POGLAVLJE 2

Uvod u okruženje .NET

U OVOM POGLAVLJU

- Prednosti zajedničkog izvršnog okruženja 59
- Deklarisanje promenljivih upotrebom zajedničkih tipova 61
- Prelazak na ASP.NET 63
- Upotreba Windowsovih obrazaca 67
- Metapodaci 69
- Organizovanje biblioteka objekata pomoću imenskih prostora 71
- Prednosti IL koda 74
- Pakovanje .NET aplikacije u sklop 76
- Moć upravljane memorije i skupljanja smeća 77
- Verzije .NET objekata 80
- Standardizovana obrada grešaka 81

Ako ste slični većini programera koji prelaze na Visual Basic .NET, onda se ne suočavate samo sa zadatkom da naučite nov programski jezik nego i pokušavate da razumete prednosti okruženja .NET i kako da iskoristite te prednosti u programima koje pravite. Okruženje .NET pruža mnoge nove mogućnosti:

- Biblioteku izvršnih komponanata nezavisnu od programskog jezika koju programeri zovu zajedničko izvršno okruženje (engl. *Common Language Runtime*, *CLR*) i koje mogu koristiti programi napisani u programskim jezicima C#, Visual Basic .NET i Jscript.
- Zajednički sistem tipova podataka, što znači da svi .NET programski jezici koriste isti način predstavljanja zajedničkih tipova podataka.
- Model ASP.NET za Web stranice u kome se koristi prevedeni programski jezik (za razliku od VBScripta) i koji podržava izvršavanje skriptova na serverskoj strani.
- Web obrasci koji omogućavaju programerima da razviju interfejs za Web stranice tehnikom prevlačenja kontrola u obrazac.
- ADO.NET, koji proširuje mogućnosti programiranja baza podataka.
- Upotreba metapodataka (podataka koji opisuju podatke) koji omogućavaju programima da šalju upite objektima o njihovim mogućnostima.
- Upotreba imenskih prostora radi organizovanja klasa i umanjivanja mogućnosti nastajanja dupliranih imena.
- Prevodioce čiji rezultat je međujezik koji poseban softver, nazvan pravovremeni prevodilac (engl. *just in time*, *JIT*), prevodi u mašinski kôd u trenutku kada se aplikacija pokrene.
- Upotreba posebnih datoteka nazvanih sklopovi u koje se smeštaju aplikacije i biblioteke klasa.
- Pravljenje Web servisa, posebnog mrežnog softvera kojeg je moguće pozivati preko Interneta iz udaljene aplikacije ili ASP.NET stranice.
- Upotreba informacija o broju verzije u datoteci sklopa aplikacije smanjuje probleme koji nastaju usled postojanja više verzija istog softvera (kao što su sukobi između datoteka tipa .DLL). Pošto postoji podatak o verziji koji sadrže i programi i komponente, različite verzije istih komponenata mogu postojati i koristiti se jedne uporedo sa drugima.
- Podršku za otkrivanje i obradu izuzetaka pomoću strukturirane obrade grešaka.

Čitanjem ove knjige detaljno ćete se upoznati sa mogućnostima .NET okruženja. Da biste počeli, u ovom poglavlju upoznaćete se sa ključnim osobinama okruženja .NET. Kada završite čitanje ovog poglavlja imaćete solidnu osnovu na kojoj možete nastaviti da stičete znanje o okruženju .NET.

Prednosti zajedničkog izvršnog okruženja

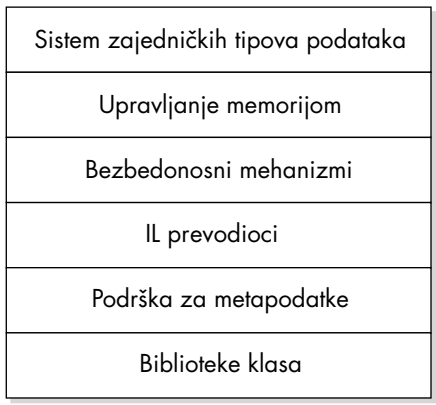
Godinama su programeri veoma često koristili funkcije iz raznih biblioteka izvršnih komponenata da bi izvodili operacije u svojim programima kao što su rad s datotekama i direktorijumima, podešavanje ili učitavanje sistemskog datuma i vremena, izvođenje

aritmetičkih operacija kao što su računanje sinusa nekog ugla ili računanje kvadratnog korena datog broja.

Biblioteke izvršnih komponenata sadrže stotine i hiljade funkcija i procedura koje programeri mogu koristiti da bi obavljali uobičajene operacije. U prošlosti, većina biblioteka su bile predviđene za određeni programski jezik, što znači da biblioteka koju su mogli koristiti programeri u Visual Basicu mogla je biti, a često je i bila, drugačija od one koju su koristili C++ programeri.

Okruženje .NET je sagrađeno na osnovama dve nove i moćne biblioteke izvršnih komponenata, nezavisne od programskih jezika i koje se zovu zajedničko izvršno okruženje (engl. *Common Language Runtime*, *CLR*) i biblioteka osnovnih klasa (engl. *base-class library*, *BCL* u kojoj se nalaze hiljade definicija klasa). Pošto su ove biblioteke nezavisne od programskog jezika, program napisan u Visual Basicu .NET koristi isti skup procedura iz biblioteke kao i C# program. Iako programeri često misle o bibliotekama u smislu funkcija koje one nude, CLR biblioteka definiše i zajedničke tipove podataka u okruženju .NET, čime se obezbeđuje jedinstven skup tipova podataka, nezavisnih od programskog jezika, za standardne tipove podataka kao što celi brojevi i brojevi u pokretnom zarezu.

U odeljku „Prednosti IL koda“ naučićete da okruženje .NET prevodi aplikacije u međujezik (engl. *Intermediate Language*, *IL*) koji će ciljni sistem kasnije prevesti (pomoću pravovremenog prevodioca) u lokalni mašinski kod koji odgovara skupu instrukcija ciljnog sistema. Sam CLR je u formatu međujezika. Na slici 2-1 prikazane su najvažnije komponente CLR-a.



Slika 2-1 Najvažnije komponente zajedničkog izvršnog okruženja.

PRIMER Zajedničko izvršno okruženje (CLR) ne zavisi od programskog jezika koji koristite. Sledeći Visual Basic .NET i C# programi ilustruju upotrebu metode GetFiles klase Directory da bi prikazali imena datoteka koja se nalaze u korenskom direktorijumu na disku C. Pored toga, kôd koristi metodu WriteLine klase Console da bi prikazao imena datoteka na ekranu i metodu ReadLine da bi sačekao da korisnik pritisne taster <ENTER> i tako završio rad. Naredbe koje slede čine program ShowRoot.vb, napisan u Visual Basicu .NET.

```
Imports System.IO

Module Module1

    Sub Main()
        Dim Filename As String

        For Each Filename in Directory.GetFiles("C:\")
            Console.WriteLine(Filename)
        Next

        Console.ReadLine()  'Zaustavljamo program da bismo
                            'videli rezultat
    End Sub
End Module
```

Slično tome, naredbe koje slede čine program CS_ShowRoot.cs, napisan u jeziku C#:

```
using System;
using System.IO;
namespace CS_ShowRoot
{
  class Class1
    {
      static void Main(string[] args)
        {
          foreach (string Filename in Directory.GetFiles("C:\\"))
            Console.WriteLine (Filename)

          Console.ReadLine()  'Zaustavljamo program da bismo videli
                               rezultat
        }
    }
}
```

Kao što vidite, zahvaljujući upotrebi zajedničkog izvršnog okruženja, programi napisani u Visual Basicu .NET i C# veoma su slični po obliku i istovetni po funkcionalnosti.

Deklarisanje promenljivih upotrebom zajedničkih tipova

Tip promenljive određuje opseg vrednosti koje promenljiva može sadržati, kao i skup operacija koji program može činiti sa promenljivom. Na primer, promenljiva tipa Integer može sadržati vrednosti u opsegu -2.147,483,648 do 2,147,483,647. Osim toga, promenljive tipa Integer program može sabirati, oduzimati, množiti i deliti. Nasuprot tome, promenljiva tipa String u Visual Basicu .NET može da sadrži vrednosti znakovnog tipa (u dvobajtnom Unicode formatu). Program može upoređivati i spojiti dve promenljive tipa String, ali ne može množiti i deliti sadržaj promenljive tipa String.

U prošlosti, svaki programski jezik je definisao svoje tipove podataka. Na nesreću, način na koji su u jednom jeziku definisani tipovi podataka nisu uvek odgovarali načinu u nekom drugom jeziku. Na primer, u programskom jeziku C++, za definisanje tipa Integer koriste se 32 bita (što znači da se u promenljivoj tipa Integer može čuvati vrednost u

opsegu od -2,147,483,648 do 2,147,483,647). Visual Basic definiše tip Integer upotrebom 16 bita (što znači da se u promenljivoj tipa Integer mogla čuvati vrednost u opsegu od -32768 do 32767).

U okruženju .NET, CLR definiše skup tipova koji programeri zovu zajedničkim tipovima. Programski jezici kao što su C# i Visual Basic .NET definišu svoje tipove na osnovu CLR-ovih tipova podataka. U tabeli 2-1 ukratko su opisani svi zajednički (CLR-ovi) tipovi.

Tip podataka	Predstavlja
Char	16-bitni Unicode znakovi
DateTime	Vrednosti tipa datum i/ili vreme
Decimal (decimalna)	Vrednosti u opsegu od 0 do +/- 79,228,162,514,264,337,593,543,950.335 bez decimalnog zareza ili od 0 do +/- 7.92281625142643375935439503 35 sa 28 mesta iza decimalnog zareza.
Double	Vrednosti u pokretnom zarezu, prikazane sa 64 bita.
GUID	Jedinstveni globalni identifikator koji se sastoji od 128 bita.
Int16	Vrednosti u opsegu od -32768 do 32767
Int32	Vrednosti u opsegu -2,147,483,648 do 2,147,483,647
Int64	Vrednosti u opsegu od -9,223,372,036,036,854,775,808 do 9,223,372,036,036,854,775,807
Single	Vrednosti u pokretnom zarezu, predstavljene sa 32 bita.
TimeSpan	Pozitivni ili negativni vremenski interval

Tabela 2-1 Standardni zajednički tipovi podataka u okruženju .NET

Zahvaljujući zajedničkim tipovima, CLR je postao nezavisan od programskog jezika. Na primer, ako određena funkcija CLR-a zahteva parametar tipa Integer, programi napisani i u Visual Basicu .NET i u C# proslediće isti broj bitova za vrednost tog parametra.

PRIMER Kada napišete program u programskom jeziku Visual Basic .NET ili C#, prevodilac će u pozadini povezati imena tipova iz programskog jezika, kao što je Integer u Visual Basicu .NET ili int u jeziku C#, sa odgovarajućim CLR-ovim tipom. Naredbe koje slede čine program VBShowCommonTypes.vb, napisan u Visual Basicu .NET.

```
Module Module1
    Sub Main()
        Dim A As Short
        Dim B As Integer
        Dim C As Int64
        Dim D As Single
        Dim E As Double
        Dim F As Char

        Console.WriteLine("A: {0}", A.GetType().FullName)
        Console.WriteLine("B: {0}", B.GetType().FullName)
        Console.WriteLine("C: {0}", C.GetType().FullName)
```

```
        Console.WriteLine("D: {0}", D.GetType().FullName)
        Console.WriteLine("E: {0}", E.GetType().FullName)
        Console.WriteLine("F: {0}", F.GetType().FullName)
        Console.ReadLine()
    End Sub
End Module
```

Nakon što prevedete i pokrenete navedeni program, na ekranu računara pojaviće se ovi rezultati:

```
A: System.Int16
B: System.Int32
C: System.Int64
D: System.Single
E: System.Double
F: System.Char
```

Slično tome, naredbe koje slede čine program ShowCommonTypes.cs, napisan u jeziku C#:

```
using System;

namespace ShowCommonTypes
{
   class Class1
     {
       static void Main(string[] args)
        {
          short A = 0;
          int B = 0;
          long C = 0;
          float D = 0;
          double E = 0;
          char F = 'A';

         Console.WriteLine("A: {0}", A.GetType().FullName)
         Console.WriteLine("B: {0}", B.GetType().FullName)
         Console.WriteLine("C: {0}", C.GetType().FullName)
         Console.WriteLine("D: {0}", D.GetType().FullName)
         Console.WriteLine("E: {0}", E.GetType().FullName)
         Console.WriteLine("F: {0}", F.GetType().FullName)

         Console.ReadLine()
       }
   }
}
```

Prelazak na ASP.NET

Na Web lokacijama je raširena upotreba ASP tehnologije za prikazivanje dinamičkog sadržaja (koji je promenljiv ili se prilagođava korisniku). Godinama Web programeri koriste jezik VBScript da bi primenili tehnologiju Active Server Pages. Iako postoje desetine miliona Web stranica zasnovanih na tehnologiji, Active Server Pages i VBScriptu, ta tehnologija ipak ima nedostataka, kao što su ograničenja programskog jezika VBScript

čiji se kôd interperetira a ne prevodi direktno u mašinski kôd; zatim, lošiji interfejs od uobičajenih Windows aplikacija, kao i ograničenja u vezi sa održavanjem informacija o stanju korisnika.

Okruženje .NET donosi nov model za pravljenje dinamičkih Web stranica koji se zove ASP.NET. Prva velika promena koju će Web programeri uočiti je da više nema VBScripta! Da bi napravili Web stranicu pomoću ASP.NET-a, Web programeri će koristiti programske jezike Visual Basic .NET, C# ili JScript .NET. Za razliku od ASP stranica zasnovanih na VBScriptu, koje server prvo interpretira a zatim izvršava, ASP.NET stranice su prevedeni programi tako da su bezbednost i performanse stranice poboljšane.

Usled velikog broja postojećih ASP stranica napravljenih pomoću VBScripta, vrlo je verovatno da će u narednim godinama ASP.NET stranice i ASP stranice postojati jedne pored drugih na istim serverima. U poglavlju 13 detaljno je opisan ASP.NET. Tu ćete saznati da stranica napravljena u ASP.NET-u može pozivati ASP stranicu i obrnuto.

Web programeri smeštaju ASP stranice u datoteke s nastavkom .asp. Datoteke ASP.NET stranica imaju nastavak .aspx. Koriste se različiti nastavci imena datoteka da bi server mogao da razlikuje ASP.NET stranicu od ASP stranice. Da bi server podržao ASP.NET stranice, na serveru mora raditi program Microsoft Internet Information Services (IIS) verzija 5 ili novija.

Godinama su Web programeri koristili skript jezike kao što su VBScript i JavaScript da bi obradili događaje na klijentskoj strani (događaje u čitaču Weba). Okruženje ASP.NET uvodi serversku podršku za događaje. Drugim rečima, u okruženju ASP.NET mnoge operacije se izvršavaju na serveru. Osim toga, u okruženju .NET postoje nove kontrole koji omogućavaju programerima da na Web stranici ponude korisnički interfejs veoma sličan Windowsovom obrascu, kao što je prikazano na slici 2-2. U opštem slučaju, Web obrazac treba shvatiti kao serversku kontrolu koja omogućava serveru da reaguje na događaje koje generiše ASP.NET stranica. U poglavlju 15 detaljno su opisani Web obrasci.

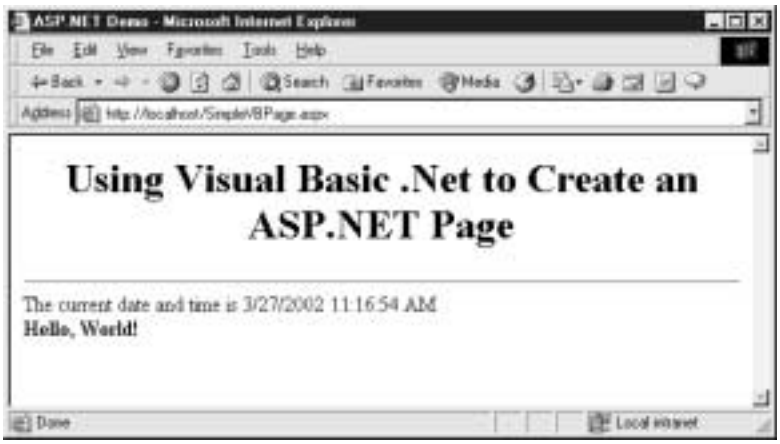
PRIMER Sledeća ASP.NET stranica, SimpleVBPage.aspx, koristi Visual Basic .NET kôd da bi prikazala tekući datum, vreme i određenu poruku.

```
<% @ Page Language="VB" %>

<script runat="server">
Sub GreetUser()
  Dim Seconds As Integer

  Seconds = Now.Second

   If ((Seconds Mod 3) = 0) Then
     Response.Write("Hello, World!<br/>")
   ElseIf (Seconds Mod 3) = 1 Then
     Response.Write("Hello, Visual Basic .NET World!<br/>")
   ElseIf (Seconds Mod 3) = 2 Then
     Response.Write("Hello, ASP.NET World!<br/>")
   End If
End Sub
</script>
<html>
```

```
<head>
  <title>ASP.NET Demo </title>
</head>
<body>
  <center><h1>Using Visual Basic .NET to Create ASP.NET Page</h1>
  </center><hr/>
The current date and time is
  <% =Now %>

  <b><br/>
  <%
      GreetUser()
  %>
  </b>
</body>
</html>
```

Slika 2-2 Okruženje .NET uvodi Web obrasce, pomoću kojih programeri mogu praviti Web stranice koje prikazuju obrasce.

Kada korisnik pregleda ovu ASP.NET stranicu, njegov čitač Weba prikazaće rezultat nalik ovom na slici 2-3.



Slika 2-3 Prikaz rezultata ASP.NET stranice SimpleVBPage.aspx.

Slično tome, naredna stranica, SimpleCsPage.aspx, realizuje istu stranicu ali u jeziku C#:

```
<% @ Page Language ="C#" %>

<script runat="server">
voidGreetUser()
 {
   int Seconds;
   Seconds = DateTime.Now.Second;

   if ((Seconds % 3) = = 0)
     Response.Write("Hello, World!<br/>");
   elseIf ((Seconds % 3) = = 1)
     Response.Write("Hello, C# World!<br/>");
   elseIf ((Seconds % 3) = = 2)
     Response.Write("Hello, ASP.NET World!<br/>");
 }
</script>
<html>
<head>
   <title>ASP.NET demo</title>
</head>
<body>
  <center><h1>Using C# to Create ASP.NET Page</h1></center><hr/>
The current date and time is
  <% =DateTime.Now %>

  <b><br/>
  <%
```

```
        GreetUser()
    %>
    </b>
</body>
</html>
```

Upotreba Windowsovih obrazaca

Godinama su Visual Basic programeri koristili obrasce da bi brzo napravili ujednačen korisnički interfejs. Mnogi programeri će se složiti da je mogućnost prevlačenja kontrola u obrazac prilikom pravljenja korisničkog interfejsa bila ključni razlog velike popularnosti i uspeha Visual Basica. Programeri u jezicima C i C++ pravili su korisnički interfejs koristeći biblioteke Microsoft Foundation Classes (MFC) i Win 32 API. Pravljenje korisničkog interfejsa u nekom drugom programskom jeziku sem u Visual Basicu oduzimalo je najviše vremena tokom razvoja aplikacije.

Okruženje .NET donosi sa sobom i okruženje za razvoj korisničkog interefejsa zasnovanog na Windowsovim obrascima i kontrole koje programeri koji koriste programske jezike Visual Basic .NET, C# ili JScript .NET mogu upotrebiti da bi tehnikom prevlačenja napravili korisnički interfejs zasnovan na obrascima. Na slici 2-4 prikazan je postupak pravljenja obrasca u okruženju Visual Studija. Kao što biste i očekivali, mogu se prevlačiti kontrole u obrazac. U pozadini, Visual Studio generiše kôd pomoću kojeg program komunicira sa kontrolama.

Slika 2-4 Windowsovi obrasci šire mogućnost razvoja korisničkog interfejsa upotrebom tehnike prevlačenja i na programske jezike C# i JScript i, naravno, na Visual Basic .NET.

PRIMER Sledeći izvorni kôd (ako ne računamo komentare) jeste kôd koji je Visual Studio generisao za obrazac prikazan na slici 2-4.

```
Public Class Form1
    Inherits System.Windows.Forms.Form

#Region "Windows Form Designer generated code"

    Public Sub New()
        MyBase.New()

        InitializeComponent()
    End Sub

    Protected Overloads Overrides Sub Dispose (ByVal disposing _
    As Boolean)
        If disposing Then
            If Not (components Is Nothing) Then
                Components.Dispose()
            End If
        End If
        MyBase.Dispose(disposing)
    End Sub

    'Required by the Windows Form Designer
    Private components As System.ComponentModel.Icontainer

    Friend WithEvents Label1 As System.Windows.Forms.Label
    Friend WithEvents TextBox1 As System.Windows.Forms.TextBox
    Friend WithEvents CheckBox1 As System.Windows.Forms.CheckBox
    Friend WithEvents Button1 As System.Windows.Forms.Button
    <System.Diagnostics.DebuggerStepThrough() > Private Sub
InitializeComponent()
        Me.Label1 = New System.Windows.Forms.Label()
        Me.TextBox1 = New System.Windows.Forms.TextBox()
        Me. CheckBox1 = New System.Windows.Forms.CheckBox()
        Me. Button1 = New System.Windows.Forms.Button()
        Me.SuspendLayout()

       'Label1
        Me.Label1.Location = New System.Drawing.Point(24, 56)
        Me.Label1.Name = "Label1"
        Me.Label1.TabIndex = 0
        Me.Label1.Text = "Book Name"

        'TextBox1
        Me.TextBox1.Location = New System.Drawing.Point(144, 56)
        Me.TextBox1.Name = "TextBox1"
        Me.Label1.TabIndex = 0
        Me.TextBox11.Text = ""

       'CheckBox1
        Me.CheckBox1.Location = New System.Drawing.Point(24, 96)
```

```
        Me.CheckBox1.Name = "CheckBox1"
        Me.Label1.TabIndex = 2
        Me.TextBox11.Text = "Rush Order"

       'Button1
        Me.Button1.Location = New System.Drawing.Point(96, 144)
        Me.Button1.Name = "Button1"
        Me.Button1.TabIndex = 3
        Me.Button1.Text = "Seacrh"

       'Form1
        Me.AutoScaleBaseSize = New System.Drawing.Size(5, 13)
        Me.ClientSize = New System.Drawing.Size(280, 221)
        Me.Controls.AddRange (New System.Windows.Forms.Control() _
          {Me.Button1, Me.CheckBox1. Me.TextBox1, Me.Label1})
        Me.Name = "Form1"
        Me.Text = "Get Book Infomation"
        MeResumeLayout(False)
    End Sub

#End Region
End Class
```

Integracijom Windowsovih obrazaca u platformu nezavisnu od programskog jezika, okruženje .NET omogućava programerima da, bez obzira na programski jezik koji koriste naprave korisnički interfejs programa pomoću tehnike prevlačenja.

Metapodaci

Reč metapodaci programeri koriste da bi opisali podatke o podacima. U okruženju .NET rasprostranjena je upotreba metapodataka kojim se opisuju objekti, sklopovi, ADO.NET skupovi podataka i još puno toga. Programeri tvrde da je jedno od glavnih poboljšanja u okruženju .NET to što objekti sami sebe opisuju; drugim rečima, program može zahtevati od objekta da opiše svoje metode, polja i svojstva koja program može koristiti.

Okruženje .NET smešta metapodatke u sam objekat. Svaki objekat sadrži podatke koji ga opisuju. Kada prevodite .NET, aplikaciju, prevodilac generiše metapodatke i uključuje ih u aplikaciju. Za svaku .NET aplikaciju postoji posebna datoteka, koja se zove datoteka sklopa, u kojoj su u obliku metapodataka zadate informacije kao što su klase koje aplikacija stavlja na raspolaganje, kao i informacije o drugim komponentama koje su neophodne aplikaciji (kao što su DLL datoteke). Kada programeri razgovaraju o metapodacima, onda se vrlo brzo pomene proširivi jezik za označavanje (engl. *Extensible Markup Language*, *XML*), zato što jezik XML pruža odlične mogućnosti za strukturiranje metapodataka.

PRIMER Sledeći program, ClassInfo.vb, definiše klasu Book koja ima metode i svojstva. Kôd zatim koristi refleksijski API okruženja .NET da bi ispitao klasu o mogućnostima koje ona pruža (drugim rečima, program „pita“ klasu, koja je samoopisujuća, da saopšti podatke o sebi):

```
Module Module1
    Class Book
        Public Title As String
        Public Author As String
        Public Price As Double

        Public Sub New (ByVal Title As String, ByVal Author As String, _
          ByVal Price As Double)
            Me.Title = Title
            Me.Author = Author
            Me.Price = Price
        End Sub

        Public Sub ShowTitle()
            Console.WriteLine("Title: " & Title)
        End Sub

        Public Sub ShowBook()
            Console.WriteLine("Title: " & Title)
            Console.WriteLine("Author: " & Author)
            Console.WriteLine("Price: " & Price)
        End Sub
    End Class

    Sub Main()
        Dim NetBook As _
          New Book("Visual Basic .NET kroz praktične primere", _
          "Jamsa", 49.99)

        Console.WriteLine("Imena metoda")
        Dim Info As Reflection.MethodInfo
        For Each Info In NetBook.GetType.GetMethods()
            Console.WriteLine(Info.Name)
        Next

        Console.WriteLine()
        Console.WriteLine("Imena članova")
        Dim Info As Reflection.MemberInfo

        For Each Member In NetBook.GetType.GetMembers()
            Console.WriteLine(Member.Name)
        Next

        Console.ReadLine()
    End Sub
End Module
```

Nakon što prevedete i pokrenete ovaj program, na ekranu računara pojaviće se ovi rezultati:

```
Imena metoda
GetHashCode
Equals
ToString
```

```
ShowTitle
ShowBook
GetType

Imena članova
Title
Author
Price
GetHashCode
Equals
ToString
ShowTitle
ShowBook
GetType
.ctor
```

Pomoću API-ja Reflection programi mogu ispitivati svaki objekat .NET klase o mogućnostima koje on pruža. U poglavlju 18 detaljno se obrađuje odraz (engl. *reflection*)

Organizovanje biblioteka objekata pomoću imenskih prostora

U objektno orijentisanim programskim okruženjima, kao što je okruženje .NET, programeri veoma mnogo koriste klase da bi opisali objekte. Da bi bolje organizovali klase i pojednostavili upotrebu klasa, programeri često grupišu klase u biblioteke klasa. Kao što je i logično, sve klase u datoj bibiloteci će obavljati slične zadatke. Možete, na primer, imati biblioteku klasa Matematika u kojoj se nalaze funkcije koje izvode aritmetičke operacije. Ili možete imati biblioteku klasa Komunikacije koju vaši programi koriste za komunikaciju sa udaljenim serverom.

Kako raste broj klasa koje dodajete biblioteci, tako raste i verovatnoća dupliranja imena (tj. da dve klase imaju isto ime). Da bi smanjili mogućnost takvog problema, programeri koriste imenske prostore da bi organizovali objekte. Imenski prostor grupiše objekte (koji se mogu nalaziti u više datoteka sklopa) i dodeljuje imena objektima koja su takva da eliminišu dupliranje imena u dve ili više klasa koje ne dele isti imenski prostor. Pretpostavimo da koristite klasu Address da biste čuvali IP adresu (kao što je 111.212.211.112) i ime domena (kao što je www.mikroknjiga.co.yu).

```
Class Address
  Public DottedName As String
  Public DomainName As String
End Class
```

Slično tome, pretpostavimo da u programu koristite i sledeću klasu Address da čuvate kućne adrese zaposlenih:

```
Class Address
  Public Street As String
  Public City As String
  Public State As String
  Public Zip As String
End Class
```

Da bi programer eliminisao sukob koji nastaje usled dupliranja imena između dve klase Address, on ih može smestiti u dva različita imenska prostora. U kodu se, na primer, može napraviti imenski prostor Komunikacije u koji se smešta klasa Address koja sadrži podatke o IP adresi. Slično tome, u programu se može napraviti imenski prostor Poruke u koji se smešta klasa Address sa informacijama o ulici u kojoj živi zaposleni. Pošto napravite imenski prostor, ispred imena klase napišite ime imenskog prostora, a zatim operator tačka; na primer, Poruke. Address ili Komunikacije. Address. Navođenjem imenskog prostora kao prefiksa formirali ste jedinstveno ime.

U okruženju .NET programi će veoma često koristiti zajedničko izvršno okruženje (CLR) i biblioteku osnovnih klasa (BCL). Da bi se bolje organizovale hiljade procedura u bibliotekama, u okruženju .NET se često koriste imenski prostori. U tabeli 2-2 ukratko su opisani najvažniji imenski prostori okruženja .NET.

Imenski prostor	Svrha
System.CodeDOM	Klase koje predstavljaju dokument sa izvornim kodom.
System.Collection	Klase koje opisuju strukture podataka tipa kolekcija, kao što su nizovi, povezane liste i redovi.
System.Configuration	Klase koje programi mogu koristiti da bi pristupili konfiguracionim podacima okruženja .NET-ovog Frameworka.
System.Data	Klase koje podržavaju učitavanje podataka iz izvora kao što je, na primer, ADO.NET
System.Diagnostics	Klase koje možete koristiti da biste otkrivali greške u .NET aplikacijama upotrebom sistemskih brojača, dnevnika događaja i brojača performansi.
System.DirectoryServices	Klase koje obezbeđuju programima interfejs ka aktivnom imeniku (engl. *Active Directory*).
System.Drawing	Klase koje se mogu koristiti u programima za rad sa grafičkim objektima (GDI+).
System.Globalization	Klase koje podržavaju internacionalne mogućnosti, kao što su lokalni formati za vreme, datum i novčanu jedinicu.
System.IO	Klase koje podržavaju U/I operacije.
System.Management	Klase koje obezbeđuju interfejs ka WMI servisima.
System.Messaging	Klase koje se mogu koristiti za rad sa redovima za poruke.
System.NET	Klase koje se mogu koristiti za izvršavanje mrežnih operacija.
System.Reflection	Klase koje se mogu koristiti za ispitivanje objekta o mogućnostima koje on pruža.
System.Runtime	Klase koje programi mogu pozivati radi upotreba datoteka sklopova, objekata tipa COM i udaljenih objekata kao što su objekti InteropServices i Remoting.
System.Security	Klase koje programi mogu pozivati radi šifrovanja, kontrole pristupa itd.

Tabela 2-2 Najvažniji imenski prostori u .NET okruženju

Imenski prostor	Svrha
System.Text	Klase koje programski mogu pozivati za rad sa tekstom u formatima ASCII, Unicode UTF-7 ili UTF-8.
System.Web	Klase koje programi mogu pozivati za rad sa lokalnim čitačem Weba.
System.Windows.Forms	Klase koje programi mogu pozvati radi pravljenja Windowsovih obrazaca.
System.XML	Klase koje podržavaju operacije sa XML podacima, kao što je čitanje i pisanje sadržaja u XML formatu.

Tabela 2-2 Najvažniji imenski prostori u .NET okruženju (nastavak)

PRIMER Kada u svojim programima koristite klase koje su definisane u CLR-u, dešavaće se da morate zadati imenski prostor u kom su smeštene klase (prevodilac će automatski pretražiti više imenskih prostora koji se obično koriste u okruženju .NET.) Da biste naredili prevodiocu da uključi klase koje se nalaze u određenom imenskom prostoru, morate uvesti imenski prostor u programski kôd pomoću naredbe Imports. Na primer, sledeća naredba nalaže prevodiocu koda da uključi imenski prostor System.IO koji sadrži klase koje će vašem programu omogućiti da radi sa datotekama i direktorijumima:

```
Imports System.IO
```

Da biste definisali svoj imenski prostor u programu ili biblioteci klasa koje pišete, upišite kôd između naredaba Namespace i End Namespace.

```
Namespace ImeImenskogProstora
  'Upišite klase ovde
End Namespace
```

U sledećem programu, NamespaceDemo.vb, biće napravljena dva imenska prostora, jedan koji se zove Network i drugi Mailing. U svakom imenskom prostoru biće definisana u kodu klasa Address. Zatim će programski biti napravljen po jedan objekat od svake klase. Kada se ispred imena klase napiše ime imenskog prostora, a zatim operator tačka, kôd izričito opisuje prevodiocu klasu koja mu je neophodna.

```
Namespace Network
    Class Address
        Public DottedName As String
        Public DomainName As String

        Public Sub New(ByVal IPAddr As String, ByVal Domain As String)
            DottedName = IPAddr
            DomainName = Domain
        End Sub

        Public Sub ShowAddress()
            Console.WriteLine("IP: " & DottedName)
            Console.WriteLine("Domain: " & DomainName)
        End Sub
    End Class
End Namespace
```

```
Namespace Mailing
    Class Address
        Public Street As String
        Public City As String
        Public State As String
        Public Zip As String

        Public Sub New(ByVal Street As String, ByVal City As String, _
          ByVal State As String, ByVal Zip As String)
            Me.Street = Street
            Me.City = City
            Me.State = State
            Me.Zip = Zip
        End Sub

        Public Sub ShowAddress()
            Console.WriteLine("Street: " & Street)
            Console.WriteLine("City: " & City)
            Console.WriteLine("State: " & State)
            Console.WriteLine("Zip: " & Zip)
        End Sub
    End Class
End Namespace

Module Module1
Sub Main()
        Dim PC_Address As New Network.Address("122.111.222.112", _
          "www.SomeSite.com")

        Dim Employee_Address As New Mailing.Address("122 Main", _
          "Houston", "Texas", "77469")

        PC_Address.ShowAddress()
        Console.WriteLine()
        Employee_Address.ShowAddress()
        Console.ReadLine()
    End Sub
End Module
```

Prednosti IL koda

Kada prevodite program, prevodilac prevodi izvorni kôd programskog jezika visokog nivoa, kao što su Visual Basic i C++, u mašinski kôd (jedinice i nule) koji se sastoji od skupa instrukcija određenog procesora. Ovakav kôd koji je specifičan za dati tip procesora programeri zovu mašinski kôd. Pošto mašinski kôd za neki Intelov procesor koristi skup instrukcija koji je karakterističan za Intelove procesore, takav mašinski kôd neće raditi na Macintosh računarima koji koriste Motoroline procesore.

Da bi se podržao veliki broj tipova procesora i operativnih sistema koji su priključeni na Web, programski jezik Java je uveo koncept virtuelnog mašinskog jezika. Java apleti ne sadrže instrukcije koje bi bile specifične za određeni tip procesora. Umesto toga, apleti

sadrže skup generičkih instrukcija nezavisnih od hardverske platforme. Kada čitač Weba preuzme Java aplet, poseban softver u čitaču (koji programeri zovu Java virtuelna mašina) pretvara kôd virtuelne mašine u mašinski kôd koji odgovara procesoru i operativnom sistemu koji se koristi na lokalnom računaru. Upotrebom formata virtuelnog mašinskog jezika, programeri mogu napraviti jedan izvršni program koji će raditi na svim tipovima mašina. Nedostatak modela jezika virtuelne mašine jeste to što da bi računar mogao da izvrši program, on mora prvo pretvoriti kôd virtuelne mašine u pravi mašinski kôd što dovodi do dodatnog opterećenja i kašnjenja pre nego što program počne da se izvršava.

Slično tome, aplikacije napisane u .NET okruženju ne prevode se u instrukcije lokalnog mašinskog jezika. Umesto toga, .NET prevodioci generišu kôd u međujeziku (engl. *Intermediate language*, *IL*). Kasnije, kada korisnik pokrene .NET program, poseban softver (koji se zove pravovremeni prevodilac) pretvara IL kôd u instrukcije lokalnog mašinskog jezika.

PRIMER Da biste pogledali IL kôd, pokrenite program koji se zove IL Disassembler, kao što je prikazano na slici 2-5. Da biste pokrenuli IL Disassembler, morate pokrenuti ILDASM.EXE sa komandne linije navodeći ime i direktorijum programa koji želite da ispitate.

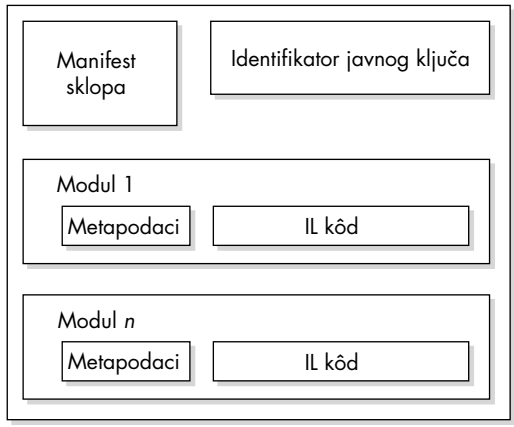
```
C:\>ildasm \PutanjaPrograma\ImePrograma.exe <ENTER>
```

Slika 2-5 Upotreba programa IL Disassembler za pregledanje IL koda.

Da biste pokrenuli IL Disassembler sa komandne linije, morate premestiti datoteku ILDASM.EXE u direktorijum koji se nalazi u podrazumevanoj putanji ili uključiti taj direktorijum u tu putanju.

Pakovanje .NET aplikacije u sklop

U okruženju .NET instalaciona jedinica programa je sklop (engl. *assembly*). Sklop može sadržati program ili biblioteku klasa. Pored toga, sklop sadrži informacije o objektu (koje programeri zovu metapodaci). Na slici 2-6 prikazane su uobičajene komponente sklopa. Jedan sklop može obuhvatiti više fizičkih datoteka (programskih modula).



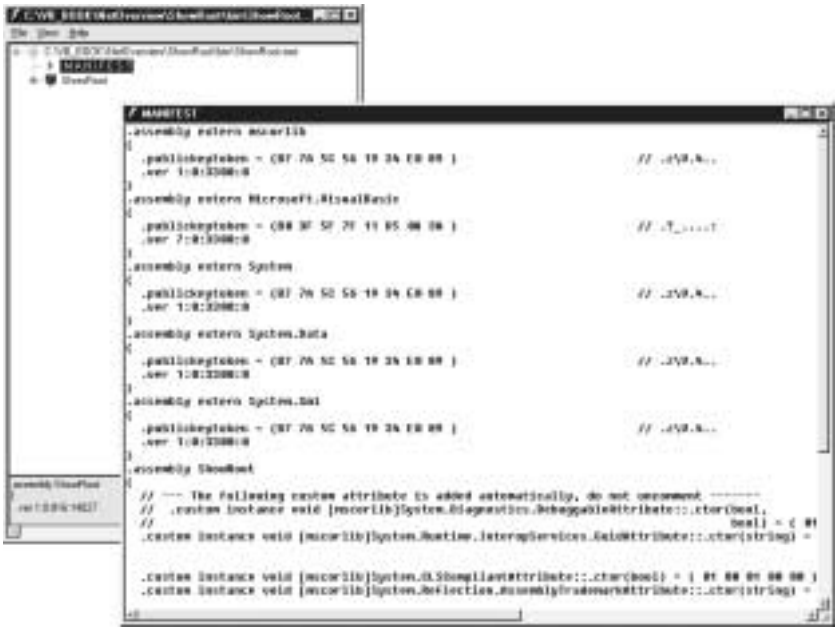
Slika 2-6 .NET aplikacije i biblioteke klasa instaliraju se u obliku sklopova (Manifest, Javni ključ-identifikator, Modul 1, Metapodaci, IL kôd, Modul n, Metapodaci, IL kôd).

Svaki sklop sadrži manifest koji opisuje sadržaj sklopa. Manifest sadrži ime sklopa i broj verzije, kao i listu datoteka koje čine sklop.

PRIMER Pomoću programa IL Disassembler koji se isporučuje uz Visual Studio možete pregledati sadržaj manifesta, kao što je prikazano na slici 2-7.

Da biste pregledali manifest pomoću alatke IL Disassembler, pokrenite program ILDASM sa komandne linije navodeći ime odgovarajućeg programa. Da biste pokrenuli taj program, treba da prebacite datoteku ILDASM.EXE u direktorijum koji se nalazi u komandnoj putanji ili da u ILDASM putanju uključite direktorijum u kom se program nalazi.

```
C:\>ildasm \DirektorijumPrograma\ImePrograma.exe
```



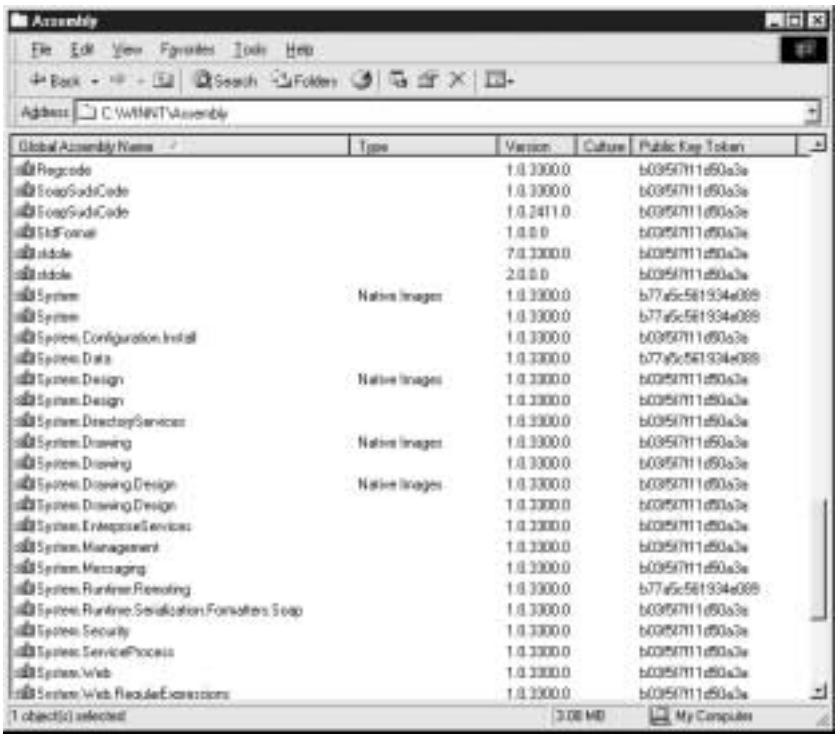
Slika 2-7 Upotreba IL Disassemblera za pregledanje manifesta sklopa.

Sklop može biti privatan za određenu aplikaciju, što znači da taj sklop koristi samo jedna aplikacija zbog koje sklop i postoji. U drugom slučaju, isti sklop mogu deliti dve ili više aplikacija. U okruženju Windows 2000, deljeni sklopovi se nalaze u direktorijumu koji programeri zovu globalni keš za sklopove (engl. *Global Assembly Cache*, *GAC*); to je najčešće direktorijum C:\WinNT\Assembly, kao što je prikazano na slici 2-8. .NET sklopovi detaljno su opisani u poglavlju 12.

Moć upravljane memorije i skupljanja smeća

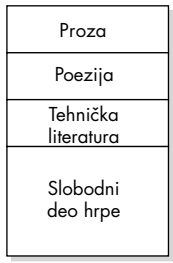
U okruženju .NET objekti programa koji se izvršava nalaze se na posebnoj lokaciji u memoriji koju programeri zovu hrpa (engl. *heap*). Ono što .NET hrpu čini posebnom jeste „upravljana memorija“. U pozadini, poseban .NET softver, nazvan skupljač smeća, briše objekte i oslobađa memoriju koju je objekat zauzeo, kada se objekat više ne koristi.

Ranije je o raspodeli i oslobađanju memorije objekta morao da se stara sam program. Na nesreću, mnogi programi su zauzimali memoriju koju kasnije više nisu vraćali operativnom sistemu. Čak i po završetku rada programa, memorija bi ostala zauzeta (dakle, bila bi nedostupna za druge programe). Programeri takve greške zovu „curenje memorije“ zato što vremenom, tokom rada takvog programa, količina memorije koja je na raspolaganju sistemu postaje sve manja.



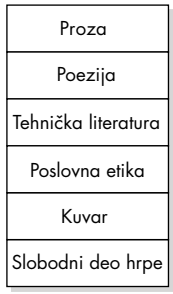
Slika 2-8 Pregled deljenih sklopova u globalnom kešu za sklopove.

U okruženju .NET svaki objekat koji napravite nalazi se u upravljanoj hrpi. Model koji upravljana hrpa koristi za dodeljivanje memorije objektu prilično je jednostavan i zahvaljujući tome prilično brz. Pretpostavimo da vaš kôd pravi tri objekta klase Knjiga koje smo nazvali Proza, Poezija i Tehnička literatura. Upravljana hrpa raspodeliće memoriju kao na slici 2-9.

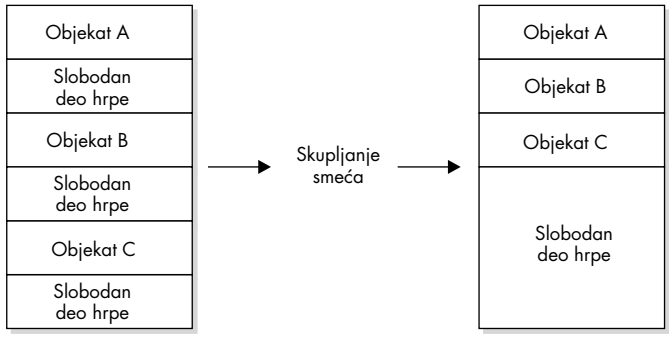


Slika 2-9 Raspodela memorije u hrpi za tri objekta tipa Knjiga.

Zatim, pretpostavimo da vaš program više ne koristi objekat Poezija i da je napravio dva nova objekta koja smo nazvali Poslovna etika i Kuvari. Softver za upravljanje hrpom oslobodiće memoriju koju je koristio objekat Poezija, i raspodeliće memoriju za dva nova objekta na dnu hrpe, kao što je prikazano na slici 2-10. Softver za upravljanje hrpom ne pokušava da pronađe slobodnu oblast memorije koja je dovoljno velika da se u nju smesti novi objekat. Umesto toga, softver samo dodeljuje memoriju novom objektu počev od kraja hrpe kao što je prikazano na slici 2-11.



Slika 2-10 Softver za upravljanje hrpom raspodelio je memoriju za nove objekte od dna hrpe.



Slika 2-11 Skupljač smeća premešta blokove u hrpi i postavlja slobodan deo memorije na dno hrpe.

Tokom vremena, kako program oslobađa pojedine objekte, u hrpi mogu nastati slobodni delovi. Tada će softver za upravljanje hrpom pokrenuti skupljač smeća koji će razmestiti stavke u hrpi tako da iskoristi prethodno oslobođenu memoriju i osloboditi memoriju na dnu hrpe, kao na slici 2-11.

PRIMER U sledećem programu, HeapUse.vb, prvo se prikazuje količina memorije u hrpi koja je dodeljena programu. Zatim program raspodeljuje memoriju za dva objekta. Nakon toga, progam ponovo prikazuje količinu raspodeljene memorije u hrpi.

Zatim program proglašava jedan od objekata nepotrebnim i zahteva da se pokrene skupljač smeća i očisti hrpu (prosleđivanjem vrednosti True metodi GetTotalMemory):

```
Module Module1

    Sub Main()
        Console.WriteLine("Starting Heap Space: " & _
          GC.GetTotalMemory(False))

        Dim BigArray(50000) As Byte
        Dim BiggerArray(250000) As Byte
        Console.WriteLine("Heap Space After Arrays: " & _
          GC.GetTotalMemory(False))

        BigArray = Nothing

        Console.WriteLine("Final Heap Space: " & _
          GC.GetTotalMemory(True))
        Console.ReadLine()
    End Sub
End Module
```

Da bi naznačio da neće dalje koristiti promenljivu BigArray, program postavlja vrednost te promenljive na Nothing i tako obeležava objekat kao odbačen za buduće operacije skupljanja smeća. Kada program pozove metodu GetTotalMemory da bi utvrdio raspodeljenu memoriju u hrpi, prosleđuje joj vrednost tipa Boolean koja saopštava skupljaču smeća da li treba da prvo uništi odbačene objekte pre vraćanja rezultata. Ako mu pozivajući program prosledi vrednost True, skupljač smeća će prvo izvesti skupljanje, a zatim utvrditi veličinu prostora u hrpi koji je program zauzeo. Ako program prosledi vrednost False, skupljač smeća neće uništiti odbačene objekte i vratiće trenutnu raspodelu memorije u hrpi.

Nakon što prevedete i pokrenete ovaj program, na ekranu računara pojaviće se ovakav rezultat:

```
Starting Heap Space: 10216
Heap Space After Arrays: 322545
Final Heap Space: 268825
```

Nakon što program završi rad, softver za upravljanje hrpom oslobodiće memoriju od preostalih objekata programa.

Verzije .NET objekata

U Windows okruženju opsežniji programi se najčešće sastoje iz datoteke tipa .exe i više datoteka dinamičkih biblioteka (DLL). Nažalost, kada korisnik instalira nov program na svoj računar, dešava se da novoinstalirani program zameni postojeću DLL datoteku sa novijom verzijom (ili u nekim slučajevima, u zavisnosti od instalacionog softvera, čak i sa starijom), što može prouzrokovati da programi koji su radili pre instalacije sada više ne rade. Windows programeri zovu takve sukobe koji se pojavljuju između različitih verzija DLL datoteka „DLL pakao“.

Da bi se smanjila mogućnost takvih sukoba između .NET aplikacija, sklop u koji je upakovana aplikacija sadrži i informacije o verziji. Aplikacije mogu da zahtevaju određenu verziju biblioteke klasa (u stvari, aplikacije zahtevaju određenu verziju sklopa koji sadrži biblioteku klasa). Dalje, dve verzije istog sklopa mogu postojati uporedo na sistemu a programi mogu odabrati sklop prema verziji koja im treba. U stvari, program može čak koristiti obe verzije sklopa u isto vreme kako bi referencirao različite objekte.

U manifestu sklopa (metapodaci koji opisuju sklop) nalazi se četvorocifren broj verzije koji definiše glavni i sporedni redni broj verzije iza kog slede redni broj sastavljanja i broj revizije sastavljanja.

```
Glavni:Sporedni:Sastavljanje:Revizija
```

U sledećoj naredbi ilustrovan je oblik u kome se broj verzije može pojaviti u sklopu:

```
.ver 1:0:816:147
```

PRIMER U poglavlju 12 opisuju se detaljno verzije u okruženju .NET. Upotrebom alatke IL Disassembler, koja se razmatra u odeljku. „Prednosti IL koda“, možete videti verziju sklopa, kao što je prikazano na slici 2-12. Kada pregledate sklop, videćete da sadrži verzije svake komponente koja mu je neophodna (kao što su DLL datoteke).

Slika 2-12 Pregled informacija o verzijama sadržanim u sklopu.

Standardizovana obrada grešaka

Godinama su programeri u Visual Basicu koristili naredbu ON ERROR da bi obradili greške i svojstva ERR.Number da bi pribavili podatke o grešci. Kao što znate, u okruženju .NET rutine u CLR-u ne zavise od programskog jezika, što znači da će i program napisan na jeziku C# i onaj napisan na Visual Basicu .NET pozivati istu rutinu.

CLR ne podržava naredbu ON ERROR. Umesto toga, kada procedura u CLR-u naiđe na neočekivanu grešku, ona generiše izuzetak (engl. *exception*) koji kôd programa mora da otkrije i obradi. Ako program ne obradi grešku, prikazuje se poruka o grešci u kojoj se opisuje izuzetak, kao na sledećoj slici, a program prekida rad.

PRIMER Da bi otkrili i obradili greške, programi pisani u Visual Basicu .NET moraju zadati operacije pri kojima može doći do izuzetka unutar naredbe Try-Catch. Na primer, sledeće naredbe pokušaće da otvore tekstualnu datoteku Report koja se nalazi u direktorijumu Temp na disku C:. Ako datoteka ne postoji, metoda StreamReader će generisati izuzetak. U ovom slučaju, pošto program poziva rutinu iz bloka Try-Catch, može otkriti i obraditi grešku:

```
Try
  SourceFile = New StreamReader("C:\Temp\Report.txt")
Catch Except As Exception
  MsgBox("Greška pri otvaranju datoteke C:\Temp\Report.txt")
End Try
```

U zavisnosti od obrade koju procedura obavlja, ona može generisati različite izuzetke od kojih svaki odgovara određenoj grešci. Na primer, kada program pokuša da otvori datoteku, metoda StreamReader može naići na grešku jer datoteka ne postoji, direktorijum koji je zadat ne postoji, datoteka je zaključana, datoteka je samo za čitanje itd. U kodu možete koristiti blok Try-Catch da biste obradili određene greške, kao što je prikazano dole:

```
Try
  SourceFile = New StreamReader("C:\Temp\Report.txt")
Catch Except As DirectoryNotFoundException
  MsgBox("Greška: direktorijum Temp nije pronađen")
Catch Except As FileNotFoundException
  MsgBox("Greška: datoteka Report.txt nije pronađena")
Catch Except As Exception
  MsgBox("Greška: " & Except.Message)
End Try
```

Upotrebu naredaba Try-Catch programeri zovu strukturirana obrada grešaka. U poglavlju 9 detaljno su opisani izuzeci i strukturirana obrada grešaka.