory ukladané do súkromných dátových položiek inštancie triedy a nie do vlastností, pretože vlastnosti len sprístupňujú dátové položky, no nie sú entitami s istou alokačnou kapacitou.

Najskôr si ukážeme, ako inštanciovať triedu bežným spôsobom. Budeme pracovať s triedou Študent, ktorá má takúto deklaráciu:

```
Class Študent
    Private m_Meno, m_Priezvisko As String
    Private m_Kredity As Byte
    Public Sub New()
        m_Meno = "Štefan"
        m_Priezvisko = "Malý"
        m_Kredity = 100
    End Sub
    Public Sub New(ByVal Meno As String, ByVal Priezvisko As String, _
        ByVal Kredity As Byte)
        m_Meno = Meno
        m_Priezvisko = Priezvisko
        m_Kredity = Kredity
    End Sub
    Public Property Meno() As String
        Get
            Return m_Meno
        End Get
        Set(ByVal value As String)
            m_Meno = value
        End Set
    End Property
    Public Property Priezvisko() As String
        Get
            Return m_Priezvisko
        End Get
        Set(ByVal value As String)
            m_Priezvisko = value
        End Set
    End Property
    Public Property Kredity() As Byte
        Get
            Return m_Kredity
        End Get
        Set(ByVal value As Byte)
            m_Kredity = value
        End Set
```

```
    End Property
End Class
```

Trieda reprezentuje študenta, pričom sleduje tri vlastnosti, ktorými sú meno, priezvisko a počet kreditov, ktoré študent získal. V triede sa nachádza preťažený konštruktor (metóda s názvom New), ktorý sa objavuje najskôr v bezparametrickom vyhotovení a následne aj vo verzii s tromi formálnymi parametrami. S každou dátovou položkou je asociovaná jedna skalárna vlastnosť, ktorá je určená na čítanie i zápis (s metódami Get a Set).

Ako sme boli zvyknutí v jazyku Visual Basic 2005, deklarovanú triedu inštanciujeme aj v novej verzii programovacieho jazyka známym spôsobom:

```
Dim Bakalár As Študent = New Študent()
Dim Magister As Študent = New Študent("Peter", "Slovák", 220)
```

V obidvoch prípadoch je aplikovaná definičná inicializácia. Prvý inštanciačný príkaz definuje odkazovú premennú Bakalár, do ktorej je uložená objektová referencia na založenú inštanciu triedy Študent. Prvá inštancia je po svojej alokácii inicializovaná bezparametrickým konštruktorom. V druhom prípade je inštanciácia triedy Študent uskutočnená podobne, len s tým rozdielom, že objekt situovaný na riadenej halde je inicializovaný parametrickým konštruktorom. Pre úplnosť dodajme, že predostreté definičné inicializácie by sme dokázali prepísať tak, aby nebol použitý priraďovací operátor (=):

```
Dim Bakalár As New Študent()
Dim Magister As New Študent("Peter", "Slovák", 220)
```

Teraz si predvedieme, ako založíme inštanciu triedy Študent pomocou kľúčového slova With a inicializačného zoznamu:

```
Dim Doktorand As Študent = New Študent With {.Meno = "Milan", _
    .Priezvisko = "Nižný", .Kredity = 160}
```

Inicializačný zoznam je tvorený blokom {}, v ktorom sú situované inicializátory, ktoré sú priraďovacími príkazmi odosielané do príslušných vlastností. Tak dochádza k inicializácii dátových položiek založenej inštancie triedy. Všimnime si, že inicializačný zoznam je k operátoru New pripojený kľúčovým slovom With. (Keby sme nešpecifikovali kľúčové slovo With, nemohli by sme k vlastnostiam v inicializačnom zozname pristupovať pomocou bodkového operátora.)

Domnievame sa, že bude vhodné, keď si spracovanie celého inštanciačného príkazu s inicializačným zoznamom rozoberieme po jednotlivých krokoch. Nuž, postup je nasledujúci:

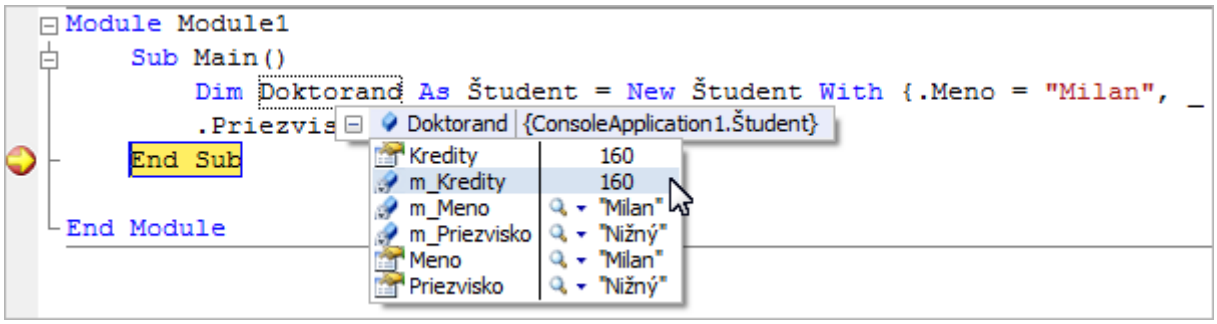
1. Na zásobník je uložená odkazová premenná s identifikátorom Doktorand a odkazovým dátovým typom Študent.

2. Operátor New zabezpečí vytvorenie inštancie triedy Študent a automaticky zavolá bezparametrický konštruktor, ktorý túto inštanciu inicializuje.

3. Keď konštruktor dokončí svoju prácu, dostáva sa k slovu ďalšia inicializácia, tentoraz tá, ktorú sme predpísali v inicializačnom zozname. Do dátových položiek inštancie triedy budú priradené inicializátory.

4. Objektová referencia identifikujúca pozíciu zrodenej inštancie triedy Študent je uložená do odkazovej premennej Doktorand.

Tip: Obsah „žijúcej“ inštancie triedy Študent môžeme sledovať prostredníctvom vizualizačných schopností ladiaceho programu (obr. 4).



Obr. 4: Inšpekcia dátových členov inštancie triedy Študent

Použitie inicializačného zoznamu je možné aj vtedy, keď je založená inštancia pomenovanej triedy inicializovaná parametrickým konštruktorom, no za týchto okolností nie je zapojenie inicializačného zoznamu spravidla potrebné. Je to preto, že všetky inicializačné aktivity vykoná parametrický konštruktor sám.

Inštanciačné príkazy, ktoré sme rozobrali, pracovali výhradne s pomenovanou triedou. Pomenovaná trieda predstavuje triedu, ktorá existuje. Ide teda o triedu, ktorá je korektne deklarovaná, má svoj identifikátor a definuje tri vlastnosti, ktoré sa volajú Meno, Priezvisko a Kredity. Pokiaľ sa v analyzovanej oblasti platnosti nachádza trieda spĺňajúca tieto požiadavky, kompilátor jazyka Visual Basic 2008 bude emitovať kód pre založenie a inicializáciu inštancie triedy s využitím inicializátorov uvedených v inicializačnom zozname.

Ak uvážime štýl použitia inicializátorov, potom určite prídeme na to, že podobný efekt by sme mohli dosiahnuť aj iným spôsobom:

```
Dim Doktorand As Študent = New Študent()
With Doktorand
    .Meno = "Peter"
    .Priezvisko = "Nižný"
    .Kredity = 160
End With
```

Áno, je to tak. Naozaj, pokiaľ rozložíme predchádzajúcu definičnú inicializáciu na dve časti, tak jednak získame objekt triedy Študent a jednak uskutočníme inicializáciu jeho dátových položiek.

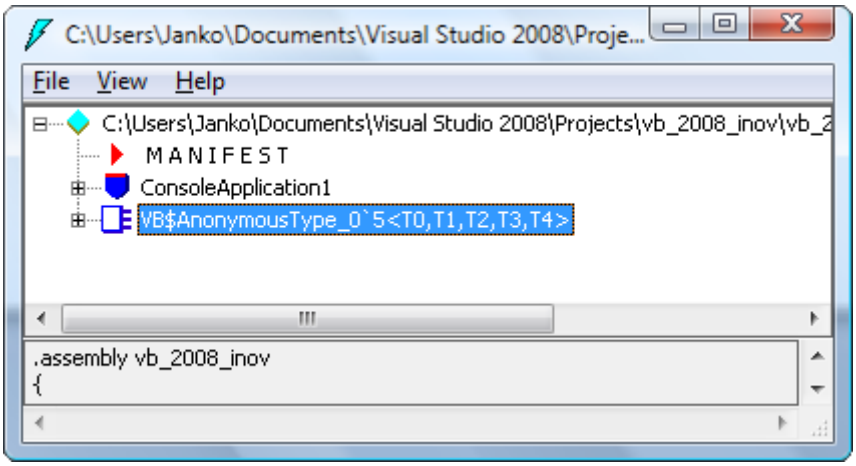
Inicializátory však dokážu pracovať aj na „vyššej úrovni“, a to vtedy, keď ich použijeme na inicializáciu inštancie anonymného odkazového dátového typu (anonymnej triedy). Zatiaľ čo pri inštanciácii pomenovanej triedy musí byť táto trieda deklarovaná a rovnako musí zavádzať definície určitých vlastností, pri anonymných triedach je to celkom inak. Jazyk Visual Basic 2008 nám dovolí založiť inštanciu aj takej triedy, ktorú nebudeme vopred deklarovať. V inštanciačnom príkaze potom obsah inicializačného zoznamu poskytne kompilátoru informácie o podobe anonymnej triedy. Kompilátor zistí, že chceme vytvoriť inštanciu triedy, ktorá má isté vlastnosti a dátové položky. Na základe dodaných informácií kompilátor sám vygeneruje triedu, založí jej inštanciu a následne ju inicializuje tak, ako predpisuje nami zapísaný inicializačný zoznam. Všetky opísané činnosti sa odohrávajú v nasledujúcom fragmente zdrojového kódu jazyka Visual Basic 2008:

```
Dim OdbornýAsistent = New With {.Meno = "Ivan", _
    .Priezvisko = "Rusnák", .Prednášky = 2, _
    .Cvičenia = 4, .Mzda = 22000}
```

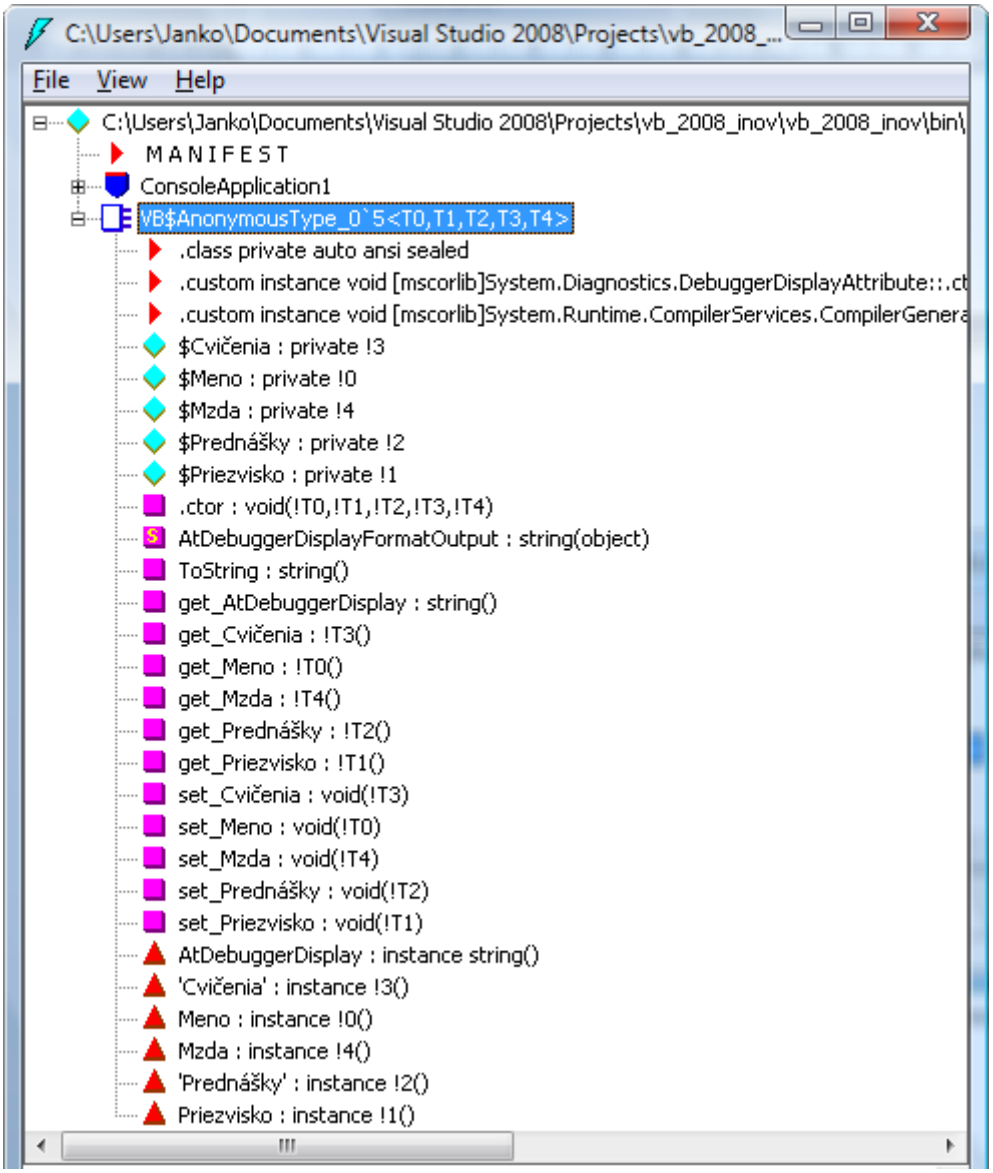
Ako môžeme postrehnúť, v definičnej inicializácii nie je explicitne zadaný žiaden odkazový dátový typ. Typ nemá ani odkazová premenná definovaná na ľavej strane priraďovacieho príkazu a žiaden typ sa nenachádza ani za operátorom New. Keď v jazyku Visual Basic 2008 uvidíme takýto syntaktický príkaz, vravíme, že ide o inštanciáciu a inicializáciu anonymného odkazového dátového typu (anonymnej triedy). Keďže trieda, ktorej inštanciu chceme založiť, je anonymná, nepoznáme jej identifikátor. Rovnako nevieme určiť typ odkazovej premennej, no táto skutočnosť, ako sa dozvieme, nepredstavuje žiaden problém. Kompilátor sám vygeneruje deklaráciu triedy, ktorá bude obsahovať päť vlastností (Meno, Priezvisko, Prednášky, Cvičenia a Mzda). Okrem vlastností vloží kompilátor do tela triedy tiež potrebný počet súkromných dátových položiek. Pritom bude platiť jednoduché asociatívne pravidlo, podľa ktorého sú dátové položky a vlastnosti vo vzťahu 1:1. Kompilátor ďalej vloží do tela triedy inštančný konštruktor a triedu pomenuje. Názov triedy nie je pre vývojárov priamo prístupný a pravdu povediac, to, aký identifikátor kompilátor anonymnej triede priradí, nie je pre nás až také dôležité. V skutočnosti môže kompilátor meniť identifikátor anonymnej triedy v jednotlivých prekladových reláciách, z čoho vyplýva, že programátori ani nemajú dôvod na to, aby implicitne vyprodukovaný názov triedy sami používali.

Vzhľadom na to, že zodpovednosť za deklarovanie anonymnej triedy vezme na svoje plecia kompilátor, my si vystačíme s konštatovaním, že pre vývojárov bude vždy pripravená trieda s príslušnou internou kompozíciou. Dobre, teraz vieme, ako je určovaný typ inštancie, no ešte musíme odpovedať na otázku, ako je určený typ odkazovej premennej umiestnenej na ľavej strane priraďovacieho príkazu. Nuž, opäť implicitne. Ak tvrdíte, že priradenie typu má na svedomí typová inferencia, tak máte samozrejme pravdu. Dátový typ odkazovej premennej je implicitne odvodený od typu inštancie triedy, ktorú zakladáme.

Hoci nie je identifikátor anonymnej triedy známy, môžeme zistiť, ako vyzerá. Stačí, ak sa pozrieme na štruktúru vygenerovaného MSIL kódu, ktorý zodpovedá kódu jazyka Visual Basic 2008 (obr. 5). V našom prípade sa anonymná trieda vystupujúca v úlohe anonymného odkazového dátového typu volá VB$AnonymousType_0`5. Deklaráciu triedy objavíme po rozvinutí príslušného uzla: my ju uvádzame na obr. 6.



Obr. 5: Kompilátorom vygenerovaná anonymná trieda v jazyku MSIL



Obr. 6: Obraz kompilátorom deklarovanej anonymnej triedy

Po preštudovaní tela triedy zistíme, že kompilátor automaticky vygeneroval päť vlastností a zhodný počet súkromných dátových položiek. Keďže vlastnosť je programová konštrukcia

združujúca dve prístupové metódy (get_ a set_), v deklarácii triedy sa vyskytuje desať prístupových metód. Tieto prístupové metódy sú určené na získanie, resp. modifikáciu súkromných dátových položiek inštancie triedy.

Tip: V programe Microsoft .NET Framework IL Disassembler (IL DASM) sú členy triedy farebne odlíšené podľa nasledujúceho vzoru:

Tab. 1: Vizualizácia členov triedy v programe IL DASM

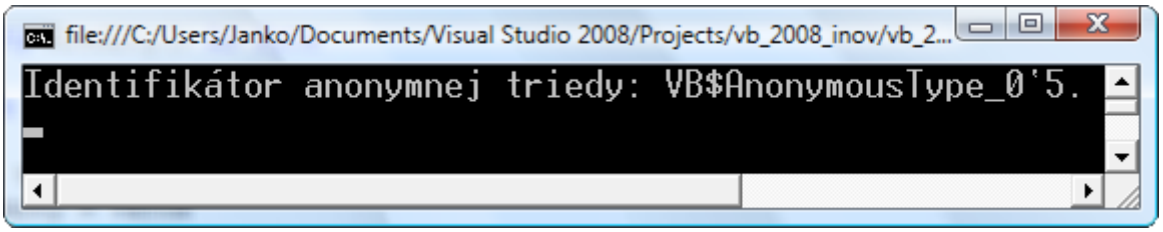
Ikona člena	Charakteristika
	Súkromná dátová položka
 .ctor	Konštruktor (inštančný)
	Metóda (inštančná)
	Vlastnosť (skalárna)

Ak budete chcieť stoj čo stoj získať identifikátor kompilátorom zostavenej anonymnej triedy, máme pre vás jeden tip – reflexiu:

```
Sub Main()
    Dim OdbornýAsistent = New With {.Meno = "Ivan", _
       .Priezvisko = "Rusnák", .Prednášky = 2, _
       .Cvičenia = 4, .Mzda = 22000}

    Dim IdentifikátorTriedy As String = _
       OdbornýAsistent.GetType().Name

    Console.WriteLine("Identifikátor anonymnej triedy: " & _
       IdentifikátorTriedy & ".")
    Console.Read()
End Sub
```

Reflexia je užitočný mechanizmus získavania informácií (metadát) o dátových typoch deklarovaných v zostavení aplikácie .NET. Ak máte skúsenosti s jazykom C++, tak v tomto prostredí sa podobný mechanizmus nazýva identifikácia dátových typov počas spracovania programu: RTTI (Run-Time Type Identification). Vďaka reflexii možno dolovať významné informácie o dátových typoch, dokonca je možné dynamicky vytvárať inštancie požadovaných typov. Mechanizmus reflexie sa viaže na prostredie CLR a vývojovo-exekučnú platformu Microsoft .NET Framework 3.5. To znamená, že jeho pozitívne aspekty môžu vo svoj prospech využívať nielen fanúšikovia jazyka Visual Basic 2008, ale takisto vývojári pracujúci v jazykoch C# a C++/CLI, ako aj v iných .NET-kompatibilných programovacích prostriedkoch.

Výstup programu, ktorý pomocou reflexie zisťuje identifikátor anonymnej triedy, je uvedený na obr. 7.

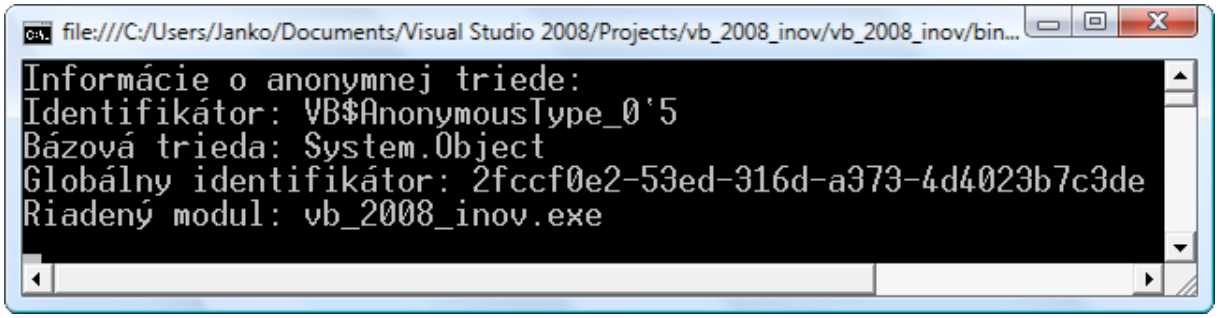


Obr. 7: Reflexívna analýza identifikátora anonymnej triedy

S reflexiou sa skamarátime veľmi rýchlo, pretože nám dokáže ponúknuť mnoho zaujímavých informácií. Zoznámme sa s rozšírenou verziou predchádzajúceho fragmentu zdrojového kódu:

```
Sub Main()
    Dim OdbornýAsistent = New With {.Meno = "Ivan", _
       .Priezvisko = "Rusnák", .Prednášky = 2, _
       .Cvičenia = 4, .Mzda = 22000}

    Dim IdentifikátorTriedy As String = _
       OdbornýAsistent.GetType().Name

    Dim BázováTrieda As String = _
       OdbornýAsistent.GetType().BaseType.FullName

    Dim GlobálnyIdentifikátor As String = _
       OdbornýAsistent.GetType().GUID.ToString()

    Dim Modul As String = _
       OdbornýAsistent.GetType().Module.Name

    Console.WriteLine("Informácie o anonymnej triede: " & _
     vbCrLf & "Identifikátor: " & IdentifikátorTriedy & _
     vbCrLf & "Bázová trieda: " & BázováTrieda & _
     vbCrLf & "Globálny identifikátor: " & GlobálnyIdentifikátor & _
     vbCrLf & "Riadený modul: " & Modul)
    Console.Read()
End Sub
```

V kóde okrem identifikátora anonymnej triedy zisťujeme tiež jej bázovú triedu, globálny identifikátor GUID a riadený modul, v ktorom je deklarácia triedy uložená (obr. 8).



Obr. 8: Ďalšie informácie o anonymnej triede získané pomocou reflexie

Ak kompilátor jazyka Visual Basic 2008 detekuje dva inštanciačné príkazy, v ktorých inicializačných zoznamoch sú uvedené rovnomenné vlastnosti s rovnakou početnosťou, vygeneruje len jednu anonymnú triedu. To sa deje napríklad v tejto ukážke:

```
'Kompilátor automaticky deklaruje len jednu anonymnú triedu.
Dim MôjPočítač = New With {.Procesor = "Intel Core 2 Duo", _
   .TaktProcesora = 4.26F, .OperačnáPamäťVGB = 2, _
   .PevnýDiskVGB = 320}

Dim KamarátovPočítač = New With {.Procesor = "AMD Turion 64 X2", _
   .TaktProcesora = 4.0F, .OperačnáPamäťVGB = 2, _
   .PevnýDiskVGB = 320}
```

V priebehu lexikálnej, syntaktickej a sémantickej analýzy kompilátor objaví, že oba inštanciačné príkazy disponujú zhodnou podobou inicializačných zoznamov. Preto kompilátor správne usúdi, že bude stačiť, ak zostaví len jednu anonymnú triedu s adekvátnou internou kompozíciou. Po skonštruovaní anonymnej triedy budú vytvorené jej dve inštancie, ktoré budú virtuálnymi modelmi počítača. Hoci vznikli z jednej anonymnej triedy, pôsobia zrodené inštancie ako dve samostatné jednotky (objekty). Podotknime, že ak by sa obsahy inicializačných zoznamov inštanciačných príkazov odlišovali, kompilátor by musel zabezpečiť generovanie dvoch anonymných tried. Táto situácia nastane vtedy, keď jemne obmeníme syntaktickú formu druhého príkazu:

```
'Kompilátor automaticky deklaruje dve odlišné anonymné triedy.
Dim MôjPočítač = New With {.Procesor = "Intel Core 2 Duo", _
   .TaktProcesora = 4.26F, .OperačnáPamäťVGB = 2, _
   .PevnýDiskVGB = 320}

Dim KamarátovPočítač = New With {.Procesor = "AMD Turion 64 X2", _
   .TaktCPU = 4.0F, .OperačnáPamäťVGB = 2, _
   .PevnýDiskVGB = 320}
```

V druhom príkaze sme zmenili identifikátor druhej vlastnosti z pôvodného (TaktProcesora) na nový (TaktCPU). Táto „maličká“ zmena z pohľadu programátora znamená niekoľkonásobne väčšie režijné náklady pre kompilátor. Keďže kompilátor si už nevystačí s jednou anonymnou triedou, nezostane mu nič iné, len pripraviť deklarácie dvoch anonymných tried, ktoré sa budú líšiť identifikátorom vlastnosti pre určenie hodinovej frekvencie procesora.

Pri práci s viacerými anonymnými triedami je nutné si uvedomiť, že tak triedy ako aj ich inštancie, nie sú v žiadnom príbuzenskom vzťahu. Preto pokus o priradenie hodnoty jednej odkazovej premennej do inej skončí chybovým hlásením o nemožnosti uskutočnenia typovej konverzie:

```
Dim MôjPočítač = New With {.Procesor = "Intel Core 2 Duo", _
   .TaktProcesora = 4.26F, .OperačnáPamäťVGB = 2, _
   .PevnýDiskVGB = 320}

Dim KamarátovPočítač = New With {.Procesor = "AMD Turion 64 X2", _
   .TaktCPU = 4.0F, .OperačnáPamäťVGB = 2, _
   .PevnýDiskVGB = 320}

'Tento priraďovací príkaz je zdrojom chyby.
MôjPočítač = KamarátovPočítač
```

Ako už vieme, tento fragment zdrojového kódu si vyžiada zhotovenie dvoch anonymných tried. Kompilátor tieto triedy ochotne pripraví, no keďže medzi nimi neexistuje relácia založená na

báze asociatívnosti alebo dedičnosti, sú obe triedy ponímané ako samostatné entity. Preto pri pokuse o priradenie objektovej referencie uchovanej v odkazovej premennej KamarátovPočítač nás kompilátor zastaví s chybovým hlásením. Toto obmedzenie sa samozrejme nevzťahuje na prípady, kedy je bez ohľadu na počet založených inštancií generovaná len jedna deklarácia anonymnej triedy.

```
Dim MôjPočítač = New With {.Procesor = "Intel Core 2 Duo", _
   .TaktProcesora = 4.26F, .OperačnáPamäťVGB = 2, _
   .PevnýDiskVGB = 320}

Dim KamarátovPočítač = New With {.Procesor = "AMD Turion 64 X2", _
   .TaktProcesora = 4.0F, .OperačnáPamäťVGB = 2, _
   .PevnýDiskVGB = 320}

'Teraz je všetko v poriadku.
MôjPočítač = KamarátovPočítač
```

Posledný priraďovací príkaz už nie je viac zdrojom vzniku chyby, pretože kompilátor vie, že priradenie je bezpečné. Hoci na riadenej halde budú situované dve inštancie, obe pochádzajú z jednej anonymnej triedy. Referencia na objekt predstavujúci kamarátov počítač tak môže byť bez akýchkoľvek problémov uložená do odkazovej premennej MôjPočítač.

S opísaným priraďovacím príkazom sa spájajú niektoré implikácie, ktoré považujeme za vhodné bližšie charakterizovať:

1. Keďže do odkazovej premennej MôjPočítač ukladáme objektovú referenciu identifikujúcu kamarátov počítač, prepisujeme pôvodnú objektovú referenciu, ktorá bola v premennej MôjPočítač uskladnená.

2. Prvá z dvoch inštancií anonymnej triedy sa tak stáva nedosiahnuteľnou z programového kódu. Je to prirodzené, pretože na túto inštanciu bola naviazaná len jedna objektová referencia, a tú sme priradením zlikvidovali. Prvá inštancia bude preto vo vhodnom okamihu automatickým správcom pamäte označená ako nepotrebná. Keď sa inštancia triedy stane nepotrebnou, bude z riadenej haldy implicitne uvoľnená (samozrejme za predpokladu, že nevyžaduje svoju explicitnú finalizáciu).

3. Vzhľadom na to, že obe prítomné odkazové premenné majú rovnaký obsah, môžeme k cieľovej inštancii anonymnej triedy pristupovať prostredníctvom ktorejkoľvek z nich. Ak by sme chceli do kamarátovho počítača nainštalovať pevný disk o kapacite pol terabajtu, tak by to mohlo vyzerať aj takto:

```
Dim MôjPočítač = New With {.Procesor = "Intel Core 2 Duo", _
   .TaktProcesora = 4.26F, .OperačnáPamäťVGB = 2, _
   .PevnýDiskVGB = 320}

Dim KamarátovPočítač = New With {.Procesor = "AMD Turion 64 X2", _
    .TaktProcesora = 4.0F, .OperačnáPamäťVGB = 2, _
    .PevnýDiskVGB = 320}

MôjPočítač = KamarátovPočítač
```

```
'Hoci používame prvú odkazovú premennú, pracujeme s druhou
'inštanciou anonymnej triedy.
 MôjPočítač.PevnýDiskVGB = 500
```

Anonymné triedy môžu byť obdarené tzv. kľúčovými vlastnosťami. Túto skutočnosť naznačíme v inštanciačnom príkaze pomocou kľúčového slova Key, ktoré uvedieme pred identifikátormi vlastností. Kľúčové vlastnosti sú určené len na čítanie, teda prostredníctvom nich nemôžu byť menené hodnoty súkromných dátových položiek inštancií anonymnej triedy.

```
Dim MojaKniha = New With {Key .Názov = "The Art of Computer Programming", _
   Key .Autor = "Donald E. Knuth", Key .Vydavateľstvo = "Addison-Wesley", _
   Key .PočetStrán = 650, Key .RokVydania = 1997}
```

Všetky verejne prístupné skalárne vlastnosti kompilátorom emitovanej anonymnej triedy sú kľúčové a určené len na čítanie. Pri anonymných triedach plnia kľúčové vlastnosti presne tú istú úlohu ako vlastnosti určené len na čítanie pri pomenovaných triedach (áno, v definícii tých sa nachádza modifikátor ReadOnly).

Akákoľvek snaha o priamu modifikáciu súkromných dátových položiek inštancie triedy pomocou kľúčovej vlastnosti bude považovaná za neprípustnú:

```
'Priradenie nie je možné, pretože vlastnosť je kľúčová.
MojaKniha.RokVydania = 1998
```

Kapitola 3

Rozširujúce metódy

Microsoft® Visual Studio® 2008

Rozširujúce metódy

Rozširujúce metódy sú novou syntakticko-sémantickou konštrukciou, ktorá nám dovoľuje rozširovať funkcionalitu už existujúcich používateľsky deklarovaných hodnotových a odkazových dátových typov bez nutnosti vytvárania odvodených typov. V tejto kapitole budeme skúmať schopnosti rozširujúcich metód najmä v súvislosti s triedami, ako hlavnými zástupcami objektových dátových typov v jazyku Visual Basic 2008. Ako rozširujúca metóda môže vystupovať buď procedúra Sub, alebo funkcia, ktorá je definovaná v module a ktorá disponuje atribútom <Extension()> z menného priestoru System.Runtime.CompilerServices. Aby sme mohli prakticky demonštrovať použitie rozširujúcich metód, potrebujeme jednu triedu. Pre potreby tohto cvičenia sme navrhli triedu BankovýÚčet, ktorá je deklarovaná nasledujúcim spôsobom:

```
Class BankovýÚčet
    'Súkromné dátové položky triedy.
    Private m_Majiteľ, Číslo As String
    Private m_MinimálnyZostatok, m_PočiatočnýVklad As UShort
    Private m_Stav As UInteger

    'Verejne prístupný parametrický konštruktor triedy.
    Public Sub New(ByVal Majiteľ As String, ByVal Číslo As String, _
       Optional ByVal MinimálnyZostatok As UShort = 500, _
       Optional ByVal PočiatočnýVklad As UShort = 1000)
        m_Majiteľ = Majiteľ
        Číslo = Číslo
        m_MinimálnyZostatok = MinimálnyZostatok
        m_PočiatočnýVklad = PočiatočnýVklad
        m_Stav = 0
        m_Stav += PočiatočnýVklad
        Console.WriteLine("Účet bol založený s počiatočným vkladom " & _
           PočiatočnýVklad & " Sk.")
    End Sub

    'Verejne prístupná vlastnosť triedy.
    Public Property Stav() As UInteger
        Get
            Return m_Stav
        End Get
        Set(ByVal value As UInteger)
            m_Stav = value
        End Set
    End Property

    'Verejne prístupná vlastnosť triedy určená len na čítanie.
    Public ReadOnly Property MinimálnyZostatok() As UShort
        Get
            Return m_MinimálnyZostatok
        End Get
    End Property
End Class
```

Komentár k triede: Inštancia triedy BankovýÚčet bude reprezentovať virtuálny účet. V procese objektovej abstrakcie sme nášmu účtu prisúdili tieto atribúty: majiteľ, číslo, minimálny zostatok, počiatočný vklad a stav. Inicializáciu objektu má na starosti verejný parametrický konštruktor, v ktorého signatúre sú posledné dva formálne parametre voliteľné (modifikátor Optional). Ak

teda vývojár nešpecifikuje argumenty pre voliteľné formálne parametre, budú absentujúce argumenty substituované implicitnými inicializačnými hodnotami. Povedané menej technicky, pokiaľ nebude stanovené inak, všetky založené bankové účty budú disponovať počiatočným vkladom vo výške 1000 Sk a o polovicu menším minimálnym zostatkom. Ak vznikne inštancia triedy BankovýÚčet, parametrický konštruktor uskutoční jej inicializáciu a informuje používateľa o vzniku účtu. Každý účet má kolekciu už spomenutých atribútov, pričom k dvom z nich sa dá pristupovať pomocou vlastností. Vlastnosť Stav slúži na zistenie aktuálneho stavu účtu, zatiaľ čo vlastnosť MinimálnyZostatok je určená len na čítanie minimálneho zostatku na účte. S takto navrhnutou deklaráciou triedy BankovýÚčet budeme ďalej pracovať.

Možnosti triedy BankovýÚčet sa chystáme zdokonaliť použitím rozširujúcim metód. Do programového modulu, v ktorom sa už nachádza trieda BankovýÚčet, vložíme príkaz Imports uskutočňujúci import menného priestoru System.Runtime.CompilerServices.

```
'Zavedenie menného priestoru, v ktorom sa nachádza atribút <Extension()>.
Imports System.Runtime.CompilerServices
```

Zavedenie menného priestoru je elegantné riešenie, vďaka ktorému nemusíme zapisovať kvalifikované meno atribútu <Extension()>. Definíciu prvej rozširujúcej metódy uvádzame ďalej:

```
<Extension()> _
Public Sub Vložiť(ByVal Účet As BankovýÚčet, _
   ByVal Suma As UInteger)
    Účet.Stav += Suma
    Console.WriteLine("Po vložení sumy " & Suma & _
       " Sk je na účte " & Účet.Stav & " Sk.")
End Sub
```

Rozširujúca metóda je syntakticky stelesnená verejne prístupnou procedúrou Sub, ktorej definícia je zreteľne označená atribútom <Extension()>. Procedúra je parametrická a pravdu povediac, rozširujúca metóda musí byť vždy parametrická, pričom dátový typ prvého formálneho parametra sa musí zhodovať s typom, ktorého funkcionalitu metóda rozširuje. V našom prípade sa metóda Vložiť viaže k triede BankovýÚčet, a preto je táto trieda typom prvého formálneho parametra metódy. Pomocou metódy Vložiť budeme chcieť uskutočniť vklad istej sumy peňazí na bankový účet. Preto sme do signatúry metódy doplnili ďalší formálny parameter, prostredníctvom ktorého určíme výšku nášho vkladu. V tele rozširujúcej metódy sú umiestnené príkazy uskutočňujúce zamýšľanú bankovú transakciu. Po úspešnom vložení sumy na účet vypíšeme správu o tom, ako sa táto akcia odrazila na aktuálnom stave účtu. Všimnime si, že stav účtu získame pomocou vlastnosti Stav, ktorá je prístupná cez prvý formálny parameter rozširujúcej metódy (odkazovú premennú Účet).

Ak za prispenia rozširujúcej metódy umožníme realizovať vklady, mali by sme analogicky dopracovať aj spracovanie úhrad. Preto pridávame zdrojový kód nasledujúcej rozširujúcej metódy:

```
<Extension()> _
Public Sub Uhradiť(ByVal Účet As BankovýÚčet, _
   ByVal Suma As UInteger)
    If ((CInt(Účet.Stav) - CInt(Suma)) < Účet.MinimálnyZostatok) Then
        Console.WriteLine("Transakcia nebola uskutočnená " & _
```

```
            "z dôvodu prekročenia " & vbCrLf & "minimálneho zostatku " & _
            "na účte.")
    Else
        Účet.Stav -= Suma
        Console.WriteLine("Po uhradení sumy " & Suma & _
            " Sk je na účte " & Účet.Stav & " Sk.")
    End If
End Sub
```

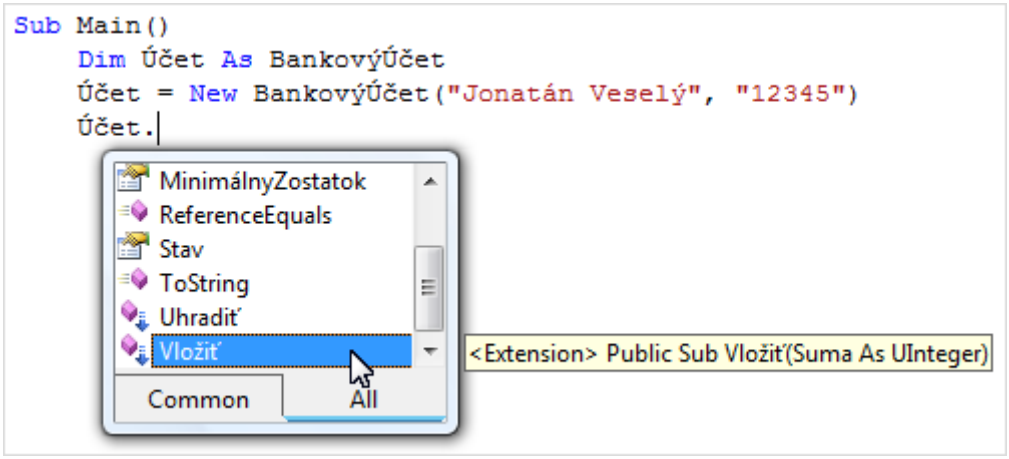
Rozširujúca metóda Uhradiť je rovnako ako jej predchodkyňa opatrená atribútom <Extension()>. Metóda je parametrická, typom prvého formálneho parametra i naďalej zostáva trieda, ktorej schopnosti zveľaďujeme. Druhý formálny parameter predstavuje sumu, ktorá bude z nášho účtu uhradená. V tele rozširujúcej metódy je implementovaná aplikačná logika na uhradenie stanovenej sumy z účtu. Keďže na bankovom účte musí vždy zostať hodnota daná minimálnym zostatkom, musíme skontrolovať, či je plánovaná transakcia vôbec realizovateľná. Ak áno, uhrádzame sumu z bankového účtu. V opačnom prípade zobrazujeme informačnú správu podávajúcu vysvetlenie príčiny neuskutočnenia bankovej operácie.

Poznámka: Možno nie je na prvý pohľad zrejmé, prečo v hlavičke rozhodovacieho príkazu If...Else vykonávame explicitné typové konverzie hodnôt uložených vo vlastnosti Stav bankového účtu a vo formálnom parametri Suma. Je to preto, že hodnoty oboch týchto entít sú neznamienkového dátového UInteger. To je koniec koncov pochopiteľné, pretože nemôžeme uhrádzať zápornú sumu a takisto nie je možné, aby stav bankového účtu bol záporný (ide o bežný bankový účet, nie kontokorent). Aby sme mohli vykonať kontrolu prípustnosti bankovej operácie, spracúvame aritmetický výraz Účet.Stav – Suma, ktorého hodnota môže byť záporná. Problém je však v tom, že entity dátového typu UInteger nemôžu nadobúdať záporné hodnoty. Preto pristupujeme k explicitnej typovej konverzii, v rámci ktorej pre potreby korektného vyhodnotenia aritmetického výrazu meníme dátový typ hodnôt z UInteger na Integer. Keby sme konverziu nespravili, v prípade pretečenia neznamienkového typu by sme boli konfrontovaní s chybou generovanou počas spracovania programu.

Výborne! Konečne sa dostávame k finálnemu bodu, ktorým je použitie rozširujúcich metód v súvislosti s inštanciou triedy BankovýÚčet.

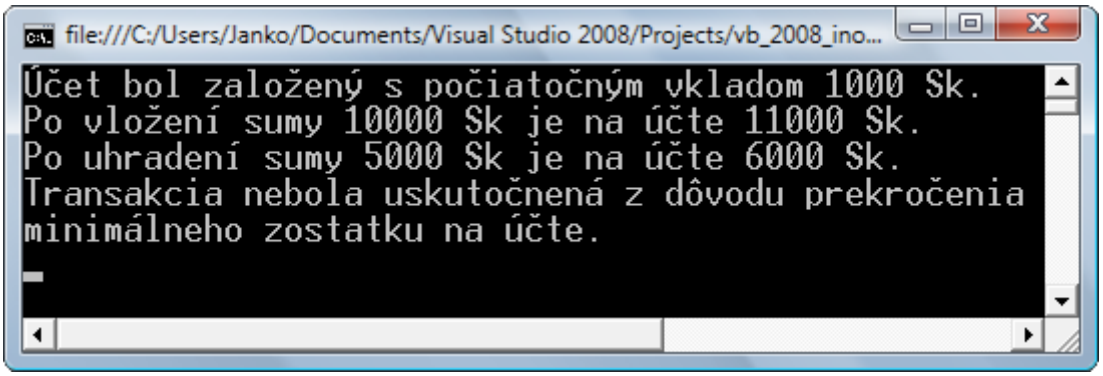
```
Sub Main()
    Dim Účet As BankovýÚčet
    Účet = New BankovýÚčet("Jonatán Veselý", "12345")
    Účet.Vložiť(10000)
    Účet.Uhradiť(5000)
    Účet.Uhradiť(7000)
End Sub
```

Tip: Keď za názvom odkazovej premennej Účet zapíšeme bodkový operátor (.), senzitívna technológia IntelliSense nám ponúkne zoznam metód, ktoré sú v danom kontexte aplikovateľné. Ako môžeme zaregistrovať, v zozname metód sa nachádzajú aj dve rozširujúce metódy, ktoré sú vizuálne stvárnené ikonou s fialovou kockou a modrou šípkou smerujúcou nadol (obr. 9).



Obr. 9: Rozširujúce metódy v zozname členov inštancie triedy, ktorý nám ponúka senzitívna technológia Microsoft IntelliSense

Pri pohľade na inštanciačný príkaz vidíme, že sme pre Jonatána Veselého vytvorili nový bankový účet s počiatočným vkladom 1000 Sk a minimálnym zostatkom 500 Sk (argumenty sme odovzdali len „povinným“ formálnym parametrom inštančného konštruktora). V ďalšom kroku sme na účet vložili 10000 korún. Potom sme uhradili 5000 korún. Ak v duchu počítate s nami, po úhrade sa na účte nachádza 6000 Sk. Nasledujúci pokus o úhradu sumy 7000 Sk preto nebude úspešný: účet túto transakciu odmietne, lebo by prišlo k prekročeniu minimálneho zostatku. Priebeh bankových operácií dokumentuje obr. 10.



Obr. 10: Realizácia bankových operácií pomocou rozširujúcich metód inštancie triedy

Rozširujúce metódy sú nepochybne interesantným programovým rysom jazyka Visual Basic 2008. Ich pravý potenciál sa však ukáže až vo chvíli, keď sa podujmeme rozšíriť funkcionalitu triedy, ktorú nebudeme sami deklarovať. Vynikajúcou vlastnosťou rozširujúcich metód je schopnosť „vylepšiť“ aj triedy, ktoré sme sami nenavrhli. Nadchádzajúca praktická ukážka predvádza, ako triedu Thread situovanú v mennom priestore System.Threading rozšíriť o metódu, ktorá bude zisťovať informácie o programovom vlákne.

Riaďte sa prosím týmito inštrukciami:

1. Príkazom Imports zavedieme menný priestor System.Threading.
2. Napíšeme definíciu rozširujúcej metódy s názvom ZistiťInformácieOVlákne:

```
<Extension()> _
Public Sub ZistiťInformácieOVlákne(ByVal Vlákno As Thread)
   Console.WriteLine("Informácie o vlákne: " & vbCrLf)
   Console.WriteLine("Meno vlákna: " & Vlákno.Name & vbCrLf & _
      "ID vlákna: " & Vlákno.ManagedThreadId & vbCrLf & _
      "Priorita vlákna: " & Vlákno.Priority.ToString())
End Sub
```

Keďže typom formálneho parametra metódy je trieda Thread, metóda bude modifikovať schopnosti tejto triedy. Ak ste sa s triedou Thread dosiaľ nestretli, tak pripájame stručnú charakteristiku. Inštancia trieda Thread predstavuje programové vlákno, ktoré definujeme ako samostatnú sekvenciu programových príkazov, ktoré sú podrobené exekúcii. Implicitne je každá aplikácia .NET pripravená v jazyku Visual Basic 2008 jednovláknová, čo znamená, že vlastní len jedno (tzv. primárne) programové vlákno. Na primárnom vlákne sa odohráva spracovanie Just-In-Time (JIT) skompilovaných programových príkazov (a teda priame spracovanie strojového kódu triedy x86 procesorom). Okrem jednovláknových aplikácií môžu vývojári programovať aj viacvláknové aplikácie: to sú aplikácie, ktoré obsahujú aspoň dve programové vlákna. Jedno vlákno však aj naďalej zostáva primárnym vláknom, zatiaľ čo novo vytvorené vlákna sú označované ako vedľajšie, resp. pracovné vlákna. Viacvláknové aplikácie majú mnoho pozitív, z ktorých najčastejšie sa spomínajú širšia dátová priepustnosť a vyššia citlivosť na pokyny používateľa. Problematika programovania viacvláknových aplikácií a ich riadenej exekúcie bohužiaľ obsahovo presahuje zameranie tejto vývojárskej príručky, a preto sa jej nebudeme bližšie venovať.

3. Predtým, ako vytvoríme nové pracovné vlákno, musíme navrhnúť definíciu metódy, ktorá bude na tomto vlákne spracúvaná. Naša metóda bude realizovať iteratívny výpočet členov Fibonacciho postupnosti:

```
Public Sub Fibonacci()
    Dim pole(79) As ULong
    pole(0) = 0
    pole(1) = 1
    For i As Integer = 2 To 79
        pole(i) = pole(i - 1) + pole(i - 2)
    Next
    Console.WriteLine("80. člen Fibonacciho postupnosti " & _
       "má hodnotu " & pole(79) & ".")
End Sub
```

Fibonacciho postupnosť je daná rekurzívno-rekurentným matematickým vzťahom:

$$fib(n) = \begin{cases} 0 & n = 1 \\ 1 & n = 2 \\ fib(n-1) + fib(n-2) & n > 2 \end{cases}$$

Okrem prvých dvoch členov (0 a 1) určíme každý ďalší člen Fibonacciho postupnosti spočítaním dvoch predchádzajúcich členov. Fibonacciho postupnosť má preto takúto podobu: 0, 1, 1, 2, 3, 5, 8, 13, 21, 34 atď. Iteratívne riešenie výpočtu Fibonacciho postupnosti je veľmi rýchle a oveľa efektívnejšie ako zodpovedajúce rekurzívne riešenie. Metóda Fibonacci zistí hodnotu 80. člena Fibonacciho postupnosti a zobrazí ju v okne

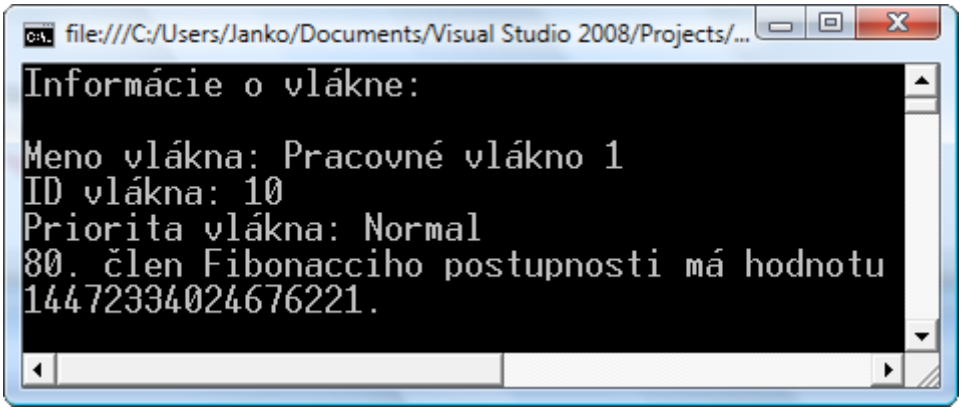
príkazového riadka. Na uskladnenie členov postupnosti používame jednorozmerné pole s 80-timi prvkami. Prvé dva prvky poľa inicializujeme okamžite a ďalšie prvky potom rekurentne vypočítavame a uchovávame ich v pripravenom poli.

4. Do tela hlavnej metódy Main vložíme príkazy vytvárajúce nové programové vlákno a volajúce rozširujúcu metódu:

```
Sub Main()
    Dim PracovnéVlákno As Thread
    'Kreácia pracovného vlákna.
    PracovnéVlákno = New Thread(New ThreadStart(AddressOf Fibonacci))
    'Pomenovanie pracovného vlákna.
    PracovnéVlákno.Name = "Pracovné vlákno 1"
    'Volanie rozširujúcej metódy.
    PracovnéVlákno.ZistiťInformácieOVlákne()
    'Spustenie vlákna.
    PracovnéVlákno.Start()
    Console.Read()
End Sub
```

Programové vlákno je inštanciou triedy Thread. Keď sa pozrieme na inštanciačný príkaz, zistíme, že parametrickému konštruktoru odovzdávame inštanciu delegáta, ktorá zapuzdruje adresu metódy Fibonacci. Adresu metódy získame pomocou operátora AddressOf, ktorý mimochodom patrí do skupiny špeciálnych operátorov (spoločne s operátormi GetType, If a TypeOf). Po vytvorení sa pracovné vlákno nachádza v nespustenom (Unstarted) stave. Príkazy na vlákne budú spracované až vtedy, keď zavoláme metódu Start. Ešte predtým, ako sa tak stane, však novo vytvorené vlákno pomenúvame. Pomenovanie pracovného vlákna je vhodnou technikou, ktorá nám uľahčí rozpoznanie vlákna a svoje silné stránky preukazuje najmä pri ladení viacvláknových aplikácií .NET. V ďalšej etape aktivujeme rozširujúcu metódu ZistiťInformácieOVlákne, ktorá zobrazí názov vlákna, identifikačné číslo (ID) vlákna a prioritu vlákna. Napokon aktivujeme metódu Start, čím zahájime exekúciu príkazov na pracovnom vlákne.

Po spustení programu nám budú najskôr ponúknuté požadované informácie o pracovnom vlákne a hneď nato spoznáme hodnotu 80. člena Fibonacciho postupnosti (obr. 11).



Obr. 11: Informácie o pracovnom vlákne ponúka rozširujúca metóda

Rozširujúce metódy spolupracujú aj s používateľsky deklarovanými hodnotovými dátovými typmi, akými sú štruktúry.

```
'Deklarácia štruktúry.
Structure Procesor
    Dim PočetJadier As Byte
    Dim PamäťL2VMB As Byte
    Dim TaktJadra As Single
    Dim HT As Boolean
End Structure
```

Štruktúra opisuje mikroprocesor, pričom sleduje počet exekučných jadier, kapacitu vyrovnávacej pamäte druhej úrovne (L2 cache), hodinový takt jadra v GHz a implementáciu technológie Hyper-Threading (HT).

```
<Extension()> _
Public Sub ZistiťPočetJadier(ByVal CPU As Procesor)
    Select Case CPU.PočetJadier
       Case 1
           Console.WriteLine("Máte 1-jadrový procesor.")
       Case 2
           Console.WriteLine("Máte 2-jadrový procesor.")
       Case 3
           Console.WriteLine("Máte 3-jadrový procesor.")
       Case 4
           Console.WriteLine("Máte 4-jadrový procesor.")
       Case Is >= 5
           Console.WriteLine("Máte viacjadrový procesor.")
    End Select
End Sub
```

Definícia rozširujúcej metódy analyzuje počet jadier procesora a vypisuje používateľovi informačnú správu. Použitie rozširujúcej metódy už snáď nemôže byť jednoduchšie:

```
Sub Main()
'Vytvorenie inštancie štruktúry.
    Dim AMD_Phenom As Procesor
'Inicializácia založenej inštancie štruktúry.
    With AMD_Phenom
        .PočetJadier = 4
        .TaktJadra = 2.3F
        .PamäťL2VMB = 2
        .HT = False
    End With
    'Aktivácia rozširujúcej metódy v súvislosti s inštanciou štruktúry.
    AMD_Phenom.ZistiťPočetJadier()
    Console.Read()
End Sub
```

Možno často potrebujete abecedne zoradiť a následne zobraziť už zoradený obsah jednorozmerného poľa textových reťazcov. Čo tak zveriť túto úlohu do rúk samostatnej rozširujúcej metóde?

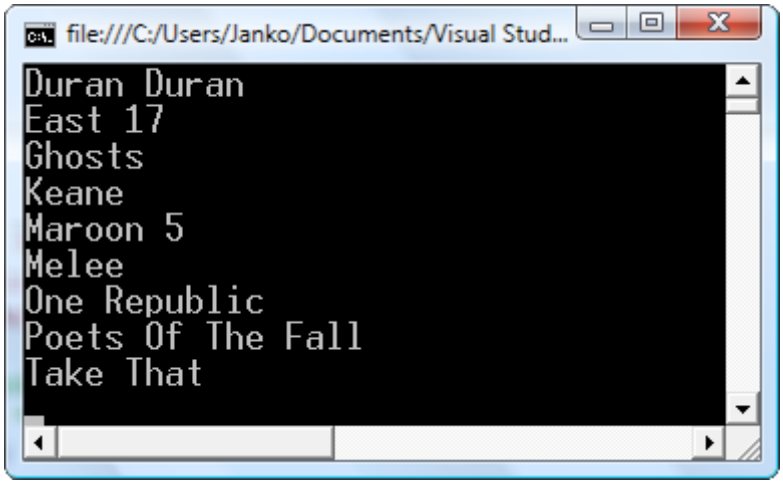
```
<Extension()> _
Public Sub ZoradiťAZobraziť(ByRef Pole() As String)
    'Zoradenie prvkov poľa podľa abecedy.
    Array.Sort(Pole)
    'Zobrazenie zoradenej postupnosti prvkov.
    For Each Prvok As String In Pole
        Console.WriteLine(Prvok)
    Next
End Sub
```

Rozširujúca metóda s názvom ZoradiťAZobraziť definuje jeden formálny parameter, ktorý je schopný prijať objektovú referenciu na jednorozmerné pole inštancií triedy System.String. Zdieľaná metóda Sort triedy Array z menného priestoru System zabezpečí zoradenie poľa pomocou algoritmu rýchleho triedenia dát (Quicksort). Algoritmus rýchleho triedenia je vhodným adeptom na zoraďovanie dát uložených v poliach, pretože v bežných podmienkach dosahuje vyhovujúcu, presnejšie lineárno-logaritmickú asymptotickú časovú zložitosť $O(n \times log(n))$, kde n je rozsah spracúvaného poľa. Dodajme, že v istých prípadoch sa časová zložitosť algoritmu rýchleho triedenia dát zhorší až na $O(n^2)$, čím sa algoritmus zaradí medzi algoritmy s kvadratickou časovou zložitosťou. Len čo sú prvky poľa zoradené od „a po z“, rozširujúca metóda ich zobrazí pomocou cyklu For Each.

Praktická ukážka použitia práve definovanej rozširujúcej metódy by mohla mať nasledujúcu podobu:

```
Sub Main()
    Dim HudobnéSkupiny() As String = _
    { _
        "Maroon 5", "One Republic", "Melee", _
        "Ghosts", "Poets Of The Fall", "East 17", _
        "Keane", "Duran Duran", "Take That" _
    }
    HudobnéSkupiny.ZoradiťAZobraziť()
    Console.Read()
End Sub
```

Výstup aplikácie uvádzame na obr. 12.



Obr. 12: Rozširujúce metódy sa kamarátia aj s poľami

Ako programátori však môžeme chcieť, aby bola pôsobnosť predchádzajúcej rozširujúcej metódy väčšia. Jazyk Visual Basic 2008 nám nebráni prepracovať štandardnú rozširujúcu metódu na generickú rozširujúcu metódu. Ak navrhneme rozširujúcu metódu ako generickú, poradí si so zoradením a zobrazením nielen textových reťazcov, ale aj hodnôt iných dátových typov.

```
'Definícia generickej rozširujúcej metódy.
<Extension()> _
Public Sub ZoradiťAZobraziť_Gen(Of T)(ByVal Vzorka() As T)
    Array.Sort(Vzorka)
    For Each Prvok As T In Vzorka
        Console.WriteLine(Prvok)
    Next
End Sub
```

Každá generická metóda má dve súpravy formálnych parametrov. Tieto pravidlá platia aj pre akúkoľvek generickú rozširovaciu metódu. Prvá súprava obsahuje jeden generický formálny parameter, ktorý sa nazýva tiež typový parameter. Ak je v signatúre metódy prítomný typový parameter, automaticky ide o generickú metódu. V druhej súprave formálnych parametrov sú definované formálne parametre generickej metódy, ktoré môžu, no nemusia, byť rovnako generické. V prípade definovanej generickej rozširujúcej metódy je jej formálny parameter pomenovaný Vzorka a je takisto generický (do tohto formálneho parametra môže byť uložená objektová referencia nasmerovaná na jednorozmerné pole generického dátového typu). Telo generickej rozširujúcej metódy sme upravili tak, aby mohla metóda uskutočňovať operácie s jednorozmerným generickým poľom.

Keď sme pôvodnú rozširujúce metódu obdarili novou generickou funkcionalitou, môžeme ju otestovať:

```
Sub Main()
    Dim HudobnéSkupiny() As String = _
    { _
        "Maroon 5", "One Republic", "Melee", _
        "Ghosts", "Poets Of The Fall", "East 17", _
        "Keane", "Duran Duran", "Take That" _
    }

    Dim Čísla() As Integer = {8, 2, 6, 3, 0, 1, 7, 4}
    'Volanie generickej rozširujúcej metódy s poľom textových reťazcov.
    HudobnéSkupiny.ZoradiťAZobraziť_Gen()
    'Volanie generickej rozširujúcej metódy s poľom celých čísel.
    Čísla.ZoradiťAZobraziť_Gen()
End Sub
```

Typový argument, teda konkrétny typ prvkov poľa, s ktorými bude rozširujúca metóda pracovať, dodáme pri jej volaní. Prvá aktivácia metódy pracuje s textovými reťazcami združenými v jednorozmernom poli. Druhé volanie metódy zase usporiada prvky celočíselného poľa.

Typová inferencia a generické metódy

Kompilátor jazyka Visual Basic 2008 nepoužíva typovú inferenciu len pri definičnej inicializácii lokálnych entít. S variantom strojového odvodzovania sa stretávame rovnako aj pri volaní generických metód s určitou množinou vstupných hodnôt (argumentov). Uvažujme túto definíciu generickej metódy:

```
Public Sub InformácieOObjekte(Of T As Class)(ByVal Objekt As T)
    Console.WriteLine("Informácie o objekte: " & vbCrLf & _
       "Trieda: " & Objekt.GetType().Name & vbCrLf & _
       "Bázová trieda: " & Objekt.GetType().BaseType.Name & vbCrLf & _
       "Hešový kód: " & Objekt.GetHashCode())
End Sub
```

Generická metóda analyzuje objekt, na ktorý je nasmerovaná referencia tejto metóde odovzdaná. Na typový parameter T je kladené obmedzenie: tento parameter môže byť inicializovaný iba odkazom na inštanciu odkazového dátového typu.

```
Sub Main()
    'Ukážka typovej inferencie: Dátový typ typového argumentu odovzdávaného
    'generickej metóde si kompilátor odvodí sám.
    InformácieOObjekte("Visual Basic 2008")
    Console.Read()
End Sub
```

Rozoberme bližšie príkaz, ktorý volá generickú metódu. Typovým argumentom je textová konštanta typu String. Konkrétny dátový typ typového argumentu kompilátor diagnostikuje v procese typovej inferencie. Z technického hľadiska je samozrejme celý proces zložitejší. Hoci my špecifikujeme len textový literál, kompilátor vie, že musí založiť inštanciu triedy System.String a do nej nakopírovať všetky textové znaky, z ktorých je literál zložený.

Kapitola 4

λ-výrazy a λ-kalkul

Microsoft® Visual Studio® 2008

λ-výrazy a λ-kalkul

V kolekcii syntakticko-sémantických inovácií jazyka Visual Basic 2008 majú svoje pevné miesto λ-výrazy (lambda výrazy). Lambda výrazy sú súčasťou širšej matematicko-informatickej vedeckej teórie, ktorá sa nazýva λ-kalkul (lambda kalkul). λ-kalkul patrí k jednej z paradigiem tvorby počítačového softvéru, ktorému sa vraví funkcionálne programovanie. Funkcionálne programovanie sa na stavbu softvéru pozerá inak ako imperatívne programovanie, na ktoré sme zvyknutí. Pripomeňme, že v ponímaní imperatívnej paradigmy je program považovaný za konečnú a neprázdnu množinu inštrukcií, ktoré sú spracúvané jedna za druhou a ktoré formujú jednotlivé kroky implementovaných imperatívnych algoritmov. Imperatívne programovanie je vývojárom blízke, pretože sa s ním stretávajú už od nepamäti: jazyky BASIC, C, C++, Visual Basic a mnohé ďalšie môžeme zaradiť do kategórie imperatívnych jazykov. Na druhej strane, funkcionálne programovanie je postavené na aparáte λ-kalkulu a počítačový program definuje z matematického hľadiska. Z pohľadu funkcionálneho programovania je program indikovaný ako konečná a neprázdna množina funkcií. Na funkcie, ktoré zapuzdrujú požadované algoritmy, sú aplikované rôzne transformácie (najmä redukcie a konverzie), vďaka ktorým sa program dopracuje k požadovanému výsledku. λ-kalkul intenzívne pracuje s λ-výrazmi, prostredníctvom ktorých sú funkcie anonymne definované.

λ-kalkul vo svojom jadre pracuje s unárnymi funkciami, teda s funkciami, ktoré sú schopné pracovať len s jedným argumentom. Argumentom unárnej funkcie môže byť ďalšia unárna funkcia a unárna funkcia môže inú unárnu funkciu vracať tiež ako svoju návratovú hodnotu. Unárna funkcia je definovaná anonymne, čo znamená, že ide o anonymnú funkciu, ktorá nemá vlastný identifikátor. Na definíciu unárnych anonymných funkcií sa používajú λ-výrazy. λ-výraz vytvára funkciu a podáva rovnako informácie o jej formálnom parametre. Napríklad matematický zápis funkcie $f(x) = 5 + x$ vyjadríme nasledujúcim λ-výrazom: $(\lambda\, \mathrm{x} \mid 5 + x)$. λ-výrazy zapisujeme vždy do zátvoriek, presne tak, ako sme uviedli. Každý λ-výraz formujú tieto štyri časti:

1. lambda symbol (λ),
2. formálny parameter funkcie (x),
3. vertikálny oddeľovač (|),
4. telo funkcie (5 + x).

V matematike by sme na výpočet hodnoty funkcie so vstupným argumentom 2 použili zápis $f(2)$. V λ-kalkule zapíšeme výraz $(\lambda\, \mathrm{x} \mid 5 + x)\ 2$, kde hodnota 2 predstavuje argument, ktorý bude dosadený do formálneho parametra (x) a použije sa v tele funkcie na určenie jej návratovej hodnoty. Návratová hodnota funkcie je v skutočnosti hodnota λ-výrazu. Je zrejmé, že hodnotou λ-výrazu bude číslo 7.

λ-výraz je z formálnej stránky definovaný nasledujúcim spôsobom:

1. X je premenná, ktorá je definovaná svojím identifikátorom.
2. $(\lambda\, \mathrm{X} \mid V)$ je abstrakcia, v rámci ktorej je X premenná a V je λ-výraz. Termín „abstrakcia“ môžeme substituovať termínom „definícia anonymnej funkcie“.

3. $(\lambda \text{ X} \mid V)\ A$ je aplikácia, teda volanie anonymnej funkcie s argumentom A.

Poznámka: Identifikátorom premennej X môže byť aj symbol zložený z viacerých znakov. Premenné, ktoré sa nachádzajú v λ-výrazoch, rozdeľujeme na viazané a voľné. Klasifikačné kritérium je celkom jednoduché: Pokiaľ je premenná asociovaná s lambda symbolom, je ponímaná ako viazaná. V opačnom prípade ide o voľnú premennú. V λ-výraze $(\lambda \text{ x} \mid 2x + 1)$ je x viazanou premennou. Keď predchádzajúci λ-výraz upravíme na $(\lambda \text{ x} \mid 2x + y)$, tak x zostáva i naďalej viazanou premennou, no premenná y je voľná (neexistuje žiaden λ-symbol, ktorý by bol s ňou spojený).

Keďže λ-kalkul dokáže pracovať len s unárnymi funkciami, musíme „reálnu" funkciu s viacerými premennými (formálnymi parametrami) v tomto prostredí modelovať ako kompozíciu jednej unárnej funkcie, ktorá ako argument prijíma ďalšiu unárnu funkciu.

Uvažujme tento príklad: Nech je počet tranzistorov umiestnených v jednom viacjadrovom procesore daný funkciou $f_1(n) = 10^8 \times n$, kde n je počet exekučných jadier uložených v procesorom balení. Povedzme, že budeme chcieť sledovať celkový počet inštalovaných tranzistorov v kolekcii viacjadrových procesorov, ktorá je vyrobená počas jedného výrobného cyklu. Tejto relácii nech zodpovedá funkcia $f_2(p, n) = p \times 10^8 \times n$, kde p je celkový počet vyprodukovaných procesorov. Definíciu funkcie f_2 by sme mohli zapísať aj takto: $f_2(p, f_1(n)) = p \times f_1(n)$. Použitím nástrojov λ-kalkulu zapíšeme vzťahy medzi funkciami nasledujúcim spôsobom:

1. abstrakcia: $(\lambda \text{ n} \mid 10^8 \times n)$
2. abstrakcia: $(\lambda \text{ p} \mid (\lambda \text{ n} \mid p \times (10^8 \times n)))$

Aplikácia pre 100 štvorjadrových procesorov, ktoré sú vyrobené v jednom cykle, bude potom vyzerať takto:

$(\lambda \text{ p} \mid (\lambda \text{ n} \mid p \times (10^8 \times n)))\ 100\ 4$

V zloženom λ-výraze možno rozpoznať vnútornú (vnorenú) abstrakciu $(\lambda \text{ n} \mid p \times (10^8 \times n))$. Vyhodnotenie zloženého λ-výrazu prebieha v týchto krokoch:

1. Najskôr nahradíme viazané premenné (p a n) konkrétnymi hodnotami:

$$(\lambda \text{ p} \mid (\lambda \text{ n} \mid 100 \times (10^8 \times 4)))$$

2. Vyhodnotíme vnútornú a vonkajšiu abstrakciu:

$$(\lambda \text{ p} \mid (\lambda \text{ n} \mid 10^{10} \times 4))$$

3. Výsledná hodnota analyzovaného λ-výrazu je 4 x 10^{10}. Na základe vypočítanej hodnoty môžeme povedať, že 100 procesorov osadených štyrmi jadrami obsahuje 40 miliárd tranzistorov.

Vyhodnocovanie λ-výrazov sa uskutočňuje pomocou vyhodnocovacích pravidiel, ktoré sa nazývajú konverzie. λ-kalkul pozná tri základné typy konverzných operácií: α-konverzia, β-redukcia a η-konverzia.

λ-výrazy v jazyku Visual Basic 2008

λ-výraz sa v jazyku Visual Basic 2008 definuje pomocou kľúčového slova Function:

```
Function(a As Integer) a + 5
```

Aby sme lepšie porozumeli predstavenému syntaktickému zápisu, budeme hľadať analógie so všeobecným zápisom funkcie v λ-kalkule. Kľúčové slovo Function substituuje symbol λ, pričom dáva kompilátoru na známosť, že sa chystáme definovať platný λ-výraz. Za kľúčovým slovom Function sa v zátvorkách nachádza definícia premennej, ktorá vystupuje ako formálny parameter anonymnej funkcie. V našom prípade je formálnym parametrom celočíselná premenná s identifikátorom a a explicitne určeným dátovým typom. Výraz a + 5 určuje manipulačnú operáciu, ktorá bude aplikovaná na hodnotu formálneho parametra. V súčinnosti s teóriou λ-kalkulu môžeme definíciu λ-výrazu stotožniť s abstrakciou unárnej funkcie.

Ako sme spomenuli, v λ-kalkule je významná najmä aplikácia unárnej funkcie, ktorá bude v jazyku Visual Basic 2008 vyzerať takto:

```
(Function(a As Integer) a + 5)(4)
```

Aplikácia je volanie unárnej anonymnej funkcie so špecifikovaným argumentom (v našom prípade je argumentom diskrétna celočíselná konštanta 4). Podotknime, že kompilátor jazyka Visual Basic 2008 nám neumožní ponechať aplikáciu unárnej funkcie vo vyššie uvedenom tvare. V záujme bezproblémového prekladu zdrojového kódu je nutné zakomponovať aplikáciu unárnej funkcie do priraďovacieho príkazu:

```
Dim lambda1 = (Function(a As Integer) a + 5)(4)
```

Ak na definíciu λ-výrazu nahliadneme z technického hľadiska, zistíme, že kompilátor vygeneruje deklaráciu anonymného delegáta a definíciu anonymnej funkcie, ktorá bude asociovaná s inštanciou zostrojeného delegáta. Vyhodnotenie λ-výrazu bude preto spočívať v inštanciácii anonymného delegáta a v aktivácii spriaznenej funkcie. V tele anonymnej funkcie bude umiestnený príkaz navracajúci hodnotu výrazu a + 5, pričom funkcii bude hodnotou odovzdaný argument 4. Je zrejmé, že návratovou hodnotou anonymnej funkcie bude 9, a práve touto hodnotou bude inicializovaná premenná definovaná na ľavej strane priraďovacieho príkazu.

Pri vyhodnocovaní λ-výrazu sa uplatňuje mechanizmus typovej inferencie. Mechanizmus analyzuje všetky entity, ktoré nedisponujú explicitnými dátovými typmi. Podľa aktuálneho kontextu sú zúčastneným entitám prisudzované implicitné dátové typy. Ako si môžeme všimnúť, jedine formálny parameter a má explicitne určený svoj dátový typ. Keďže kompilátor vie, že hodnotou výrazu a + 5 bude opäť 32-bitové celé číslo, dokáže inferovať typ hodnoty celého λ-výrazu. Presnejšie by sme mohli povedať, že kompilátor odvodí dátový typ návratovej hodnoty

anonymne definovanej unárnej funkcie. Nuž a podľa dátového typu hodnoty λ-výrazu bude determinovaný rovnako aj dátový typ cieľovej premennej. Premenná s identifikátorom lambda1 bude preto premennou typu Integer.

Upozornenie: Ak nepracujeme s aktívnou voľbou Option Strict (Option Strict Off), tak je možné v λ-výraze vynechať explicitné určenie dátového typu formálneho parametra. Potom sa λ-výraz zmení takto:

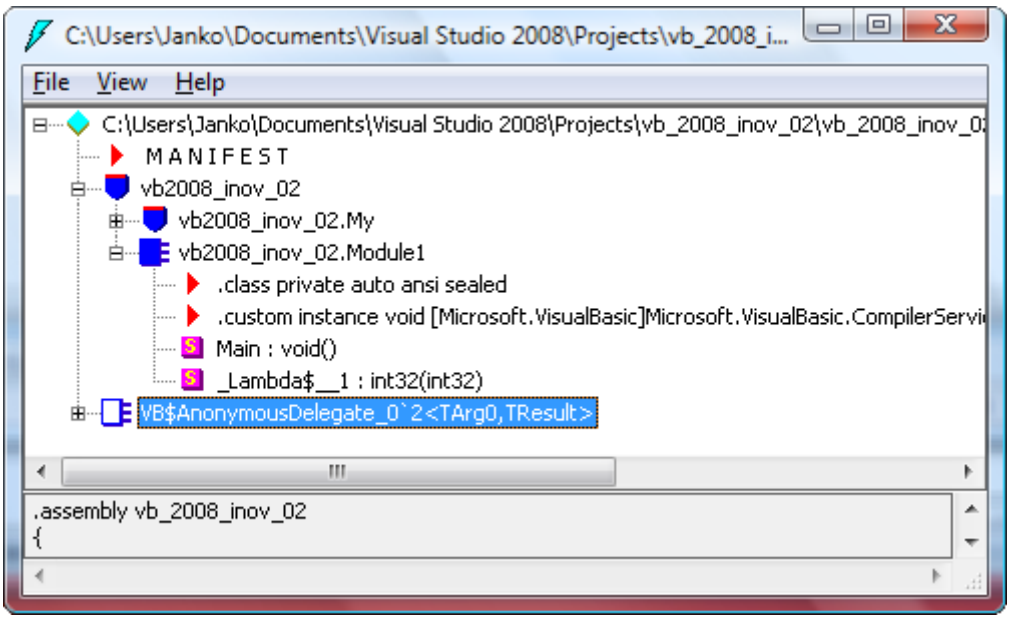
```
Dim lambda1 = (Function(a) a + 5)(4)
```

Túto modifikáciu však v žiadnom prípade neodporúčame, pretože kompilátor priradí formálnemu parametru primitívny odkazový dátový typ Object. Rovnakým typom bude disponovať aj premenná lambda1. Genericita dátového typu Object je dôvodom, prečo bude musieť byť spustený mechanizmus zjednotenia typov (boxing). Bez spracovania komplementárnych operácií totiž nie je možné priamo inicializovať odkazový formálny parameter typu Object celočíselnou konštantou hodnotového dátového typu.

Len čo je premenná lambda1 inicializovaná, môžeme jej hodnotu zobraziť kdekoľvek budeme chcieť:

```
Sub Main()
    Dim lambda1 = (Function(a As Integer) a + 5)(4)
    Console.WriteLine(lambda1)
    Console.Read()
End Sub
```

Tvrdenie, že zapísaný λ-výraz zahajuje zostavenie anonymného delegáta a anonymnej funkcie, možno verifikovať pomocou nástroja IL DASM (obr. 13).



Obr. 13: Nízkoúrovňový pohľad na anonymného delegáta a anonymnú funkciu, ktorú automaticky vytvára kompilátor pri spracovaní λ-výrazu

Anonymne zostavená unárna funkcia sa volá _Lambda$__1 a definuje jeden formálny parameter typu int32. Zdrojový kód jazyka MSIL, ktorý je uložený v tele anonymnej metódy, má nasledujúcu podobu:

```
.method private specialname static int32
        _Lambda$__1(int32 a) cil managed
{
  .custom instance void
   [mscorlib]System.Diagnostics.DebuggerStepThroughAttribute::.ctor() = (
   01 00 00 00 )
  .custom instance void
   [mscorlib]System.Runtime.CompilerServices.CompilerGeneratedAttribute::
   .ctor() = ( 01 00 00 00 )
  // Code size       8 (0x8)
  .maxstack  2
  .locals init ([0] int32 _Lambda$__1)
  IL_0000:  ldarg.0
  IL_0001:  ldc.i4.5
  IL_0002:  add.ovf
  IL_0003:  stloc.0
  IL_0004:  br.s       IL_0006
  IL_0006:  ldloc.0
  IL_0007:  ret
} // end of method Module1::_Lambda$__1
```

Anonymná funkcia je súkromná a statická (v jazyku Visual Basic 2008 by sme namiesto „statická“ povedali „zdieľaná“). V tele funkcie dochádza k inicializácii formálneho parametra, pripočítaniu hodnoty 5 a vráteniu návratovej hodnoty funkcie.

Veľmi zaujímavá je technická interpretácia akcií, ktoré súvisia s vyhodnotením λ-výrazu v jazyku MSIL:

```
.method public static void  Main() cil managed
{
  .entrypoint
  .custom instance void [mscorlib]System.STAThreadAttribute::.ctor() = ( 01
00 00 00 )
  // Code size       35 (0x23)
  .maxstack  2
  .locals init ([0] int32 lambda1)
  IL_0000:  nop
  IL_0001:  ldnull
  IL_0002:  ldftn      int32 vb2008_inov_02.Module1::_Lambda$__1(int32)
  IL_0008:  newobj     instance void class
            VB$AnonymousDelegate_0`2<int32,int32>::.ctor(object,
            native int)
  IL_000d:  ldc.i4.4
  IL_000e:  callvirt   instance !1 class
            VB$AnonymousDelegate_0`2<int32,int32>::Invoke(!0)
  IL_0013:  stloc.0
  IL_0014:  ldloc.0
  IL_0015:  call       void [mscorlib]System.Console::WriteLine(int32)
  IL_001a:  nop
  IL_001b:  call       int32 [mscorlib]System.Console::Read()
  IL_0020:  pop
  IL_0021:  nop
  IL_0022:  ret
```

```
} // end of method Module1::Main
```

Inštrukcia IL_0002: ldftn int32 vb2008_inov_02.Module1::_Lambda$__1(int32) získava smerník na anonymnú unárnu funkciu. V ďalšom kroku (IL_0008) je pomocou inštrukcie newobj založená inštancia anonymného delegáta, ktorá je inicializovaná smerníkom na anonymnú funkciu. Funkcia je napokon nepriamo, cez inštanciu anonymného delegáta, aktivovaná (inštrukcia callvirt s návestidlom IL_000e).

λ-výraz smieme zapísať aj samostatne, nemusíme vždy vytvárať premennú, do ktorej hodnotu tohto výrazu uložíme.

```
Module Module1
    Sub Main()
        'λ-výraz je určený priamo pri volaní metódy.
        Console.WriteLine((Function(a As Integer) a + 5)(4))
        Console.Read()
    End Sub
End Module
```

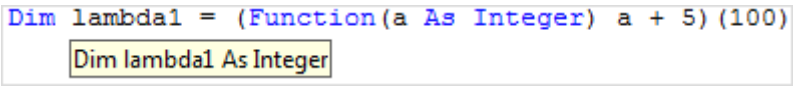
Ak λ-výraz predstavuje len abstrakciu funkcie, môžeme jej aplikáciu uskutočniť v súvislosti so spriaznenou premennou. Viac informácií poskytuje ďalší výpis zdrojového kódu jazyka Visual Basic 2008:

```
Module Module1
    Sub Main()
        'λ-výraz definuje abstrakciu unárnej anonymnej funkcie.
        Dim lambda1 = Function(a As Integer) a + 5
        'Aplikácia funkcie je realizovaná pomocou premennej.
        Dim hodnota As Integer = lambda1(100)
        Console.WriteLine(hodnota)
        Console.Read()
    End Sub
End Module
```

Hoci finálny efekt je rovnaký, medzi uvedenými variantmi existujú významné syntakticko-sémantické rozdiely. Ak v rámci definície premennej dôjde aj k jej inicializácii hodnotou λ-výrazu, tak dátový typ takejto premennej bude implicitne inferovaný kompilátorom. Typ premennej bude priamo kompatibilný s typom hodnoty λ-výrazu. Na druhej strane, ak nebude λ-výraz predstavovať aplikáciu funkcie, dátovým typom premennej bude anonymný delegát.

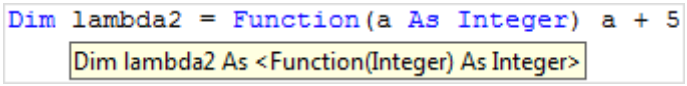
Porovnajme oba charakterizované varianty:

```
'λ-výraz zavádza aplikáciu funkcie. Dátový typ premennej je automaticky
'inferovaný, pričom je zhodný s typom hodnoty λ-výrazu.
Dim lambda1 = (Function(a As Integer) a + 5)(100)
```



Obr. 14: λ-výraz ako aplikácia funkcie

```
'λ-výraz zavádza abstrakciu funkcie. Dátový typ premennej je automaticky
'inferovaný a je to anonymný delegát.
Dim lambda2 = Function(a As Integer) a + 5
```



Obr. 15: λ-výraz ako abstrakcia funkcie

Keďže λ-výrazy sú produktívnou cestou, ako pracovať s delegátmi a anonymnými funkciami, môžeme pomocou nich v mnohých prípadoch značne sprehľadniť a zefektívniť zapísaný zdrojový kód. Na nasledujúcom príklade si ukážeme, akú flexibilitu prinášajú λ-výrazy vývojárom v jazyku Visual Basic 2008.

V našom praktickom experimente navrhneme deklaráciu delegáta a definíciu metódy, s ktorou bude inštancia deklarovaného delegáta prepojená. Metóda bude modelovať nárast výkonnosti pri paralelizácii programu podľa Amdahlovho zákona.

Paralelné programovanie a Amdahlov zákon

Americký vedec Gene Amdahl vynašiel v roku 1967 matematický vzorec pre určenie nárastu výkonnosti medzi sekvenčnou a paralelizovanou verziou počítačového programu. Amdahlov zákon je daný nasledujúcim matematickým vzťahom:

$$N_v = \frac{s + p}{s + \frac{p}{n}}$$

kde:

- N_v je nárast výkonnosti po paralelizácii pôvodne úplne sekvenčného programu.
- s je relatívna početnosť sekvenčných algoritmov počítačového programu.
- p je relatívna početnosť paralelných algoritmov počítačového programu.
- n je počet procesorov počítača, resp. počet exekučných jadier viacjadrového procesora.

Keďže súčet relatívnych početností sekvenčných a paralelných algoritmov počítačového programu bude vždy rovný 1, môžeme vzorec reflektujúci Amdahlov zákon modifikovať takto:

$$N_v = \frac{1}{s + \frac{p}{n}}$$

Uveďme príklad praktického nasadenia Amdahlovho zákona:

$$N_v = \frac{1}{0{,}5 + \frac{0{,}5}{2}} = \frac{1}{0{,}5 + 0{,}25} = \frac{1}{0{,}75} = 1{,}33$$

Príklad predstavuje program, ktorého algoritmická skladba je z 50 % tvorená sekvenčnými algoritmami a z ďalších 50 % paralelnými algoritmami. Takýto program teda presne polovicu

úloh rieši pomocou sekvenčných algoritmov a druhú polovicu pomocou paralelných algoritmov. Samozrejme, rozdiely existujú v povahách algoritmov. Zatiaľ čo sekvenčné algoritmy sú spracúvané synchrónne, pre paralelné algoritmy je príznačná asynchrónna exekúcia. Paralelné algoritmy dokážu prostredníctvom dátového alebo úlohového paralelizmu rozdeliť výpočtovo náročný problém na menšie časti (podproblémy), ktoré môžu byť realizované v rovnakom čase na viacerých procesoroch viacprocesorových strojov alebo na viacerých jadrách viacjadrových procesorov. (V tomto ponímaní abstrahujeme od pseudoparalelizmu a namiesto neho pracujeme so skutočným paralelizmom.)

Praktický príklad demonštruje situáciu, kedy je program v spomenutej algoritmickej skladbe spustený na počítači osadenom dvojjadrovým procesorom. Z výpočtu vyplýva, že maximálny teoretický nárast výkonnosti bude 33 %, čo znamená, že po paralelnej optimalizácii bude program pracovať o tretinu rýchlejšie ako predtým.

V jazyku Visual Basic 2008 najskôr deklarujeme delegáta a definujeme cieľovú metódu:

```
'Deklarácia delegáta.
Private Delegate Function Delegát(ByVal s As Single, _
    ByVal p As Single, ByVal n As Integer) As Single

'Definícia cieľovej metódy.
Private Function NárastVýkonnosti(ByVal s As Single, _
    ByVal p As Single, ByVal n As Integer) As Single
        Return 1 / (s + p / n)
End Function
```

V tele hlavnej metódy následne inštanciujeme delegáta a zabezpečíme spojenie zrodenej inštancie s cieľovou metódou:

```
Sub Main()
    'Inštanciácia deklarovaného delegáta a asociovanie metódy
    'so vzniknutou inštanciou.
    Dim Del As New Delegát(AddressOf NárastVýkonnosti)
    'Nepriama aktivácia cieľovej metódy pomocou inštancie delegáta.
    Console.WriteLine("Nárast výkonnosti je {0}.", Del.Invoke(0.5, 0.5, 2))
    Console.Read()
End Sub
```

Teraz si ukážeme, ako sa môžeme k rovnakému výsledku dopracovať použitím λ-výrazu:

```
Sub Main()
    Console.WriteLine("Nárast výkonnosti je {0}.", _
    (Function(s As Single, p As Single, n As Integer) _
        1 / (s + p / n)) _
        (0.5, 0.5, 2))
    Console.Read()
End Sub
```

Vďaka inteligentne navrhnutému λ-výrazu si dokážeme v jazyku Visual Basic 2008 značne zefektívniť prácu. Nie je totiž nutné, aby sme samostatne deklarovali delegáta a vytvárali definíciu cieľovej metódy – všetky potrebné inicializačné práce odvedieme pri definícii λ-výrazu.

Kapitola 5

Kovariancia a kontravariancia delegátov

Microsoft® Visual Studio® 2008

Kovariancia a kontravariancia delegátov

Zatiaľ čo v predchádzajúcich verziách jazyka Visual Basic bolo bezpodmienečne nutné dodržiavať zhodu medzi signatúrou deklarovaného delegáta a signatúrou definovanej cieľovej metódy, v edícii 2008 kompilátor už tak striktný nie je. Napríklad nasledujúci fragment zdrojového kódu by v jazyku Visual Basic 2005 spôsobil generovanie chybovej výnimky v režime prekladu:

```
Module Module1
    'Deklarácia delegáta.
    Friend Delegate Function Delegát(ByVal i As Short) As Integer

    'Definícia cieľovej metódy.
    Friend Function Metóda(ByVal i As Short) As Byte
        If (i Mod 2 = 0) Then
            Return CByte(i * i)
        Else
            Return 0
        End If
    End Function

    Sub Main()
        'Tento riadok je zdrojom programovej chyby.
        Dim Del As Delegát = New Delegát(AddressOf Metóda)
        Dim x As Integer = Del.Invoke(10)
        Console.WriteLine(x.ToString())
        Console.Read()
    End Sub
End Module
```

Problém je v tom, že delegát a metóda sa rozchádzajú v otázke dátového typu svojich návratových hodnôt. Zatiaľ čo z deklarácie delegáta je zrejmé, že jeho inštancia bude môcť byť asociovaná s metódou, ktorá vracia 32-bitové celé číslo, pohľad na definovanú metódu prezrádza, že jej návratovou hodnotou bude 8-bitová celočíselná hodnota. Ak kompilátor diagnostikoval nekompatibilitu dátových typov, generoval chybové hlásenie s týmto znením: *Method 'Friend Function Metóda(i As Short) As Byte' does not have the same signature as delegate 'Delegate Function Delegát(i As Short) As Integer'* [*Metóda 'Friend Function Metóda(i As Short) As Byte' nemá rovnakú signatúru ako delegát 'Delegate Function Delegát(i As Short) As Integer'*]. Keď sa chceli vývojári vyvarovať chyby prekladača, museli zjednotiť dátové typy návratových hodnôt.

Programovací jazyk Visual Basic 2008 umožňuje, aby boli vzťahy medzi deklarovanými delegátmi a definovanými metódami menej formálne. Povedané inak, v istých situáciách, ktoré sú závislé od programového kontextu, sa môžu signatúry delegátov a spriaznených metód odlišovať. Keďže tieto situácie opisujú dve nové vlastnosti delegátov, opíšeme si ich podrobnejšie.

Prvou zo skúmaných vlastností bude kovariancia delegátov. Podľa nej je možné, aby bol dátový typ návratovej hodnoty cieľovej metódy špecifickejší ako dátový typ návratovej hodnoty determinovaný pri deklarácii delegáta. Ak je táto podmienka splnená, vravíme, že delegát je voči metóde kovariantný. Ukážku kovariancie sme si už predstavili: Ak vyššie predstavený výpis

zdrojového kódu preložíme v editore jazyka Visual Basic 2008, kompilátor nebude vznášať žiadne námietky. Je to preto, že dátový typ návratovej hodnoty cieľovej metódy (Byte) je špecifickejší ako dátový typ návratovej hodnoty delegáta (Integer). V tomto kontexte máme pod pojmom „špecifickejší“ na mysli skutočnosť, že medzi typom Byte a typom Integer smie byť bez akýchkoľvek problémov uskutočnená implicitná typová konverzia. Aby sme boli úplne presní: Hodnota typu Byte môže byť v procese rozširujúcej implicitnej typovej konverzie hladko pretransformovaná na hodnotu typu Integer. Toto pretypovanie je vždy bezpečné, pretože pri rozširujúcich implicitných typových konverziách nemôže nikdy prísť ku strate dátovej informácie.

Kovarianciu delegátov možno s výhodou využiť aj pri práci s inštanciami tried, medzi ktorými je vybudované puto na báze implicitnej verejnej jednoduchej dedičnosti:

```
Module Module1
    'Deklarácia bázovej triedy.
    Class A
        'Definícia metódy, ktorá môže byť v odvodenej triede
        'prekrytá.
        Public Overridable Sub M()
            Console.WriteLine("A.M()")
        End Sub
    End Class

    'Deklarácia odvodenej triedy.
    Class B
        Inherits A
        'Definícia metódy, ktorá prekrýva rovnomennú zdedenú
        'metódu bázovej triedy.
        Public Overrides Sub M()
            Console.WriteLine("B.M()")
        End Sub
    End Class

    'Deklarácia delegáta.
    Friend Delegate Function Delegát() As A

    'Definícia metódy, ktorá bude spojená s inštanciou delegáta.
    Friend Function Metóda() As B
        Dim obj As B = New B()
        Return obj
    End Function

    Sub Main()
        'Inštanciácia delegáta.
        Dim Del As Delegát = New Delegát(AddressOf Metóda)
        'Aktivácia metódy prostredníctvom inštancie delegáta.
        Dim obj As B = CType(Del.Invoke(), B)
        obj.M()
        Console.Read()
    End Sub
End Module
```

Vďaka kovariancii delegáta bude môcť byť jeho inštancia asociovaná s cieľovou metódou, ktorá vracia odkaz na inštanciu triedy A, alebo odkaz na inštanciu ľubovoľnej podtriedy triedy A. Keďže trieda B je podtriedou triedy A, je zdrojový kód v poriadku a kompilátor nehlási žiadne varovné správy. Nemenej zaujímavý je zdrojový kód uložený v tele cieľovej metódy. Ako si môžeme všimnúť, kód inštanciuje podtriedu B a vracia odkaz na založenú inštanciu. V definícii

hlavnej metódy Main najskôr vytvárame inštanciu delegáta, ktorú prepojíme s cieľovou metódou. Potom nepriamo, pomocou zrodenej inštancie delegáta, voláme spriaznenú metódu. Vzhľadom na to, že na aktiváciu cieľovej metódy využívame inštančnú metódu Invoke, metóda bude spracovaná synchrónne. Podotknime, že v záujme uloženia návratovej hodnoty cieľovej metódy je nutné vykonať explicitnú typovú konverziu. Príkaz obj.M() napokon volá metódu inštancie odvodenej triedy.

Poznámka: Triedy A a B sú projektované tak, aby reflektovali základné použitie polymorfizmu implementovaného prostredníctvom dedičnosti. V deklarácii bázovej triedy je definovaná verejne prístupná, inštančná a bezparametrická metóda s identifikátorom M. V hlavičke tejto metódy je uvedený modifikátor Overridable, čo znamená, že odvodené triedy disponujú kompetenciami na prekrytie zdedenej metódy s totožným identifikátorom. Naša podtrieda B prekrýva pôvodnú implementáciu metódy M bázovej triedy. Túto skutočnosť dávame najavo použitím modifikátora Overrides v jej hlavičke. Uvažujme teraz situáciu, kedy existujú dve inštancie: jedna je inštanciou bázovej triedy A a druhá zase inštanciou odvodenej triedy B. Ak obom inštanciám pošleme rovnakú správu vyžadujúcu vyvolanie metódy M, tak rozhodnutie o tom, ktorá metóda (či inštancie bázovej triedy, alebo inštancie odvodenej triedy) bude aktivovaná, je dané typom zrodenej inštancie. V našom praktickom experimente zakladáme inštanciu podtriedy B, z čoho vyplýva, že príkaz obj.M() bude aktivovať metódu inštancie zmienenej podtriedy.

Druhou z predmetných vlastností delegátov je kontravariancia. Kontravariancia umožňuje, aby cieľová metóda definovala formálne parametre, ktorých dátové typy sú generickejšie ako typy formálnych parametrov uvedených v deklarácii delegáta. Tak môžeme navrhnúť jednu metódu, ktorá bude vystupovať ako spracovateľ viacerých udalostí.

```
Public Class Form1
    'Metóda pôsobí ako spracovateľ dvoch udalostí formulára.
    Friend Sub Metóda(ByVal o As Object, ByVal e As EventArgs)
        If TypeOf e Is KeyPressEventArgs Then
            MessageBox.Show("Bol aktivovaný kláves: " & _
               CType(e, KeyPressEventArgs).KeyChar.ToString())
        ElseIf TypeOf e Is FormClosingEventArgs Then
            Dim Odpoveď As DialogResult = _
            MessageBox.Show("Naozaj chcete zatvoriť formulár?", _
               "Správa", MessageBoxButtons.YesNo, _
               MessageBoxIcon.Question)
            If (Odpoveď = Windows.Forms.DialogResult.No) Then
                CType(e, FormClosingEventArgs).Cancel = True
            End If
        End If
    End Sub
    Private Sub Form1_Load(ByVal sender As Object, _
       ByVal e As System.EventArgs) Handles Me.Load
        AddHandler Me.KeyPress, AddressOf Metóda
        AddHandler Me.FormClosing, AddressOf Metóda
    End Sub
End Class
```

Komentár k zdrojovému kódu: Pomocou definovanej metódy smieme obslúžiť dvojicu udalostí formulára. Udalosť KeyPress je generovaná vždy, keď používateľ aktivuje kláves

(zameranie musí samozrejme byť na aktívnom formulári). Udalosť FormClosing vzniká v začiatočnom štádiu procesu uzatvorenia formulára. Po preštudovaní zdrojového kódu vidíme, že metóda v úlohe spracovateľa udalostí dokáže samočinne rozpoznať, ktorá udalosť nastala a náležite ju ošetriť. Keď používateľ stlačí kláves, v dialógovom okne bude zobrazená jeho hodnota. Ak používateľ vydá pokyn na uzatvorenie formulára, spracovateľ zachytí udalosť FormClosing a spýta sa používateľa, či si naozaj praje formulár zatvoriť.

Zo syntaktického hľadiska je dôležité poukázať na druhý formálny parameter metódy, ktorého dátovým typom je trieda EventArgs. Do takto definovaného formálneho parametra sa dá uložiť odkaz na inštanciu triedy EventArgs, resp. odkaz na inštanciu ľubovoľnej podtriedy triedy EventArgs.

V prípade generovania udalosti KeyPress bude skonštruovaná inštancia triedy KeyPressEventArgs, zatiaľ čo pri vzniku udalosti FormClosing bude inštanciovaná trieda FormClosingEventArgs. Obe triedy (KeyPressEventArgs a FormClosingEventArgs) sú podtriedami bázovej triedy EventArgs.

Kapitola 6

XML konštanty a XML konštanty s pridruženými programovými výrazmi

Microsoft® Visual Studio® 2008

XML konštanty

Visual Basic 2008 je prvým .NET-kompatibilným programovacím jazykom, ktorý dovoľuje programátorom priamo pracovať s XML konštantami. XML konštanta je blok kódu jazyka XML, ktorý môže byť jednoducho uložený do definovanej odkazovej premennej:

```
'Priame použitie XML konštanty v zdrojovom kóde jazyka Visual Basic 2008.
Dim XMLElement As XElement = _
<Kniha Autor="Ján Hanák">
    <Názov>C# - praktické příklady</Názov>
    <Vydavateľ>Grada Publishing</Vydavateľ>
    <RokVydania>2006</RokVydania>
    <PočetStrán>288</PočetStrán>
    <ISBN>80-247-0988-0</ISBN>
</Kniha>
```

Tento príkaz predstavuje definičnú inicializáciu odkazovej premennej s identifikátorom XMLElement a je vskutku zaujímavý. Výraz nachádzajúci sa na ľavej strane od operátora priradenia zakladá na zásobníku programového vlákna odkazovú premennú, ktorej dátovým typom je trieda XElement z menného priestoru System.Xml.Linq. Keďže vo výraze nie je zapísané kvalifikované meno triedy, musíme požadovaný menný priestor zaviesť pomocou príkazu Imports. Z výrazu vieme dedukovať, že do vytvorenej odkazovej premennej budú môcť byť uložené odkazy nasmerované na inštancie triedy XElement. Na pravej strane od operátora priradenia sa rozprestiera deklarácia XML konštanty. Ako môžeme postrehnúť, XML konštantu tvorí koreňový element Kniha, ktorý disponuje svojím atribútom a niekoľkými vnorenými elementmi. Výborné je, že XML kód je zapísaný presne v takej podobe, na akú sme zvyknutí. Visual Basic 2008 preberá pravidlá pre zápis elementov a ich atribútov dané jazykom XML verzie 1.0 a uplatňuje ich pri deklarácii XML konštánt.

Poznámka: Podpora jazykovej špecifikácie XML nie je v jazyku Visual Basic 2008 stopercentná. Existuje totiž množina rysov, ktoré môžeme použiť v „čistom“ XML, avšak už nie pri deklarácii XML konštánt v jazyku Visual Basic 2008. K nepodporovaným vlastnostiam patrí napríklad to, že XML konštanta nemôže obsahovať špecifikáciu definície typu dokumentu (DTD). Podobne, počet znakov XML konštanty združených na jednom riadku nemôže presiahnuť hodnotu 2^{16}-1 (teda 65535). Na druhej strane, XML konštanty vytvorené v kóde jazyka Visual Basic 2008 smú obsahovať pridružené programové výrazy, ktoré budú dynamicky vyhodnotené až v režime exekúcie aplikácie .NET. Jazyk XML nič podobné nedokáže.

Kompilátor jazyka Visual Basic 2008 nahliada na kód vypĺňajúci telo XML konštanty ako na platný XML kód. Na XML kód sa nevzťahujú pravidlá platiace pre rozdeľovanie entít v štandardnom zdrojovom kóde jazyka Visual Basic 2008. Povedané inak, Visual Basic 2008 nám umožní deklarovať XML konštantu na viacerých riadkoch bez nutnosti vkladania špeciálnej postupnosti symbolov pre pokračovanie programového príkazu (spomínaná postupnosť je zložená z medzery, podčiarkovníka a symbolu vloženia nového riadka).

Deklarovaná XML konštanta je zapuzdrená do inštancie triedy XElement, pričom odkaz na túto dynamicky vytvorenú inštanciu bude v rámci priraďovacieho príkazu uložený do pripravenej

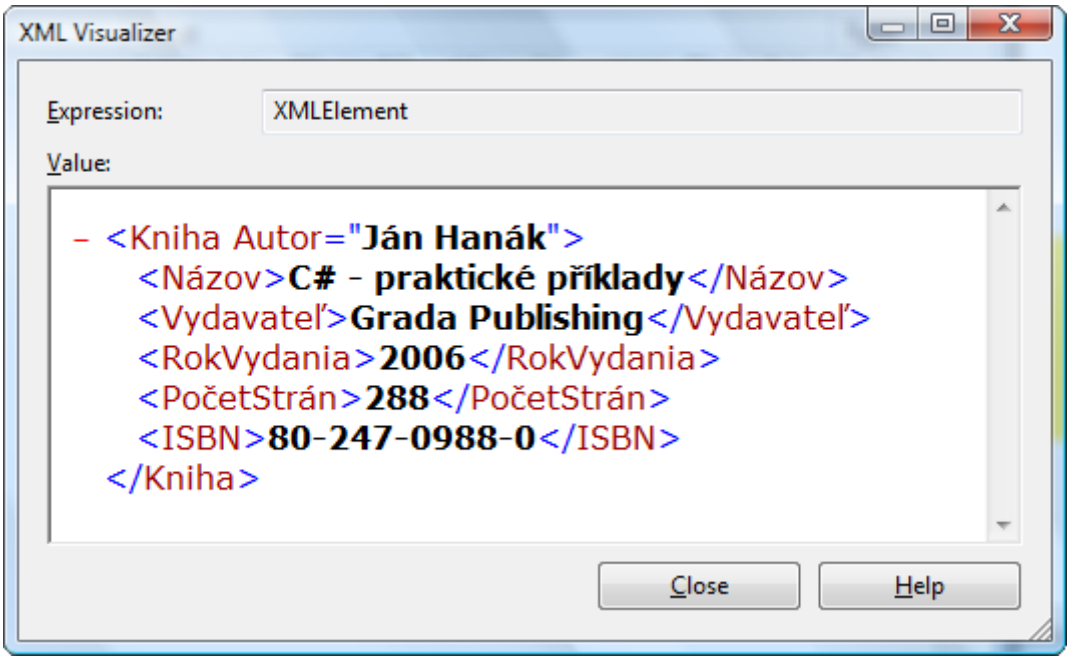
odkazovej premennej. Definičnú inicializáciu premennej XMLElement by sme mohli modifikovať nasledujúcim spôsobom:

```
'Funkčne ekvivalentná definičná inicializácia odkazovej premennej,
'ktorá uchováva odkaz na objekt obsahujúci XML konštantu.
Dim XMLElement As XElement = New XElement("Kniha", _
    New XAttribute("Autor", "Ján Hanák"), _
    New XElement("Názov", "C# - praktické příklady"), _
    New XElement("Vydavateľ", "Grada Publishing"), _
    New XElement("RokVydania", "2006"), _
    New XElement("PočetStrán", "288"), _
    New XElement("ISBN", "80-247-0988-0"))
```

Keďže Visual Basic 2008 dokáže implicitne diagnostikovať dátové typy lokálnych entít pomocou inferenčného mechanizmu, môžeme pôvodnú definičnú inicializáciu odkazovej premennej zapísať aj v tomto tvare:

```
'Dátový typ odkazovej premennej je implicitne inferovaný.
Dim XMLElement = _
<Kniha Autor="Ján Hanák">
    <Názov>C# - praktické příklady</Názov>
    <Vydavateľ>Grada Publishing</Vydavateľ>
    <RokVydania>2006</RokVydania>
    <PočetStrán>288</PočetStrán>
    <ISBN>80-247-0988-0</ISBN>
</Kniha>
```

Finálny efekt je samozrejme rovnaký, kompilátor jazyka Visual Basic 2008 inferuje dátový typ odkazovej premennej (a v konečnom dôsledku aj typ objektu, v ktorej bude XML konštanta uložená). Ak do zdrojového kódu umiestnime lokálny bod prerušenia, môžeme si dátovú sekciu inštancie triedy XElement prehliadnuť pomocou XML dátového vizualizéra (obr. 16).

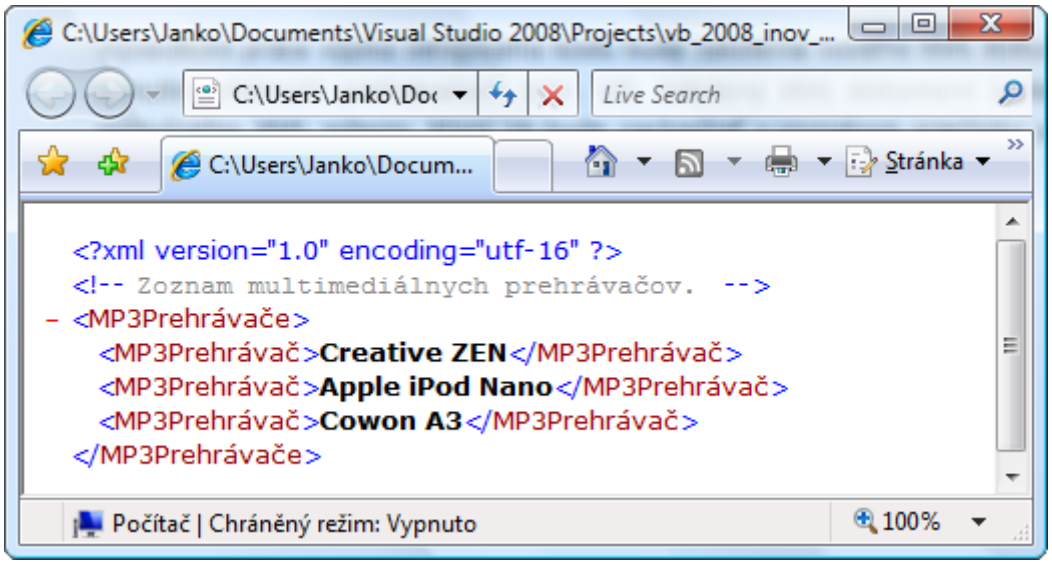


Obr. 16: Zobrazenie XML konštanty uloženej v inštancii triedy XElement pomocou XML dátového vizualizéra

XML konštanta však nemusí vytvárať len element, ale rovno celý XML dokument:

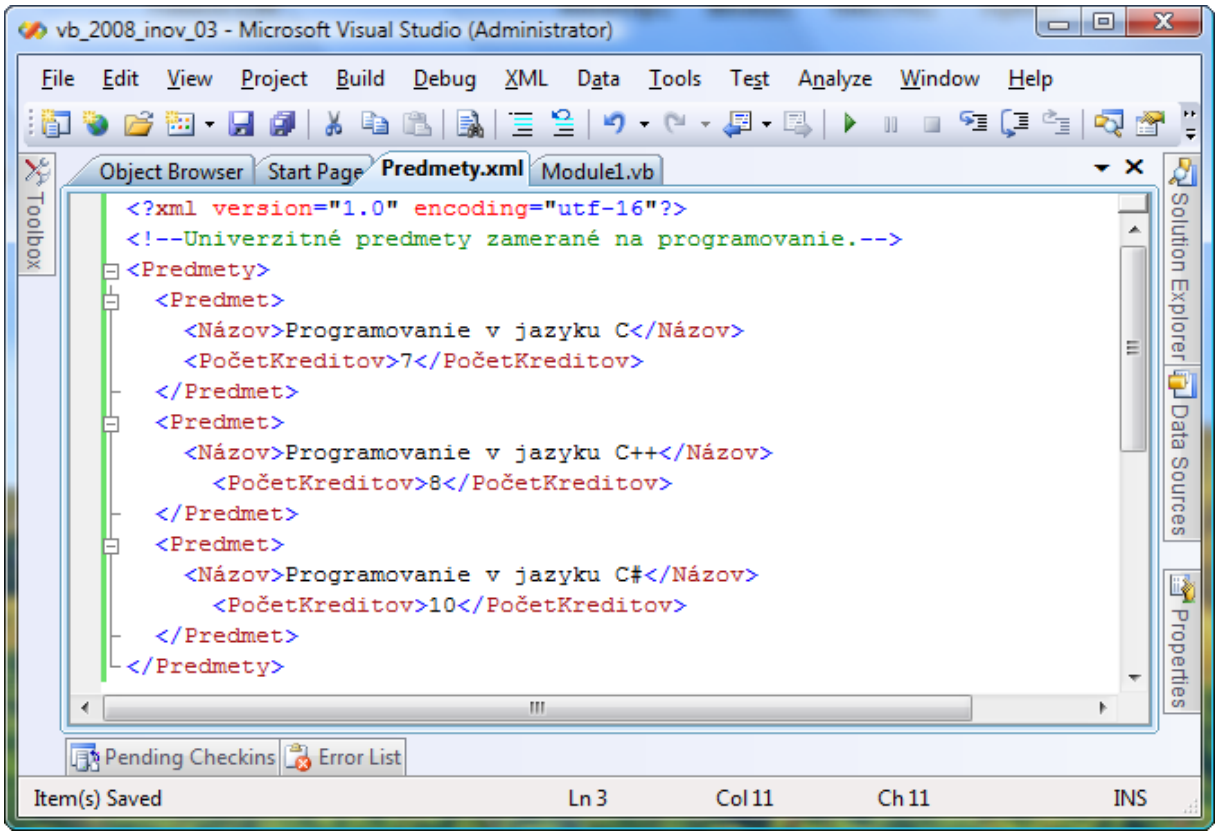
```
'Vytvorenie XML dokumentu.
Dim XMLDokument As XDocument = _
<?xml version="1.0" encoding="UTF-16"?>
<!--Zoznam multimediálnych prehrávačov.-->
<MP3Prehrávače>
    <MP3Prehrávač>Creative ZEN</MP3Prehrávač>
    <MP3Prehrávač>Apple iPod Nano</MP3Prehrávač>
    <MP3Prehrávač>Cowon A3</MP3Prehrávač>
</MP3Prehrávače>

'Uloženie XML dokumentu do súboru.
XMLDokument.Save("XMLDokument.xml")
Console.WriteLine("XML dokument bol úspešne uložený do súboru.")
```

Výsledkom práce výpisu zdrojového kódu bude založenie nového XML dokumentu s informáciami o multimediálnych prehrávačoch (obr. 17). Založený XML dokument bude vzápätí uložený do príslušného XML súboru, ktorý sa bude nachádzať v rovnakom priečinku ako spustiteľný súbor aplikácie .NET.



Obr. 17: Súbor s XML dokumentom

Ďalšia ukážka znázorňuje spôsob, akým môžeme načítať a zobraziť obsah ľubovoľného XML dokumentu. Náš vzorový XML dokument mal takúto podobu:

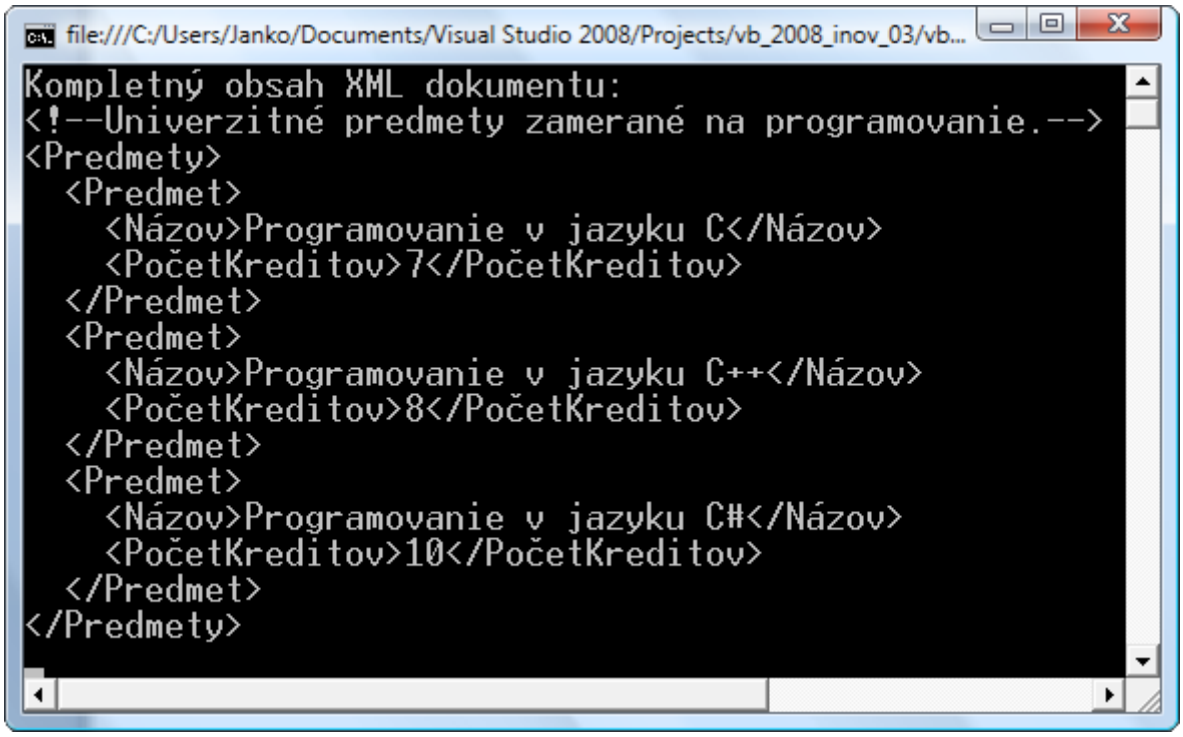


Obr. 18: Testovací XML dokument

Zdrojový kód jazyka Visual Basic 2008, ktorý zabezpečí načítanie obsahu XML dokumentu zo súboru, vyzerá takto:

```
Module Module1

    Sub Main()
        Dim XMLDokument As XDocument
        'Vytvorenie novej inštancie triedy XDocument.
        'Táto inštancia obsahuje dáta XML dokumentu.

        XMLDokument = XDocument.Load("Predmety.xml")
        Console.WriteLine("Kompletný obsah XML dokumentu:")

        'Výpis dátového obsahu XML dokumentu iterovaním cez
        'všetky uzly tvoriace strom XML elementov.
        For Each Uzol As XNode In XMLDokument.Nodes
            Console.WriteLine(Uzol.ToString())
        Next
        Console.Read()
    End Sub

End Module
```

Výstup programu uvádzame na obr. 19.



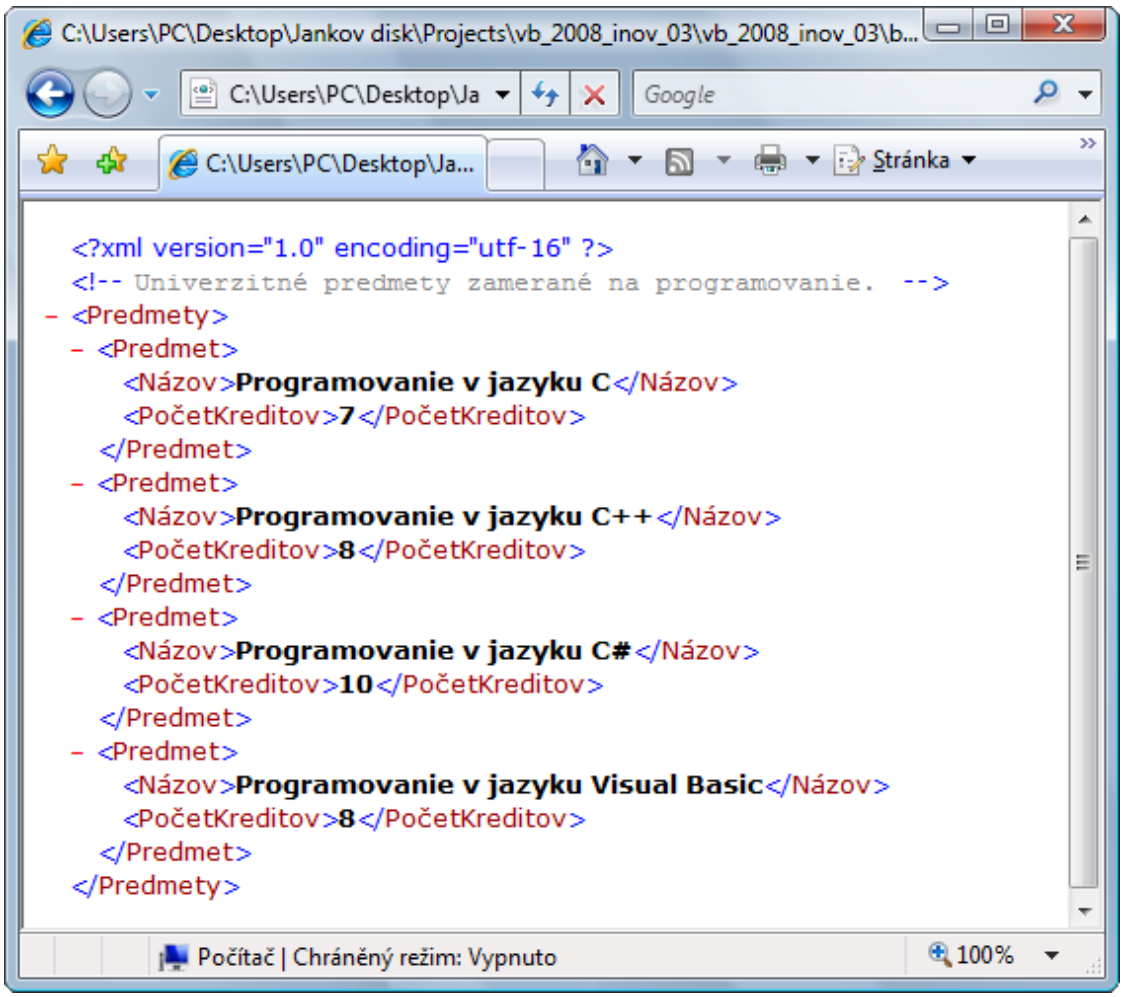
Obr. 19: Program zobrazujúci kompletný výpis XML dokumentu načítaného zo súboru

Pôvodný XML dokument môžeme doplniť o ďalší element:

```
Imports System.Xml

Module Module1
    Sub Main()
        'Inštanciácia triedy XmlDocument z menného priestoru System.XML.
        Dim XMLDokument As New XmlDocument

        'Načítanie obsahu XML dokumentu.
        XMLDokument.Load("Predmety.xml")

        'Vytvorenie nového elementu pre ďalší univerzitný predmet.
        Dim NovýPredmet As XmlNode
        NovýPredmet = XMLDokument.CreateElement("Predmet")

        'Vytvorenie vnorených elementov.
        Dim NázovNovéhoPredmetu As XmlNode
        Dim PočetKreditovNovéhoPredmetu As XmlNode
        NázovNovéhoPredmetu = XMLDokument.CreateElement("Názov")
        PočetKreditovNovéhoPredmetu = _
            XMLDokument.CreateElement("PočetKreditov")

        'Pridanie vnorených elementov do rodičovského elementu.
        NovýPredmet.AppendChild(NázovNovéhoPredmetu)
        NovýPredmet.AppendChild(PočetKreditovNovéhoPredmetu)

        'Naplnenie vnorených elementov dátami.
        NázovNovéhoPredmetu.AppendChild( _
            XMLDokument.CreateTextNode("Programovanie " & _
                "v jazyku Visual Basic"))
        PočetKreditovNovéhoPredmetu.AppendChild( _
            XMLDokument.CreateTextNode("8"))
```

```
        'Pridanie nového elementu do XML dokumentu.
        XMLDokument.DocumentElement.AppendChild(NovýPredmet)

        'Uloženie modifikácie do XML súboru.
        XMLDokument.Save("Predmety_02.xml")

        'Ak všetko prebehlo v poriadku, vypisujeme informačnú správu.
        Console.WriteLine("XML dokument bol modifikovaný a uložený.")
        Console.Read()
    End Sub

End Module
```

Po spustení tohto programu jazyka Visual Basic 2008 bude načítaný XML dokument doplnený o nový univerzitný predmet (nový element). Upravená podoba XML dokumentu bude nakoniec uložená do nového XML súboru s identifikátorom Predmety_02.xml. Podobu modifikovaného XML dokumentu prináša obr. 20.



Obr. 20: Obsah programovo modifikovaného XML dokumentu

XML konštanty s pridruženými programovými výrazmi

Novinkou voči bežným možnostiam jazyka XML vo verzii 1.0 je schopnosť XML konštánt jazyka Visual Basic 2008 obsahovať pridružené programové výrazy. Pridružené programové výrazy predstavujú fragmenty zdrojového kódu, ktoré budú automaticky vyhodnotené v režime exekúcie aplikácie .NET. Ak analyzátor diagnostikuje XML konštantu s pridruženým programovým výrazom, uskutoční vyhodnotenie výrazu a vypočítanú hodnotu výrazu dosadí do entity deklarovanej v XML konštante.

Pre praktickú aplikáciu XML konštánt s pridruženými výrazmi sme zvolili jednoduchý ekonomický program, ktorý sa zameriava na výpočet kritického bodu rentability.

Najskôr, s dovolením, uvedieme stručný úvod do skúmanej ekonomickej problematiky. Predstavme si, že máme podnik (strategickú podnikateľskú jednotku), ktorý vyrába a predáva určité portfólio produktov. V závislosti od diverzifikácie výrobného programu môže podnik vyrábať a predávať veľa rôznych výrobkov a ich variantov. Nech existuje konečná neprázdna množina vyrábaných produktov V, ktorá je tvorená jednotlivými výrobkami. Túto skutočnosť môžeme formalizovať nasledujúcim spôsobom: $V = \{v_1, v_2, \dots, v_n\}$. My si z uvedenej množiny vyberieme jeden výrobok, na ktorom uskutočníme našu ekonomickú analýzu. Ekonómovia obvykle chcú vedieť, kedy sa celkové náklady na výrobu daného výrobku budú zhodovať s celkovými tržbami dosiahnutými predajom istého množstva tohto výrobku. Práve túto situáciu, teda keď sa náklady na výrobu výrobku rovnajú tržbám z jeho predaja, charakterizuje tzv. kritický bod rentability. Tento bod označuje množstvo vyrobených a súčasne predaných jednotiek výrobku, pričom musí platiť táto matematická rovnica:

$$T = N$$

kde:

- T sú celkové tržby generované predajom istého množstva výrobku.
- N sú celkové náklady vynaložené na výrobu istého množstva výrobku.

Keď sa celkové tržby zhodujú s celkovými nákladmi, tak podniku zatiaľ neprináša výroba a následný predaj výrobku žiadny zisk. Je totiž zrejmé, že zisk generovaný predajom výrobku je v okamihu dosiahnutia kritického bodu rentability nulový ($Z = 0$). Preto sa niekedy kritický bod rentability nazýva tiež jednoducho „nulovým bodom“. Vzhľadom na to, že veličiny T a N sú agregované, musíme ich pre potreby ďalšej analýzy bližšie špecifikovať. Dovolíme si pripomenúť, že v našom praktickom experimente pracujeme s najjednoduchším možným vyjadrením ekonomických javov, pričom abstrahujeme od akýchkoľvek nepresností, ktorých sa môžeme vo vzťahu ku skutočnosti dopustiť. Kritický bod rentability vypočítame pomocou tohto matematického algoritmu:

$$\begin{gathered} T = N \\ p \times q = FN + b \times q \\ p \times q - b \times q = FN \\ q(p - b) = FN \\ q = \frac{FN}{p - b} \end{gathered}$$

Celkové tržby získame, keď predáme istý počet výrobkov (q), pričom každý výrobok má svoju predajnú cenu (p). Celkové náklady sa skladajú z nákladov fixných (FN) a variabilných (VN). Zatiaľ čo úroveň fixných nákladov nie je závislá od objemu vyrobenej produkcie (prinajmenšom z krátkodobého hľadiska nie), u variabilných nákladov táto závislosť existuje. Čím viac jednotiek skúmaného výrobku podnik vyprodukuje, tým väčšie variabilné náklady bude nutné alokovať. Aby sme znázornili závislosť variabilných nákladov na objeme produkcie, zavádzame vzťah $VN = b\ x\ q$. V ďalších krokoch sa už matematický algoritmus sústreďuje na osamostatnenie premennej q, pretože jej hodnota (za takto stanovených podmienok) predstavuje množstvo vyrobených a predaných jednotiek produktu, teda nami hľadaný kritický bod rentability.

Samozrejme, študenti ekonomických vied bežne operujú s finálnym vzorcom, ktorý môžeme zapísať takto:

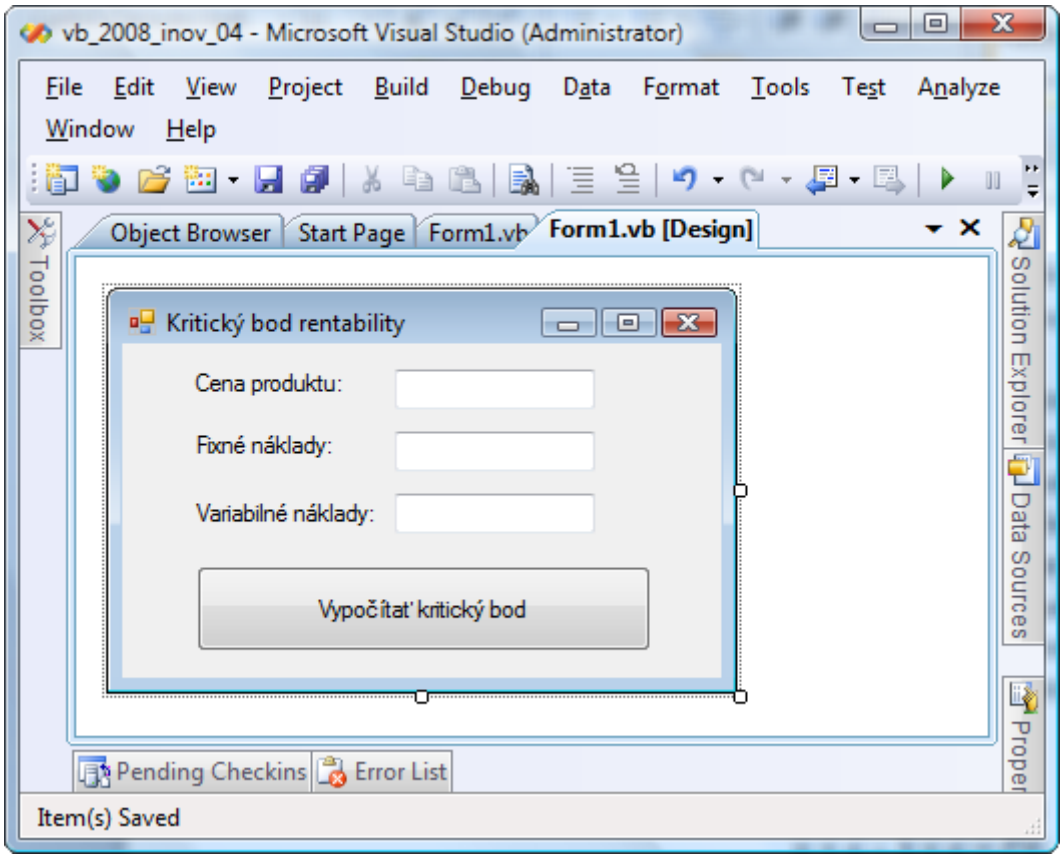
$$q_{kb} = \frac{FN}{p - b}$$

kde:

- q_{kb} je množstvo vyprodukovaných a predaných výrobkov, pri ktorom je dosiahnutý kritický bod rentability.
- FN sú fixné náklady na výrobu príslušného množstva výrobkov.
- p je predajná cena jedného výrobku.
- b sú variabilné náklady na výrobu jedného výrobku (tzv. jednotkové náklady).

Ak nie je pre analyzovaný výrobok dosiahnuté také množstvo produkcie a predaja, aké stanovuje premenná q_{kb}, tak takýto výrobok spôsobuje podniku stratu. Naopak, pri prekročení kritického bodu sa už výrobok dostáva do „ziskovej oblasti“, pretože celkové tržby začínajú prevyšovať celkové náklady.

Aplikácia na určenie kritického bodu rentability disponuje grafickým používateľským rozhraním, ktoré je zobrazené na obr. 21.



Obr. 21: Grafické používateľské rozhranie aplikácie .NET na určenie kritického bodu rentability

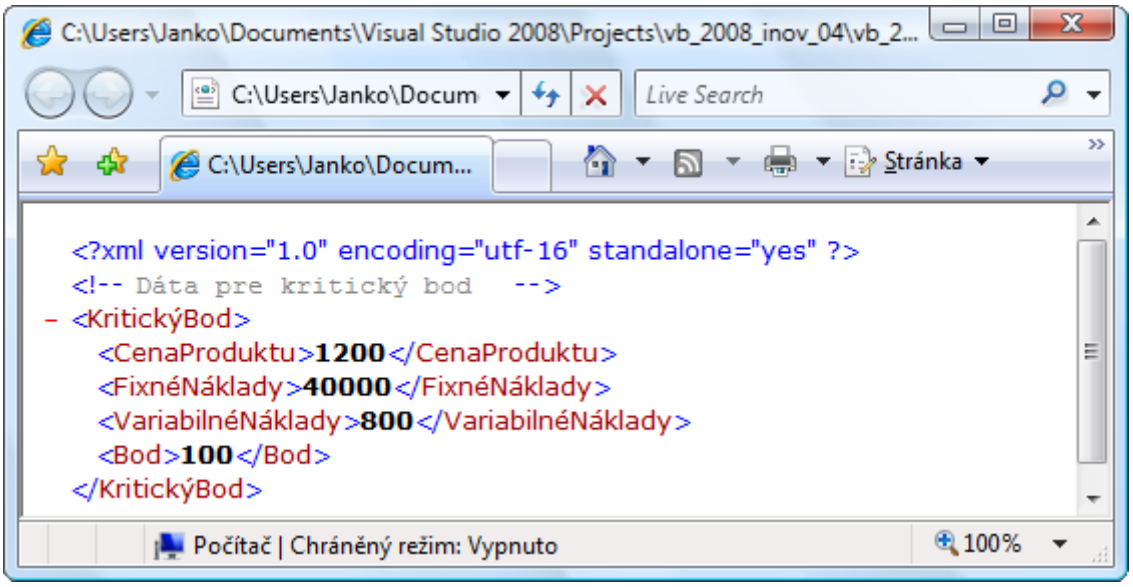
Nás bude zaujímať predovšetkým zdrojový kód jazyka Visual Basic 2008, ktorý je uložený v tele spracovateľa udalosti Click tlačidla a realizuje výpočet kritického bodu rentability:

```
Dim CenaProduktu, FixnéNáklady, VariabilnéNáklady As Integer

'Inicializácia premenných podľa hodnôt, ktoré používateľ zadal
'do textových polí.
CenaProduktu = CInt(txtCenaProduktu.Text)
FixnéNáklady = CInt(txtFixnéNáklady.Text)
VariabilnéNáklady = CInt(txtVariabilnéNáklady.Text)

'Použitie XML konštanty s pridruženými programovými výrazmi.
Dim XMLKritickýBod As XDocument = _
<?xml version="1.0" encoding="UTF-16" standalone="yes"?>
<!--Dáta pre kritický bod -->
<KritickýBod>
    <CenaProduktu><%= CenaProduktu %></CenaProduktu>
    <FixnéNáklady><%= FixnéNáklady %></FixnéNáklady>
    <VariabilnéNáklady><%= VariabilnéNáklady %></VariabilnéNáklady>
    <Bod><%= FixnéNáklady / (CenaProduktu - VariabilnéNáklady) %></Bod>
    </KritickýBod>

'Uloženie finálneho XML dokumentu do súboru.
XMLKritickýBod.Save("KritickýBod.xml")
MessageBox.Show("Dáta boli úspešne uložené do XML súboru.", _
    "Informačná správa", MessageBoxButtons.OK, MessageBoxIcon.Information)
```

V deklarácii XML konštanty sú umiestnené pridružené programové výrazy, ktoré sa nachádzajú v blokoch vymedzených symbolmi <%= %>. Výrazy môžu obsahovať nielen samostatné

premenné, ale aj syntakticky a sémanticky korektne zapísané postupnosti operandov a operátorov či volania procedúr. Napríklad pri deklarácii elementu Bod manipulujeme s aritmetickým výrazom, ktorého úlohou je na základe dát uložených v použitých premenných vypočítať hodnotu prislúchajúcu kritickému bodu rentability. Ako sme už spomenuli, pridružené programové výrazy budú vyhodnotené až počas behu aplikácie .NET, čo je zrejmé, pretože až v tomto štádiu sa od používateľa dozvieme konkrétne hodnoty vstupných dát. Po vyhodnotení budú výrazy nahradené svojimi hodnotami. V našom prípade tak budú elementy CenaProduktu, FixnéNáklady a VariabilnéNáklady substituované konkrétnymi hodnotami zadanými používateľom. Analogicky, hodnota aritmetického výrazu predstavujúca množstvo výrobkov, ktoré zodpovedá kritickému bodu rentability, bude dosadená do elementu Bod.

Po vyhodnotení všetkých pridružených programových výrazov budú XML elementy obsahovať konkrétne dáta, ktoré môžu byť uložené do súboru. To sa v konečnom dôsledku aj deje: inštančná metóda Save triedy XDocument garantuje zaliatie XML dokumentu do cieľového súboru. Jeden z možných obrazov vytvoreného XML dokumentu uvádzame na obr. 22.



Obr. 22: Obraz vytvoreného XML dokumentu po vyhodnotení pridružených programových výrazov

Pri pohľade na výstupný XML dokument môžeme konštatovať, že ak podnik vyrába výrobok, na výrobu ktorého vynaloží fixné náklady vo výške 40000 Sk a variabilné náklady vo výške 800 Sk, tak za predpokladu, že bude podnik tento výrobok predávať za cenu 1200 Sk, kritický bod rentability bude dosiahnutý pri výrobe a následnom predaji 100 kusov (ks) výrobku. Ak podnik vyrobí a predá menej ako 100 ks výrobku, výroba nebude rentabilná, pretože v tomto prípade budú celkové tržby menšie ako celkové náklady. Naopak, pri výrobe viac ako 100 ks výrobku začnú celkové tržby prevyšovať celkové náklady, čím sa výrobok dostane do ziskovej oblasti a stane sa pre podnik profitabilným zdrojom.

Nemenej interesantným programovým rysom je možnosť použiť v pridruženom programovom výraze volanie procedúry, resp. metódy. Preštudujme si ďalší výpis zdrojového kódu jazyka Visual Basic 2008:

```
Private Sub btnKritickýBod_Click(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles btnKritickýBod.Click
    Dim CenaProduktu, FixnéNáklady, VariabilnéNáklady As Integer

    CenaProduktu = CInt(txtCenaProduktu.Text)
    FixnéNáklady = CInt(txtFixnéNáklady.Text)
    VariabilnéNáklady = CInt(txtVariabilnéNáklady.Text)

    Dim XMLKritickýBod As XDocument = _
    <?xml version="1.0" encoding="UTF-16" standalone="yes"?>
    <!--Dáta pre kritický bod -->
    <KritickýBod>
        <CenaProduktu><%= CenaProduktu %></CenaProduktu>
        <FixnéNáklady><%= FixnéNáklady %></FixnéNáklady>
        <!--Pridružený programový výraz volá metódu pre výpočet
            kritického bodu rentability.-->
        <VariabilnéNáklady><%= VariabilnéNáklady %></VariabilnéNáklady>
        <Bod><%= VypočítaťKritickýBod(CenaProduktu, FixnéNáklady, _
                 VariabilnéNáklady) %></Bod>
        </KritickýBod>

    XMLKritickýBod.Save("KritickýBod.xml")
    MessageBox.Show("Dáta boli úspešne uložené do XML súboru.", _
       "Informačná správa", MessageBoxButtons.OK, _
       MessageBoxIcon.Information)
End Sub

'Definícia metódy pre stanovenie kritického bodu rentability.
Public Function VypočítaťKritickýBod(ByVal CenaProduktu As Integer, _
    ByVal FixnéNáklady As Integer, ByVal VariabilnéNáklady As Integer) _
    As Integer
        Dim KritickýBodRentability As Integer
        KritickýBodRentability = _
            FixnéNáklady \ (CenaProduktu - VariabilnéNáklady)
        Return KritickýBodRentability
End Function
```

V tomto fragmente definujeme samostatnú funkciu VypočítaťKritickýBod, ktorá realizuje operáciu určenia kritického bodu rentability vo svojej vlastnej réžii. Funkciu explicitne voláme z pridruženého programového výrazu, ktorý je uvedený v deklarácii XML konštanty. Keď príde k aktivácii funkcie, odohrá sa nasledujúci reťazec udalostí:

1. Vstupné dáta (argumenty) budú hodnotou odovzdané formálnym parametrom funkcie.

2. Spracujú sa všetky príkazy umiestnené v tele funkcie a určí sa jej návratová hodnota (množstvo výrobkov predstavujúce kritický bod rentability).

3. Návratová hodnota funkcie je hodnotou výrazu, v ktorom dochádza k volaniu funkcie. Návratová hodnota bude preto uložená do elementu Bod.

Kapitola 7

LINQ (Language Integrated Query): Unifikovaný dopytovací jazyk

Microsoft® Visual Studio® 2008

LINQ (Language Integrated Query): Unifikovaný dopytovací jazyk

Verzia 2008 jazyka Visual Basic je prvou, v ktorej sa objavuje podpora pre unifikovaný dopytovací jazyk nazývaný LINQ (Language Integrated Query). Hlavná vízia pri tvorbe nového dopytovacieho jazyka bola poháňaná snahou o zjednotenie programátorských prác pri manipulácii s dátami, ktoré sú uložené v typovo rôznych dátových zdrojoch. Dnešní vývojári môžu dáta získavať zo vskutku širokej množiny entít, ku ktorým patria objekty, objektové kolekcie, polia, dátové súpravy, SQL databázy či XML dokumenty. S každým dátovým zdrojovom sa spravidla viaže určité prístupové komunikačné rozhranie, ktoré umožňuje prehľadávať, filtrovať a vo všeobecnosti uskutočňovať manipulačné operácie s predmetnými dátami. Nadobudnuté skúsenosti z minulých čias nám dávajú jasne najavo, že čím viac rôznych variantov dátových zdrojov a ich komunikačných rozhraní existuje, tým dlhšia je doba ich osvojenia a tým menšia je pracovná produktivita vývojárov. Tí sa totiž museli naučiť zaobchádzať s rôznymi modelmi realizujúcimi diferencovanú správu dát.

Softvéroví inžinieri v americkom Redmonde si uvedomili, že priaznivý synergický efekt pri práci s dátami uskladnenými v rozličných dátových zdrojoch, budú môcť vývojári dosiahnuť len vtedy, keď budú pracovať s unifikovaným komunikačným rozhraním. Zmienené komunikačné rozhranie má byť natoľko generické, aby pomocou neho bolo možné pracovať s objektovými kolekciami, poľami, dátovými súpravami či XML dokumentmi. Na druhej strane, komunikačné rozhranie musí byť typovo bezpečné, teda vždy priamo aplikovateľné na konkrétne zvolený dátový zdroj. Výsledným produktom inovatívneho úsilia sa stal nový unifikovaný dopytovací jazyk LINQ, ktorý vývojárom prináša nasledujúcu kolekciu výhod:

1. LINQ je dopytovací jazyk, ktorý je priamo zakomponovaný do jazykovej špecifikácie jazyka Visual Basic 2008. Všetky syntakticko-sémantické súčasti jazyka LINQ sú explicitne dosiahnuteľné pomocou príslušných programových konštrukcií jazyka Visual Basic 2008. Okrem jazyka Visual Basic 2008 je LINQ natívne podporovaný aj jazykom C# vo verzii 3.0. Môžeme teda vyhlásiť, že aj vývojári v C# 3.0 môžu ťažiť z plnej podpory jazyka LINQ.

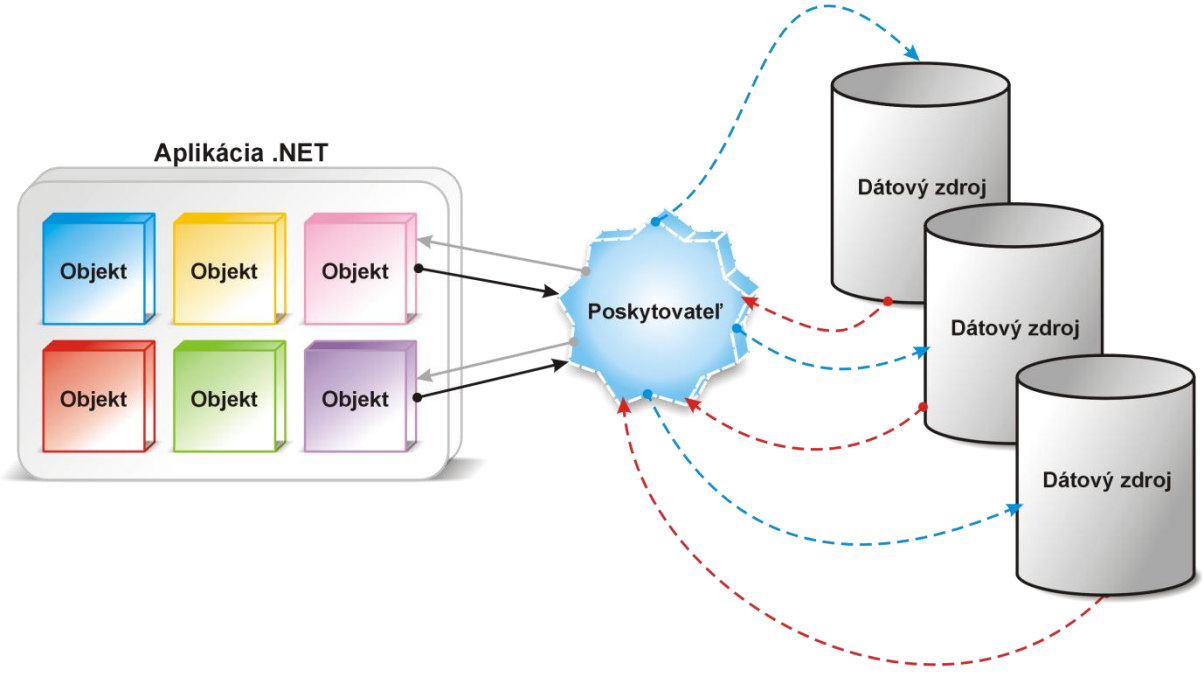
Tip: Hoci Visual Basic 2008 a C# 3.0 sú zatiaľ jediné .NET-kompatibilné programovacie prostriedky priamo podporujúce jazyk LINQ, neznamená to, že sa s týmto dopytovacím jazykom nemôžeme stretnúť aj v intenciách iných programovacích jazykov. LINQ sa dá napríklad použiť tiež v jazyku C++/CLI, aj keď je nutné poznamenať, že jeho uplatnenie v tomto prostredí nie je ani zďaleka také intuitívne ako v jazykoch Visual Basic 2008 a C# 3.0.

2. Ak sa vývojári rozhodnú použiť jazyk LINQ, môžu sa sústrediť výhradne na abstraktno-analytickú časť realizácie dátovo-manipulačných operácií. Povedané inak, vývojári nemusia do podrobností poznať všetky technické detaily nízkoúrovňových dátových operácií, ani fyzickú reprezentáciu dát v príslušnom dátovom zdroji. Vďaka jazyku LINQ sa môžeme koncentrovať skôr na to „čo chceme urobiť“, než na to „ako to chceme urobiť“. Toto je jedna z najviac citovaných konkurenčných výhod jazyka LINQ, pretože

dátovo orientované programové dopyty smieme vytvárať konzistentným spôsobom bez ohľadu na to, či budú smerované na množiny objektov, polia, relačné databázy, alebo XML dokumenty.

3. Dátová operácia, ktorá je vyjadrená v syntakticky správne zapísanej postupnosti operátorov jazyka LINQ, predstavuje dopyt, ktorý bude zaslaný cieľovému dátovému zdroju. Výsledok dátovej operácie bude zapuzdrený do typovo silného objektu. Keďže typ vráteného objektu s požadovanými dátami bude vždy konkrétny, senzitívna technológia IntelliSense je schopná pomáhať vývojárom už v čase tvorby zdrojového kódu. Tak sa citeľne uľahčuje nielen proces tvorby dopytov, ale aj spracovanie ich výsledkov.

Jazyk LINQ spolupracuje s niekoľkými entitami, ktoré zabezpečujú komunikáciu medzi aplikáciou .NET napísanou v jazyku Visual Basic 2008 a cieľovým dátovým zdrojom. Tieto entity sú známe ako poskytovatelia. Hlavnou úlohou poskytovateľov je nadviazať obojsmerný informačný dialóg medzi aplikáciou a dátovým zdrojom. Ak vývojár zapíše v jazyku Visual Basic 2008 programový unifikovaný dopyt pomocou jazyka LINQ, poskytovateľ pretransformuje tento unifikovaný dopyt na konkrétny dopyt, ktorý bude zaslaný cieľovému dátovému zdroju. Výsledkom zaslaného dopytu budú dáta, resp. dátové kolekcie, ktoré budú poskytovateľom zapuzdrené do objektov tvoriacich výsledok aktivácie dopytu. S objektmi naplnenými požadovanými dátami môžu vývojári pracovať tak, ako s akýmikoľvek inými objektmi. Poskytovatelia jazyka LINQ prinášajú cennú pridanú hodnotu, ktorá vývojárom umožňuje pracovať na vyššej úrovni abstrakcie ako kedykoľvek predtým. Kooperácia medzi aplikáciou jazyka Visual Basic 2008, poskytovateľom a dátovými zdrojmi je schematicky znázornená na obr. 23.



Obr. 23: Komunikácia medzi aplikáciou .NET a dátovými zdrojmi pomocou poskytovateľa

V základnom vyhotovení obsahuje jazyk LINQ nasledujúcich poskytovateľov:

- **LINQ to Objects.** Poskytovateľ komunikuje s kolekciami objektov a poľami, ktoré vznikli z tried implementujúcich rozhranie IEnumerable z menného priestoru System.Collections, resp. generické rozhranie IEnumerable(Of T) z menného priestoru System.Collections.Generic. Dopyty môžu byť smerované na objekty tvoriace hierarchické kolekcie a grafy. Poskytovateľ LINQ to Objects je primárnym poskytovateľom pri práci s jazykom LINQ.

- **LINQ to ADO.NET**. V skutočnosti je LINQ to ADO.NET množina poskytovateľov, do ktorej patria títo poskytovatelia:

    - **LINQ to SQL:** Poskytovateľ slúži na manipuláciu s dátami, ktoré sú uložené v relačných databázach spravovaných databázovým serverom Microsoft SQL Server.

    - **LINQ to Entities:** Poskytovateľ s podobným funkčným záberom ako LINQ to SQL, len s tým rozdielom, že miesto fyzickej perzistencie dát je aplikovaná konceptuálna perzistencia dát prostredníctvom konceptuálneho modelu EDM (Entity Data Model).

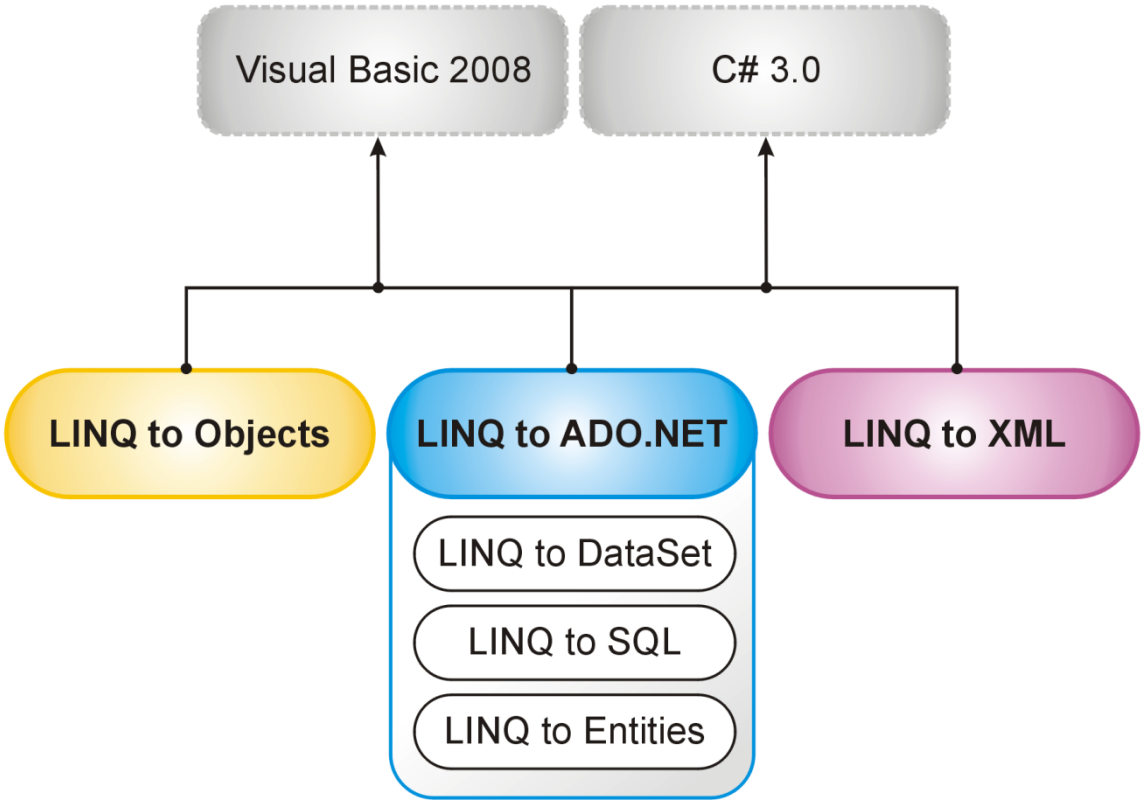
    - **LINQ to DataSet:** Poskytovateľ umožňuje posielať dopyty dátovým súpravám, ktoré sú základným komponentom softvérovej technológie ADO.NET. Dátové súpravy obsahujú pamäťovú reprezentáciu dát, ktoré boli získané z jedného, alebo aj viacerých dátových zdrojov. Pripomeňme, že dátové súpravy sú „odpojené“ od cieľových dátových zdrojov a predstavujú tak pamäťové kópie dát, ktoré sú fyzicky uložené v príslušných dátových zdrojoch.

- **LINQ to XML.** Poskytovateľ sa zameriava na prácu s elementmi a atribútmi XML stromov, ktoré tvoria obsahy XML dokumentov. Podobne ako DOM (Document Object Model), aj poskytovateľ LINQ to XML podporuje možnosti pamäťovej modifikácie načítaných XML dokumentov. Dopyty smerované na XML dokumenty sa zo syntaktickej stránky ponášajú na SQL dopyty zasielané relačným databázam. Po svojej zmene môžu byť pamäťové reprezentácie XML dokumentov uložené späť do súboru, alebo serializované (dodajme, že pri XML serializácii ide o tzv. plytkú serializáciu XML dokumentov). Medzi najčastejšie dopyty patria tie, ktoré prehľadávajú XML dokumenty a získavajú z nich požadované množiny elementov a atribútov. Analyzované XML stromy môžu byť opäť podrobené transformácii, ktorá je spojená so zmenou ich pôvodnej kompozície. Za asistencie poskytovateľa LINQ to XML smieme vykonávať veľa štandardných aj menej frekventovaných operácií s dátami uskladnenými v XML dokumentoch. Zo všetkých uvádzame aspoň niektoré:

    - Načítavanie XML dokumentov zo súborov.
    - Ukladanie (serializovanie) XML dokumentov do súborov.
    - Vytváranie XML elementov a XML dokumentov pomocou XML konštánt.

- Manipulovanie s XML dokumentmi dynamicky pomocou pridružených programových výrazov.
- Uskutočňovanie validácie XML stromov pomocou schém XML (XSD).

Symbiózu medzi jazykmi Visual Basic 2008 a C# 3.0 a dátovými poskytovateľmi unifikovaného jazyka LINQ znázorňuje obr. 24.



Obr. 24: Visual Basic 2008 a C# 3.0 v spoločnosti dátových poskytovateľov unifikovaného jazyka LINQ

LINQ to Objects: Praktické experimenty

Dopyty v jazyku LINQ sa vytvárajú podľa nasledujúcej triády:

1. Získanie prístupu k dátovému zdroju.
2. Vytvorenie dopytu.
3. Spracovanie dopytu na dátovom zdroji.

Nadchádzajúci fragment zdrojového kódu predvádza praktickú aplikáciu spomenutej triády:

```
Module Module1

    Sub Main()
        'Dátovým zdrojom je pole celých čísel.
        Dim Pole() As Integer = {1, 2, 3, 4, 5}
        'Vytvorenie dopytu.
        Dim Dopyt = From Prvok In Pole _
```

```
                    Where Prvok > 3 _
                    Select Prvok
        'Spracovanie dopytu na dátovom zdroji.
        For Each Prvok As Integer In Dopyt
            Console.WriteLine(Prvok)
        Next
        Console.Read()
    End Sub

End Module
```

Dátovým zdrojom je v našom prípade jednorozmerné pole celých čísel, ktoré je počas svojej definície okamžite inicializované pomocou inicializačného zoznamu. Skonštruované vektorové pole je inštanciou podtriedy triedy System.Array, ktorá implementuje generické rozhranie IEnumerable(Of T) z menného priestoru System.Collections.Generic. Akákoľvek trieda, ktorá implementuje rozhranie IEnumerable alebo rozhranie IEnumerable(Of T), resp. akákoľvek trieda, ktorá je odvodená od triedy implementujúcej jedno z uvedených rozhraní, môže slúžiť na vytváranie inštancií, ktoré budú môcť byť použité pri tvorbe a spracovaní dopytov jazyka LINQ. Triede, ktorá spĺňa tieto náležitosti, vravíme trieda s možnosťou dopytovania.

Dopyt jazyka LINQ vytvárame pomocou predpísanej syntaktickej konštrukcie:

```
'Vytvorenie dopytu.
Dim Dopyt = From Prvok In Pole _
            Where Prvok > 3 _
            Select Prvok
```

Tento príkaz predstavuje definičnú inicializáciu odkazovej premennej s identifikátorom Dopyt. Ako môžeme vidieť, dátový typ premennej nie je špecifikovaný. To je ale v poriadku, pretože tento príkaz (a pravdu povediac, drvivá väčšina všetkých ostatných príkazov jazyka LINQ) sa spolieha na implicitnú typovú inferenciu lokálnych entít. Samozrejme, k determinácii dátového typu definovanej odkazovej premennej sa dostaneme, no teraz skúsme upriamiť našu pozornosť na výraz, ktorý sa nachádza na pravej strane priraďovacieho príkazu. Ide o tzv. dopytovací výraz, ktorý sa skladá z troch štandardných dopytovacích operátorov From, Where a Select. Celý dopytovací výraz si teraz rozdelíme na menšie podvýrazy, ktoré súvisia s jednotlivými dopytovacími operátormi. Prvý podvýraz pracuje s operátorom From a jeho úlohou je určiť dátový zdroj, ktorý mienime použiť. Naším dátovým zdrojom je jednorozmerné pole, ktorého identifikátor uvádzame za kľúčovým slovom In. Premenná Prvok stojaca za operátorom From zohráva podobnú úlohu ako riadiaca premenná v cykloch For a For Each. Zmyslom existencie tejto premennej je získanie prístupu k položkám dátového zdroja, avšak až vo chvíli, keď bude dopyt reprezentovaný dopytovacím výrazom spracovaný. Dátový typ premennej Prvok je implicitne inferovaný, takže ho nemusíme priamo uvádzať. V každom prípade ale musí platiť, že dátový typ premennej Prvok musí byť kompatibilný s dátovým typom každej položky dátového zdroja. V našom prípade je dátovým zdrojom celočíselné pole, takže je zrejmé, že implicitne priradeným dátovým typom premennej Prvok bude Integer.

Druhý podvýraz obsahuje štandardný dopytovací operátor Where. Tento operátor je asociovaný s predikátom, teda podmienkou, podľa ktorej budú filtrované analyzované dáta dátového zdroja. V našej ukážke bude hodnotu dopytovacieho výrazu tvoriť množina pozostávajúca iba z tých prvkov poľa, ktorých hodnota je väčšia ako 3. Posledný podvýraz je vymedzený operátorom Select

a práve pomocou neho dochádza k určeniu hodnoty celého dopytovacieho výrazu (a k navráteniu tých položiek dátového zdroja, ktoré zodpovedajú zadanému predikátu). Nie je tajomstvom, že náš dopytovací výraz bude navracať množinu celočíselných hodnôt väčších ako 3.

Dátový typ odkazovej premennej Dopyt musí byť kompatibilný s dátovým typom hodnoty podvýrazu s operátorom Select. V zobrazenom zdrojovom kóde bude typ premennej Dopyt automaticky inferovaný ako IEnumerable(Of Integer), pričom hodnota podvýrazu Select Prvok bude rovnako typu Integer.

Vytvorený dopyt je uložený v premennej Dopyt. Radi by sme zdôraznili, že v uvedenej premennej je uložený dopytovací výraz, presnejšie definícia dopytovacieho výrazu. V premennej teda nie je uložená hodnota dopytovacieho výrazu, pretože tento výraz bude vyhodnotený až po svojom spracovaní. Spracovanie dopytovacieho výrazu sa uskutočňuje spravidla v tele iteratívneho príkazu:

```
'Spracovanie dopytu na dátovom zdroji.
For Each Prvok As Integer In Dopyt
    Console.WriteLine(Prvok)
Next
```

Programový cyklus prechádza množinu dátových položiek, ktoré vyhovovali zadaniu dopytovacieho výrazu a zobrazuje ich hodnoty v okne príkazového riadka. Tento praktický experiment demonštruje techniku, ktorá je známa ako odložené spracovanie dopytovacieho výrazu (dopytu) unifikovaného jazyka LINQ. Podstatou tohto mechanizmu je, že definíciu dopytovacieho výrazu najskôr uložíme do pripravenej odkazovej premennej implicitne inferovaného dátového typu a až neskôr tento dopytovací výraz spracujeme.

Nasledujúci fragment zdrojového kódu jazyka Visual Basic 2008 používa dopyt na nájdenie všetkých procesov, ktoré aktuálne bežia v operačnom systéme:

```
Imports System.Diagnostics
Module Module1

    Sub Main()
        Dim Procesy = From Proces In Process.GetProcesses() _
                      Order By Proces.ProcessName Ascending _
                      Select Proces.ProcessName
        For Each Proces In Procesy
            Console.WriteLine(Proces)
        Next
        Console.Read()
    End Sub

End Module
```

Všade, kde sa to len dá, sa spoliehame na implicitnú typovú inferenciu. Novinkou je použitie štandardného dopytovacieho operátora Order By, ktorý zaistí lexikografické zoradenie kolekcie procesov.

Jemnou obmenou získame množinu aktívnych procesov, ktorých názvy sa začínajú písmenom „W“:

```
Imports System.Diagnostics
Module Module1

    Sub Main()
        Dim Procesy = From Proces In Process.GetProcesses() _
                      Where Proces.ProcessName.StartsWith("W") _
                      Select Proces _
                      Order By Proces.ProcessName Ascending
        For Each Proces In Procesy
            Console.WriteLine(Proces.ProcessName)
        Next
        Console.Read()
    End Sub

End Module
```

Programátori môžu dopyty vysielať tiež k poliam, ktorých prvkami sú inštancie vopred deklarovanej triedy. Povedzme, že v jazyku Visual Basic 2008 vytvoríme triedu reflektujúcu základné charakteristiky softvérového vývojára:

```
Public Class Vývojár

    'Dátová sekcia triedy.
    Private m_Meno As String
    Private m_Priezvisko As String
    Private m_Jazyky() As String

    'Verejne prístupný parametrický inštančný konštruktor.
    Public Sub New(ByVal Meno As String, ByVal Priezvisko As String, _
        ByRef Jazyky() As String)
        Me.m_Meno = Meno
        Me.m_Priezvisko = Priezvisko
        Me.m_Jazyky = Jazyky
    End Sub


    'Verejne prístupné skalárne vlastnosti určené len na čítanie.
    Public ReadOnly Property Meno() As String
        Get
            Return Me.m_Meno
        End Get
    End Property

    Public ReadOnly Property Priezvisko() As String
        Get
            Return Me.m_Priezvisko
        End Get
    End Property

    Public ReadOnly Property ProgramovacieJazyky() As String()
        Get
            Return Me.m_Jazyky
        End Get
    End Property
End Class
```

V tele hlavnej metódy Main založíme a inicializujeme pole inštancií triedy Vývojár:

```
'Založenie a inicializovanie poľa inštancií triedy Vývojár.
Dim Vývojári() As Vývojár = { _
    New Vývojár("Jozef", "Modrý", New String() {"Visual Basic"}), _
    New Vývojár("Milan", "Lenner", New String() {"C#"}), _
    New Vývojár("Augustín", "Hrnčiar", New String() {"C#"})}
```

V ďalšom kroku zhotovíme dopyt, prostredníctvom ktorého budeme môcť získať informácie len o tých vývojároch, ktorí sa špecializujú na programovací jazyk C#:

```
'Vytvorenie dopytu.
Dim TímVývojárov = From Vývojár In Vývojári _
                   Where Vývojár.ProgramovacieJazyky.Contains("C#") _
                   Select Vývojár
```

Dopyt spracujeme v cykle For Each:

```
'Spracovanie dopytu a zobrazenie výsledku.
For Each Vývojár In TímVývojárov
    Console.WriteLine("Meno a priezvisko vývojára: {0} {1}.", _
        Vývojár.Meno, Vývojár.Priezvisko)
Next
```

Implicitne inferovaný dátový typ odkazovej premennej TímVývojárov bude IEnumerable(Of Vývojár) z menného priestoru System.Collections.Generic. Implicitne inferovaný dátový typ riadiacej premennej cyklu For Each bude používateľsky deklarovaný odkazový dátový typ s názvom Vývojár.

Hoci to nie je potrebné, ak budeme chcieť, môžeme dátové typy použitých premenných určiť sami:

```
Imports System.Collections.Generic

Dim Vývojári() As Vývojár = { _
    New Vývojár("Jozef", "Modrý", New String() {"Visual Basic"}), _
    New Vývojár("Milan", "Lenner", New String() {"C#"}), _
    New Vývojár("Augustín", "Hrnčiar", New String() {"C#"})}


'Explicitná determinácia dátových typov premenných.
Dim TímVývojárov As IEnumerable(Of Vývojár) = _
    From Vývojár As Vývojár In Vývojári _
    Where Vývojár.ProgramovacieJazyky.Contains("C#") _
    Select Vývojár

For Each Vývojár As Vývojár In TímVývojárov
    Console.WriteLine("Meno a priezvisko vývojára: {0} {1}.", _
        Vývojár.Meno, Vývojár.Priezvisko)
Next
```

Okamžité spracovanie dopytu unifikovaného jazyka LINQ

Hoci v praxi sa najčastejšie stretneme s odloženým spracovaním dopytov jazyka LINQ, existujú situácie, kedy môžeme uprednostniť okamžité vyhodnotenie dopytovacieho výrazu. Termínom „okamžité vyhodnotenie“ máme na mysli vyhodnotenie dopytovacieho výrazu ihneď po jeho korektnej definícii. Vzhľadom na to, že dopytovací výraz bude bezprostredne po svojej definícii spracovaný, získame množinu položiek dátového zdroja, ktoré vyhovujú nášmu zadaniu. Napríklad, predchádzajúcu ukážku kódu s jednorozmerným poľom inštancií triedy Vývojár by sme mohli prepracovať nasledujúcim spôsobom:

```
Dim Vývojári() As Vývojár = { _
    New Vývojár("Jozef", "Modrý", New String() {"Visual Basic"}), _
    New Vývojár("Milan", "Lenner", New String() {"C#"}), _
    New Vývojár("Augustín", "Hrnčiar", New String() {"C#"})}

Dim TímVývojárov() As Vývojár = _
    (From Vývojár In Vývojári _
    Where Vývojár.ProgramovacieJazyky.Contains("C#") _
    Select Vývojár).ToArray()

For Each Vývojár In TímVývojárov
    Console.WriteLine("Meno a priezvisko vývojára: {0} {1}.", _
        Vývojár.Meno, Vývojár.Priezvisko)
Next
```

Napriek tomu, že modifikácia je vo vzťahu k svojmu originálu funkčne ekvivalentná, syntakticko-sémantické vyjadrenie je úplne odlišné. Predovšetkým musíme poukázať na použitie rozširujúcej generickej metódy ToArray, ktorej všeobecná definícia vyzerá takto:

```
<ExtensionAttribute> _
Public Shared Function ToArray(Of TSource)( _
      source As IEnumerable(Of TSource) _
) As TSource()

    'Telo rozširujúcej metódy je vynechané.

End Function
```

Metóda ToArray spôsobí okamžité spracovanie definovaného dopytovacieho výrazu, pričom v podobe svojej návratovej hodnoty vracia odkaz na pole, ktoré je naplnené kópiami dátových objektov dátového zdroja, ktoré spĺňajú podmienky dopytu. Ako sme už uviedli, metóda ToArray je generická, pričom ako implicitný argument prijíma odkaz na kolekciu objektov, ktorá implementuje generické rozhranie IEnumerable(Of T). Rovnako si musíme uvedomiť, že metódu ToArray môžeme volať len v súvislosti s dopytovacím výrazom, ktorého definícia musí byť uzatvorená do zátvoriek. Po okamžitom spracovaní dopytu nám metóda ToArray poskytne odkaz na pole typovo silných objektov. Ponúknutý odkaz ukladáme do odkazovej premennej TímVývojárov, ktorej dátový typ explicitne definujeme. Zvyšok zdrojového kódu je už potom ľahko pochopiteľný.

LINQ to DataSet: Praktické experimenty

S príchodom programovacieho jazyka Visual Basic .NET sme mohli požadované množiny dátových položiek jednopoužívateľských databáz (ku ktorým patria najmä databázy programu Microsoft Office Access) i viacpoužívateľských databáz (predovšetkým databázy umiestnené na databázovom serveri Microsoft SQL Server) načítavať pomocou objektov dátovo orientovanej technologickej platformy ADO.NET. Počas svojej evolúcie sa databázová platforma ADO.NET dostala v roku 2005 do svojej druhej verzie (2.0). Platforma ADO.NET, s ktorou sa stretávame v jazyku Visual Basic 2008, absolvovala ďalšie inovatívne vylepšenia, vďaka ktorým sa z nej stala prvotriedna kolekcia aplikačných programových rozhraní, prostredníctvom ktorých môžu vývojári realizovať rozmanité manipulačné operácie s dátami uskladnenými v lokálnych alebo vzdialených dátových zdrojoch.

Prvá praktická ukážka nás zoznamuje s prístupom, pomocou ktorého smieme získať zoznam produktových kategórií ponúkaných obchodnou spoločnosťou:

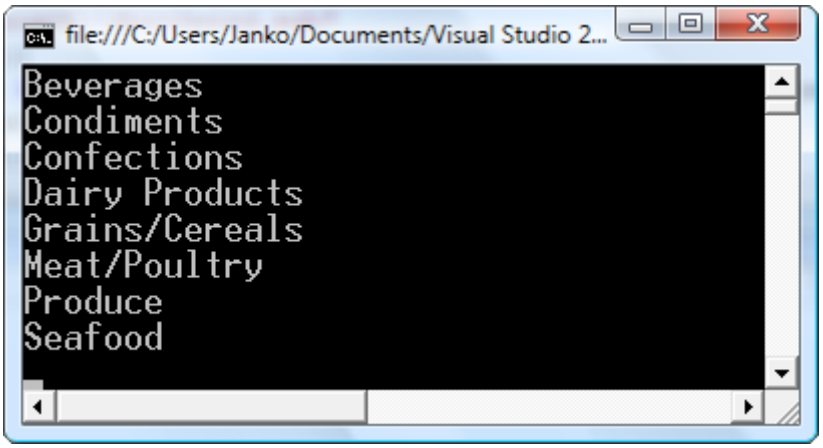
```
'Zavedenie menného priestoru pre prácu s databázami, s ktorými manipulujeme
'pomocou dátového poskytovateľa OLE DB.
Imports System.Data.OleDb
Module Module1
    Sub Main()
        'Vytvorenie objektu, ktorý realizuje spojenie s cieľovou databázou.
        Dim Spojenie As New OleDbConnection()

        'Špecifikácia pripojovacieho reťazca.
        Spojenie.ConnectionString = "Provider=Microsoft.Jet.OLEDB.4.0;" & _
            "Data Source=C:\Northwind.mdb"

        'Otvorenie spojenia s databázou.
        Spojenie.Open()

        'Projektovanie príkazu, ktorý zapuzdruje dopytovací výraz
        'zabezpečujúci načítanie všetkých produktových kategórií
        'obchodnej spoločnosti.
        Dim Príkaz As New OleDbCommand("SELECT * FROM Categories", _
           Spojenie)

        'Vytvorenie dátoveho čítača, spracovanie dopytu a navrátenie
        'množiny dátových položiek, ktoré vyhovujú kritériam zadaným
        'v dopytovacom výraze.
        Dim DátovýČítač As OleDbDataReader = Príkaz.ExecuteReader()

        'Iterovanie medzi položkami získanej množiny a zobrazovanie
        'názvov jednotlivých produktových kategórií.
        While DátovýČítač.Read()
            Console.WriteLine(DátovýČítač("CategoryName").ToString())
        End While

        'Uzatvorenie dátového čítača.
        DátovýČítač.Close()

        'Ukončenie spojenia s cieľovou databázou.
        Spojenie.Close()
        Console.Read()
    End Sub
End Module
```

Experiment používa populárnu databázu Northwind, ktorá obsahuje údaje o rôznych entitách obchodnej spoločnosti ako sú zamestnanci, klienti, produktové kategórie, produkty, objednávky, dodávatelia, odberatelia a iné. K databáze budeme pristupovať pomocou OLE DB dátového poskytovateľa (ak by bola naša databáza nainštalovaná na Microsoft SQL Serveri, použili by sme poskytovateľa pre SQL Server). Jazyk Visual Basic 2008 je plne objektovo orientovaný, a preto neprekvapuje, že s databázami budeme komunikovať pomocou súpravy objektov, ktorých triedne deklarácie sa nachádzajú v mennom priestore System.Data.OleDb. Ak chceme z databázy získať množinu požadovaných dát (zoznam produktových kategórií v našom prípade), potrebujeme tri objekty:

1. Spojovací objekt (inštancia triedy OleDbConnection).
2. Príkazový objekt (inštancia triedy OleDbCommand).
3. Dátový čítač (inštancia triedy OleDbDataReader).

Spojovací objekt nadviaže pomocou vhodne konfigurovaného pripojovacieho reťazca spojenie s cieľovou databázou. Z pripojovacieho reťazca vieme vyčítať, že naším dátovým poskytovateľom bude Microsoft.Jet.OLEDB.4.0 (patrí do rodiny dátových poskytovateľov OLE DB). Rovnako sa dozvedáme, že súbor s databázou má identifikátor Northwind.mdb a je umiestnený na disku C. Spojovací objekt sprístupňuje inštančnú metódu Open, ktorá nadviaže spojenie s cieľovou databázou. V tomto okamihu je spojenie s databázou úspešne otvorené, takže môžeme zahájiť komunikačný dialóg. Vzhľadom na to, že naším zámerom je zhotoviť zoznam produktových kategórií ponúkaných obchodnou spoločnosťou, vytvárame príkazový objekt. Príkazový objekt nám pomôže vyjadriť naše myšlienky vo forme príkazu jazyka SQL, ktorý bude spracovaný na databáze. Príkazový objekt vystupuje ako inštancia triedy OleDbCommand, pričom zapuzdruje dopytovací výraz, ktorý je tvorený príkazom jazyka SQL. Výraz SELECT * FROM Categories vyberá všetky dátové záznamy, ktoré sú situované v dátovej tabuľke Categories. Ako si môžeme všimnúť, príkazový objekt sa v plnej miere spolieha na spojovací objekt. To je zrejmé podľa jednoznačnej indície: inštančný konštruktor triedy OleDbCommand prijíma ako svoj druhý argument odkaz na objekt triedy OleDbConnection.

Príkazový objekt nám poskytuje zmes inštančných metód, z ktorých si vyberáme jednu – ExecuteReader. Táto metóda spracuje databázový dopyt a sama nám ponúkne odkaz na objekt triedy OleDbDataReader. Tento objekt môžeme s výhodou použiť pri iterovaní výslednej množiny dátových položiek, ktoré vyhovujú stanoveným požiadavkám. Prechádzanie medzi názvami produktových kategórií realizujeme v tele cyklu While, pričom počet opakovaní determinuje metóda Read dátového čítača. Na výstupe budú zobrazené všetky záznamy, ktoré sú uvedené v stĺpci CategoryName dátovej tabuľky Categories (obr. 25).



Obr. 25: Výstup dátovej operácie

Vo chvíli, keď sme úspešne splnili náš cieľ, explicitne uzatvárame dátový čítač a ukončujeme spojenie s databázou.

Upozornenie: Dátový čítač (inštancia triedy OleDbDataReader) vykonáva prúdový prístup k výslednej množine databázových dopytov. Je potrebné si uvedomiť, že dátový čítač je schopný uskutočňovať iba dopredné čítanie hodnôt dátových položiek uložených v analyzovanej dátovej množine. Objekt nedokáže meniť obsah vrátených dátových položiek, ani propagovať modifikácie do cieľovej databázy.

Aj keď spojovací objekt, príkazový objekt a dátový čítač tvoria umne navrhnutú trojicu pomocníkov, v praxi sa častejšie stretávame s priamym použitím dátových súprav a dátových adaptérov. Zatiaľ čo dátová súprava (inštancia triedy DataSet z menného priestoru System.Data) predstavuje pamäťovú kópiu vymedzeného fragmentu databázy, dátový adaptér (inštancia triedy OleDbDataAdapter z menného priestoru System.Data.OleDb, resp. inštancia triedy SqlDataAdapter z menného priestoru System.Data.SqlClient) plní úlohu prostredníka medzi dátovou súpravou a fyzickým dátovým zdrojom (databázou). Hlavná úloha dátového adaptéra spočíva v poskytnutí vyššej vrstvy abstrakcie pri vykonávaní činností, ktoré sú spojené s dátovými súpravami a dátovými zdrojmi.

Nasledujúci výpis zdrojového kódu jazyka Visual Basic 2008 predvádza praktickú aplikáciu dátového adaptéra a dátovej súpravy. Dovolíme si konštatovať, že program je funkčne ekvivalentný s programom pripraveným v predchádzajúcej praktickej ukážke.

```
'Zavedenie menného priestoru pre prácu s databázami, s ktorými manipulujeme
'pomocou dátového poskytovateľa OLE DB.
Imports System.Data.OleDb
Module Module1
    Sub Main()
        Dim DátovýAdaptér As OleDbDataAdapter

        'Založenie dátového adaptéra a jeho konfigurácia.
        DátovýAdaptér = New OleDbDataAdapter( _
            New OleDbCommand("SELECT * FROM Categories", _
            New OleDbConnection("Provider=Microsoft.Jet.OLEDB.4.0;" & _
                                "Data Source=C:\Northwind.mdb")))
```

```
        'Vytvorenie dátovej súpravy.
        Dim DátováSúprava As New DataSet()

        'Naplnenie dátovej súpravy výslednou množinou dátových záznamov.
        DátovýAdaptér.Fill(DátováSúprava, "Categories")

        'Iterovanie medzi položkami výslednej množiny dátových záznamov.
        For Each Riadok As DataRow In _
            DátováSúprava.Tables("Categories").Rows
                Console.WriteLine(Riadok.Item("CategoryName"))
        Next

        'Zapísanie obsahu dátovej súpravy do XML súboru.
        DátováSúprava.WriteXml(New IO.StreamWriter("C:\Dáta_XML.xml"), _
            XmlWriteMode.WriteSchema)
        Console.WriteLine("Obsah dátovej súpravy bol úspešne " & _
            "zapísaný do XML súboru.")
        Console.Read()
    End Sub
End Module
```

Pri konštrukcii dátového adaptéra delegujeme určité právomoci na vytvorený spojovací a príkazový objekt. Len čo zostrojíme dátovú súpravu, voláme inštančnú metódu Fill dátového adaptéra. Táto metóda inicializuje dátovú súpravu množinou dátových záznamov, ktoré tvoria výsledok dopytu zaslaného databáze. V našom prípade pracujeme iba s jednou dátovou tabuľkou, takže zobrazenie záznamov zvládneme v jednom cykle For Each. Obsah dátovej súpravy nakoniec vkladáme do vytvoreného XML dokumentu, ktorý odosielame do fyzického XML súboru.

Tip: Istotne stojí za zmienku, že program nikde explicitne neotvára ani neuzatvára spojenie s databázou. V skutočnosti je spojenie s databázou udržiavané len tak dlho, kým metóda Fill dátového adaptéra získa výslednú množinu dátových záznamov. V momente, keď je výsledok dopytu spracovaný, dátový adaptér sa odpája od databázy a vkladá získanú množinu dátových záznamov do dátovej súpravy. Z uvedeného je zrejmé, že dátová súprava pôsobí ako odpojený dátový zdroj, ktorý nedisponuje žiadnymi priamymi väzbami na cieľový fyzický dátový zdroj.

Poskytovateľ LINQ to DataSet dovoľuje programátorom používať unifikovaný jazyk LINQ na deklaratívne vytváranie dopytov, ktoré budú smerované na inicializované dátové súpravy. Ukážme si, ako by zobrazenie produktových kategórií obchodnej spoločnosti vyzeralo vtedy, ak by sme použili jazyk LINQ:

```
Imports System.Data.OleDb

Module Module1
    Sub Main()
        Dim DátovýAdaptér As OleDbDataAdapter
        DátovýAdaptér = New OleDbDataAdapter("SELECT * FROM Categories", _
            "Provider=Microsoft.Jet.OLEDB.4.0;" & _
            "Data Source=C:\Northwind.mdb")
        Dim DátováSúprava As New DataSet()
```

```
        DátovýAdaptér.Fill(DátováSúprava, "Categories")

        'Získanie prístupu k dátovej tabuľke.
        Dim DátováTabuľka As DataTable = DátováSúprava.Tables("Categories")

        'Definícia dopytovacieho výrazu jazyka LINQ.
        Dim Kategórie = From Kategória In DátováTabuľka.AsEnumerable() _
                    Select Kategória.Field(Of String)("CategoryName")

        'Spracovanie definovaného dopytovacieho výrazu jazyka LINQ.
        For Each Kategória In Kategórie
            Console.WriteLine(Kategória)
        Next

    End Sub
End Module
```

Keďže chceme vysielať dopyty voči dátovej súprave, je logické, že takáto dátová súprava musí byť v pamäti inštanciovaná a zároveň aj inicializovaná. Na inicializáciu dátovej súpravy používame overený variant, ktorý spolupracuje s dátovým adaptérom. Naša dátová súprava obsahuje len jednu dátovú tabuľku. Povedané technickejšie, objektová kolekcia DataSet.Tables je zložená len z jedného objektu triedy DataTable. Pozrime sa bližšie na definíciu dopytovacieho výrazu jazyka LINQ. Pretože dátová tabuľka nie je inštanciou triedy, ktorá by implementovala generické rozhranie IEnumerable(Of T), musíme v súvislosti s ňou (a teda inštanciou triedy DataTable) aktivovať rozširujúcu metódu AsEnumerable. Táto metóda ako svoj prvý a implicitný argument prijíma odkaz na dátovú tabuľku, ktorá bude predmetom jej spracovania. Metóda AsEnumerable preskenuje dátová tabuľku, vytvorí novú kolekciu záznamov uložených v dátovej tabuľke a odkaz na túto kolekciu vráti v podobe svojej návratovej hodnoty typu EnumerableRowCollection(Of DataRow). Pretože objekt kolekcie vzišiel z triedy, ktorá implementuje generické rozhranie IEnumerable(Of T), môže byť použitý pri tvorbe dopytovacieho výrazu jazyka LINQ. Dátový typ odkazovej premennej Kategória je implicitne inferovaný ako DataRow.

Štandardný dopytovací operátor Select uskutočňuje selekciu všetkých dátových záznamov, ktoré tvoria stĺpec s názvom produktovej kategórie. Výberový podvýraz operuje s rozširujúcou parametrickou metódou Field, ktorá nám dovoľuje získať prístup k typovo silnej hodnote dátového záznamu v špecifikovanom stĺpci dátovej tabuľky. Dopyt jazyka LINQ teda vyberá všetky záznamy, ktoré reprezentujú názvy kategórií predávaných produktov. Spracovanie dopytu sa odohráva v cykle For Each.

Ďalší praktický fragment zdrojového kódu jazyka Visual Basic 2008 zisťuje hodnotu kumulatívnych prepravných nákladov vynaložených na objednávky, ktoré boli realizované v roku 1997:

```
Imports System.Data.OleDb

Module Module1
    Sub Main()
        Dim DátovýAdaptér As OleDbDataAdapter
        DátovýAdaptér = New OleDbDataAdapter("SELECT * FROM Orders", _
            "Provider=Microsoft.Jet.OLEDB.4.0;" & _
            "Data Source=C:\Northwind.mdb")
```

```
        Dim DátováSúprava As New DataSet()
        DátovýAdaptér.Fill(DátováSúprava, "Orders")

        Dim DátováTabuľka As DataTable = DátováSúprava.Tables("Orders")
        Dim Objednávky = From Objednávka In DátováTabuľka.AsEnumerable() _
            Where Objednávka.Field(Of Date)("OrderDate").Year = 1997 _
            Select Objednávka.Field(Of Object)("Freight")

        Dim PrepravnéNáklady As Double
        For Each Objednávka In Objednávky
            PrepravnéNáklady += CType(Objednávka, Double)
        Next

        Console.WriteLine("Hodnota kumulatívnych prepravných " & _
            "nákladov je " & PrepravnéNáklady & " USD.")
        Console.Read()
    End Sub

End Module
```

LINQ to XML: Praktické experimenty

O tom, že Visual Basic 2008 si veľmi dobre rozumie s XML konštantami, sme sa dozvedeli v kapitole 6: XML konštanty a XML konštanty s pridruženými programovými výrazmi. Pomocou unifikovaného jazyka LINQ však dokážeme omnoho viac než len deklarovať XML konštanty. S intuitívnymi syntaktickými konštrukciami môžeme vysielať dopyty voči XML stromom. Uvažujme nasledujúci program:

```
'Import menného priestoru pre rozhranie LINQ to XML.
Imports System.Xml.Linq

Module Module1
    Sub Main()
        'Deklarácia XML konštanty.
        Dim XMLPočítač As XElement = _
        <Počítače>
            <Počítač>
                <Procesor>AMD Turion 64 X2 </Procesor>
                <Takt>4 GHz</Takt>
                <ViacJadier>Áno</ViacJadier>
                <RAM>2 GB</RAM>
                <HDD>320 GB</HDD>
                <VGA>NVidia Geforce 8600M GS</VGA>
            </Počítač>
            <Počítač>
                <Procesor>Intel Pentium 4 s HT technológiou</Procesor>
                <Takt>3 GHz</Takt>
                <ViacJadier>Nie</ViacJadier>
                <RAM>512 MB</RAM>
                <HDD>160 GB</HDD>
                <VGA>ATI Radeon X1950 XTX</VGA>
            </Počítač>
            <Počítač>
                <Procesor>Intel Core 2 Duo</Procesor>
                <Takt>4.26 GHz</Takt>
                <ViacJadier>Áno</ViacJadier>
```

```
                <RAM>4 GB</RAM>
                <HDD>500 GB</HDD>
                <VGA>Nvidia GeForce 8800 GT</VGA>
            </Počítač>
        </Počítače>

        'Definícia dopytovacieho výrazu, ktorý získa len tie počítače,
        'ktoré sú osadené viacjadrovými procesormi.
        Dim Počítače = From Počítač In XMLPočítač...<Počítač> _
                       Where Počítač.<ViacJadier>.Value = "Áno" _
                       Select Počítač

        'Spracovanie dopytovacieho výrazu. Na výstupe zobrazujeme
        'produktové názvy viacjadrových procesorov nainštalovaných
        'v analyzovaných počítačoch.
        For Each Počítač In Počítače
            Console.WriteLine(Počítač.<Procesor>.Value)
        Next
        Console.Read()
    End Sub

End Module
```

Komentár k programu: Navzdory tomu, že explicitne zavádzame menný priestor System.Xml.Linq, táto akcia má v kontexte aplikácií bežiacich na vývojovo-exekučnej platforme Microsoft .NET Framework 3.5 len fakultatívny charakter. Uvedený menný priestor je totiž automaticky zaradený do projektu jazyka Visual Basic 2008 okamžite po jeho vytvorení. XML konštanta, ktorú deklarujeme, disponuje koreňovým elementom Počítače, v ktorom sa nachádza trojica vnorených elementov s identifikátorom Počítač. Každý z týchto elementov charakterizuje počítač, pričom sledujeme vlastnosti ako je typ procesora (element Procesor), takt procesora (element Takt, pri viacjadrových procesoroch ide o súčet taktov všetkých inštalovaných exekučných jadier), prítomnosť jedného alebo viacerých exekučných jadier procesora (element ViacJadier), veľkosť operačnej pamäte (element RAM), kapacita pevného disku (element HDD) a typ grafickej karty (element VGA).

Dopytovací výraz, ktorý nám navráti kolekciu elementov predstavujúcich počítače s viacjadrovými procesormi, používa syntakticko-sémantické novinky jazyka Visual Basic 2008, ktorými sú tzv. osové vlastnosti. Osové vlastnosti sú na syntaktickej úrovni reprezentované špeciálnymi výrazmi, prostredníctvom ktorých môžeme priamo pristupovať k rôznym entitám XML stromov, predovšetkým k elementom a ich atribútom. Osová vlastnosť ...<Počítač> zabezpečuje prístup ku všetkým elementom s identifikátorom Počítač. Keďže náš XML strom je tvorený kolekciou objektov triedy XElement, môžeme konštatovať, že osová vlastnosť ...<Počítač> nám poskytne odkaz na kolekciu objektov triedy XElement, ktoré reprezentujú elementy s názvom Počítač. Dodajme, že získaný odkaz na kolekciu objektov triedy XElement bude dátového typu IEnumerable(Of XElement). Objektová kolekcia teda implementuje generické rozhranie IEnumerable(Of T), čo je nutnou podmienkou pre prácu s dopytmi jazyka LINQ v prostredí XML.

Keď máme kolekciu elementov, aplikujeme operátor Where a zapisujeme ďalšiu osovú vlastnosť, pomocou ktorej jednoznačnej determinujeme cieľový element, s ktorým chceme pracovať. V našej ukážke je týmto elementom element ViacJadier. Všimnime si, že osová vlastnosť priamo sprístupňujúca konkrétny element má tvar Počítač.<ViacJadier>. Cieľový

element je samozrejme tiež objektom triedy XElement. Ak chceme zistiť jeho hodnotu, zavoláme v súvislosti s týmto objektom rozširujúcu metódu Value. Táto metóda uskutoční analýzu cieľového elementu a vráti nám jeho hodnotu v podobe textového reťazca. Z deklarácie XML konštanty vieme určiť, že hodnotou elementu ViacJadier môže byť buď hodnota „Áno“, alebo hodnota „Nie“. Pretože podvýraz dopytovacieho výrazu zapísaný za operátorom Where sa zameriava na nájdenie počítačov s viacjadrovými procesormi, hodnotu navrátenú rozširujúcou metódou Value vložíme do relačného výrazu. Operátor Select potom do výslednej dátovej množiny zaraďuje len vyhovujúce elementy s identifikátorom ViacJadier.

Spracovanie definovaného dopytovacieho výrazu realizujeme v tele cyklu For Each, čo je neklamný dôkaz toho, že náš experiment demonštruje odložené spracovanie dopytov jazyka LINQ. Keďže chceme, aby boli na výstupe zobrazené produktové názvy viacjadrových procesorov, ktoré sú v zúčastnených počítačoch nainštalované, uplatňujeme osovú vlastnosť s explicitným určením elementu, ktorého hodnota sa má zobraziť. Je zrejmé, že týmto elementom bude Procesor, čiže získavame hodnotu objektu triedy XElement, ktorý uchováva textovú charakteristiku procesora.

Ak budeme abstrahovať od deklaratívnej povahy dopytovacích výrazov jazyka LINQ a pozrieme sa na ich nízkoúrovňovú technickú implementáciu, dôjdeme k pozoruhodným záverom. Počas našej pátracej akcie totiž zistíme, že osové vlastnosti sú v skutočnosti inteligentnými metódami. Predchádzajúce výpisy zdrojového kódu jazyka Visual Basic 2008, ktoré definujú a spracúvajú dopytovací výraz, by sme preto mohli, samozrejme so zachovaním pôvodnej funkcionality, prepracovať nasledujúcim spôsobom:

```
Dim Počítače = From Počítač In XMLPočítač.Descendants("Počítač") _
               Where Počítač.Element("ViacJadier").Value = "Áno" _
               Select Počítač

For Each Počítač In Počítače
    Console.WriteLine(Počítač.Element("Procesor").Value)
Next
```

Je iba na vývojároch, ktorý z uvedených štýlov sa rozhodnú preferovať. Výsledky sú vždy také isté, aj keď deklaratívne zadaný dopytovací výraz sa nám zdá predsa len pôsobivejší.

Ďalší program jazyka Visual Basic 2008 poukazuje na možnosti, ktoré nám dáva osová vlastnosť pre prístup k atribútom elementu.

```
Module Module1
    Sub Main()
        Dim XMLPočítač As XElement = _
        <Počítače>
            <Počítač>
                <Procesor L2cache="1024">AMD Turion 64 X2 </Procesor>
                <Takt>4 GHz</Takt>
                <ViacJadier>Áno</ViacJadier>
                <RAM>2 GB</RAM>
                <HDD>320 GB</HDD>
                <VGA>NVidia Geforce 8600M GS</VGA>
            </Počítač>
            <Počítač>
                <Procesor L2cache="1024">Intel Pentium 4 s HT
```

```
                                        technológiou</Procesor>
                <Takt>3 GHz</Takt>
                <ViacJadier>Nie</ViacJadier>
                <RAM>512 MB</RAM>
                <HDD>160 GB</HDD>
                <VGA>ATI Radeon X1950 XTX</VGA>
            </Počítač>
            <Počítač>
                <Procesor L2cache="2048">Intel Core 2 Duo</Procesor>
                <Takt>4.26 GHz</Takt>
                <ViacJadier>Áno</ViacJadier>
                <RAM>4 GB</RAM>
                <HDD>500 GB</HDD>
                <VGA>Nvidia GeForce 8800 GT</VGA>
            </Počítač>
        </Počítače>

        Dim Procesory = From Procesor In XMLPočítač...<Procesor> _
                        Where CType(Procesor.@<L2cache>, Integer) = 2048 _
                        Select Procesor

        For Each Procesor In Procesory
            Console.WriteLine("Procesor: {0}", Procesor.Value)

        Next
        Console.Read()
    End Sub

End Module
```

Do elementov Procesor jazyka XML deklarovaných v XML konštante sme doplnili znenie atribútov s identifikátorom L2cache. Tento atribút definuje veľkosť vyrovnávacej pamäte druhej úrovne procesora (ak ide o viacjadrový procesor, hodnota je kumulatívna). Pozornosť si zaslúži definícia dopytovacieho výrazu. Pomocou štandardného dopytovacieho operátora From a osovej vlastnosti ...<Procesor> získame prístup k všetkým elementom so zadaným identifikátorom. Operátor Where realizuje selekciu, pričom vyberá iba tie procesory, ktorých vyrovnávacia pamäť druhej úrovne je rovná 2 MB. Celý podvýraz dopytovacieho výrazu, ktorého je operátor Where súčasťou, je veľmi zaujímavý. Najskôr používame osovú vlastnosť pre priamy prístup k atribútu L2cache, ktorý je syntakticky stvárnený zápisom Procesor.@<L2cache>. K želanému atribútu sa dostaneme pomocou symbolu zavináča (@) a názvu atribútu, pričom samotný názov atribútu môže (ale nemusí) byť uzatvorený do lomených zátvoriek (<>). Atribút je súčasťou XML elementu a ako taký je zapuzdrený do objektu triedy XAttribute. Osová vlastnosť nám okamžite poskytne hodnotu atribútu, avšak v podobe textového reťazca. Keďže my chceme celočíselnú hodnotu, voláme konverzný operátor CType a vykonávame explicitnú typovú konverziu postupnosti textových znakov na odpovedajúcu celočíselnú hodnotu. Tú vzápätí porovnávame s konštantou predstavujúcou veľkosť vyrovnávacej pamäte druhej úrovne.

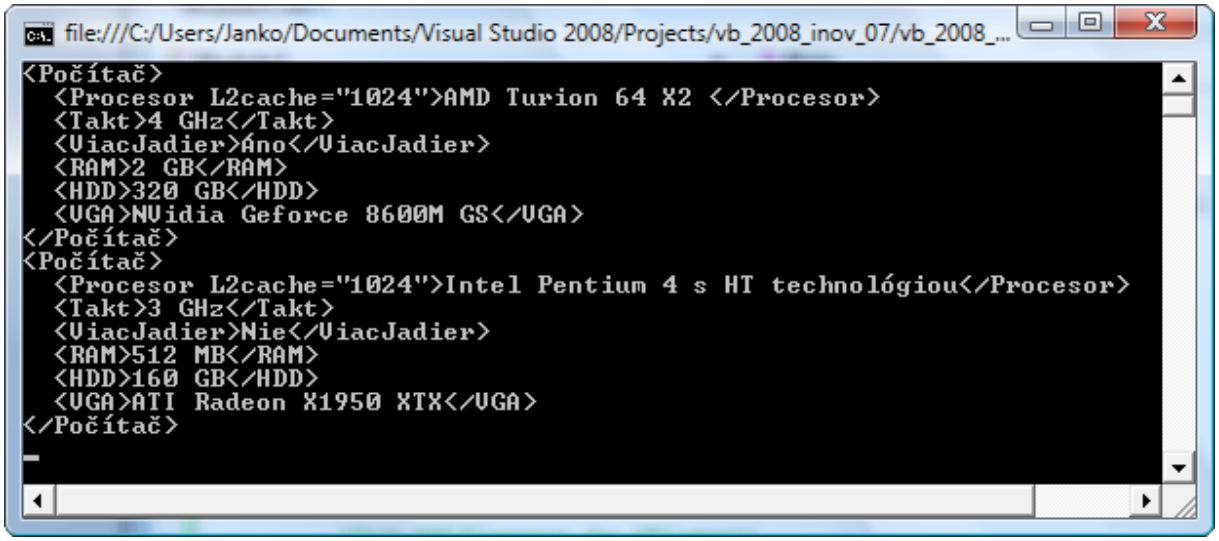
Dobre, no čo ak budeme chcieť zobraziť na výstupe nielen mená vyhovujúcej množiny procesorov, ale aj bližšie informácie o nich? Nuž, za týchto okolností zostavíme zložený dopyt, ktorý pre nás zariadi všetko potrebné:

```
'Definícia zloženého dopytovacieho výrazu.
Dim Počítače = From Počítač In XMLPočítač...<Počítač> _
                   From Procesor In Počítač...<Procesor> _
                   Where CType(Procesor.@<L2cache>, Integer) = 1024 _
                   Select Počítač

For Each Počítač In Počítače
    Console.WriteLine(Počítač)
Next
```

Spracovanie dopytu prebieha takto:

1. Najskôr získame kolekciu elementov s identifikátorom Počítač.
2. Zo získanej kolekcie vyberieme tie vnorené elementy s identifikátorom Procesor, ktorých atribúty L2cache majú hodnotu 1024.
3. Na výstupe zobrazíme informácie o počítačoch spĺňajúcich zadané kritéria.

Výstup programu uvádzame na obr. 26.



Obr. 26: Výstup programu, ktorý operuje so zloženým dopytom jazyka LINQ

Záver

Vážení vývojári, programátori a softvéroví experti,

práve ste dočítali poslednú stranu knihy **Inovácie v jazyku Visual Basic 2008**. Sme potešení, že ste s nami absolvovali exkurziu naprieč primárnymi syntakticko-sémantickými inováciami jazyka Visual Basic 2008. Nazdávame sa, že nebudete namietať, keď vyhlásime, že v novej verzii je Visual Basic vskutku robustným softvérovým nástrojom, ktorý patrí k tomu najlepšiemu, čo ľudstvo doposiaľ vynašlo.

Samozrejme, v našej knihe sme sa sústredili len na základný prehľad inovácií, zdokonalení a zmien, s ktorými Visual Basic 2008 prichádza. Ak budete mať po absolvovaní nášho základného kurzu chuť načerpať ďalšie poznatky, dovoľujeme si vám predložiť niekoľko hodnotných informačných zdrojov, o ktorých sme presvedčení, že dokážu uspokojiť vaše potreby:

1. Visual Basic Developer Center → http://msdn.microsoft.com/en-us/vbasic/default.aspx.
2. Visual Basic 2008 Language Specification → http://msdn.microsoft.com/en-us/library/ms234437.aspx.
3. The Visual Basic Team Blog → http://blogs.msdn.com/vbteam/default.aspx.
4. Visual Basic 2008 Samples → http://code.msdn.microsoft.com/vbsamples/.
5. Visual Basic Technical Articles → http://msdn.microsoft.com/en-us/vbasic/bb466163.aspx.
6. How Do I Videos → http://msdn.microsoft.com/en-us/vbasic/bb466226.aspx?wt.slv=RightTail.
7. Visual Studio 2008 and .NET Framework 3.5 Training Kit → http://www.microsoft.com/downloads/details.aspx?FamilyID=8bdaa836-0bba-4393-94db-6c3c4a0c98a1&DisplayLang=en.
8. MSDN Forums – Visual Basic → http://forums.msdn.microsoft.com/en-US/tag/visualbasic/forums/.
9. Lisa Feigenbaum on Visual Basic 2008 IDE Enhancements → http://channel9.msdn.com/posts/Dan/C9-Bytes-Lisa-Feigenbaum.
10. Webcasty v českom a slovenskom jazyku → http://www.microsoft.com/cze/msdn/webcasts/default.mspx.
11. Praktické cvičenia v slovenskom jazyku → http://www.microsoft.com/slovakia/msdn/hols/default.mspx.
12. MSDN Magazine → http://msdn.microsoft.com/en-us/magazine/default.aspx.

Prajeme vám veľa úspechov pri vytváraní tých najlepších aplikácií v jazyku Visual Basic 2008!

Autor

O autorovi

Ing. Ján Hanák, MVP vyštudoval Ekonomickú univerzitu v Bratislave. Tu, na Katedre aplikovanej informatiky Fakulty hospodárskej informatiky (KAI FHI), pracuje ako vysokoškolský pedagóg. Prednáša a vedie semináre týkajúce sa programovania a vývoja počítačového softvéru v programovacích jazykoch C, C++ a C#. Okrem spomenutej trojice jazykov patrí k jeho obľúbeným programovacím prostriedkom tiež Visual Basic a C++/CLI. Aktívne vynachádza nové postupy tvorby softvéru, ktorý bude pomáhať nielen študentom, ale aj širokej verejnosti.

Je nadšeným autorom odbornej počítačovej literatúry. V jeho portfóliu môžete nájsť nasledujúce knižné tituly:

1. **Programovanie B – Zbierka prednášok (Učebná pomôcka na programovanie v jazyku C++)**. Bratislava: Vydavateľstvo EKONÓM, 2008.
2. **Programovanie A – Zbierka prednášok (Učebná pomôcka na programovanie v jazyku C)**. Bratislava: Vydavateľstvo EKONÓM, 2008.
3. **Expanzívne šablóny: Príručka pre tvorbu "code snippets" pre Visual Studio**. Bratislava: Microsoft Slovakia, 2008.
4. **Kryptografia: Príručka pre praktické odskúšanie symetrického šifrovania v .NET Framework-u**. Bratislava: Microsoft Slovakia, 2007.
5. **Príručka pre praktické odskúšanie vývoja nad Windows Mobile 6.0**. Bratislava: Microsoft Slovakia, 2007.
6. **Príručka pre praktické odskúšanie vývoja nad DirectX**. Bratislava: Microsoft Slovakia, 2007.
7. **Príručka pre praktické odskúšanie automatizácie aplikácií Microsoft Office 2007**. Bratislava: Microsoft Slovakia, 2007.
8. **Visual Basic 2005 pro pokročilé**. Brno: Zoner Press, 2007.
9. **C# - praktické příklady**. Praha: Grada Publishing, 2006 (kniha získala ocenenie „Najúspešnejšia novinka vydavateľstva Grada v oblasti programovania za rok 2006“).
10. **Programujeme v jazycích C++ s Managed Extensions a C++/CLI**. Praha: Microsoft, 2006.
11. **Přecházíme z jazyka Visual Basic 6.0 na jazyk Visual Basic 2005**. Praha: Microsoft, 2005.
12. **Visual Basic .NET 2003 – Začíname programovat**. Praha: Grada Publishing, 2004.

V rokoch 2006, 2007 a 2008 bol jeho prínos vývojárskym komunitám ocenený celosvetovými vývojárskymi titulmi **Microsoft Most Valuable Professional (MVP)** s kompetenciou **Visual Developer – Visual C++**.

Kontakt s vývojármi a programátormi udržiava najmä prostredníctvom technických seminárov a odborných konferencií, na ktorých vystupuje. Za všetky vyberáme tieto:

- Konferencia **Software Developer 2007**, príspevok na tému „Představení produktu Visual C++ 2005 a jazyka C++/CLI“. Praha 19. 6. 2007.
- Technický seminár **Novinky vo Visual C++ 2005**. Microsoft Slovakia. Bratislava 3. 10. 2006.
- Technický seminár **Visual Basic 2005 a jeho cesta k Windows Vista**. Microsoft Slovakia. Bratislava 27. 4. 2006.

V súčasnosti pôsobí ako stály spolupracovník počítačových časopisov PC World, ComputerWorld a Connect!. V minulosti zastával pozíciu odborného redaktora rovnako v magazínoch Software Developer, Infoware, PC Revue a Chip.

Dovedna publikoval viac ako 250 odborných a populárnych prác venovaných vývoju počítačového softvéru.

Ak sa chcete s autorom spojiť, môžete využiť jeho adresu elektronickej pošty: hanja@stonline.sk.

Akademický blog autora môžete sledovať na adrese: http://blog.aspnet.sk/hanja/.

www.microsoft.sk