

22 Serialisierung von Objekten

| | |
|------------|---|
| 658 | Einführung in Serialisierungstechniken |
| 666 | Flaches und tiefes Klonen von Objekten |
| 671 | Serialisieren von Objekten mit Zirkelverweisen |
| 674 | XML-Serialisierung |

Wenn Sie Anwendungen entwickeln, kommen Sie irgendwann zwangsläufig an den Punkt, an dem Sie die Objekte, mit deren Hilfe Sie die Daten Ihrer Anwendung verwalten, für den späteren Gebrauch sichern oder zur Weiterverarbeitung an eine andere Instanz übertragen müssen. Vom aktuellen »Zustand« eines Objektes muss in diesem Fall eine Art Momentaufnahme gemacht werden; der Speicherinhalt aller Eigenschaften und aller untergeordneten Objekte muss dabei gesichert werden. Dabei spielt es natürlich erst einmal keine Rolle, auf welche Weise die Daten gesichert werden: Sie können bytewise – so, wie sie im Arbeitsspeicher stehen – direkt dort ausgelesen und in eine Datei geschrieben werden; denkbar wäre auch, dass sie zuvor durch entsprechende Algorithmen komprimiert und erst dann in einer Datei gespeichert werden. Die Daten eines Objektes, wie Zahlen oder Datumswerte, könnten auch zuvor in ein vom Anwender lesbares Format umgewandelt und speziell formatiert werden, sodass ein Transfervorgang sie auch beispielsweise im *XML*- oder *Soap*-Format direkt über das Internet zu einem anderen Server transportieren könnte.

Ganz gleich wie die Daten aus einem Objekt »geholt« und anschließend an eine andere Stelle verfrachtet werden – die Prozedur, die diese Aufgabe erfüllt, kann nur nach einem bestimmten Schema vorgehen: Sie muss *der Reihe nach* alle Eigenschaften und Variablen des Objektes abfragen, sie aufbereiten und anschließend mit den aufbereiteten Daten irgendetwas anstellen. Bei diesem Vorgang spricht man vom Serialisieren von Objekten.

Auch der umgekehrte Weg ist notwendig: Wenn Sie beispielsweise eine Reihe von Adressobjekten in eine Textdatei serialisiert haben, damit die Daten nach Beenden des Adressprogramms und Ausschalten des Computers erhalten bleiben, muss der umgekehrte Prozess stattfinden, sobald der Anwender den Computer einschaltet, die Adressverwaltung startet und mit den zuvor erfassten Adressen weiterarbeiten möchte. In diesem Fall müssen die Objekte wieder den gleichen Zustand annehmen, den sie vor dem Serialisieren hatten. Sie müssen jetzt aus der Textdatei *deserialisiert* werden.

Nun wäre das Framework nicht das Framework, wenn es Sie bei diesen Vorgängen, die natürlich in jeder Anwendung vorkommen, nicht unterstützen würde. Sie brauchen also nicht selbst Hand anzuheben und jede einzelne Eigenschaft eines Objektes auszulesen und in eine Textdatei zu schreiben. Umgekehrt müssen Sie Adressobjekte auch nicht selbst in Ihrer Anwendung komplett neu instanzie-

ren, während Sie sie aus einer Textdatei wieder einlesen. Das Framework hält für diesen Zweck einige geniale Werkzeuge bereit, die Sie in den folgenden Abschnitten kennen lernen sollten, da sie Ihnen die Arbeit erheblich erleichtern können – nicht nur, wenn Sie die Adressen Ihrer Adressverwaltung auf der Festplatte Ihres Computers speichern wollen.

Einführung in Serialisierungstechniken

Bevor Sie sich anschauen, was das Framework an Serialisierungstechniken zu bieten hat, lassen Sie uns zunächst einen Blick auf ein ganz simples Beispiel werfen. Dieses Beispielprogramm macht nichts weiter, als den Anwender eine Adresse eingeben zu lassen und diese durch Klick auf eine Schaltfläche in eine Datei zu serialisieren (sprich: zu sichern).

BEGLEITDATEIEN: Sie finden dieses Beispiel im Verzeichnis `.\VB 2005 - Entwicklerbuch\E - Datentypen\Kap22\Serialization01\`.



Abbildung 22.1: Die Nur-Lesen-Eigenschaften werden beim Serialisieren nicht übernommen

Das Beispiel nutzt dabei zunächst noch keine Serialisierungstechniken des Frameworks, sondern liest die einzelnen Eigenschaften sozusagen manuell aus und speichert sie als Text in einer Datei. Beim Klicken auf die Schaltfläche *Serialisieren* generiert die Prozedur, die das Click-Ereignis behandelt, ein Adresse-Objekt aus den Feldinhalten des Dialoges. Dieses Objekt übergibt die Routine einer weiteren Prozedur, die eine Datei zum Schreiben öffnet, die Eigenschaften nacheinander ausliest und sie als String in die Datei schreibt:

```
Private Sub btnSerialisieren_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) _
    Handles btnSerialisieren.Click
    Dim locAdresse As New Adresse(txtVorname.Text, _
        txtNachname.Text, _
        txtStraße.Text, _
        txtPLZOrt.Text)
    'Einen "unmöglichen" Dateinamen verwenden, damit, wie es der Zufall will,
    'nicht eine andere wichtige Datei gleichen Namens überschrieben wird.
    Adresse.SerializeToFile(locAdresse, "C:\serializedemo_f4e3w21.txt")

    'Info über den Datensatz anzeigen.
    txtErstelltAm.Text = locAdresse.ErfasstAm.ToString("dd.MM.yyyy HH:mm:ss")
    txtErstelltVon.Text = locAdresse.ErfasstVon
End Sub
```

Die Adresse-Klasse hält die notwendigen Elemente zum Speichern der Adressdaten und die statische Prozedur zum Serialisieren der Objektdaten in eine Datei bereit:

```
Public Class Adresse
    Private myName As String
    Private myVorname As String
    Private myStraße As String
    Private myPLZOrt As String
    Private myErfasstAm As DateTime
    Private myErfasstVon As String

    Sub New(ByVal Vorname As String, ByVal Name As String, ByVal Straße As String, ByVal PLZOrt As String)
        'Konstruktor legt alle Member-Daten an.
        myName = Name
        myVorname = Vorname
        myStraße = Straße
        myPLZOrt = PLZOrt
        myErfasstAm = DateTime.Now
        myErfasstVon = Environment.UserName
    End Sub

    'Alle öffentlichen Felder in die Datei schreiben.
    Public Shared Sub SerializeToFile(ByVal adr As Adresse, ByVal Filename As String)
        Dim locStreamWriter As New StreamWriter(Filename, False, System.Text.Encoding.Default)
        With locStreamWriter
            .WriteLine(adr.Vorname)
            .WriteLine(adr.Name)
            .WriteLine(adr.SträÙe)
            .WriteLine(adr.PLZOrt)
            .Flush()
            .Close()
        End With
    End Sub

    'Aus der Datei lesen und daraus ein neues Adressobjekt erstellen.
    Public Shared Function SerializeFromFile(ByVal Filename As String) As Adresse
        'Interessiert an dieser Stelle nicht, deswegen ausgelassen.
    End Function

    Public Property Name() As String
        Get
            Return myName
        End Get
        Set(ByVal Value As String)
            myName = Value
        End Set
    End Property
End Class
```

```

'Die beiden folgenden Eigenschaften haben "Nur-Lesen-Status", da auch
'der Entwickler das Anlegen-Datum nicht manipulieren darf!
Public ReadOnly Property ErfasstAm() As DateTime
    Get
        Return myErfasstAm
    End Get
End Property

Public ReadOnly Property ErfasstVon() As String
    Get
        Return myErfasstVon
    End Get
End Property

#Region "Die anderen Eigenschaften"
'Aus Platzgründen ausgelassen.
#End Region
End Class

```

Sie sehen anhand dieses Codelistings, dass es zwei Eigenschaften gibt, die zwar gelesen, aber nicht geschrieben werden dürfen: Diese Eigenschaften, die Auskunft darüber geben, wer das Adresse-Objekt zu welchem Zeitpunkt angelegt hat, dürfen ausschließlich bei der Objekterstellung definiert werden.

Bei Serialisieren auf die herkömmliche Art und Weise, wie hier im Beispiel zu sehen, ergibt sich hier schon ein Problem: Wenn Sie das Programm nämlich beenden, anschließend erneut starten und die Deserialisieren-Schaltfläche betätigen, dann wird zwar aus der Datei ein neues Adresse-Objekt erzeugt. Dieses Objekt entspricht dem ursprünglichen aber nicht in allen Details, denn: Das Erstellungsdatum und der »Ersteller« des Objektes selber hätten zwar noch mitgesichert werden können; der Deserialisierungs-Algorithmus hat aber auf Grund der Beschaffenheit dieser zusätzlichen Eigenschaften keine Möglichkeit, die Originalzustände dieser Eigenschaft wieder einzulesen. Er könnte diese Zusatzinformationen dem Objekt schlichtweg nicht zuordnen.

Diese Tatsache spiegelt sich im Programm wider: Wann immer Sie das ursprünglich gespeicherte Objekt durch Klick auf die entsprechende Schaltfläche deserialisieren, steht in den unteren Infofeldern nicht das ursprüngliche Erstellungsdatum, sondern das Erstellungsdatum des Objektes zum Zeitpunkt des Deserialisierens.

Serialisieren mit SoapFormatter und BinaryFormatter

Anders sieht es aus, wenn Sie das Serialisieren und das Deserialisieren mit bestimmten Hilfsmitteln aus dem Framework durchführen. Sie brauchen sich in diesem Fall nämlich nicht um das Auslesen der Eigenschaften der Objekte selbst zu kümmern – das Framework macht das automatisch für Sie. Und noch mehr: Einige Klassen des Frameworks, die für das Serialisieren von Objektdaten zuständig sind, erlauben auch das Serialisieren und Deserialisieren von privaten Eigenschaften einer Klasse.

WICHTIG: Für alle Objekte, die serialisiert werden sollen, gilt: Sie müssen mit einem besonderen Attribut namens `Serializable` gekennzeichnet werden. Wenn diese Voraussetzungen erfüllt sind, können Sie zwei der Serialisier-Klassen für die Serialisierung und Deserialisierung einer so gekennzeichneten Objekt-Instanz verwenden, die auch private Member-Variablen einer Klasse berücksichtigen.

- **SoapFormatter-Klasse:** Stellt Serialisierungs- und Deserialisierungsfunktionen zur Verfügung, die das SOAP-Format verwenden. Die Dateninhalte eines Objektes werden dabei in reinen Text umgewandelt, der auf der einen Seite auch für den Anwender verständlich mit einem Texteditor gelesen und angezeigt und auf der anderen Seite – da er das SOAP-Format einhält – auch problemlos über das Internet transportiert werden kann. Der Nachteil: Diese Art der Datenspeicherung ist nicht sonderlich effizient, eben durch die Konvertierung nativer Daten in lesbaren Text.
- **BinaryFormatter-Klasse:** Stellt Serialisierungs- und Deserialisierungsfunktionen zur Verfügung, die das Binärformat verwenden. Die Dateninhalte eines Objektes werden dabei so verwendet, wie sie im Arbeitsspeicher vorliegen. Werden die Daten eines Objektes unter Verwendung dieses *Serializers* beispielsweise in einer Datei gespeichert, erfolgt die Datenspeicherung sehr kompakt. Eine auf diese Weise generierte Datei ist aber für den Anwender direkt nicht lesbar, da sie die Daten des Objektes im binären Format enthält.

BEGLEITDATEIEN: Das folgende Beispiel demonstriert den Einsatz mit diesen beiden Klassen. Sie finden dieses Beispiel unter `.\VB 2005 - Entwicklerbuch\E - Datentypen\Kap22\Serialization02\`.

Abbildung 22.2: In dieser Version können Sie zwischen Soap- und Binärserialisierung wählen. Die Nur-Lesen-Eigenschaften bleiben jetzt beim Deserialisieren erhalten

Wenn Sie dieses Programm starten, wählen Sie für das Serialisieren bzw. Deserialisieren das gewünschte Format aus. Wenn Sie sich für das Soap-Format entscheiden, generiert die SoapFormatter-Klasse eine Datei, die der folgenden entspricht (vorausgesetzt natürlich, Sie haben die gleichen Daten eingegeben, wie in Abbildung 22.2 zu sehen):

```
<SOAP-ENV:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:clr="http://schemas.microsoft.com/soap/encoding clr/1.0" SOAP-
ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
<SOAP-ENV:Body>
<a1:Adresse id="ref-1"
xmlns:a1="http://schemas.microsoft.com/clr/nsassem/Serialization02/Serialization02%2C%20Version%3D1.0.0.0%2C%20Culture%3Dneutral%2C%20PublicKeyToken%3Dnu11">
<myName id="ref-3">Thiemann</myName>
<myVorname id="ref-4">Uwe</myVorname>
<myStraße id="ref-5">Autorenstraße 34</myStraße>
<myPLZOrt id="ref-6">99999 Buchhausen</myPLZOrt>
<myErfasstAm>2006-02-15T11:29:24.1295355+01:00</myErfasstAm>
<myErfasstVon id="ref-7">ACTIVEDEVELOP\loeffel</myErfasstVon>
</a1:Adresse>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Sie werden überrascht sein, wenn Sie sehen, mit wie wenig Aufwand die Serialisierung im Programm realisiert wurde. Wenn Sie einen Blick in den Projektmappen-Explorer werfen, finden Sie dort im Gegensatz zum vorherigen Beispiel zwei weitere Klassendateien. Sie enthalten die Kapselungen von SoapFormatter und BinaryFormatter zur Serialisierung (und Deserialisierung) beliebiger, serialisierbarer Objekte in Dateien, die folgendermaßen aussehen.

Universeller Soap-Datei-De-/Serializer

```
Imports System.Runtime.Serialization
Imports System.Runtime.Serialization.Formatters.Soap
Imports System.IO

Public Class ADSoapSerializer

    Shared Sub SerializeToFile(ByVal FileInfo As FileInfo, ByVal [Object] As Object)

        Dim locFs As FileStream = New FileStream(FileInfo.FullName, FileMode.Create)
        Dim locSoapFormatter As New SoapFormatter(Nothing, _
            New StreamingContext(StreamingContextStates.File))
        locSoapFormatter.Serialize(locFs, [Object])
        locFs.Flush()
        locFs.Close()

    End Sub

    Shared Function DeserializeFromFile(ByVal FileInfo As FileInfo) As Object

        Dim locObject As Object

        Dim locFs As FileStream = New FileStream(FileInfo.FullName, FileMode.Open)
        Dim locSoapFormatter As New SoapFormatter(Nothing, _
            New StreamingContext(StreamingContextStates.File))
        locObject = locSoapFormatter.Deserialize(locFs)
        locFs.Close()
        Return locObject

    End Function
End Class
```

Wenn Sie SoapFormatter verwenden wollen, müssen Sie einen Verweis in das entsprechende Projekt einbauen. Dazu öffnen Sie im Projektmappen-Explorer über dem Projektnamen oder dem Ordner *Verweise* (wenn das Symbol *alle Dateien anzeigen aktiviert* ist) mit der rechten Maustaste das Kontextmenü und wählen dort den Eintrag *Verweise*. Im Dialog, der sich jetzt öffnet, wählen Sie den Eintrag `System.Runtime.Serialization.Formatters.Soap` per Doppelklick aus (etwa wie in Abbildung 22.3 zu sehen).

Verlassen Sie den Dialog mit *OK*. Achten Sie ebenfalls darauf, die Anweisungen

```
Imports System.Runtime.Serialization
Imports System.Runtime.Serialization.Formatters.Soap
```

als erste Zeilen in der Codedatei zu platzieren, in der Sie SoapFormatter verwenden möchten.

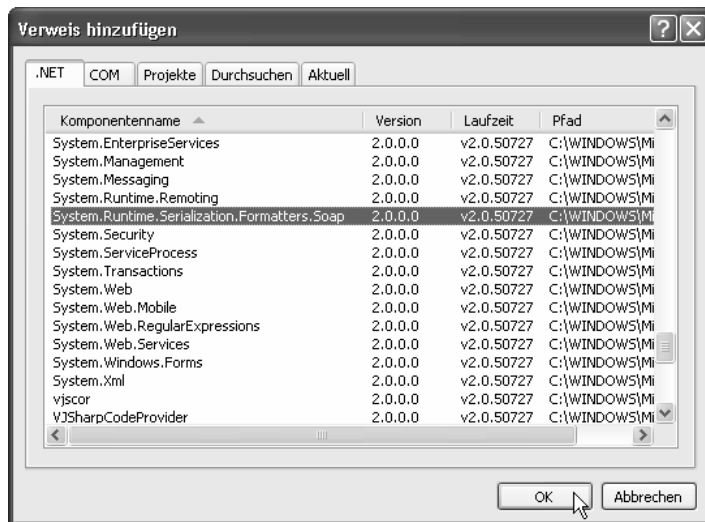


Abbildung 22.3: Wenn Sie mit der *SoapFormatter*-Klasse Objektdaten im Soap-Format serialisieren wollen, binden Sie diesen Verweis in Ihr Projekt ein

Universeller Binary-Datei-De-/Serializer

```
Imports System.Runtime.Serialization
Imports System.Runtime.Serialization.Formatters.Binary
Imports System.IO
```

```
Public Class ADBinarySerializer
```

```
    Shared Sub SerializeToFile(ByVal FileInfo As FileInfo, ByVal [Object] As Object)
```

```
        Dim locFs As FileStream = New FileStream(FileInfo.FullName, FileMode.Create)
        Dim locBinaryFormatter As New BinaryFormatter(Nothing, _
            New StreamingContext(StreamingContextStates.File))
        locBinaryFormatter.Serialize(locFs, [Object])
        locFs.Flush()
        locFs.Close()
```

```
    End Sub
```

```
    Shared Function DeserializeFromFile(ByVal FileInfo As FileInfo) As Object
```

```
        Dim locObject As Object

        Dim locFs As FileStream = New FileStream(FileInfo.FullName, FileMode.Open)
        Dim locBinaryFormatter As New BinaryFormatter(Nothing, _
            New StreamingContext(StreamingContextStates.File))
        locObject = locBinaryFormatter.Deserialize(locFs)
        locFs.Close()
        Return locObject
```

```
    End Function
```

```
End Class
```

Achten Sie darauf, die Anweisungen

```
Imports System.Runtime.Serialization
Imports System.Runtime.Serialization.Formatters.Binary
```

als erste Zeilen in der Codedatei zu platzieren, in der Sie BinaryFormatter verwenden möchten. Einen speziellen Verweis brauchen Sie, anders als bei SoapFormatter, nicht in das Projekt einzubinden.

Funktionsweise der Datei-Serializer-Klassen

Beide Klassen funktionieren exakt nach dem gleichen Prinzip, sie verwenden lediglich unterschiedliche Formatter (also Klassen, die Datenaufbereitungslogiken zur Verfügung stellen), um unterschiedliche Formate zu erzeugen bzw. für die Rekreation des Ursprungsobjektes zu verwenden.

Sie öffnen beim Serialisieren zunächst einen Dateistrom, über den die Daten über den jeweiligen Formatter aus dem Objekt in die Datei gelangen. In der anschließenden Instanzierung des Formatters wird dieser durch die Übergabe einer StreamingContext-Instanz darauf vorbereitet, welches Ziel die Objektserialisierung haben wird (in diesem Fall werden die Objektdaten in eine Datei serialisiert). Die eigentliche Serialisierung des Objektes geschieht dann durch die einzige Zeile:

```
locSoapFormatter.Serialize(locFs, [Object])
```

Der Rest der Prozedur ist obligatorisch: Alle Stromdaten werden mit Flush aus dem internen Puffer geleert, und die Datei wird anschließend geschlossen. Das Ergebnis: Die Objektdaten wurden einschließlich ihrer privaten Member in die Datei geschrieben. Der umgekehrte Weg beim Deserialisieren erfolgt äquivalent.

TIPP: Wenn Sie diese Art der Serialisierung in Dateien in Ihren eigenen Programmen verwenden wollen, kopieren Sie die Codedateien ADBinarySerializer.vb bzw. ADSoapSerializer.vb einfach in Ihr Projektverzeichnis und fügen sie anschließend Ihrem Projekt hinzu. Sie können sie dann auf genauso einfache Weise verwenden, wie sie im Beispielprojekt vom Adressobjekt verwendet wurden (siehe in Fettschrift gesetzte Zeilen am Ende des Codelistings):

```
<Serializable(> _
Public Class Adresse
```

```
    Private myName As String
    Private myVorname As String
    Private myStraße As String
    Private myPLZOrt As String
    Private myErfasstAm As DateTime
    Private myErfasstVon As String
```

```
    Sub New(ByVal Vorname As String, ByVal Name As String, ByVal Straße As String, ByVal PLZOrt As String)
        'Konstruktor legt alle Member-Daten an.
        myName = Name
        myVorname = Vorname
        myStraße = Straße
        myPLZOrt = PLZOrt
        myErfasstAm = DateTime.Now
        myErfasstVon = Environment.UserName
    End Sub
```

```

'Alle öffentlichen Felder in die Datei schreiben - Soap-Format
Public Shared Sub SerializeSoapToFile(ByVal adr As Adresse, ByVal Filename As String)
    ADSoapSerializer.SerializeToFile(New FileInfo(Filename), adr)
End Sub

'Aus der Datei lesen und daraus ein neues Adressobjekt erstellen - Soap-Format.
Public Shared Function SerializeSoapFromFile(ByVal Filename As String) As Adresse
    Return CType(ADSoapSerializer.DeserializeFromFile(New FileInfo(Filename)), Adresse)
End Function

'Alle öffentlichen Felder in die Datei schreiben - Binary-Format.
Public Shared Sub SerializeBinToFile(ByVal adr As Adresse, ByVal Filename As String)
    ADBinarySerializer.SerializeToFile(New FileInfo(Filename), adr)
End Sub

'Aus der Datei lesen und daraus ein neues Adressobjekt erstellen - Binary-Format.
Public Shared Function SerializeBinFromFile(ByVal Filename As String) As Adresse
    Return CType(ADBinarySerializer.DeserializeFromFile(New FileInfo(Filename)), Adresse)
End Function
.
.
.

```

WICHTIG: Wenn Sie die Objektserialisierung anwenden, achten Sie darauf, dass alle Objekte, die Sie serialisieren wollen, das `Serializable`-Attribut tragen – so wie auch im vorherigen Codelistings zu sehen (siehe erste beiden Zeilen). Wenn die Klasse, die Sie serialisieren möchten, oder eine Objektinstanz, die diese Klasse einbindet, dieses Attribut nicht trägt, löst das Framework beim Serialisierungsversuch durch eines der zu serialisierenden Objekte eine Ausnahme aus, etwa wie in Abbildung 22.4 zu sehen.

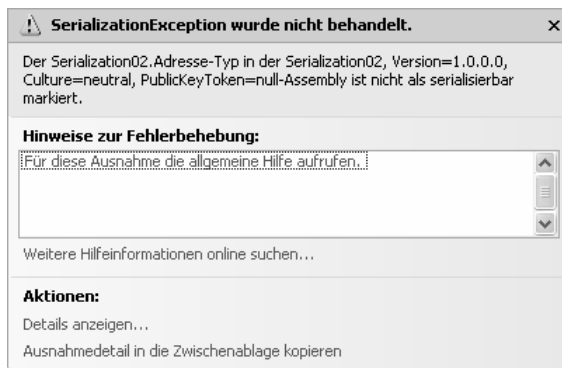


Abbildung 22.4: Wenn eine zu serialisierende Klasse oder ein Objekt, das sie einbindet, nicht mit dem Attribut `Serializable` gekennzeichnet ist, löst das Framework beim Serialisierungsversuch diese Ausnahme aus

Flaches und tiefes Klonen von Objekten

Der Vorteil beim Serialisieren über Funktionen des Frameworks ist, dass die Serialisierungsalgorithmen in der Lage sind, automatisch so genannte »tiefe Klons« (vollständige Kopien) eines Objektes zu erstellen.

Dazu folgender Hintergrund: Angenommen, Sie haben ein Objekt, das die Daten Ihrer Applikation speichert. Dieses Objekt verfügt dann beispielsweise über eine Eigenschaft, die eine ArrayList zur Verfügung stellt, in der weitere Elemente gespeichert sind. Um eine komplette Kopie dieses Objektes zu erstellen, würde es nicht ausreichen, die Elemente, die diese ArrayList-Eigenschaft beinhaltet, zu kopieren, wie das folgende Beispiel zeigt:

BEGLEITDATEIEN: Sie finden dieses Beispiel unter `.\VB 2005 - Entwicklerbuch\E - Datentypen\Kap22\DeepCloning\`.

```
Module md1Main
  Sub Main()
    Dim locAdrOriginal As New Adresse("Hans", "Mustermann", "Musterstraße 22", "59555 Lipstadt")
    Dim locAdrKopie As Adresse
    With locAdrOriginal.BefreundetMit
      .Add(New Adresse("Uwe", "Thiemann", "Autorstr. 33", "49595 Buchhausen"))
      .Add(New Adresse("Gaby", "Halek", "Autorstr. 34", "49595 Buchhausen"))
      .Add(New Adresse("Gabriele", "Löffelmann", "Autorstr. 35", "49595 Buchhausen"))
    End With
    'Originaladresse ausgeben.
    Console.WriteLine("Original:")
    Console.WriteLine("=====")
    Console.WriteLine(locAdrOriginal)

    'Kopie anlegen.
    With locAdrOriginal
      locAdrKopie = New Adresse(.Vorname, .Name, .Straße, .PLZOrt)
      locAdrKopie.Name += " (Kopie)"
      locAdrKopie.BefreundetMit = .BefreundetMit
    End With

    'Kopie ausgeben.
    Console.WriteLine("Kopie:")
    Console.WriteLine("=====")
    Console.WriteLine(locAdrKopie)

    'Änderungen im Original:
    CType(locAdrOriginal.BefreundetMit(1), Adresse).Name = "Löffelmann-Halek"
    CType(locAdrOriginal.BefreundetMit(2), Adresse).Name = "Löffelmann-Halek"

    'Kopie nach Änderungen im Original:
    Console.WriteLine("Kopie nach Änderung im Original:")
    Console.WriteLine("=====")
    Console.WriteLine(locAdrKopie)
    Console.ReadLine()
  End Sub
End Module
```

Dieses Beispiel nutzt die leicht abgewandelte Adresse-Klasse, die Sie schon aus dem vorherigen Beispiel kennen. Sie verfügt über eine zusätzliche Eigenschaft namens BefreundetMit. Diese Eigenschaft entspricht einer ArrayList mit der Aufgabe, andere Adressen aufzunehmen, die bestimmen, mit wem dieser Kontakt befreundet ist.

```
<Serializable(> _
Public Class Adresse

    Private myName As String
    Private myVorname As String
    Private myStraße As String
    Private myPLZOrt As String
    Private myErfasstAm As DateTime
    Private myErfasstVon As String
    Private myBefreundetMit As ArrayList

    Sub New(ByVal Vorname As String, ByVal Name As String, ByVal Straße As String, ByVal PLZOrt As String)
        'Konstruktor legt alle Member-Daten an.
        myName = Name
        myVorname = Vorname
        myStraße = Straße
        myPLZOrt = PLZOrt
        myErfasstAm = DateTime.Now
        myErfasstVon = Environment.UserName
        myBefreundetMit = New ArrayList
    End Sub

    Public Property BefreundetMit() As ArrayList
        Get
            Return myBefreundetMit
        End Get
        Set(ByVal Value As ArrayList)
            myBefreundetMit = Value
        End Set
    End Property

    #Region "Die anderen Eigenschaften"
        'Aus Platzgründen ausgelassen.
    #End Region

    Public Overrides Function ToString() As String
        Dim locTemp As String
        locTemp = Name + ", " + Vorname + ", " + Straße + ", " + PLZOrt + vbNewLine
        locTemp += "--- Befreundet mit: ---" + vbNewLine
        For Each locAdr As Adresse In BefreundetMit
            locTemp += " * " + locAdr.ToStringShort() + vbNewLine
        Next
        locTemp += vbNewLine
        Return locTemp
    End Function
End Class
```

```

    Public Function ToStringShort() As String
        Return Name + ", " + Vorname
    End Function
End Class

```

Zusätzlich gibt es in dieser Klasse zwei Funktionen – ToString und ToStringShort – die eine Adresse-Instanz in einen String umwandeln, damit die Ausgabe einfacher wird.

Das Programm macht nun Folgendes: Es legt eine Originaladresse an und fügt ihr weitere Adressobjekte hinzu, die in der ArrayList, die durch BefreundetMit offen gelegt wird, gespeichert werden. Anschließend erzeugt es eine identische Kopie der Originaladresse im Objekt locAdrKopie.

Das Problem: An dieser Stelle wird eine so genannte flache Kopie des Objektes erstellt. Nur die Eigenschaften der oberen Ebene werden in die Kopie übernommen. Das wird auch deutlich, wenn Sie das Beispielpogramm starten:

```

Original:
=====
Mustermann, Hans, Musterstraße 22, 59555 Lippstadt
--- Befreundet mit: ---
* Thiemann, Uwe
* Halek, Gaby
* Löffelmann, Gabriele

```

```

Kopie:
=====
Mustermann (Kopie), Hans, Musterstraße 22, 59555 Lippstadt
--- Befreundet mit: ---
* Thiemann, Uwe
* Halek, Gaby
* Löffelmann, Gabriele

```

```

Kopie nach Änderung im Original:
=====
Mustermann (Kopie), Hans, Musterstraße 22, 59555 Lippstadt
--- Befreundet mit: ---
* Thiemann, Uwe
* Löffelmann-Halek, Gaby
* Löffelmann-Halek, Gabriele

```

Beim Ändern der Elemente der BefreundetMit-ArrayList werden auch die Elemente der Kopie verändert. Und das muss auch so sein, denn: Die Eigenschaft BefreundetMit stellt ja nicht wirklich selbst eine ArrayList dar, sondern verweist lediglich auf sie. Tatsächlich gibt es die Elemente ArrayList nur ein einziges Mal. In diesem Beispiel ist also nur eine flache Kopie (»shallow clone« bzw. »shallow copy« auf englisch) des Adresse-Objektes erstellt worden.

Anders wird das, wenn Sie eine tiefe Kopie (»deep clone« bzw. »deep copy« auf englisch) erstellen. Hier werden die Elemente der ArrayList, um beim Beispiel zu bleiben, wirklich kopiert – es gibt nach Abschluss der Kopieerstellung wirklich zwei völlig unabhängige Objekte, mit ebenso unabhängigen Elementen.

Was bei diesem Beispiel mit noch vergleichsweise wenig Aufwand durchführbar wäre, sieht bei wirklich komplexen Objekten schon anders aus. Wollten Sie eine DeepCopy-Routine selber implementieren, bedeutete dies einen enormen Aufwand. Obendrein könnten Sie eine solche Routine mit normalen Mitteln nicht universal gestalten; sie würde sich nur auf das aktuelle Objekt beziehen. Bei einer Klassenableitung, die die Basisklasse um weitere Eigenschaften ergänzen würde, müssten Sie die DeepCopy-Routine schon wieder modifizieren.

Doch diesen Aufwand müssen Sie auch gar nicht betreiben, denn mit den Serialisierungsfunktionen nimmt Ihnen das Framework diesen kompletten Aufwand ab. Beim Serialisieren erstellt das Framework nämlich eine tiefe Kopie.¹ Und da Serialisierung natürlich nicht notwendigerweise bedeutet, das Objekt in einer Datei zwischenzuspeichern, können Sie mit einem kleinen Trick eine universelle DeepCopy-Routine kreieren, die mit jedem serialisierbaren Objekt funktioniert, wie das folgende Beispiel zeigt.

Universelle DeepClone-Methode

Das folgende Beispiel verwendet ebenfalls den BinarySerializer für das Serialisieren und das Deserialisieren eines Objektes, doch nutzt er einen anderen Träger im Vergleich zum letzten Beispiel. Ein Objekt, das es zu serialisieren gilt, wird nicht in eine Datei, sondern in ein MemoryStream-Objekt geschrieben, das schließlich in ein Byte-Array konvertiert wird. Im umgekehrten Fall erzeugt der BinarySerializer aus einem Byte-Array, das in ein MemoryStream-Objekt umgewandelt wird, wieder das ursprüngliche Objekt. Da der BinarySerializer grundsätzlich auch verschachtelte Eigenschaften verarbeitet, lässt sich daraus auf einfache Art und Weise eine Klasse erstellen, die von jedem beliebigen Objekt, das selbst als serialisierbar gekennzeichnet ist und auch nur selbst serialisierbare Objekte verwendet, eine tiefe Kopie anfertigen kann.

BEGLEITDATEIEN: Sie finden dieses Beispiel unter `.\VB 2005 - Entwicklerbuch\E - Datentypen\Kap22\UniversalDeepCloning\`.

```
Imports System.Runtime.Serialization
Imports System.Runtime.Serialization.Formatters.Binary
Imports System.IO

Public Class AObjectCloner

    Public Shared Function DeepCopy(ByVal [Object] As Object) As Object
        Return DeserializeFromArray(SerializeToArray([Object]))
    End Function

    Shared Function SerializeToArray(ByVal [Object] As Object) As Byte()

        Dim retByte() As Byte
        Dim locMs As MemoryStream = New MemoryStream
```

¹ Wobei diese Aussage nicht hundertprozentig richtig ist, denn eigentlich erstellt das Framework ja keine Kopie des Objektes, sondern liest beim Serialisieren zunächst nur seine Daten aus. Das macht es aber bis in die unterste Ebene, sodass man den kombinierten Vorgang von Serialisieren eines Objektes in einen Datenstrom und Deserialisieren dieses Datenstroms in ein neues Objekt durchaus »tiefes Kopieren« des Objektes nennen kann.

```

Dim locBinaryFormatter As New BinaryFormatter(Nothing, _
    New StreamingContext(StreamingContextStates.Clone))
locBinaryFormatter.Serialize(locMs, [Object])
locMs.Flush()
locMs.Close()
retByte = locMs.ToArray()
Return retByte

End Function

Shared Function DeserializeFromByteArray(ByVal by As Byte()) As Object

    Dim locObject As Object

    Dim locFs As MemoryStream = New MemoryStream(by)
    Dim locBinaryFormatter As New BinaryFormatter(Nothing, _
        New StreamingContext(StreamingContextStates.File))
    locObject = locBinaryFormatter.Deserialize(locFs)
    locFs.Close()
    Return locObject

End Function

End Class

```

Das Hauptprogramm verwendet im folgenden Beispiel nun nicht mehr eigenen Code zum Kopieren des Objektes, sondern benutzt die DeepCopy-Funktion der Klasse (die geänderte Passage ist fett hervorgehoben):

```

Module mdlMain

    Sub Main()
        Dim locAdrOriginal As New Adresse("Hans", "Mustermann", "Musterstraße 22", "59555 Lipstadt")
        Dim locAdrKopie As Adresse
        With locAdrOriginal.BefreundetMit
            .Add(New Adresse("Uwe", "Thiemann", "Autorstr. 33", "49595 Buchhausen"))
            .Add(New Adresse("Gaby", "Halek", "Autorstr. 34", "49595 Buchhausen"))
            .Add(New Adresse("Gabriele", "Löffelmann", "Autorstr. 35", "49595 Buchhausen"))
        End With

        'Originaladresse ausgeben.
        Console.WriteLine("Original:")
        Console.WriteLine("=====")
        Console.WriteLine(locAdrOriginal)

        'Kopie anlegen.
        locAdrKopie = CType(ADObjectCloner.DeepCopy(locAdrOriginal), Adresse)

        'Kopie ausgeben.
        Console.WriteLine("Kopie:")
        Console.WriteLine("=====")
        Console.WriteLine(locAdrKopie)
    End Sub
End Module

```

```

'Änderungen im Original:
CType(locAdrOriginal.BefreundetMit(1), Adresse).Name = "Löffelmann-Halek"
CType(locAdrOriginal.BefreundetMit(2), Adresse).Name = "Löffelmann-Halek"

'Kopie nach Änderungen im Original:
Console.WriteLine("Kopie nach Änderung im Original:")
Console.WriteLine("=====")
Console.WriteLine(locAdrKopie)
Console.ReadLine()

End Sub

```

End Module

Wenn Sie das Programm anschließend starten, erkennen Sie den Unterschied zum vorherigen Beispiel. Die beiden erzeugten Objekte sind wirklich komplett unabhängig voneinander. Änderungen an der ArrayList des Ausgangsobjektes beeinflussen das Ergebnis der Ausgabe der Objektkopie in keiner Weise:

```

Original:
=====
Mustermann, Hans, Musterstraße 22, 59555 Lipstadt
--- Befreundet mit: ---
* Thiemann, Uwe
* Halek, Gaby
* Löffelmann, Gabriele

```

```

Kopie:
=====
Mustermann, Hans, Musterstraße 22, 59555 Lipstadt
--- Befreundet mit: ---
* Thiemann, Uwe
* Halek, Gaby
* Löffelmann, Gabriele

```

```

Kopie nach Änderung im Original:
=====
Mustermann, Hans, Musterstraße 22, 59555 Lipstadt
--- Befreundet mit: ---
* Thiemann, Uwe
* Halek, Gaby
* Löffelmann, Gabriele

```

Serialisieren von Objekten mit Zirkelverweisen

So genannte Zirkelverweise bereiten Speicheralgorithmen erfahrungsgemäß die größten Schwierigkeiten. Zirkelverweise kennen Sie vielleicht aus der Tabellenkalkulation. Sie treten dann auf, wenn beispielsweise Zelle A auf Zelle B, diese auf Zelle C, und diese wieder auf Zelle A verweisen soll. Was bei Tabellenkalkulationen grundsätzlich verboten ist, gestattet das Framework mit Objektreferenzen

dagegen sehr wohl. Und – um bei unserem bisherigen Beispiel zu bleiben – im Szenario des Programms aus dem letzten Abschnitt könnte das sogar recht schnell passieren, denn: Es ist nicht nur denkbar, sondern wahrscheinlich, dass Person A Person B zu seinem Freundeskreis zählt und umgekehrt.

Welche Auswirkungen Zirkelverweise normalerweise haben, zeigt das folgende Beispiel.

BEGLEITDATEIEN: Sie finden dieses Beispiel unter `.\VB 2005 - Entwicklerbuch\E - Datentypen\Kap22\Zirkelverweise\`.

In diesem Beispielpogramm sind im Gegensatz zum vorherigen die folgenden Änderungen vorgenommen worden, was die Adresse-Klasse betrifft. Die ToString-Funktion benutzt selbst nicht mehr die Kurzform der Adressen für die Ausgabe der BefreundetMit-Eigenschaft, sondern erzeugt den Ausgabe-String ebenfalls mit der ToString-Funktion, die allerdings eine kleine Änderung erfahren hat. Welche das ist, sehen Sie, wenn Sie das Programm starten und sich anschließend das Ergebnis anschauen:

Erste Adresse:

=====

Halek, Gaby, Musterstraße 24, 32132 Buchhausen

--- Befreundet mit: ---

Raubein, Petra, Autorenstr. 12, 32132 Buchhausen

Thiemann, Uwe, Autorenstr. 22, 32132 Buchhausen

--- Befreundet mit: ---

Koch, Manuela, Autorenstr. 22, 32132 Buchhausen

Zweite Adresse:

=====

Thiemann, Uwe, Autorenstr. 22, 32132 Buchhausen

--- Befreundet mit: ---

Koch, Manuela, Autorenstr. 22, 32132 Buchhausen

Das Programm, das diese Ausgabe hervorgebracht hat, sieht folgendermaßen aus:

```
Module mdlMain
```

```
Sub Main()
```

```
Dim locErsteAdr As New Adresse("Gaby", "Halek", "Musterstraße 24", "32132 Buchhausen")
```

```
Dim locZweiteAdr As New Adresse("Uwe", "Thiemann", "Autorenstr. 22", "32132 Buchhausen")
```

```
locErsteAdr.BefreundetMit.Add(New Adresse("Petra", "Raubein", "Autorenstr. 12", "32132 Buchhausen"))
```

```
locErsteAdr.BefreundetMit.Add(locZweiteAdr)
```

```
locZweiteAdr.BefreundetMit.Add(New Adresse("Manuela", "Koch", "Autorenstr. 22", "32132 Buchhausen"))
```

```
'Wenn Sie diese Zeile einbauen, erstellen Sie einen Zirkelverweis.
```

```
'locZweiteAdr.BefreundetMit.Add(locErsteAdr)
```

```
'Originaladresse ausgeben.
```

```
Console.WriteLine("Erste Adresse:")
```

```
Console.WriteLine("=====")
```

```
Console.WriteLine(locErsteAdr)
```

```
Console.WriteLine("Zweite Adresse:")
```

```
Console.WriteLine("=====")
```

```

Console.WriteLine(locZweiteAdr)

'Kopie anlegen.
Dim locAdrKopie As Adresse = CType(ADObjectCloner.DeepCopy(locErsteAdr), Adresse)
Console.ReadLine()

End Sub
End Module

```

Sie erkennen am Ergebnis und dem Programmtext, dass das Programm auch verschachtelte Daten in der Eigenschaft `BefreundetMit` (und der dahinter steckenden `ArrayList`) bei der Ausgabe berücksichtigt. Wenn es die Liste der befreundeten Personen erstellt, und eine Person dieser Liste wieder Personeneinträge unter `BefreundetMit` führt, werden diese bei der Ausgabe ebenfalls berücksichtigt.

Eine solche Vorgehensweise kann für ein Programm allerdings wirklich fatale Folgen haben. Denn wenn eine der Personen der `BefreundetMit`-Liste in ihrer Liste die Person führt, die ihr ebenfalls zugeordnet ist (Person A ist befreundet mit Person B, und Person B ist ebenfalls befreundet mit Person A – was ja durchaus nichts Ungewöhnliches ist), dann haben Sie es mit einem Zirkelverweis zu tun.

Der `ToString`-Algorithmus versagt in diesem Fall, weil er sich durch den Zirkelverweis immer wieder selbst aufruft. Sie können dieses Verhalten testen, indem Sie die Auskommentierung der im Listing fett gesetzten Zeile aufheben und den Auflösungsversuch eines Zirkelverweises initiieren. Wenn Sie das Programm anschließend starten, erscheint nach einer Weile im Ausgabefenster die Meldung

Eine Ausnahme (erste Chance) des Typs "System.OutOfMemoryException" ist in `mscorlib.dll` aufgetreten.

Das Programm »hängt« eine Weile, und je nachdem, wie gut oder schlecht Ihr System mit der nur noch ungenügend zur Verfügung stehenden Menge an Speicher klarkommt, sehen Sie kurz darauf folgende Ausnahme:

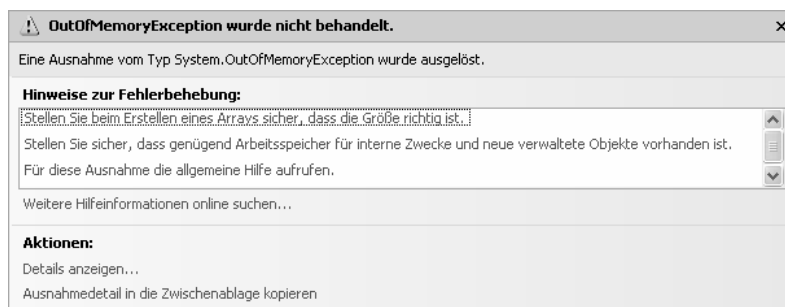


Abbildung 22.5: Durch den Zirkelverweis versagt `ToString` der Beispielklasse, da es sich immer wieder selbst aufruft und dabei allen verfügbaren Speicher verbraucht

Viele Serialisierungsalgorithmen versagen ebenfalls durch eine solche Konstellation der Datenklassen. Dass die `DeepCopy`-Methode jedoch funktioniert und der `.NET`-Serializer keine Fehlermeldung auslöst, können Sie ganz einfach testen, indem Sie alle `Console.WriteLine`-Befehle auskommentieren. Das Programm passiert nach einem erneuten Start die Zeile

```
Dim locAdrKopie As Adresse = CType(ADObjectCloner.DeepCopy(locErsteAdr), Adresse)
```

anstandslos, beendet anschließend ordnungsgemäß und beweist so, dass der Serialisierungsalgorithmus des Frameworks über Zirkelverweise erhaben ist.

Serialisierung von Objekten unterschiedlicher Versionen beim Einsatz von BinaryFormatter oder SoapFormatter-Klassen

Einer sehr wichtigen Sache sollten Sie sich bewusst sein: Wenn Sie zur Serialisierung BinaryFormatter oder SoapFormatter verwenden, um Objekte in Dateien zu serialisieren, dann können Sie bei einem Update eines Programms leicht Gefahr laufen, in Versionskonflikte zu geraten.

Angenommen, Sie verfügen über eine Datenebene in Ihrer Applikation, die durch ein Objekt repräsentiert wird. Dieses Objekt kapselt alle Daten Ihrer Anwendung. Da Sie dem Anwender natürlich die Möglichkeit geben müssen, die Daten als Datei auf einem Datenträger zu speichern, nutzen Sie dafür die Serialisierungsfunktionen von .NET.

Nun erweitern Sie durch ein Update Ihr Programm, und es kommen einige Eigenschaften zur Datenklasse hinzu. Da sich die Deserialisierungsfunktionen beider Serialisierungsklassen am Objekt orientieren, erwartet es die Daten zu einigen Eigenschaften, die es im alten Objektmodell natürlich nicht gegeben hat. Der Deserialisierungsversuch schlägt in diesem Fall mit einer Ausnahme fehl.

Eine Möglichkeit, das zu verhindern, besteht durch den Einsatz der so genannten *SerializationBinder*-Klasse, die für die Lösung des Versionsproblems konzipiert wurde. Mehr über dieses Objekt erfahren Sie in der Visual Studio-Online-Hilfe.

Eine andere Möglichkeit, das Versionsproblem in den Griff zu bekommen, ist die Anwendung der XML-Serialisierung, die im folgenden Abschnitt besprochen wird.

XML-Serialisierung

Die XML-Serialisierung erfolgt generell nach dem gleichen Prinzip, wie Sie es beim Einsatz von BinaryFormatter und SoapFormatter kennen gelernt haben. Die XML-Serialisierung hat im Gegensatz dazu entscheidende Vorteile:

- Das Problem mit der Serialisierung von Klassen unterschiedlicher Versionen betrifft sie nicht, denn es werden nur die Daten deserialisiert, die einerseits vorhanden sind, andererseits auch in den zu deserialisierenden Klassen zur Verfügung stehen.
- XML-Code ist nicht nur leichter lesbar, sondern kann auch von den verschiedensten Programmen importiert und weiterverarbeitet werden.

Allerdings müssen Sie beim Erstellen von Klassen, die Sie im XML-Format serialisieren wollen, auch einige Kompromisse eingehen:

- Wie bei anderen Klassen, die Sie mit .NET-Serialisierungstechniken serialisieren, müssen die Klassen mit dem `Serializable`-Attribut gekennzeichnet werden.
- Eine Klasse, die im XML-Format serialisiert werden soll, muss über einen parameterlosen Konstruktor (`Sub New()`) verfügen.
- Nur Member-Variablen bzw. Eigenschaften, die öffentlichen Zugriff haben, können serialisiert werden.

Wenn Ihre Klassen diese Voraussetzungen erfüllen, steht einer Serialisierung nichts mehr im Weg.

BEGLEITDATEIEN: Sie finden das folgende Beispielprojekt im Verzeichnis `.\VB 2005 - Entwicklerbuch\E – Datentypen\Kap22\XMLSerialization\` unter dem Projektnamen `Adresso.sln`.

Dieses Beispielprojekt entspricht im Großen und Ganzen dem letzten Beispiel aus ► Kapitel 20. Es erlaubt das Generieren von Zufallsadressen und deren Darstellung in einer Liste. Im Gegensatz zum Beispiel aus Kapitel 20 verfügt es jedoch über eine richtige Menüstruktur und in dieser Ausbaustufe über ein paar zusätzliche Funktionen, die über dieses Menü erreichbar sind:

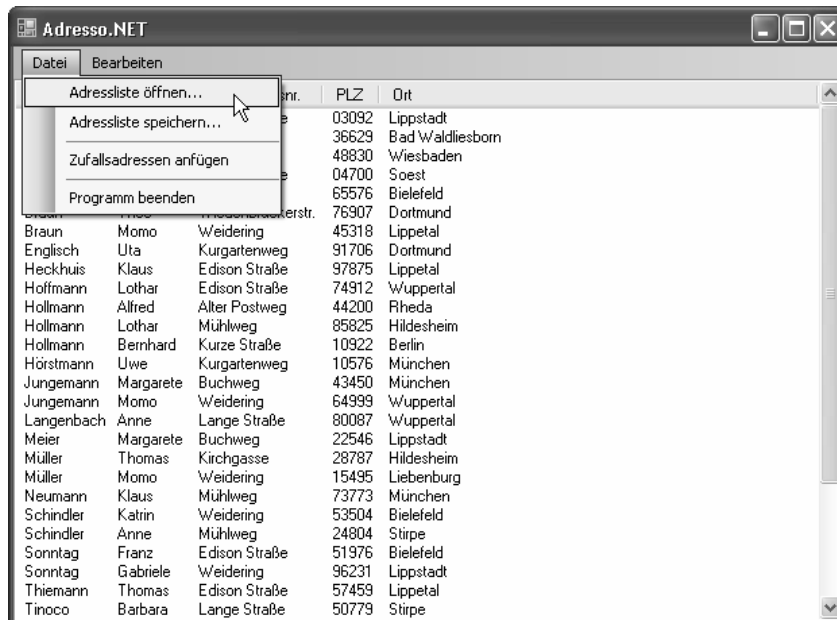


Abbildung 22.6: Die Adressenverwaltung »Adresso.Net« erlaubt es in dieser Ausbaustufe Zufallsadressen zu generieren und erstellte Listen in Dateien abzuspeichern und zu laden

Sie können eine beliebige Anzahl zufälliger Adressen erstellen, indem Sie den Menüpunkt *Zufallsadressen anfügen* wiederholt aufrufen. Mit *Adressliste speichern* erstellen Sie die so generierte Adressliste als Datei im XML-Format. Über den umgekehrten Weg können Sie eine Liste, die als XML-Datei vorliegt, über den Menüpunkt *Adressliste öffnen* in die entsprechenden Adresse-Objekte umwandeln, die dann wieder im `Listview`-Steuerelement des Programms angezeigt werden.

Neben der Darstellung der Liste und der Auswertung der Menüereignisse, die weitestgehend dem Beispiel aus ► Kapitel 20 entsprechen, und auf die ich aus diesem Grund an dieser Stelle nicht nochmals eingehen möchte, besteht der Code dieses Beispiels aus zwei zentralen Klassen.

Die Klasse `Adresse` kennen Sie aus vorherigen Beispielen bereits, und ihre Codebesonderheiten sind schnell erklärt:

```
<Serializable(>
```

```
Public Class Adresse
```

```
    'Membervariablen, die die Daten halten:
```

```
    Protected myMatchcode As String
```

```

Protected myName As String
Protected myVorname As String
Protected myStraße As String
Protected myPLZ As String
Protected myOrt As String

''' <summary>
''' Erstellt eine neue Instanz dieser Klasse.
''' </summary>
''' <remarks>Ein parameterloser Konstruktor wird benötigt,
''' um XML-Serialisierung zu ermöglichen.</remarks>
Sub New()
    MyBase.New()
End Sub

'Konstruktor - legt eine neue Instanz an
Sub New(ByVal Matchcode As String, ByVal Name As String, ByVal Vorname As String, _
    ByVal Straße As String, ByVal Plz As String, ByVal Ort As String)
    myMatchcode = Matchcode
    myName = Name
    myVorname = Vorname
    myStraße = Straße
    myPLZ = Plz
    myOrt = Ort
End Sub

Public Overridable Property Matchcode() As String
    Get
        Return myMatchcode
    End Get
    Set(ByVal Value As String)
        myMatchcode = Value
    End Set
End Property

.
. 'Der Code der anderen Eigenschaften wurde aus Platzgründen ausgelassen.
.
End Class

```

Drei Dinge sind wichtig, damit die XML-Serialisierung mit Auflistungen von Elementen dieses Typs funktionieren:

- Das Attribut `Serializable` muss auf die Klasse angewendet werden (siehe oberste, fett hervorgehobene Listingzeile).
- Die Klasse muss über einen parameterlosen Konstruktor verfügen (darunter, ebenfalls in Fettschrift).
- Alle Eigenschaften, die serialisiert werden sollen, müssen öffentlich zugänglich sein. Im Beispiellisting ist die `Matchcode`-Eigenschaft (die im Übrigen in der Formularliste nicht dargestellt wird) deswegen auch mit dem `Public`-Modifizierer ausgestattet.

Gespeichert werden die einzelnen Adresse-Elemente in einer Auflistung namens Adressen, die aus der generischen Klasse List(Of Adresse) abgeleitet wird (mehr zur generischen Auflistung erfahren Sie ebenfalls in ► Kapitel 20). Diese Klasse sieht folgendermaßen aus:

```

<Serializable(> _
Public Class Adressen
    Inherits List(Of Adresse)

    ''' <summary>
    ''' Erstellt eine neue Instanz dieser Klasse.
    ''' </summary>
    ''' <remarks></remarks>
    Sub New()
        MyBase.New()
    End Sub

    ''' <summary>
    ''' Erstellt eine neue Instanz dieser Klasse
    ''' und ermöglicht das Übernehmen einer vorhandenen Auflistung in diese.
    ''' </summary>
    ''' <param name="collection">Die generische Adresse-Auflistung, deren Elemente
    ''' in diese Auflistungsinstanzen übernommen werden sollen.</param>
    ''' <remarks></remarks>
    Sub New(ByVal collection As ICollection(Of Adresse))
        MyBase.New(collection)
    End Sub

    ''' <summary>
    ''' Serialisiert alle Elemente dieser Auflistung in eine XML-Datei.
    ''' </summary>
    ''' <param name="Dateiname">Der Dateiname der XML-Datei.</param>
    ''' <remarks></remarks>
    Sub XMLSerialize(ByVal Dateiname As String)
        Dim locXmlWriter As New XmlSerializer(GetType(Adressen))
        Dim locXmlDatei As New StreamWriter(Dateiname)
        locXmlWriter.Serialize(locXmlDatei, Me)
        locXmlDatei.Flush()
        locXmlDatei.Close()
    End Sub

    ''' <summary>
    ''' Generiert aus einer XML-Datei eine neue Instanz dieser Auflistungsklasse.
    ''' </summary>
    ''' <param name="Dateiname">Name der XML-Datei, aus der die Daten für
    ''' diese Auflistungsinstanzen entnommen werden sollen.</param>
    ''' <returns></returns>
    ''' <remarks></remarks>
    Public Shared Function XmlDeserialize(ByVal Dateiname As String) As Adressen
        Dim locXmlLeser As New XmlSerializer(GetType(Adressen))
        Dim locXmlDatei As New StreamReader(Dateiname)
        Return CType(locXmlLeser.Deserialize(locXmlDatei), Adressen)
    End Function

```

```

''' <summary>
''' Liefert eine Instanz dieser Klasse mit Zufallsadressen zurück.
''' </summary>
''' <param name="Anzahl">Anzahl der Zufallsadressen, die erzeugt werden sollen.</param>
''' <returns></returns>
''' <remarks></remarks>
Public Shared Function ZufallsAdressen(ByVal Anzahl As Integer) As List(Of Adresse)
.
. ' Ausgelassener Code; unwichtig an dieser Stelle
.
End Function
End Class

```

Auch diese Klasse, die die Adresse-Elemente speichert, erfüllt die Voraussetzungen für die XML-Serialisierung – Sie verfügt über das notwendige Attribut, über einen parameterlosen Konstruktor und über öffentliche Eigenschaften. Die einzige öffentliche Eigenschaft, auf die es bei dieser Klasse ankommt, ist übrigens im Code nicht zu erkennen: Es ist Items, die durch die Basisklasse List(Of Adresse) definiert wurde und per Vererbung natürlich auch in der neuen Adressen-Klasse zur Verfügung steht. Sie erlaubt dem XML-Serialisierer Zugriff auf die einzelnen Adresse-Elemente und damit letzten Endes auf jedes einzelne Datenfeld von Adresse.

Beim ersten Blick auf den Code dieser Klasse mag es vielleicht verwirren, dass die Methode zum Serialisieren des Klasseninhalts eine nicht statische, die zum Deserialisieren hingegen eine statische ist. Doch das ist klar: Beim Deserialisieren gibt es noch keine Instanz der Adressen-Klasse; also gibt es auch keine Member-Methode die aufgerufen werden könnte, denn die Instanz mit den Adresse-Elementen geht ja erst durch den Deserialisierungsprozess hervor!

Anders ist es beim Serialisieren: Hier sind die einzelnen Elemente bereit in der Instanz von Adressen vorhanden, und aus diesem Grund kann es auch eine Member-Methode sein, die das Serialisieren vornimmt; sie greift auf ihre eigenen Elemente zu.

Das Anwenden dieser beiden Methoden ist schließlich eine Kleinigkeit. Die beiden folgenden Ereignisbehandlungsmethoden, die das Auswählen der entsprechenden Menüeinträge auswerten, sind dafür verantwortlich (relevante Codeteile sind wieder fett hervorgehoben):

```

'Wird aufgerufen, wenn Anwender Datei/Adresslist öffnen wählt.
Private Sub tsmAdresslisteLaden_Click(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles tsmAdresslisteLaden.Click

    Dim dateiÖffnenDialog As New OpenFileDialog
    With dateiÖffnenDialog
        .CheckFileExists = True
        .CheckPathExists = True
        .DefaultExt = "*.xml"
        .Filter = "Adresso XML-Dateien" & _
            " (" & "*.adrxml" & ")|" & "*.adrxml" & "|Alle Dateien (*.*)|*.*"
    End With
    Dim dialogErgebnis As DialogResult = .ShowDialog
    If dialogErgebnis = Windows.Forms.DialogResult.Cancel Then
        Exit Sub
    End If

```

```

        'Adressen deserialisieren
        myAdressen = Adressen.XmlDeserialize(.FileName)

        'Liste neu darstellen
        ElementeDarstellen()
    End With
End Sub

'Wird aufgerufen, wenn Anwender Datei/Adressliste speichern wählt.
Private Sub tsmAdresslisteSpeichern_Click(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles tsmAdresslisteSpeichern.Click

    Dim dateiSpeichernDialog As New SaveFileDialog

    With dateiSpeichernDialog
        .CheckPathExists = True
        .DefaultExt = "*.xml"
        .Filter = "Adresso XML-Dateien" & " (" & "*.adrxml" & ")|" & _
            "*.adrxml" & "|Alle Dateien (*.*)|*.*"
        Dim dialogErgebnis As DialogResult = .ShowDialog
        If dialogErgebnis = Windows.Forms.DialogResult.Cancel Then
            Exit Sub
        End If
    End With

    'Adressen serialisieren
    myAdressen.XmlSerialize(.FileName)
End Sub
End Class

```

Prüfen der Versionsunabhängigkeit der XML-Datei

Die XML-Datei, die dieses Programm generiert, sieht auszugsweise etwa wie in Abbildung 22.7 aus, und Sie können sich »Ihre« Version mit dem Notepad anschauen.

Wenn Sie ein Datenfeld aus dieser XML-Datei entfernen, etwa wie in Abbildung 22.7 zu sehen, und anschließend abspeichern, können Sie diese XML-Datei dennoch deserialisieren lassen – für die Ort-Eigenschaft des betroffenen Adresse-Objektes wird dann beim Deserialisieren eben keine Zuordnung vorgenommen. Der Deserialisierungsprozess schlägt aber nicht fehl.

Abbildung 22.7: Auch wenn Sie ein Datenfeld aus der XML-Datei entfernen, lässt sich die Datei anschließend noch deserialisieren

Genau so könnten Sie der Adresse-Klasse eine zusätzliche Eigenschaft hinzufügen. »Alte« Versionen der Serialisierungsdateien ließen sich dennoch weiterverwenden.

TIPP: Wenn Sie XML-Serialisierungslogik in eigenen Klassen implementieren wollen, nutzen Sie am besten die Codeausschnittsbibliothek (über die Sie in ► Kapitel 4 mehr erfahren). Sie finden die entsprechenden Codeausschnitte unter *XML/Klassendaten aus einer XML-Datei lesen* sowie *XML/Klassendatei in eine XML-Datei schreiben*.

Serialisierungsfehler bei KeyedCollection

Die generische KeyedCollection-Klasse eignet sich als Auflistungsbasis für Geschäftsobjekte ideal, denn:

- Sie stellt, da sie generisch ist, eine typsichere Auflistung des jeweiligen Elementes dar.
- Objekte lassen sich wahlweise über einen Index *oder* über einen Schlüssel abrufen. Den letzteren muss das Objekt allerdings selber zur Verfügung stellen – ► Kapitel 20 liefert im entsprechenden Abschnitt mehr zu diesem Thema.
- Da die KeyedCollection auch über einen Indexer verfügt, stellt sie auch eine Enumerierungstechnik für For/Each zur Verfügung, obwohl Sie zusätzlich Wörterbuchfunktionalität implementiert.

An Flexibilität ist diese Klasse also in Sachen Auflistung kaum zu übertreffen.

Allerdings ist diese Kombination auch gerade dann eine Crux, wenn sich Schlüsseltyp und Indexer typtechnisch ins Gehege kommen, denn:

Stellen Sie sich vor, Sie haben eine Klasse wie die folgende, die den eindeutigen Schlüssel eines Geschäftsobjektes über eine ID zur Verfügung stellt (und dieser Anwendungsfall ist keinesfalls konstruiert, denn gerade in der Datenbankprogrammierung haben Tabellen oft den Datentyp Integer als primären Schlüssel und Hauptindex):

BEGLEITDATEIEN: Das Projekt zu diesem Beispiel finden Sie im Verzeichnis `.\VB 2005 - Entwicklerbuch\E - Datentypen\Kap22\KeyedCollection\`.

```
Public Class Adresse

    Private myIDAdresse As Integer
    Private myNachname As String
    Private myVorname As String

    Sub New(ByVal IDAdresse As Integer, ByVal Nachname As String, ByVal Vorname As String)
        myIDAdresse = IDAdresse
        myNachname = Nachname
        myVorname = Vorname
    End Sub

    Public Property IDAdresse() As Integer
        Get
            Return myIDAdresse
        End Get
        Set(ByVal value As Integer)
            myIDAdresse = value
        End Set
    End Property

    Public Property Nachname() As String
        Get
            Return myNachname
        End Get
        Set(ByVal value As String)
            myNachname = value
        End Set
    End Property

    Public Property Vorname() As String
        Get
            Return myVorname
        End Get
        Set(ByVal value As String)
            myVorname = value
        End Set
    End Property
End Class
```

Dieses Geschäftsobjekt speichert also eine simple Adresse – wesentliche Eigenschaften sind hier aus Platzgründen ausgelassen.

Was viel wichtiger ist: Ein Geschäftsobjekt dieses Typs identifiziert sich eindeutig durch seine IDAdresse-Eigenschaft, und das Problem dabei ist: Diese ist vom Typ Integer. Warum das ein Problem darstellt, zeigt die Implementierung einer KeyedCollection-Auflistung auf Basis dieser Klasse, denn es bietet sich mehr als an, dass IDAdresse auch als Quelle für die Schaffung des eindeutigen Schlüssels verwendet wird:

```
Public Class Adressen
    Inherits System.Collections.ObjectModel.KeyedCollection(Of Integer, Adresse)

    Protected Overrides Function GetKeyForItem(ByVal item As Adresse) As Integer
        Return item.IDAdresse
    End Function
End Class
```

Während der Aufbau einer KeyedCollection in dieser Struktur noch ohne Probleme möglich ist,

```
Protected Overrides Sub OnLoad(ByVal e As System.EventArgs)
    MyBase.OnLoad(e)
    myAdressen.Add(New Adresse(2, "Löffelmann", "Klaus"))
    myAdressen.Add(New Adresse(4, "Heckhuis", "Jürgen"))
    myAdressen.Add(New Adresse(6, "Sonntag", "Miriam"))
    myAdressen.Add(New Adresse(8, "Heckhuis", "Jürgen"))
    myAdressen.Add(New Adresse(10, "Vielstedde", "Anja"))
End Sub
```

zeigt sich beim Abrufen der Elemente bereits eine Besonderheit, wenn es sich beim Schlüssel um den Datentyp Integer handelt:

```
Private Sub btnObjekteAusgeben_Click(ByVal sender As Object, ByVal e As EventArgs)
    For count As Integer = 0 To 4
        debug.Print(myAdressen(1))
    Next
End Sub
```

Item (key As Integer) As KeyedCollection.Adresse
key:
Der Schlüssel des abzurufenden Elements.

Abbildung 22.8: Handelt es sich um einen Integer-Schlüssel, bietet die KeyedCollection-Auflistung nur eine Signatur zum Abrufen der Elemente an

Soweit macht das auch Sinn. Da Indexer und Schlüssel in diesem Fall vom gleichen Datentyp sind, könnten weder Compiler noch Framework unterscheiden, welche »Version« der Abruffunktion sie verwenden sollen – die durch den Indexer oder die über den Schlüssel. Einzig und allein die Methodensignatur (also welche Parametertypen übergeben werden) entscheidet nämlich darüber, und da es in diesem Fall die gleichen Typen sind, kann keine Signaturenunterscheidung stattfinden. Ergo: Es gibt nur eine Möglichkeit, an das Element heranzukommen.

Anders wäre das, wenn unser Schlüssel beispielsweise vom Datentyp Long wäre.² In diesem Fall würde der Indexer über einen Integer, der Schlüssel über den Datentyp Long angesprochen werden, und dementsprechend würde Ihnen schon IntelliSense zwei Möglichkeiten zum Abrufen der Elemente anbieten:

² Lesen Sie aber bitte diesen Abschnitt zunächst zu Ende, bevor Sie jetzt schon alle Ihre KeyedCollection-basierenden Auflistungsklassen auf den Datentyp Long als Schlüssel umstellen!

```

Private Sub btnObjekteAusgeben_Click(ByVal sender As
  For count As Integer = 0 To 4
    debug.Print(myAdressen(
  Next
End Sub

```

▲ 1 von 2 ▼ Item (**key As Long**) As KeyedCollection.Adresse
key: Der Schlüssel des abzurufenden Elements.

▲ 2 von 2 ▼ Item (**index As Integer**) As KeyedCollection.Adresse
index: Der nullbasierte Index des Elements, das abgerufen oder festgelegt werden soll.

Abbildung 22.9: Unterscheiden sich Schlüssel und Indexer beim Datentyp, gibt es bei der *KeyedCollection* tatsächlich zwei Möglichkeiten, an ein einzelnes Element heranzukommen

Diese Integer/Integer-Problematik wäre noch nicht weiter tragisch, denn durch den von *KeyedCollection* implementierten Enumerator gäbe es immer noch die Möglichkeit, die Objekte einerseits per Schlüssel, andererseits nacheinander abzurufen. Für letztere Methode würden Sie einfach eine *For/Each*-Schleife verwenden.

Richtig problematisch wird es aber, wenn Sie versuchen, die Daten zu serialisieren. Denn dann passiert Folgendes:

```

Private Sub btnObjekteSerialisieren_Click(ByVal sender As System.Object, ByVal e As Sy
  Dim writer As New XmlSerializer(GetType(Adressen))
  Dim file As New StreamWriter("c:\testdaten_xml.xml")
  writer.Serialize(file, myAdressen)
  file.Close()
End Sub

```

InvalidOperationException wurde nicht behandelt.

Beim Generieren des XML-Dokuments ist ein Fehler aufgetreten.

Hinweise zur Fehlerbehebung:

Weitere Informationen finden Sie in der *InnerException*-Eigenschaft.
 Für diese Ausnahme die allgemeine Hilfe aufrufen.
 Für die interne Ausnahme die allgemeine Hilfe aufrufen.

Aktionen:

Details anzeigen
 Ausnahme...

Details anzeigen

Ausnahme-Snapshot:

| | |
|--|---|
| System.InvalidOperationException | { "Beim Generieren des XML-Dokuments ist ein Fehler aufgetreten." } |
| Data | { System.Collections.ListDictionaryInternal } |
| HelpLink | Nothing |
| InnerException | { "Der angegebene Schlüssel war nicht im Wörterbuch angegeben." } |
| [System.Collections.Generic.KeyNotF... | { "Der angegebene Schlüssel war nicht im Wörterbuch angegeben." } |
| Data | { System.Collections.ListDictionaryInternal } |
| HelpLink | Nothing |
| InnerException | Nothing |
| Message | "Der angegebene Schlüssel war nicht im Wörterbuch angegeben." |
| Source | "mscorlib" |
| StackTrace | " bei System.ThrowHelper.ThrowKeyNotFoundException() bei System.Co (System.Reflection.RuntimeMethodInfo) |
| TargetSite | |

OK

```

Private Sub btnObjekt
  Dim reader As New
  Dim file As New S
  Dim fileData As A
  fileData = CType(
End Sub
Class
Protected Overrides Functio
  Return item.IDAdresse
End Function
Class
Public Class Adresse
  Private myIDAdresse As Inte

```

Abbildung 22.10: Beim Serialisieren stellen sich massive Probleme ein, wenn Schlüsseltyp und Indexer gleichermaßen vom Typ *Integer* sind!

Der Serialisierungsalgorithmus kommt hier offensichtlich nicht mehr mit der Doppelbelegung der Datentypen zurecht. Offensichtlich versucht er über eine Zählvariable der Reihe nach auf die Objekte zuzugreifen, die aber der Indexer von *KeyedCollection* fälschlicherweise als Schlüssel interpretiert. Da in unserem Beispiel die IDs nicht der Nummerierungsreihenfolge der Objekte entsprechen, schlägt dieser Versuch fehl und das Programm steigt mit einer Ausnahme, wie in der Abbildung zu sehen, aus.

Workaround

Abhilfe können Sie bei diesem Szenario auf folgende Weise schaffen:

Sie schreiben sich eine Helfer-Klasse, die nur zur Signaturenunterscheidung dient, die aber nichts weiter macht, als den Integer-Datentyp zu kapseln. Diese könnte beispielsweise folgendermaßen ausschaun:

```
<Serializable(> _
Public Class IntKey

    Private myKey As Integer

    'Wichtig für die XML-Serialisierung: Parameterloser Konstruktor!
    Sub New()
        MyBase.new()
    End Sub

    Sub New(ByVal Key As Integer)
        myKey = Key
    End Sub

    Public ReadOnly Property Key() As Integer
        Get
            Return myKey
        End Get
    End Property
End Class
```

HINWEIS: Auf den ersten Blick könnte man meinen, dass auch die Verwendung des Long-Datentyps als Schlüssel ausreichend sein müsste, um das Problem in den Griff zu bekommen. Ich persönlich wäre dabei aber vorsichtig, da durch die implizite Konvertierungsmöglichkeit von Integer in Long unter Umständen dieselben Effekte zu Tage treten könnten. Mit dem hier vorgestellten Workaround funktioniert es jedoch einwandfrei.

Die geänderte Eigenschaft in der Adresse-Klasse sieht nun wie folgt aus:

```
Public Class Adresse

    Private myIDAdresse As IntKey
    Private myNachname As String
    Private myVorname As String

    'Den brauchen wir fürs Deserialisieren!
    Sub New()
        MyBase.new()
    End Sub

    Sub New(ByVal IDAdresse As IntKey, ByVal Nachname As String, ByVal Vorname As String)
        myIDAdresse = IDAdresse
        myNachname = Nachname
        myVorname = Vorname
    End Sub
```

```

Public Property IDAdresse() As IntKey
    Get
        Return myIDAdresse
    End Get
    Set(ByVal value As IntKey)
        myIDAdresse = value
    End Set
End Property

```

Und dementsprechend hat sich auch die KeyedCollection geändert:

```

Public Class Adressen
    Inherits System.Collections.ObjectModel.KeyedCollection(Of IntKey, Adresse)

    Protected Overrides Function GetKeyForItem(ByVal item As Adresse) As IntKey
        Return item.IDAdresse
    End Function
End Class

```

Das Hinzufügen von Elementen zur Auflistung bereitet zwar ein kleines bisschen mehr Aufwand:

```

Protected Overrides Sub OnLoad(ByVal e As System.EventArgs)
    MyBase.OnLoad(e)
    myAdressen.Add(New Adresse(New IntKey(2), "Löffelmann", "Klaus"))
    myAdressen.Add(New Adresse(New IntKey(4), "Heckhuis", "Jürgen"))
    myAdressen.Add(New Adresse(New IntKey(6), "Sonntag", "Miriam"))
    myAdressen.Add(New Adresse(New IntKey(8), "Heckhuis", "Jürgen"))
    myAdressen.Add(New Adresse(New IntKey(10), "Vielstedde", "Anja"))
End Sub

```

Aber dafür können Sie die einzelnen Elemente nunmehr zielsicher und ohne Zweideutigkeiten sowohl über den Indexer (siehe Grafik) *als auch* über den Schlüssel abrufen:

```

Private Sub btnObjekteAusgeben_Click(ByVal sender As System.Object,
    For count As Integer = 0 To 4
        Debug.Print(myAdressen(count).Nachname & ", " & myAdressen(
    Next
End Sub

```

2 von 2 Item (index As Integer) As KeyedCollection.Adresse
 index: Der nullbasierte Index des Elements, das abgerufen oder festgelegt werden soll.

Abbildung 22.11: Mit der Workaround-Lösung funktioniert nun der Abruf über Schlüssel und Indexer perfekt

BEISPIELDATEIEN: Eine Version des Programm, die dieses Verfahren nutzt, finden Sie im Verzeichnis `.\VB 2005 - Entwicklerbuch\E - Datentypen\Kap22\KeyedCollection2`, und Sie werden sehen, dass es auch beim Serialisieren und Deserialisieren einwandfrei funktioniert.
