الباب الرابع

تطوير أصناف التعامل مع البيانات

DEVELOPING DATA ACCESS CLASSES

- الفصل الثالث عشر: مقدمة عن أصناف التعامل مع البيانات والكائنات المستمرة
- الفصل الرابع عشر: تطوير تطبيقات قواعد بيانات أكثر تعقيداً

الفصل الثالث عشر

مقدمة عن أصناف التعامل مع البيانات والكائنات المستمرة

INTRODUCTION TO DATA ACCESS CLASSES AND PERSISTENCE

أهداف الفصل:

- التعرف على مدخلات لغة VB .NET ومخرجاتها.
- جعل الكائنات مستمرة.
- تصميم أصناف التعامل مع البيانات (Data Access Classes).
- التفاعل مع أصناف التعامل مع البيانات.
- استخدام قاعدة بيانات علاقية بواسطة برامج VB .NET.

لقد تعلمنا في الباب الثاني من هذا الكتاب كيفية تصميم أصناف مجال المشكلة (PD Classes) وتعريفها، كما تعلمنا في الباب الثالث كيف نصمم أصناف واجهة الاستخدام (GUI Classes) ونعرفها. وسوف نتعلم في هذا الباب كيف نصمم أصناف التعامل مع البيانات (DA Classes) ونعرفها والتي تقدم خدمات تخزين الكائنات واسترجاعها لكل من تطبيقات الويندوز وتطبيقات الويب. أما الباب الخامس فسوف يوضح لك كيف تربط الأنواع الثلاثة من الأصناف داخل نظام يعتمد على الكائنات (OO System).

تخزن معظم الشركات والمؤسسات بياناتها داخل قواعد البيانات علاقية (Relational Data Base). ولكن يجب أيضاً أن نعطي فكرة عن التعامل مع الملفات التي تقدم إمكانية تخزين بيانات داخل ملف (File Input) وإمكانية استرجاع هذه البيانات من الملف (File Output) وتختصر هذه العمليات بالرمز (I/O). مع العلم أن هذه الفقرة يشغل حيزاً كبيراً في أي لغة برمجة لأن التطبيقات الحقيقية تتطلب تخزين البيانات واسترجاعها. ولكي نشرح هذا الجزء بالتفصيل باستخدام لغة VB .NET نحتاج إلى كتاب منفصل. ولذلك سوف نتعرف في هذا الفصل فقط على أساسيات مدخلات البيانات ومخرجاتها من وإلى الملفات.

ويعد الانتهاء من هذا الفصل سوف تتمكن عزيزي القارئ من استخدام مدخلات لغة VB .NET ومخرجاتها لتخزين البيانات واسترجاعها – مشتملاً على قيم الكائنات وصفاتها داخل ملفات تعاقبية (Sequential Files) بالإضافة إلى تخزين الكائنات واسترجاعها باستخدام النشر التسلسلي (Serialization). سوف تتمكن أيضاً من تصميم أصناف التعامل مع البيانات (DA Classes) وتعريفها لكي تخزن كائنات أصناف مجال المشكلة وتسترجعها بأحد أسلوبين وهما: تخزين قيم صفات الكائنات، وتخزين الكائنات نفسها.. وأخيراً سوف تتمكن من تصميم قاعدة بيانات علاقية لتخزين قيم صفات كائنات أصناف مجال المشكلة واسترجاعها باستخدام أصناف التعامل مع البيانات.

التعرف على مدخلات لغة VB .NET ومخرجاتها

Examining VB .NET Input and Output

تستخدم لغة VB .NET مفهوم التدفق (Stream) لتقديم كل من خدمة الإدخال (Input) والإخراج (Output) من وإلى الملفات. ويعني هذا المفهوم ببساطة انتقال مجموعة من البايتات من وإلى الملف. مع العلم أننا سوف نركز في الأمثلة الأولى القليلة من هذا الفصل على التعامل مع الملفات التعاقبية، وذلك من خلال كل من الصنف StreamWriter والصنف StreamReader المشتقان من الأصناف الأساسية: الصنف TextWriter والصنف TextReader على التوالي. يتواجد كل من الصنف StreamWriter والصنف StreamReader في الفضاء المسمى "System.IO". ولذلك يجب أن تستدعي الأمر Imports لجلب هذا الفضاء داخل تطبيقات معالجة الملفات التعاقبية.

يستخدم المصطلح "file" (ملف) والمصطلح "Record" (سجل) والمصطلح "field" (حقل) في تطبيقات معالجة الملفات مثل معالجة ملف يحتوي على سجلات مراكب كما يلي:

- يمثل الملف مجموعة من البيانات، وبمعنى آخر يتكون الملف من مجموعة من السجلات.
- نستطيع أن ننظر للسجل على أنه صف (Row) في ملف الذي يتكون من مجموعة من الصفوف؛ ولذلك يتم تخزين بيانات الكائن الواحد (مثل بيانات مركب) داخل صف.
- يتكون السجل الواحد من مجموعة من الحقول التي تقابل صفات الكائن (مثل الصفة stRegNo والصفة length).

يوضح الجدول رقم (١٣.١) ملفاً تعاقبياً يخزن بيانات المراكب، حيث تمثل الصفوف سجلات الملف وتمثل الأعمدة حقول الملف.

الجدول رقم (١٣,١). ملف تعاقبي يخزن بيانات المراكب.

State Reg No	Boat Length	Manufacturer	Model Year	Cust Phone No
MO12345	26	Ranger	1976	765-4321
MO223344	24	Tracker	1996	467-1234
MO34561	35	Tartan	1998	123-4567
...	...	...	...	...

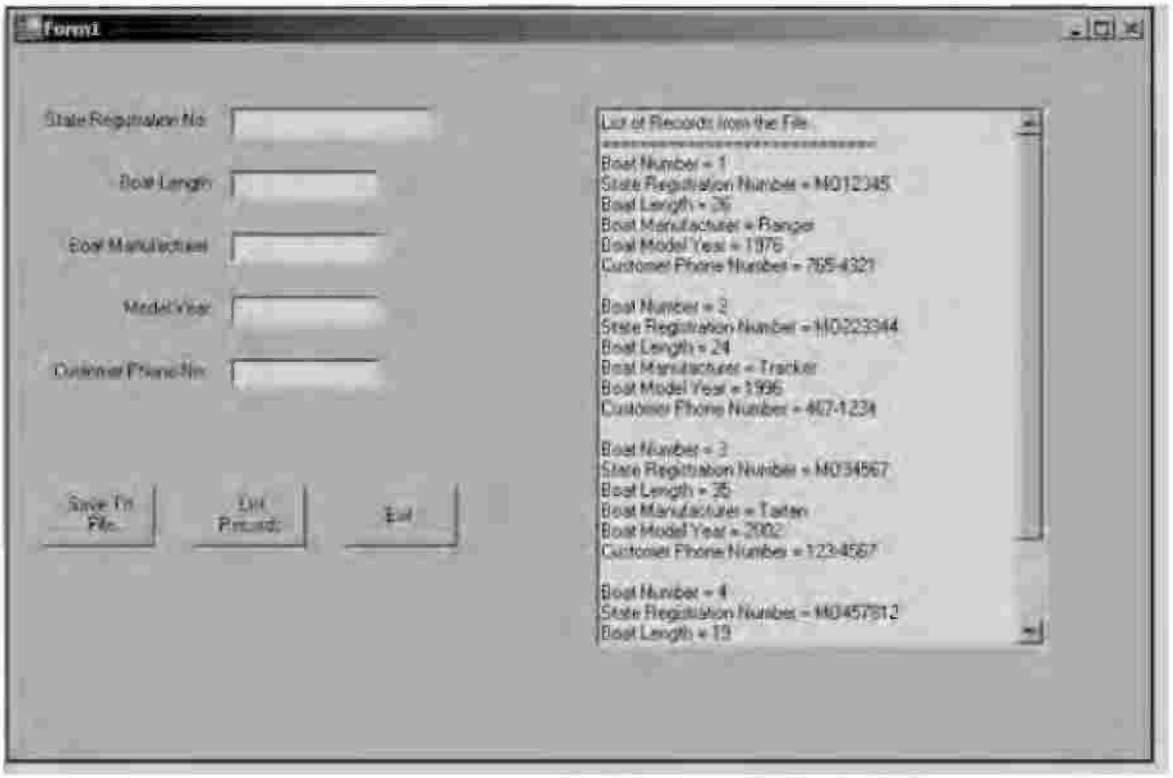
إن طريقة قراءة وكتابة سجل ثم سجل كما هو واضح في سجلات المراكب تستخدم مفهوم التدفق (Stream) ولكن فقط نقرأ أو نكتب مجموعة من البايتات من وإلى الملف في كل مرة التعامل الواحدة مع الملف. يوضح المثال الأول من هذا الفصل تطبيق معالجة ملفات تعاقبية باستخدام الصنف StreamWriter والصنف StreamReader حيث يتم استخدام مفهوم التدفق المعرف داخل لغة VB .NET مع عناصر واجهة استخدام مناسبة لتطوير هذا التطبيق.

معالجة الملفات التعاقبية
Sequential File Processing

يستخدم هذا المثال، كما هو واضح في الشكل رقم (١٣,١)، خمسة صناديق نصية لاستقبال بيانات مركب ثم تخزينها داخل ملف تعاقبي وذلك عند الضغط على الزر "Save to File". وإذا ضغط المستخدم على الزر "List Records" سيتم استرجاع سجلات المراكب من الملف التعاقبي ثم عرضها داخل صندوق النص المتعدد السطور الذي يحتوي على عمود تصفح رأسي (Vertical Scroll Bar) لتصفح محتوياته. أما الزر الثالث "Exit" فيستخدم لإنهاء البرنامج.

الصيغة العامة لاستخدام الصنف StreamWriter والصنف StreamReader
General Format for StreamWriter and StreamReader

سوف نتعرض في هذا المثال على عدد قليل من إجراءات الصنف StreamWriter (الإجراء Write() والإجراء WriteLine() والإجراء Close())، كما سنتعرض لعدد آخر من إجراءات الصنف StreamReader (الإجراء Read() والإجراء ReadLine() والإجراء ReadToEnd() والإجراء Peek() والإجراء Close()). يوضح الشكل رقم (١٣,٢) كلاً من الصيغة العامة ومثالاً لتعريف كائنات من كل من الصنف StreamWriter والصنف StreamReader.



الشكل رقم (١٣,١). تصميم لنموذج مثال StreamWriter و StreamReader.

StreamWriter

Format

```
Dim objectName as New StreamWriter("Filename")
Dim objectName as New StreamWriter("Filename", BooleanAppend)
```

Examples

```
Dim swInventory as New StreamWriter("a:\boats.txt")
Dim swInventory as New StreamWriter("a:\boats.txt", True)
```

StreamReader

Format

```
Dim objectName as New StreamReader("Filename")
```

Examples

```
Dim srInventory as New StreamReader("a:\boats.txt")
```

الشكل رقم (١٣,٢). الصيغة العامة ومثال لتعريف StreamWriter و StreamReader.

يمثل الفهرس Bin داخل المشروع الحالي الموقع الافتراضي لمعامل اسم الملف (Filename) الموضح في الشكل رقم (١٣,٢)، ولكن بطبيعة الحال يمكنك أن تحدد المسار الكامل للملف إذا أردت ذلك. فعلى سبيل المثال ربما تريد

أن تحدد اسم ملف المثال الحالي وموقعه هكذا "C:\Chap13\boats.txt" والذي يعني أن الملف سوف يتم تخزينه داخل المجلد "Chap13" المتواجد داخل القرص الصلب "C"، أما المعامل BooleanAppend الموضح في إجراء إنشاء الصيغة العامة الثانية فإنه يستخدم للسماح بإضافة البيانات إلى نهاية الملف وذلك عند إسناده بالقيمة True. عندما تنشئ كائناً من الصنف StreamWriter يتم فتح الملف إذا كان موجوداً في المسار المحدد. أما إذا لم يكون الملف موجوداً فيتم إنشاء ملف جديد ثم يتم فتحه؛ وذلك لأن الملف يجب أن يتم فتحه قبل التعامل معه. يمكنك تعريف كائن من الصنف StreamWriter في الجزء المخصص للتعريفات أو تعريفه داخل إجراء ما. يوضح الشكل رقم (١٣,٣) أوامر المثال الحالي لبرنامج الملف التعاقبي. لاحظ الأمر Imports في قمة أوامر البرنامج والذي يستخدم لجلب الفضاء المسمى System.IO:

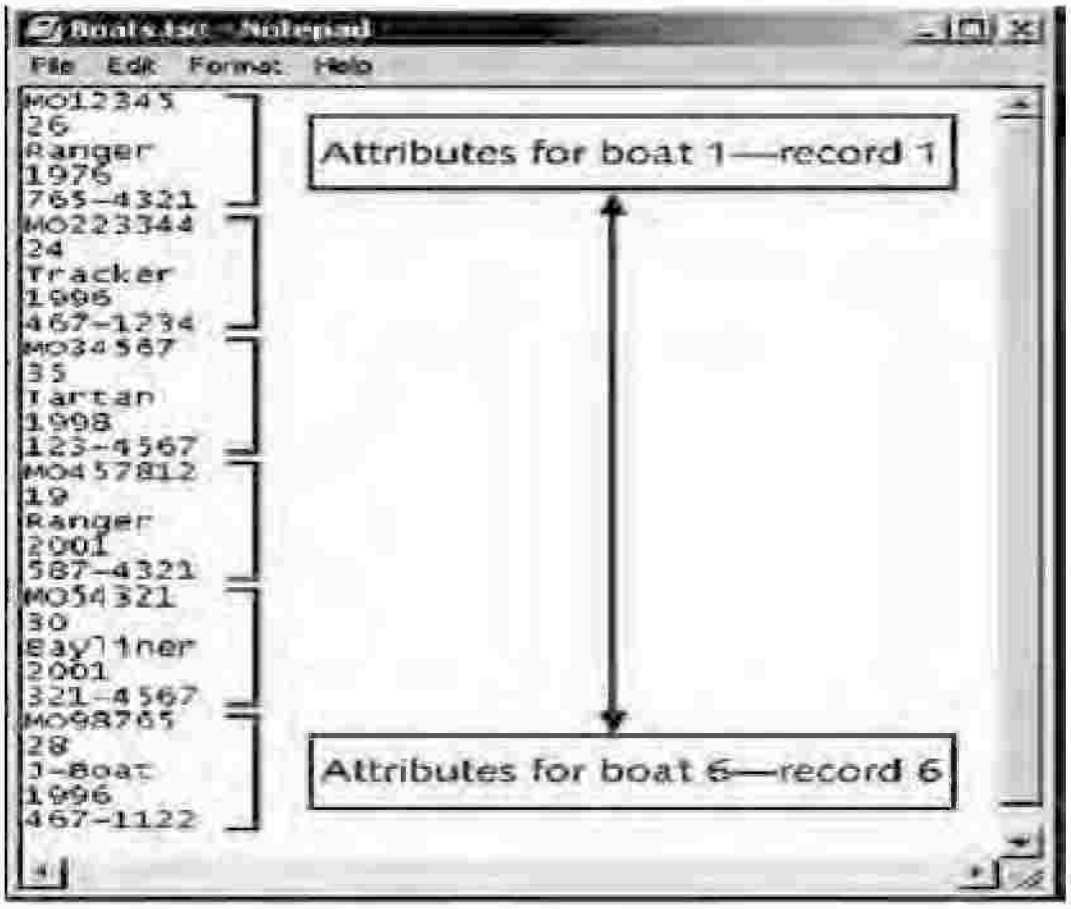
```
Imports System.IO
Public Class Form1
        Inherits System.Windows.Forms.Form
```

تبدأ شفرة إجراء معالجة الزر "Save to File" بتوصيف كائن swBoats من الصنف StreamWriter للكتابة داخل الملف a:\boats.txt. بفرض إلحاق بيانات جديدة في نهاية الملف أسندنا القيمة True للصفة Append، مع العلم أنه يمكنك فتح هذا الملف لاحقاً بعد إنشائه بأحد برامج تحرير النصوص آخر مثل برنامج المفكرة الذي ننصح باستخدامه. ثم تستخدم الأوامر التالية للكائن swBoats لكتابة بيانات مركب جديد (القيم الجديدة المدخلة في صناديق النص) إلى الملف باستخدام الإجراء WriteLine() بدلاً من الإجراء Write() حيث يتم كتابة القيمة التي يتم تمريرها إليها في سطر منفصل (أي إدراج علامة سطر جديد بعد كتابة القيمة الممررة إليها).

ثم تأتي أوامر مسح صناديق النص التي تصبح جاهزة لاستقبال بيانات مركب جديد ووضع المؤشر داخل صندوق النص "State Registration No"، ثم غلق الكائن swBoats باستدعاء الإجراء StreamWriter الذي ينتهي بكتابة وتحرير موارد النظام. وبعد الانتهاء من تخزين مجموعة سجلات إلى الملف يمكن أن تستخدم برنامج المفكرة لاستعراض هذه السجلات. يوضح الشكل رقم (١٣,٤) بعض السجلات المخزنة في الملف.

```
'SeqFile1--Boat Attributes

'Chapter 13 -- Example 1
Imports System.IO
Public Class Form1
    Inherits System.Windows.Forms.Form
Private Sub btnSave_Click(ByVal sender As System.Object, _
   ByVal e As System.EventArgs) Handles btnSave.Click

    ' Declare StreamWriter
   Dim swBoats As New StreamWriter("A:\Boats.txt", True)

    ' Save a boat record in the file
   swBoats.WriteLine(txtStRegNo.Text)
   swBoats.WriteLine(txtLength.Text)
   swBoats.WriteLine(txtManufacturer.Text)
   swBoats.WriteLine(txtYear.Text)
   swBoats.WriteLine(txtCustPhoneNo.Text)

    ' Clear text boxes and set focus to state registration no text box
   txtStRegNo.Clear()
   txtLength.Clear()
   txtManufacturer.Clear()
   txtYear.Clear()
   txtCustPhoneNo.Clear()
   txtStRegNo.Focus()

    ' Close the StreamWriter
   swBoats.Close()
End Sub
Private Sub btnListRecords_Click(ByVal sender As System.Object, _
   ByVal e As System.EventArgs) Handles btnListRecords.Click

   Dim boatRecord As New System.Text.StringBuilder()
   Dim stRegNo, manuf, custPhoneNo As String
   Dim length As Single
   Dim year, recNbr As Int32
   Dim myFile As String = "a:\Boats.txt"
   Try
      recNbr = 0
      boatRecord.Append("List of Records from the File" & vbCrLf)
      boatRecord.Append("============================" & vbCrLf)
      ' Check to see if file exists
      If File.Exists(myFile) And myFile.Length > 0 Then
         ' Declare the StreamReader and set record counter to 0
         Dim srBoats As StreamReader = New StreamReader(myFile)

         ' Loop over records until end of the file is reached
         Do Until srBoats.Peek = -1
            ' Read a record
            stRegNo = srBoats.ReadLine
            length = srBoats.ReadLine
            manuf = srBoats.ReadLine
            year = srBoats.ReadLine
            custPhoneNo = srBoats.ReadLine
            recNbr += 1

            ' Build line for display in multiline text box
            boatRecord.Append("Boat Number = " & _
               recNbr & vbCrLf)
            boatRecord.Append("State Registration Number = " & _
               stRegNo & vbCrLf)
            boatRecord.Append("Boat Length = " & _
               length & vbCrLf)
            boatRecord.Append("Boat Manufacturer = " & _
               manuf & vbCrLf)
            boatRecord.Append("Boat Model Year = " & _
               year & vbCrLf)
            boatRecord.Append("Customer Phone Number = " & _
               custPhoneNo & vbCrLf & vbCrLf)

         Loop
         srBoats.Close()
         boatRecord.Append("End of File")
         txtDisplay.Text = boatRecord.ToString

         txtStRegNo.Focus()
         Else
            MessageBox.Show("File does not exist, enter records")
      End If
   Catch ex As Exception
      MessageBox.Show(ex.ToString)
   End Try
End Sub

Private Sub btnExit_Click(ByVal sender As System.Object, _
   ByVal e As System.EventArgs) Handles btnExit.Click

   Me.Close()
End Sub
```

الشكل رقم (١٣,٣). أوامر مثال برنامج الملف التعاقبي.



الشكل رقم (١٣,٩). ملف خصائص المركب Boat.

تتجول شقرة إجراء الزر "List Records" داخل سطور الملف حيث تقرأ قيمة تلو الأخرى ثم تلحق هذه القيم إلى كائن من الصنف StringBuilder يسمى boatRecord من خلال متغيرات مؤقتة، حيث سيتم إسناد هذا الكائن إلى الصفة Text لصندوق النص متعدد السطور (txtDisplay) لعرض جميع سجلات الملف. نستخدم خمسة متغيرات مؤقتة لاستقبال بيانات سجل واحد من الملف وهي: المتغير strRegNo، والمتغير length، والمتغير manuf، والمتغير year، والمتغير custPhoneNo.

```
Dim boatRecord As New System.Text.StringBuilder()
Dim strRegNo, manuf, custPhoneNo As String
Dim length As Single
Dim year, recNbr As Integer
Dim myFile As String = "e:\Boats.txt"
```

نسند نص بداية الملف إلى المتغير boatRecord أولاً وذلك قبل الدخول في شفرة التكرار Do-Until لكي نلحق بيانات الملف عليه، ثم نلحق عليه نص نهاية الملف بعد التكرار. يستخدم المتغير myFile لتخزين اسم الملف ومساره الذي أنشئ أثناء تنفيذ الزر Save، كما يستخدم هذا المتغير في كل من توصيف كائن الصنف StreamReader (srBoats) واختبار وجود الملف في الأوامر اللاحقة.

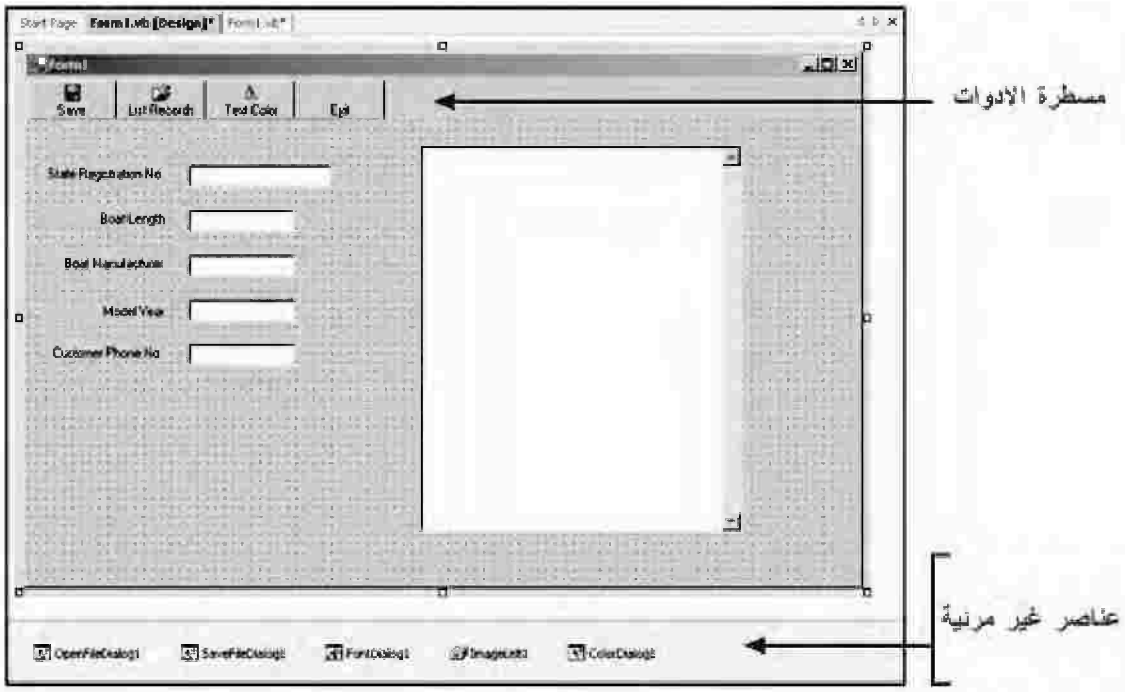
وكما تلاحظ من شفرة عرض سجلات الملف فإنه يجب التحقق من وجود الملف قبل أن نقرأ محتوياته لكي لا تحدث أخطاء وذلك باستخدام الإجراء Exists المعرف داخل الصنف File. فإذا تحققنا من وجود الملف عندئذ يمكننا أن نتجول داخل الملف لكي نستخلص قيمة ونعالجها إلى أن نصل إلى نهاية الملف. يمكّننا الإجراء Read() المعرف داخل الصنف StreamReader من معرفة الوصول إلى نهاية الملف بإرجاع المؤشر 1- الذي يسنده نظام مدخلات ملفات .نت ومخرجاتها عند الوصول إلى نهاية الملف. وأخيراً نستدعي الإجراء Close() لكي نغلق الملف بعد قراءة محتوياته.

نستخدم الإجراء ReadLine لقراءة خمس قيم داخل التكرار ثم تخزينها في متغيرات مؤقتة وهي: المتغير stRegNo، والمتغير length، والمتغير manuf، والمتغير year، والمتغير custPhoneNo.

بعد قراءة خمسة سطور من الملف (التي تناظر صفات مركب واحد) داخل التكرار، يتم زيادة العداد recNbr بمقدار واحد لكي نعرض عدد السجلات الكلي للملف، مع العلم أن القيم المقروءة يتم إلحاقها إلى المتغير boatRecord باستدعاء الإجراء Append المعرف لدى الصنف StringBuilder. كما يتم إلحاق اسم كل حقل قبل القيمة المقروءة ثم يتم إلحاق الرمز vbCrLf الذي يعني إدراج سطر جديد بعد كل قيمة مقروءة. وبعد الخروج من أمر التكرار، سيتم غلق كائن الصنف StreamReader ثم إلحاق العبارة "End of File" إلى المتغير boatRecord الذي سيتم إسناده إلى الخاصية Text لصندوق النص المتعدد السطور txtDisplay ومن ثم تظهر محتويات الملف داخل صندوق النص كما هو موضح في الشكل رقم (١٣,١).

إضافة مساطر أدوات وصناديق حوار
Adding Toolbars and Dialog Controls

عادة ما نستخدم كلاً من مساطر الأدوات (Toolbars) وصناديق الحوار (Dialog Boxes) لبناء واجهة استخدام مميزة حيث تقدم لغة VB .NET عدداً من عناصر صناديق الحوار لكي تغني جودة واجهة استخدام التطبيقات. سوف نطور المثال السابق بإضافة مسطرة أدوات أولاً ثم بعض عناصر صناديق الحوار كما هو واضح في الشكل رقم (١٣,٥).



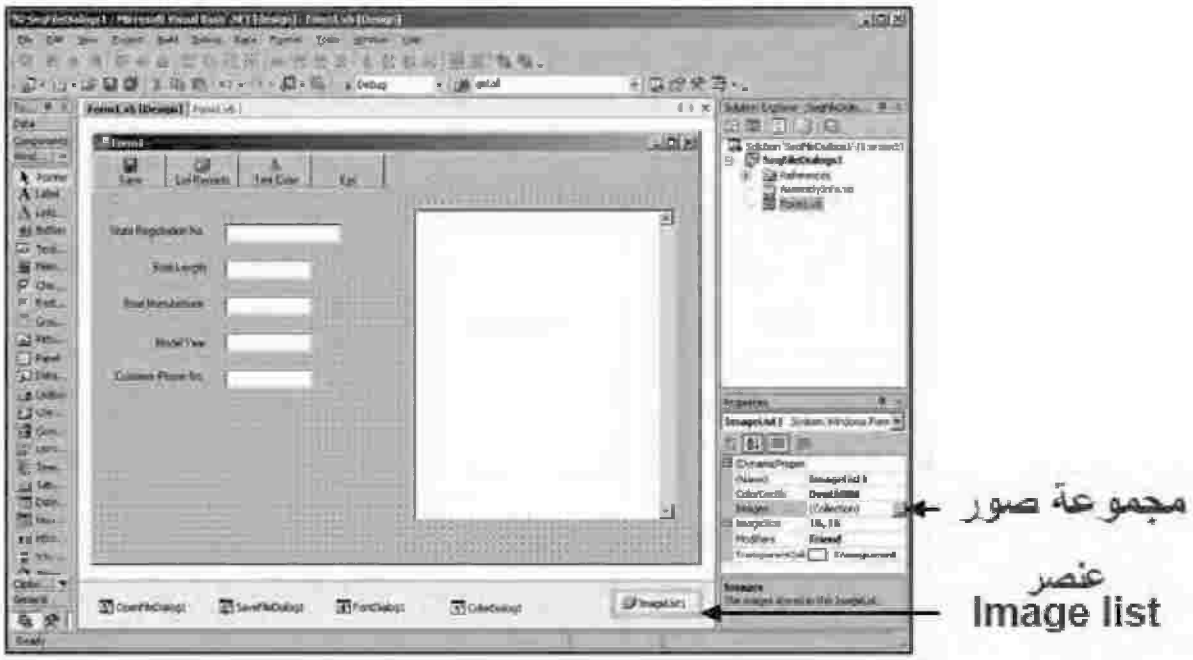
الشكل رقم (١٣,٥). إضافة شريط أدوات.

نوضح في هذه الفقرة كيف نضيف كلاً من عنصر مسطرة الأدوات وعنصر قائمة الصور (ImageList) إلى المثال الحالي (أما إضافة عناصر صناديق الحوار فسوف نناقش في الفقرة التالية). وكما هو ملاحظ أنه قد يتم تغيير واجهة استخدام المثال السابق حيث تم إضافة عنصر مسطرة أدوات عناصر الأزرار وحذفها وكذلك إضافة بعض عناصر صناديق الحوار وعنصر قائمة الصور اللذان يظهران في نافذة منفصلة أسفل النموذج، وذلك على عكس الجيل السابق من لغة VB .NET حيث توضع العناصر التي ليس لها واجهة ظاهرة أثناء التنفيذ في نافذة منفصلة أسفل النموذج وليس داخل النموذج نفسه.

لاحظ أنه عند إضافة عنصر مسطرة أدوات من صندوق الأدوات إلى النموذج لا تظهر مقابض تغيير حجم مسطرة الأدوات وتستمر في قمة النموذج.

عنصر قائمة الصور وعنصر مسطرة الأدوات
The ImageList and Toolbar Controls

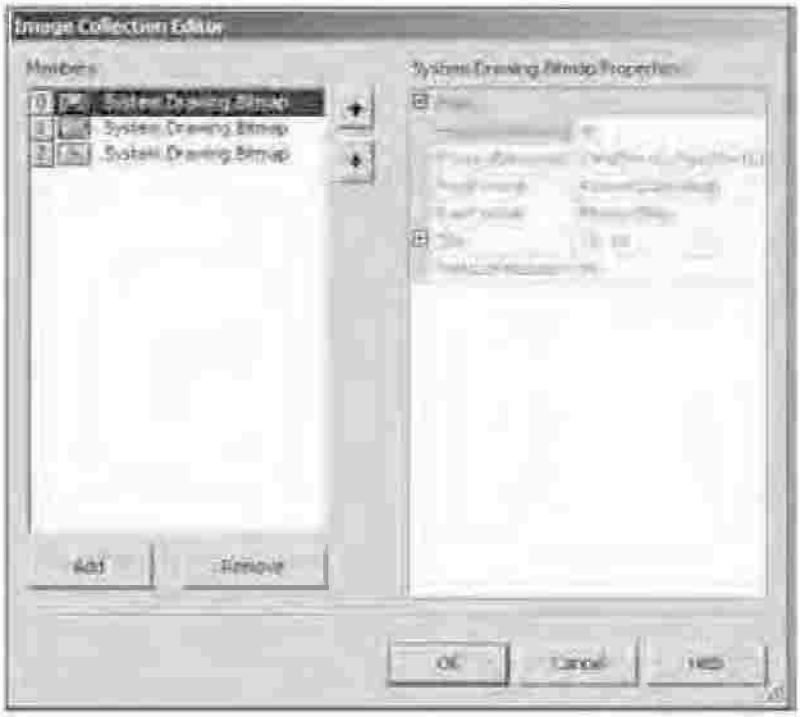
إن استخدام عنصر مسطرة الأدوات ذات الأزرار الرسومية يتطلب استخدام عنصر قائمة الصور (ImageList) الذي يحتوي على صور الأزرار؛ لذلك اضغط ضغطاً مزدوجاً على العنصر ImageList داخل صندوق الأدوات، فعندئذ سيضاف العنصر في نافذة منفصلة أسفل النموذج، ثم اختر هذا العنصر وابحث على الخاصية Images داخل نافذة الخصائص كما هو واضح في الشكل رقم (١٣.٦).



الشكل رقم (١٣,٦). مجموعة خاصية الصور عنصر تحكم ImageList.

تسمح تلك الخاصية Images بتكوين مجموعة من الصور تستخدم كواجهة لأزرار مسطرة الأدوات. ولتكوين هذه المجموعة اضغط على الزر "..."، فعندئذ ستظهر نافذة بعنوان "Image Collection Editor" أي محرر مجموعة الصور كما هو واضح في الشكل رقم (١٣,٧). يمكّنك الزر "Add" من إضافة صورة إلى المجموعة، مع العلم أن كل صورة تحتوي على فهرس يوضح ترتيبها داخل المجموعة (لاحظ ظهور الفهرس في الركن الأيسر العلوي من نافذة Members). يمكنك تغيير ترتيب الصور باستخدام أزرار الأسهم المتواجدة بين النافذتين، حيث يمكن اختيار الصورة المراد تغييرها من زر السهم العلوي أو زر السهم السفلي، فعندئذ يتغير موضع الصورة. ولحذف صورة من المجموعة قم باختيارها أولاً ثم اضغط على الزر "Remove".

تنتمي معظم أنواع صور مايكروسوفت إلى النوع ".bmp"، حيث يمكنك البحث داخل جهازك أو البحث على شبكة الإنترنت لإيجاد الصور المناسبة لأزرار عناصر المسطرة. كما يمكنك استخدام أنواع أخرى من الصور مثل النوع ".tif" أو ".gif" أو "jpg" (لاحظ أن بعض الصور التي تتواجد على شبكة الإنترنت تحتاج إلى موافقة لاستخدامها).



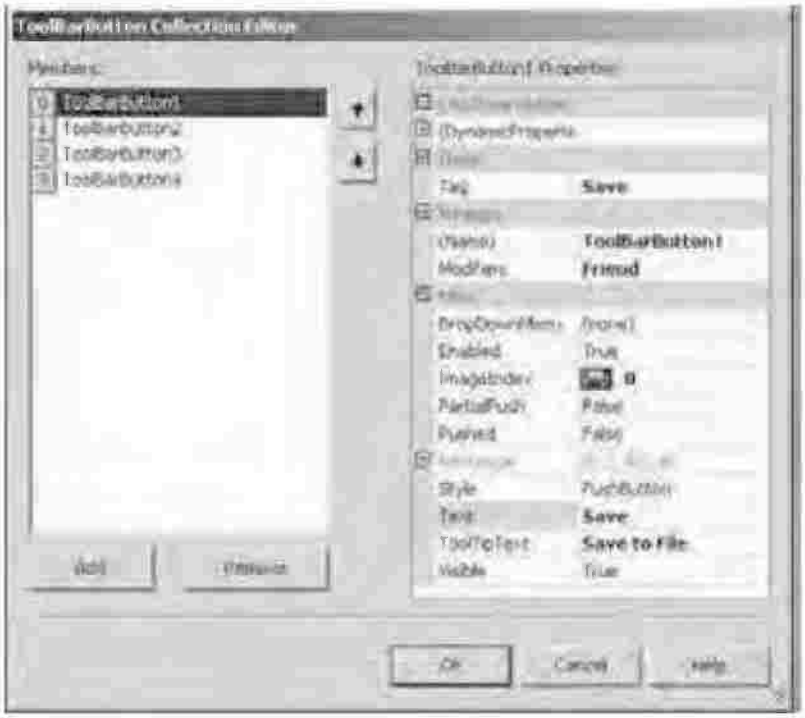
الشكل رقم (١٣,٧). معالج مجموعة الصور.

عنصر مسطرة الأدوات
The Toolbar Control

بعد اختيار الصور لعنصر قائمة الصور وإضافتها، نحتاج أن نضيف عنصر مسطرة الأدوات (Toolbar) للنموذج ومن ثم نسند خاصية ImageList الخاصة بعنصر مسطرة الأدوات بعنصر قائمة الصور التي تحتوي على الصور التي تم اختيارها لتظهر كواجهة لأزرار مسطرة الأدوات. ولعمل ذلك يتم اختيار عنصر مسطرة الأدوات ثم اختيار خاصية ImageList واستخدام القائمة المنسدلة المرفقة لهذه الخاصية لاختيار عنصر قائمة الصور المطلوب كما هو واضح في الشكل رقم (١٣,٨) (لاحظ أن الاسم في هذه الحالة هو ImageList).

تتشابه الصفة Buttons الخاصة بمسطرة الأدوات مع الصفة Images الخاصة بعنصر قائمة الصور، فعندما تضغط على الزر "..." تظهر نافذة بعنوان "ToolBarButton Collection Editor" والتي تسمح لك أن تضيف أزرار تناظر إلى قائمة الصور المتاحة داخل المجموعة Image لعنصر قائمة الصور. وكما هو واضح من الجزء الأيسر لنافذة الشكل رقم (١٣,٩) فقد تم إضافة أربعة أزرار (بداية من فهرس صفر إلى ٣). ويوضح الجزء الأيمن للنافذة أن كلاً من الصفة Tag والصفة ImageIndex والصفة Text والصفة ToolTipText يمكن ضبطها كما هو مطلوب.

الشكل رقم (١٣,٨). خاصية شريط الأدوات ImageList.



الشكل رقم (١٣,٩). معالج مجموعة ToolBarButton.

وكما هو واضح من الشكل رقم (١٣,٩) أن الزر ToolBarButton1 هو الزر المختار حالياً. وقد تم ضبط خواصه حيث أسند للخاصية Tag القيمة "Save" وأسند للخاصية ImageIndex القيمة صفر التي تناظر فهرس الصورة الأولى داخل مجموعة الصور المعرفة داخل مجموعة العنصر ImageList. كما أسند للخاصية Text القيمة "Save" وأسند إلى الخاصية ToolTipText القيمة "Save to File". وسوف يتم استخدام كل من الخاصية Tag والخاصية Text والخاصية ImageIndex في تحديد أي من الأزرار قد يتم الضغط عليه.

وبذلك يتم ضبط خواص الأزرار الأخرى بقيم مناسبة على نحو الزر السابق نفسه ثم نضغط على الزر OK لغلق النافذة الحالية والعودة إلى عنصر مسطرة الأدوات (انظر الشكل رقم ١٣,٥). لاحظ أن عرض كل زر سيختلف عن الآخرين ؛ وذلك يرجع إلى النص المسند للخاصية Text وبسبب الصورة التي ستظهر كواجهة له. لكن يمكنك إسناد عرض متساوٍ لجميع الأزرار ؛ وذلك بتوسعة عنصر مسطرة الأدوات ذاته، حيث نسند قيمة مناسبة للخاصية ButtonSize لعنصر مسطرة الأدوات (القيمة ٨٠ مناسبة لهذا المثال).

وبشكل عام إن أزرار عنصر مسطرة الأدوات تمثل بديلاً لأزرار نموذج المثال السابق ؛ ولذلك سوف نقوم بتعديلات طفيفة على أوامر المثال السابق، ومن ثم استخدامها في هذا المثال، حيث نحتاج فقط إلى وضع أوامر إجراءات معالجة أحداث أزرار المثال السابق داخل إجراءات خاصة والتي سيتم استدعاؤها عند الضغط على أزرار عنصر مسطرة الأدوات. ولعمل ذلك، تم إنشاء ثلاثة إجراءات خاصة فارغة كما يلي:

```
Private Sub SaveToFile()
   ' save code goes here
End Sub

Private Sub   ListRecordsFromFile()
   ' code to extract and list records goes here
End Sub

Private Sub  StopProgram()
   ' code to exit goes here
End Sub
```

قم بنسخ أوامر إجراءات أزرار المثال السابق إلى الإجراءات الخاصة والمناظرة لها ثم احذف الأزرار من النموذج. لاحظ أننا تركنا اسم كل من عنصر مسطرة الأدوات وعنصر قائمة الصور بأسمائها الافتراضية. والآن نريد أن نكتب أوامر استدعاء الإجراءات الخاصة وذلك عند الضغط على أزرار مسطرة الأدوات.

تحديد أي زر من مسطرة الأدوات قد تم الضغط عليه
Determining a Toolbar Button Click

اضغط ضغطاً مزدوجاً في أي مكان على مسطرة الأدوات، عندئذ ستظهر نافذة محرر الأوامر موضحاً الإجراء ToolBar1.ButtonClick() الذي سيستقبل المعامل "e" المعرف من النوع ToolBarButtonClickEventArgs والذي يسمح لك أن تحدد أي من أزرار مسطرة الأدوات قد تم الضغط عليه، حيث يوضح الشكل رقم (١٣,١٠) أحد الطرق التي تستخدم لتحديد ذلك.

لقد استخدمنا في هذه الأوامر الخاصية Tag لمعرفة أي زر قد تم الضغط عليه، مع العلم أنه توجد طريقتان أخريان يمكن استخدامهما للقيام بالغرض نفسه، حيث يمكن استخدام الخاصية Text ومن ثم يصبح رأس أمر Select هكذا e.Button.Tag وأول أمر يصبح Case "Save": SaveToFile(). كما يمكن استخدام الخاصية ImageIndex ويصبح رأس أمر Select هكذا e.Button.ImageIndex وأول أمر هكذا: Case 0: SaveToFile. والآن نتوقع أن يعمل هذا البرنامج مثل برنامج المثال السابق لأنه سوف يستدعي الإجراءات الخاصة المعرفة سابقاً بناء على الزر الذي تم الضغط عليه.

```
Private Sub ToolBar1_ButtonClick(ByVal sender As System.Object, _
   ByVal e As System.Windows.Forms.ToolBarButtonClickEventArgs) _
   Handles ToolBar1.ButtonClick

   Select Case e.Button.Tag
      Case "Save" : SaveToFile()
      Case "Open" : ListRecordsFromFile()
      Case "Color" : SetDisplayText()
      Case "Exit" : StopProgram()
      Case Else
   End Select
End Sub
```

الشكل رقم (١٣,١٠). كود شريط الأدوات.

إضافة صناديق حوارية للبرنامج
Adding Dialogs to the Application

تقدم الصناديق الحوارية (Dialog Boxes) واجهة استخدام غنية ومثالية وخصوصاً لبعض الوظائف الشائعة مثل وظيفة فتح ملف، ووظيفة تخزين ملف، ووظيفة اختيار ألوان، ووظيفة اختيار خط. وهي الوظائف التي يجب أن تكون موحدة المظهر بغض النظر عن البرنامج الذي سوف يحتويها؛ ولذلك تقدم لغة VB .NET مجموعة من صناديق الحوار الجاهزة لهذا الغرض. سوف نشرح في هذه الفقرة كيف نضيف الصندوق الحواري "Open File" لفتح

ملف والصندوق الحواري "Save File" لتخزين ملف. كذلك سوف نضيف أيضاً الصندوق الحواري "Color" لاختيار لون والصندوق الحواري "Font" لاختيار خط إلى البرنامج الحالي. وبالرغم من أننا سوف نناقش الصندوق الحواري "Font" في هذه الفقرة، إلا أننا سوف نؤجل استخدامه إلى تمرين تالي.

تظهر عناصر صناديق الحوار في نهاية صندوق الأدوات (Toolbox)، ولذلك قم بإيجادها ثم اضغط على النموذج الحالي. ومن الملاحظ أن هذه العناصر ستضاف إلى نافذة منفصلة تظهر أسفل النموذج كما هو واضح في الشكل رقم (١٣.٥) الذي يحتوي على الصناديق الأربعة الحوارية.

الصندوق الحواري "Save File"
The Save File Dialog Box

تذكر أن أوامر تخزين ملف انتقلت إلى الإجراء الخاص SaveToFile المعرف في الفقرة السابقة، ويجب أن يعدل لكي يستخدم الصندوق الحواري "Save File" كما هو واضح في الشكل رقم (١٣.١١).

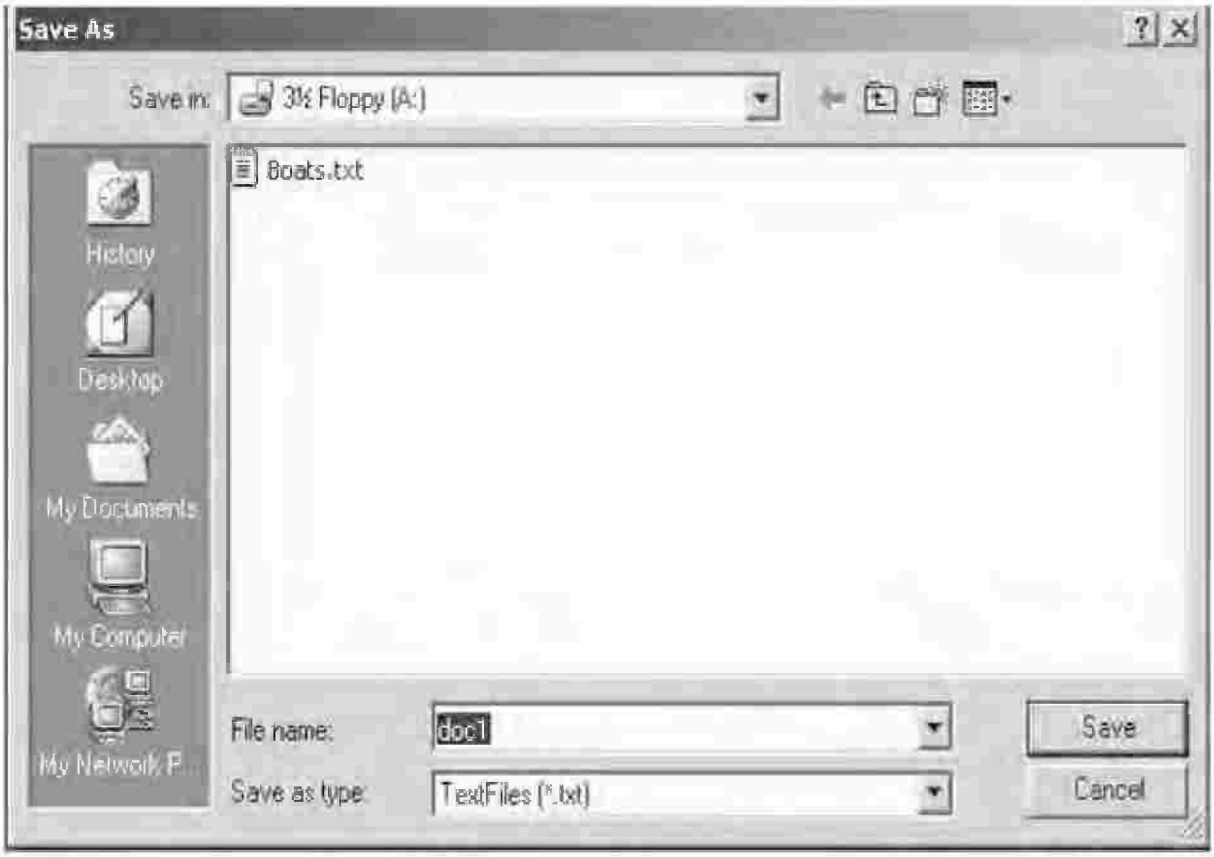
```
Private Sub SaveToFile()
    ' SaveFileDialog to get filename and save boat record
    With SaveFileDialog1
        .Filter = "TextFiles (*.txt)|*.txt|All Files (*.*)|*.*"
        .DefaultExt = "txt"
        .InitialDirectory = "c:\VBFiles"
        If .ShowDialog() = DialogResult.OK Then
            Dim swBoats As New StreamWriter(.FileName, True)

            swBoats.WriteLine(txtStRegNo.Text)
            swBoats.WriteLine(txtLength.Text)
            swBoats.WriteLine(txtManufacturer.Text)
            swBoats.WriteLine(txtYear.Text)
            swBoats.WriteLine(txtCustPhoneNo.Text)
            ' Close the StreamWriter
            swBoats.Close()
        End If
    End With
    ' Clear text boxes and set focus to state registration no textbox
    txtStRegNo.Clear()
    txtLength.Clear()
    txtManufacturer.Clear()
    txtYear.Clear()
    txtCustPhoneNo.Clear()
    txtStRegNo.Focus()
End Sub
```

الشكل رقم (١٣.١١). كود تخزين ملف.

لاحظ أن هذه الأوامر تستخدم الاسم الافتراضي SaveFileDialog1 للصندوق الحواري "Save File". ومن الملاحظ أيضاً استخدام العبارة With..End التي تبسط استدعاء أكثر من خاصية أو إجراء من كائن، خصوصاً إذا

كان اسم هذا الكائن طويلاً نوعا ما (كما في هذه الحالة)، حيث تم استدعاء كل من الخاصية Filter والخاصية DefaultExt والخاصية InitialDirectory من الكائن SaveFileDialog1 ولم يتم تكرار اسم الكائن. تتطلب الخاصية Filter سلسلة من الحروف محاطة بالرمز (") مع تقسيمها باستخدام الرمز (|)، حيث يمثل كل قسم نصاً يحدد نوع الملف (الاسم المعتمد له) المسموح أن يظهر في هذا الصندوق. ولقد أسندنا القيمة "*.txt" للخاصية DefaultExt التي تحدد أن الملفات ذات الامتداد txt. ستظهر عند ظهور الصندوق الحواري للمرة الأولى. كما أسندنا الفهرس الافتراضي للخاصية InitialDirectory والذي سيظهر عند بداية ظهور الصندوق الحواري حيث يمكن تغييره بواسطة المستخدم لتخزين الملف في مكان آخر.

يستخدم الإجراء ShowDialog لفتح أي صندوق حواري، ولقد استخدمناه في هذا المثال لفتح الصندوق الحواري "Save File". ولكن يجب أن تلاحظ أن هذا الصندوق قد تم فتحه في صيغة "Save As" لتقديم وظيفة الحفظ باسم كما هو واضح في الشكل رقم (١٣.١٢).



الشكل رقم (١٣.١٢). الصندوق الحواري "حفظ باسم".

لاحظ أن هذه النافذة تشبه أي صندوق حواري داخل أي تطبيق يقدم وظيفة تخزين الملف، حيث تظهر قائمة "Save In" في أعلى النافذة لتمكين المستخدم من تغيير القرص أو اختيار المجلد، وكذلك القائمة "File Name" التي تمكّن المستخدم من كتابة اسم الملف. كما يمكن للمستخدم أن يختار الملف من قائمة الملفات وعندئذ سيظهر اسمه في هذه القائمة. كما أن القائمة "File Type" تمكّن المستخدم من اختيار نوع الملف المراد (يظهر النوع txt. كنوع افتراضي في هذه الحالة).

وبالعودة إلى أوامر الشكل رقم (١٣,١١) نجد أن الأمر If يستخدم لاختبار ما إذا كان المستخدم قد ضغط على الزر OK أم لا. فإذا ضغط المستخدم على الزر Cancel، عندئذ سيتوقف تنفيذ الأوامر. أما إذا ضغط المستخدم على الزر OK، عندئذ سوف يتم تعريف كائن من الصنف StreamWriter باستخدام الخاصية FileName المعرفة داخل الصنف SaveFileDialog1 والتي تحتوي على اسم الملف الذي أدخله المستخدم. بعد ذلك تتشابه بقية الأوامر مع التطبيق الأول من هذا الفصل (راجع الشكل رقم ١٣,٣).

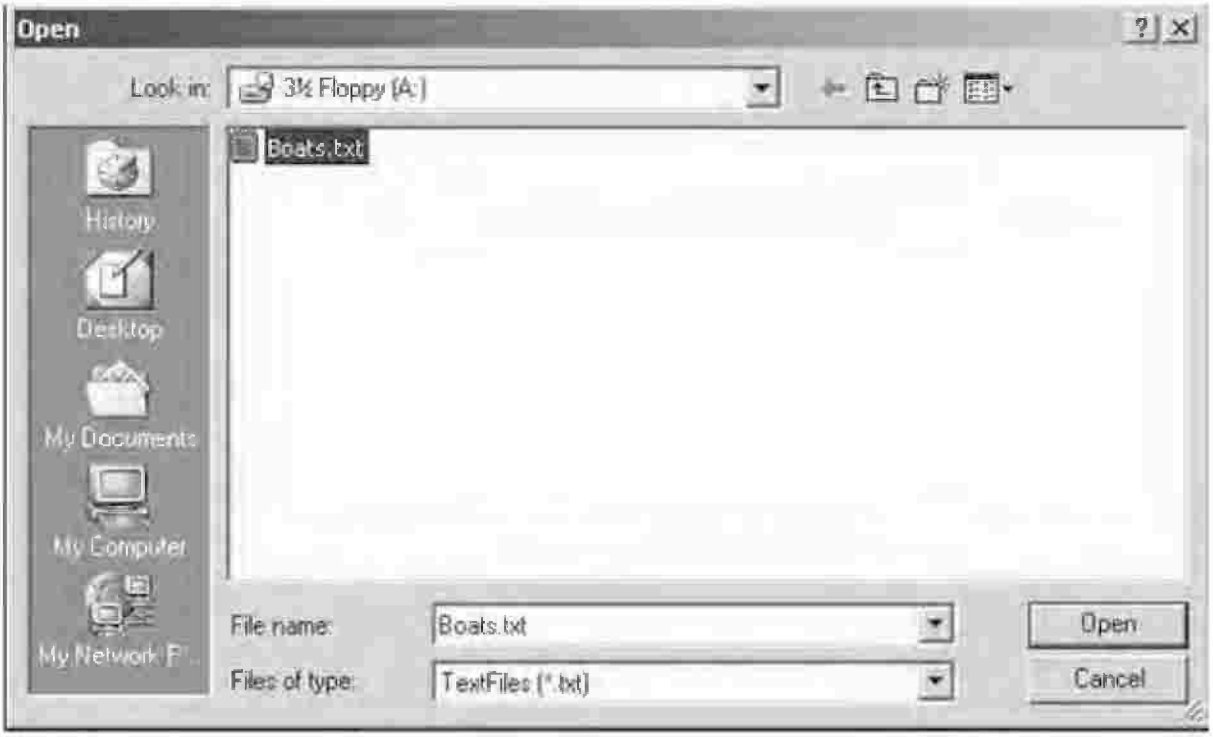
الصندوق الحواري "OpenFile"
The OpenFile Dialog Box

يعمل الصندوق الحواري "OpenFile" مثل الصندوق الحواري "SaveFile" ولكن بشكل عكسي، حيث نريد هنا اختيار الملف لقراءة محتوياته وليس لتخزينه كما هو واضح في أوامر الشكل رقم (١٣,١٣). تبدأ هذه الأوامر بالعبارة With OpenFileDialog1 ثم أوامر تتشابه مع أوامر فتح الصندوق الحواري "SaveFile"، والتي يجب أن تكون سهلة بالنسبة لك، خصوصاً بعد الاطلاع على أوامر المثال السابق حيث تستخدم الخاصية Filter والخاصية DefaultExt والخاصية InitialDirectory مثل استخدامها في مثال الصندوق الحواري "SaveFile"، كما يستخدم الإجراء ShowDialog لفتح الصندوق الحواري "OpenFile" كما هو موضح في الشكل رقم (١٣,١٤).

كما يستخدم الأمر If في هذه الأوامر مثل استخدامه في مثال فتح الصندوق الحواري "SaveFile" وذلك لاختبار هل المستخدم قد ضغط على الزر OK أم لا. فإذا ضغط المستخدم على الزر OK، عندئذ سيتم إنشاء كائن من الصنف StreamReader ثم نكمل الأوامر مثل أوامر المثال الأول.

```
Private Sub ListRecordsFromFile()
   Dim stRegNo, manuf, custPhoneNo, myFile As String
   Dim length As Single
   Dim year, recNbr As Int32

   ' OpenFileDialog1 gets filename
   With OpenFileDialog1
      .Filter = "TextFiles (*.txt)|*.txt|All Files (*.*)|*.*"
      .DefaultExt = "txt"
      .InitialDirectory = "c:\VBFiles"
      If .ShowDialog() = DialogResult.OK Then
         myfile = .FileName
         ' Declare Stringbuilder, StreamReader
         Dim boatRecord As New System.Text.StringBuilder()
         recNbr = 0
         boatRecord.Append("List of Records from the File" & vbCrLf)
         boatRecord.Append("============================" & vbCrLf)
         ' Check to see if file exists
         If File.Exists(myfile) And myfile.Length > 0 Then
            Dim srBoats As StreamReader = _
               New StreamReader(.FileName)
            ' Loop over records until end of the file is reached
            ' -1 denotes end of file
            Do Until srBoats.Peek = -1
               ' Read a record
               stRegNo = srBoats.ReadLine
               length = srBoats.ReadLine
               manuf = srBoats.ReadLine
               year = srBoats.ReadLine
               custPhoneNo = srBoats.ReadLine
               recNbr += 1

               ' Build line for display in multiline text box
               boatRecord.Append("Boat Number = " & _
               recNbr & vbCrLf)
               boatRecord.Append("State Registration Number = " & _
                    stRegNo & vbCrLf)
               boatRecord.Append("Boat Length = " & _
                    length & vbCrLf)

               boatRecord.Append("Boat Manufacturer = " & _
                    manuf & vbCrLf)
               boatRecord.Append("Boat Model Year = " & _
                    year & vbCrLf)
               boatRecord.Append("Customer Phone Number = " & _
                    custPhoneNo & vbCrLf & vbCrLf)

            Loop
            srBoats.Close()
            boatRecord.Append("End of File")
            txtDisplay.Text = boatRecord.ToString
            txtStRegNo.Focus()
         End If
      End If
   End With
End Sub
```

الشكل رقم (١٣,١٣). كود فتح ملف.



الشكل رقم (١٣,١٤). الصندوق الحواري الخاص بفتح ملف.

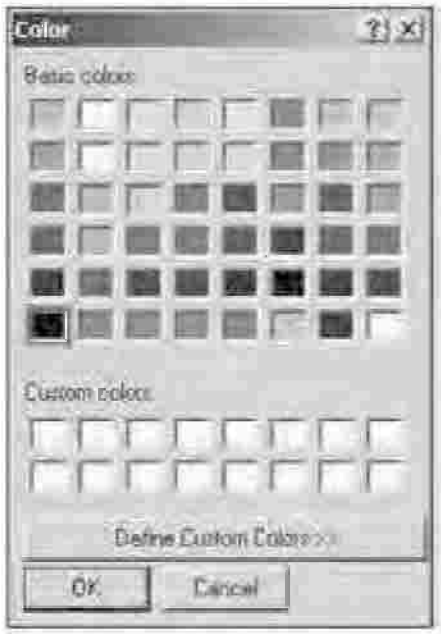
إضافة الصندوق الحواري "Color"

Adding the Color Dialog Box

يستخدم الصندوق الحواري Color لتمكين المستخدم من اختيار لون، والذي نستطيع أن نستخدمه لتغيير لون نص متواجد داخل صندوق نص مثلاً. يوضح الشكل رقم (١٣,١٥) أقل أوامر ممكنة لتغيير لون نص داخل صندوق نص متعدد السطور، حيث استخدمنا الإجراء ShowDialog لفتح الصندوق الحواري. ثم أسندنا قيمة الخاصية Color المعرفة داخل الصنف ColorDialog والتي تحتوي على اللون الذي اختاره المستخدم داخل الخاصية ForeColor لصندوق النص TextDisplay. يظهر الصندوق الحواري Color في الشكل رقم (١٣,١٦).

```
Private Sub SetDisplayText()
    With ColorDialog1
        .ShowDialog()
        txtDisplay.ForeColor = .Color
    End With
End Sub
```

الشكل رقم (١٣,١٥). كود الصندوق الحواري الخاص باللون.



الشكل رقم (١٣,١٦). صندوق الألوان.

يجب أن يعمل البرنامج الآن بشكل جيد بعد التعديلات التي أجريناها عليه لاستدعاء كل من الصندوق الحواري "SaveFile"، والصندوق الحواري "OpenFile"، والصندوق الحواري "Color". يوضح الشكل رقم (١٣,١٧) شفرة أوامر البرنامج كاملة.

```
' Chapter 13--Example2

' Sequential file processing with a toolbar and dialogs
Imports System.IO
Imports System.Text.StringBuilder
Public Class Form1
    Inherits System.Windows.Forms.Form

    Private Sub StopProgram()
        Me.Close()
    End Sub

Private Sub SaveToFile()
    ' SaveFileDialog to get filename and save host record
    With SaveFileDialog1
      .Filter = "TextFiles (*.txt)|*.txt|All Files (*.*)|*.*"
      .DefaultExt = "txt"
      .InitialDirectory = "c:\VBFiles"
```

الشكل رقم (١٣,١٧). شفرة أوامر البرنامج كاملة.

```
        If .ShowDialog() = DialogResult.OK Then
            Dim swBoats As New StreamWriter(.FileName, True)

            swBoats.WriteLine(txtStRegNo.Text)
            swBoats.WriteLine(txtLength.Text)
            swBoats.WriteLine(txtManufacturer.Text)
            swBoats.WriteLine(txtYear.Text)
            swBoats.WriteLine(txtCustPhoneNo.Text)
            ' Close the StreamWriter
            swBoats.Close()
        End If
    End With
    ' Clear text boxes and set focus to state registration no TextBox
    txtStRegNo.Clear()
    txtLength.Clear()
    txtManufacturer.Clear()
    txtYear.Clear()
    txtCustPhoneNo.Clear()
    txtStRegNo.Focus()
End Sub

Private Sub ListRecordsFromFile()
    Dim stRegNo, manuf, custPhoneNo, myFile As String
    Dim length As Single
    Dim year, recNbr As Int32

    ' OpenFileDialog1 gets filename
    With OpenFileDialog1
        .Filter = "TextFiles (*.txt)|*.txt|All Files (*.*)|*.*"
        .DefaultExt = "txt"
        .InitialDirectory = "c:\VBFiles"
        If .ShowDialog() = DialogResult.OK Then
            myFile = .FileName
            ' Declare StringBuilder, StreamReader
            Dim boatRecord As New System.Text.StringBuilder()
            recNbr = 0
            boatRecord.Append("List of Records from the File" & vbCrLf)
            boatRecord.Append("============================" & vbCrLf)
            ' Check to see if file exists
            If File.Exists(myFile) And myFile.Length > 0 Then
                Dim srBoats As StreamReader =
                    New StreamReader(.FileName)
                ' Loop over records until end of the file is reached
                ' -1 denotes end of file
                Do Until srBoats.Peek = -1
                    ' Read a record
                    stRegNo = srBoats.ReadLine
                    length = srBoats.ReadLine
                    manuf = srBoats.ReadLine
                    year = srBoats.ReadLine
                    custPhoneNo = srBoats.ReadLine
                    recNbr += 1
```

تابع الشكل رقم (١٣,١٧).

```
                ' Build line for display in multiline text box
                boatRecord.Append("Boat Number = " & _
                recNbr & vbCrLf)
                boatRecord.Append("State Registration Number = " & _
                    stRegNo & vbCrLf)
                boatRecord.Append("Boat Length = " & _
                    length & vbCrLf)
                boatRecord.Append("Boat Manufacturer = " & _
                    manuf & vbCrLf)
                boatRecord.Append("Boat Model Year = " & _
                    year & vbCrLf)
                boatRecord.Append("Customer Phone Number = " & _
                    custPhoneNo & vbCrLf & vbCrLf)

            Loop
            srBoats.Close()
            boatRecord.Append("End of File")
            txtDisplay.Text = boatRecord.ToString
            txtStRegNo.Focus()
          End If
        End If
    End With
End Sub
Private Sub SetDisplayText()
    With ColorDialog1
        .ShowDialog()
        txtDisplay.ForeColor = .Color
    End With
End Sub

'Region " Windows Form Designer generated code "

Private Sub ToolBar1_ButtonClick(ByVal sender As System.Object, _
    ByVal e As System.Windows.Forms.ToolBarButtonClickEventArgs) _
    Handles ToolBar1.ButtonClick

    Select Case e.Button.Tag
        Case "Save" : Call SaveToFile()
        Case "Open" : Call ListRecordsFromFile()
        Case "Color" : Call SetDisplayText()
        Case "Exit" : Call StopProgram()
        Case Else
    End Select

 End Sub
End Class
```

تابع الشكل رقم (١٣,١٧).

الصندوق الحواري "FontDialog"

The FontDialog Control

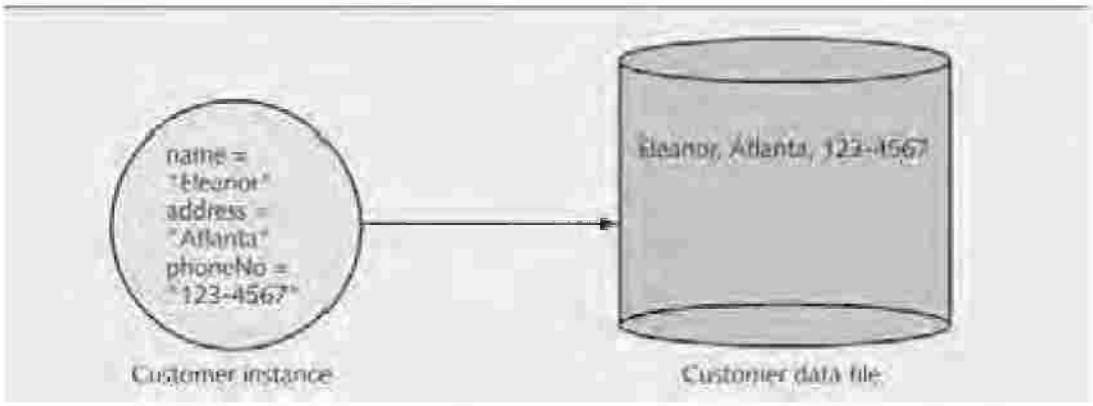
يستخدم عنصر الصندوق الحواري FontDialog في تغيير نوع الخط (Font) ولون الخط في تطبيق ما. يتم ذلك بإسناد قيمة الخاصية Font والخاصية ForeColor لعناصر التطبيق مثل عنصر العنوان (Label) وعنصر صندوق النص (Text Box). توضح الأوامر التالية كيف نغير نوع الخط ولون الخط لصندوق النص TextDialog.

```
' Set FontDialog control to include color
FontDialog1.ShowColor() = True

' Open the FontDialog window & check if user clicked OK
If FontDialog1.ShowDialog() = DialogResult.OK Then
   txtDisplay.Font = FontDialog1.Font
   txtDisplay.ForeColor = FontDialog1.Color
End If
```

كيف نجعل الكائنات مستمرة

Making Objects Persistent

لقد تعلمنا من الأمثلة السابقة كيف نخزن معلومات من الملفات ونسترجعها والتي ستساعدنا كي نحفظ بكائنات النظام مستمرة. فإلى الآن لا يمكن الاحتفاظ بالكائنات بعد انتهاء تنفيذ البرنامج. وبالطبع إن أي تطبيقات واقعية تتطلب إمكانية تخزين هذه الكائنات واسترجاعها حيث توجد طريقتان للاحتفاظ بالكائنات وهما: تخزين قيم صفات الكائنات (Attribute Storage)، أو تخزين الكائنات نفسها (Object Storage)، حيث تتطلب الطريقة الأولى استرجاع قيم صفات الكائنات من الكائنات ثم تخزينها داخل ملف كما هو واضح في الشكل رقم (١٣,١٨) الذي يمثل أسلوب تخزين قيم صفات كائن الصنف Customer.



الشكل رقم (١٣,١٨). أسلوب تخزين قيم صفات كائن الصنف Customer.

سوف نتعلم كيف نخزن كائنات ونسترجعها باستخدام أكثر من أسلوب:

- الاحتفاظ بقيم صفات الكائنات واسترجاعها باستخدام كل من الصنف StreamWriter والصنف StreamReader.
- نشر الكائنات إلى القرص الصلب (Object Serialization).
- الاحتفاظ بقيم صفات الكائنات باستخدام قواعد البيانات (Databases).

تستخدم الطريقة الأولى الملفات التعاقبية مع كل من الصنف StreamWriter والصنف StreamReader للاحتفاظ بقيم صفات الكائنات. لقد تعرفنا على هذا الأسلوب سابقاً في هذا الفصل ولكن سوف نطبقه هنا مع أصناف التعامل مع البيانات. فعلى سبيل المثال إذا أردت الاحتفاظ ببيانات كائن الصنف Customer سوف نستدعي إجراءات الوصول أولاً لكي نسترجع قيم الصفة Name والصفة Address، والصفة PhoneNo ثم تخزن هذه القيم داخل ملف بيانات العملاء. كما يمكنك استرجاع الكائن مرة ثانية بإجراء العملية نفسها ولكن بطريقة عكسية حيث نقرأ بيانات العميل من ملف البيانات ثم ننشئ كائن الصنف Customer باستخدام هذه القيم.

وعلى أي حال إن أسلوب تخزين قيم صفات الكائنات إلى ملفات تعاقبية ربما لا يكون أفضل أسلوب للاحتفاظ بالكائنات. وبسبب أهمية الموضوع وتعقيد عملية التخزين والاسترجاع، فقد قدم إطار عمل .نت طريقة سهلة لتخزين الكائنات واسترجاعها تسمى Object Serialization وتعني نشر الكائنات أنفسها إلى القرص الصلب، حيث يتم تحويل الكائن إلى سلسلة متدفقة (Stream) يمكن تخزينها داخل ملف تعاقبي. وتحول العملية العكسية (Deserialization) السلسلة المتدفقة من الملف إلى كائن مرة ثانية. ويتميز هذا الأسلوب أنه لا توجد حاجة لإعادة إنشاء الكائن كما نفعل في أسلوب الاحتفاظ بقيم صفات الكائن ولكن يتم استرجاع الكائن مباشرة.

يوفر إطار عمل .نت طريقتين لوصف شكل السلسلة المتدفقة التي تنتج أثناء عملية النشر التسلسلي للكائنات هما: صفة النظام الثنائي (Binary Formatter) وبرتوكول SOAP (Simple Object Application Protocol). يدعم برتوكول SOAP لغة XML التي تستخدم بكثرة في تطبيقات خدمات الويب (Web Services). سوف نعرض لكل من برتوكول SOAP ولغة XML وخدمات الويب خلال الفصل السادس عشر.

وأخيراً نستخدم قواعد البيانات للاحتفاظ بالكائنات، حيث تتكون قواعد البيانات من مجموعة من الملفات المنظمة التي تسهل عملية الاستعلام. تنظم البيانات في قواعد البيانات العلاقية داخل جداول مرتبطة ببعضها بعضاً، حيث يمثل كل عمود داخل جدول صفة لكائن وكل صف يمثل سجلاً. تتكون لغة الاستعلام الهيكلية SQL (Structured Query Language) من كلمات محجوزة وأوامر والتي تستخدم مع قواعد البيانات العلاقية. تحتوي لغة VB .NET على مجموعة أصناف وإجراءات التي يمكن استدعاؤها أثناء التعامل مع قواعد البيانات العلاقية. سوف نتعرض للغة SQL لاحقاً حسب الاحتياج.

تصميم صنف التعامل مع البيانات
Designing a Data Access Class

إن الغرض الرئيس من أصناف التعامل مع البيانات هو تقديم مجموعة من الإجراءات التي تستخدم لتخزين البيانات واسترجاعها (صفات الكائن). ومن ثم تجعل كائنات أصناف مجال المشكلة كائنات مستمرة

(Persistent Objects). وكما رأينا في الفصل الخامس فإن أسلوب تصميم الأنظمة المعتمدة على الكائنات (OO Systems) وتطويرها يستخدم طريقة الطبقات الثلاثة (Three-Tiers) والتي تتكون من ثلاث طبقات من الأصناف وهي: أصناف واجهة الاستخدام (GUI Classes) التي تقدم واجهة رسومية لاستقبال البيانات وعرض المعلومات، وأصناف مجال المشكلة (PD Classes) التي تمثل كائنات المشكلة محل الاهتمام، وأصناف التعامل مع البيانات (DA Classes) التي تقدم وظيفة تخزين البيانات واسترجاعها. كما تستخدم هذه الطريقة نموذج العميل والخادم (Client-Server Model) في تطوير الأنظمة الذي يسهل عملية التركيب الأنظمة وصيانتها.

ترجع أسباب فصل مهام تخزين البيانات واسترجاعها ووضعها في أصناف التعامل مع البيانات إلى سببين. السبب الأول هو عزل هذه البيانات بعيداً عن الأصناف الأخرى مما يقلل من مشكلات عملية الصيانة، حيث إن تغيير طريقة تخزين البيانات لا تتطلب أي تغيير في كل من أصناف مجال المشكلة وأصناف واجهة الاستخدام بل تؤثر فقط على أصناف التعامل مع البيانات. وبالمثل فإن تعديل كل من واجهة الاستخدام وأصناف مجال المشكلة لا يتطلب أدنى تعديل في أصناف التعامل مع البيانات. السبب الثاني هو إن هذا الأسلوب يدعم نموذج العميل-الخادم، حيث يمكن نشر الأنواع الثلاثة من الأصناف على ثلاثة أجهزة منفصلة وهو ما يؤدي إلى سهولة التركيب ونشر النظم على بيئة تدعم مفهوم العميل-الخادم (للحصول على تفاصيل أكثر راجع كلاً من الفصل الخامس عشر والفصل السادس عشر من هذا الكتاب). كما يؤدي هذا الفصل إلى عدم استدعاء إجراءات أصناف التعامل مع البيانات إلا بواسطة أصناف مجال المشكلة. ومن ثم فإن الخدمات التي يتم تقديمها بواسطة أصناف التعامل مع البيانات تبدو وكأنها تقدم بواسطة أصناف مجال المشكلة لأصناف واجهة الاستخدام التي لا تعلم شيئاً عن أصناف التعامل مع البيانات.

إجراءات التعامل مع البيانات
Data Access Methods

بشكل عام يتم تطوير صنف تعامل بيانات واحد لكل صنف من أصناف مجال المشكلة. فعلى سبيل المثال، سيتم تطوير الصنف CustomerDA في هذا الفصل لتخزين كائنات الصنف Customer واسترجاعها. ولكن تذكر أن الصنف الذي لديه الحق باستدعاء إجراءات الصنف CustomerDA هو الصنف Customer فقط. ومن ثم سوف تطلب أصناف واجهة الاستخدام هذه الخدمات من الصنف Customer وليس من الصنف CustomerDA. وسوف نوضح هذه الفكرة لاحقاً في موضوع "الاتصال بأصناف التعامل مع البيانات".

تذكر من الفصل السادس أن لغة VB .NET تقدم نوعين من الإجراءات وهما: إجراءات مشتركة (Shared Methods)، وإجراءات غير مشتركة (Nonshared Methods). الإجراءات المشتركة هي التي لا يتطلب

استدعاؤها تعريفاً بل تستدعى من الصنف مباشرة؛ ولذلك يطلق عليها أحياناً اسم إجراءات أصناف (Class Methods) ويتم تعريفها في بعض اللغات الأخرى باستخدام الكلمة المحجوزة "Static". والإجراءات غير المشتركة هي التي يجب أن تستدعيها من كائن محدد وليس من صنف. ولأننا لا نحتاج أن نعرف كائنات من أصناف التعامل مع البيانات فسيتم تعريف إجراءات أصناف التعامل مع البيانات من نوع الإجراءات المشتركة.

سوف يقوم الصنف CustomerDA بأربع مهام وهي: استرجاع بيانات عميل، وتخزين بيانات عميل، وتغيير بيانات عميل، وحذف بيانات عميل. وسنطور أربعة إجراءات تناظر هذه المهام الأربعة وهي: الإجراء Find، والإجراء AddNew، والإجراء Update، والإجراء Delete. وبالرغم من أن أوامر هذه الإجراءات ستعتمد على الأسلوب المتبع في تخزين البيانات إلا أن تعريف توقيع هذه الإجراءات سيكون موحداً. بمعنى آخر إن الكائنات التي سوف تستدعي هذه الإجراءات لم يكن لديها أي فكرة عن كيفية تخزين البيانات واسترجاعها. ولكن يجب أن تعلم فقط كيف تستدعي هذه الإجراءات (أي توقيع الإجراءات).

سوف نصف في الفقرات الآتية كيف نعرف رؤوس هذه الإجراءات. كما سنعرف صنفين من نوع الاستثناء (Exception) لاستخدامهما داخل هذه الإجراءات وهو الصنف NotFoundException والصنف DuplicateException. أما أوامر هذه الإجراءات فسوف نشرحها في الفقرات التالية.

البحث عن عميل
Finding a Customer

يستخدم الإجراء Find في البحث عن عميل بواسطة رقم هاتفه؛ لذلك يمثل رقم هاتف العميل الصفة المميزة له والتي يجب ألا تتكرر. يستقبل الإجراء Find معاملاً يحتوي على رقم هاتف العميل المراد البحث عنه. فإذا تم إيجاد العميل، عندئذ سوف يعيد هذا الإجراء مؤشراً لكائن من الصنف Customer يحتوي على بيانات هذا العميل. أما إذا لم يتم إيجاده، فسوف يعيد هذا الإجراء كائناً من صنف الاستثناء الخاص NotFoundException. توضح الأوامر التالية تعريف رأس الإجراء Find.

```
Public Shared Function Find(ByVal PhoneNo As String) As Customer
```

إضافة عميل
Adding a Customer

يستخدم الإجراء AddNew لإضافة بيانات عميل إلى النظام، حيث يستقبل كائناً من الصنف Customer كمعامل ولم تُعِد بيانات إلى الكائن الذي قام باستدعائها. ولأننا لا نريد أن نخزن أكثر من عميل لهم نفس رقم

الهاتف، فسوف يقوم ذلك الإجراء بالتأكد أولاً من عدم وجود عميل مخزن سابقاً يملك نفس رقم الهاتف. فإذا وجد عميل مسبقاً يملك نفس رقم الهاتف، عندئذ سوف ينشئ كائناً من صنف الاستثناء الخاص DuplicateException ثم يعيده. يبدو رأس تعريف الإجراء AddNew هكذا:

```
Public Shared Sub AddNew(ByVal aCustomer As Customer)
```

تعديل بيانات عميل
Updating a Customer

يستخدم الإجراء Update لتعديل بيانات العميل، حيث يمكن للعميل أن يغير عنوانه أو رقم هاتفه. يستقبل الإجراء Update مؤشراً لكائن الصنف Customer يحتوي على البيانات الجديدة ثم يبحث عن هذا العميل ثم يغير بياناته إذا وجده داخل النظام. فإذا لم يجده، سوف ينشئ كائناً من صنف الاستثناء الخاص NotFoundException ويعيده، مع العلم أن هذا الإجراء لم يقم بإعادة بيانات. يوضح الأمر التالي رأس تعريف الإجراء Update:

```
Public Shared Sub Update(ByVal aCustomer As Customer)
```

حذف بيانات عميل
Deleting a Customer

يستخدم الإجراء Delete لحذف بيانات عميل من النظام، حيث يستقبل (مثل الإجراء Update) مؤشراً لكائن الصنف Customer ثم يبحث عنه لكي يقوم بحذفه من النظام إذا وجده. فإن لم يجده، فسوف ينشئ كائناً من صنف الاستثناء الخاص NotFoundException ويعيده، مع العلم أن هذا الإجراء أيضاً لا يقوم بإعادة بيانات. يوضح الأمر التالي تعريف رأس الإجراء Delete:

```
Public Shared Sub Delete(ByVal aCustomer As Customer)
```

إجراءات أصناف التعامل مع البيانات الإضافية
Additional Data Access Methods

يحتوي صنف التعامل مع البيانات CustomerDA على ثلاثة إجراءات إضافية وهي: الإجراء Initialize، والإجراء Terminate، والإجراء GetAll. وكما تشير أسمائها يقوم الإجراء Initialize بالتهيئة المطلوبة لتخزين البيانات أو استرجاعها، كما يقوم الإجراء Terminate بمهام الإنهاء وذلك عند الانتهاء من تخزين البيانات أو استرجاعها، ويسترجع الإجراء GetAll جميع العملاء المخزنين في النظام. لا يستقبل كل من الإجراء Initialize أو

الإجراء Terminate أي بيانات ولا يعيدها. أما أوامر هذه الإجراءات فإنها ستعتمد على الطريقة المستخدمة لتخزين البيانات واسترجاعها، حيث نشرح في هذا الفصل أربعة أساليب لهذا الغرض. توضح الأوامر التالية تعريف رأس كل من الإجراء Initialize و الإجراء Terminate:

```
Public Shared Sub Initialize()
Public Shared Sub Terminate()
```

أما الإجراء GetAll فيعيد جميع العملاء المخزنين في النظام ومن ثم سوف يعيد كائناً من الصنف ArrayList الذي سيحتوي على كائنات من الصنف Customer. يوضح الأمر التالي تعريف رأس هذا الإجراء:

```
Public Shared Function GetAll() As ArrayList
```

تعيد الأوامر التالية تعريف رؤوس إجراءات الصنف CustomerDA:

```
Public Shared Function Find(ByVal PhoneNo As String) As Customer
Public Shared Sub AddNew(ByVal aCustomer As Customer)
Public Shared Sub Update(ByVal aCustomer As Customer)
Public Shared Sub Delete(ByVal aCustomer As Customer)
Public Shared Sub Initialize()
Public Shared Sub Terminate()
Public Shared Function GetAll() As ArrayList
```

الاتصال بأصناف التعامل مع البيانات
Communicating With a Data Access Class

لقد أوضحنا في الفقرة السابقة أن إجراءات صنف التعامل مع البيانات لا يتم استدعاؤها إلا من صنف مجال المشكلة المناظر له، حيث إن الصنف Customer هو صنف مجال المشكلة الوحيد الذي يستدعي إجراءات الصنف CustomerDA. ولتقديم خدمات تخزين العملاء واسترجاعها مثلاً، يتم استدعاء إجراءات الصنف Customer التي بدورها تستدعي إجراءات الصنف CustomerDA. وبذلك تبدو هذه الخدمات للأصناف الأخرى (أصناف واجهة الاستخدام) وكأنها تقدم بواسطة أصناف مجال المشكلة.

ولكي يمكن لأصناف واجهة الاستخدام مثلاً أن تستفيد من خدمات أصناف التعامل مع البيانات (الإجراءات السبعة) فيجب علينا تعريف سبعة إجراءات مناظرة في صنف مجال المشكلة الذي يلعب دوراً وسيطاً فقط بينها. وعندما نعرف هذه الإجراءات السبعة داخل الصنف Customer، فيجب علينا أن نعرفها بشكل مستقل عن الأسلوب المتبع داخل أصناف التعامل مع البيانات لتخزين البيانات واسترجاعها.

البحث عن عميل
Finding a Customer

إن الغرض الرئيس من الإجراء Find المعرف داخل الصنف Customer هو استدعاء الإجراء Find من الصنف CustomerDA. ولأن هذا الإجراء يعيد مؤشراً لكائن من الصنف Customer، فيجب أيضاً على الإجراء Find المعرف داخل الصنف CustomerDA أن يعيد مؤشراً لكائن من الصنف Customer. كما يجب عليه أيضاً أن يعيد كائناً من صنف الاستثناء NotFoundException إذا استقبله الإجراء Find المعرف داخل الصنف CustomerDA وذلك إذا لم يتم إيجاد العميل. ويجب أن يعرف الإجراء Find من النوع Shared لأنه يجب أن يتم استدعاؤه من صنف وليس من كائن.

يحتوي الإجراء Find، كما هو واضح في الأوامر التالية، على أمر استدعاء الإجراء Find المعرف داخل الصنف CustomerDA ممرراً إليه رقم هاتف العميل الممرر إليه ثم يعيد مؤشراً لكائن الصنف Customer الذي استقبله من الإجراء Find المعرف في الصنف CustomerDA عند استدعائه.

```
Public Shared Function Find(ByVal PhoneNo As String) As Customer
   Return CustomerDA.Find(PhoneNo)
End Function
```

إضافة عميل
Adding a Customer

يستدعي الإجراء AddNew المعرف داخل الصنف Customer الإجراء AddNew المعرف داخل الصنف CustomerDA لكي يخزن بيانات عميل جديد (بيانات كائن الصنف Customer). سوف يستدعي هذا الإجراء من كائن الصنف Customer الذي نريد أن نخزن بياناته؛ ولذلك لم نعرف هذا الإجراء من النوع Shared. كما لا يستقبل هذا الإجراء بيانات ولا يعيدها. ولكن لاحظ أن الإجراء AddNew المعرف داخل الصنف CustomerDA سوف يعيد كائناً من صنف الاستثناء الخاص DuplicateException عند وجود عميل في النظام يمتلك نفس رقم الهاتف. ولاحظ أيضاً أن الإجراء AddNew سوف يمرر كائن الصنف Customer الجاري تنفيذ أوامره (المراد تخزين قيم صفاته)؛ ولذلك سوف يمرر له الكلمة Me هكذا:

```
Public Sub AddNew()
   CustomerDA.AddNew(Me)
End Sub
```

تغيير بيانات عميل
Changing a Customer

يستدعي الإجراء Update المعرف داخل الصنف Customer الإجراء Update المعرف داخل الصنف CustomerDA. سوف يستدعي هذا الإجراء من كائن الصنف Customer المراد تعديل بياناته ؛ ولذلك لم يتم تعريفه من النوع Shared. كما سوف يمرر هذا الكائن إلى الإجراء Update للصنف CustomerDA. لاحظ أيضاً أن الإجراء Update المعرف داخل الصنف CustomerDA ربما يعيد كائناً من صنف الاستثناء الخاص NotFoundException:

```
Public Sub Update()
   CustomerDA.Update(Me)
End Sub
```

حذف عميل
Deleting a Customer

يستدعي الإجراء Delete المعرف داخل الصنف Customer الإجراء Delete المعرف داخل الصنف CustomerDA (كما هو واضح في الأوامر التالية). ومثل الإجراء Update، لم يتم تعريف الإجراء Delete من النوع Shared وربما يعيد كائناً من صنف الاستثناء الخاص NotFoundException عند عدم وجود العميل. كما يمرر كائن الصنف Customer الحالي الكلمة Me إلى الإجراء Delete المعرف داخل الصنف CustomerDA:

```
Public Sub Delete()
   CustomerDA.Delete(Me)
End Sub
```

إجراءات صنف مجال المشكلة الإضافية
Additional Problem Domain Methods

لأن الصنف CustomerDA يحتوي على كل من الإجراء Initialize والإجراء Terminate والإجراء GetAll، يجب أن يتم تعريفها داخل الصنف Customer، حيث يتم استدعاء الإجراء المناظر له من الصنف Customer (كما هو واضح في الأوامر التالية). كما سيتم تعريفها جميعاً من النوع Shared. لاحظ أن الإجراء GetAll سوف يعيد كائناً من الصنف ArrayList:

```
Public Shared Sub Terminate()
   CustomerDA.Terminate()
End Sub
Public Shared Sub Initialize()
   CustomerDA.Initialize()
End Sub
Public Shared Function GetAll() As ArrayList
   Return CustomerDA.getAll
End Function
```

تطوير الكائنات المستمرة باستخدام الملفات المتعاقبة
Implementing Persistence with a Sequential File

سوف نستخدم أسلوب تخزين بيانات الكائن (Attribute Storage) إلى ملف تعاقبي داخل المثال التالي. ولتقليل عملية القراءة والكتابة من وإلى الملف، سوف نجعل الإجراء Initialize يقرأ سجلات العملاء المخزنة داخل الملف وتحويلها إلى كائنات من الصنف Customer ثم تخزينها داخل مصفوفة ArrayList. ومن ثم سوف نتعامل مع هذه المصفوفة عند القراءة والتخزين مؤقتاً إلى أن نستدعي الإجراء Terminate الذي سيتجول داخل كائنات هذه المصفوفة وقراءة صفات كل كائن ثم إعادة تخزينها إلى الملف.

يبدأ تعريف الصنف CustomerDA بالأوامر Imports للإشارة إلى كل من الفضاء المسمى Collection الذي يحتوي على الصنف ArrayList والفضاء المسمى IO الذي يحتوي على أصناف المدخلات والمخرجات.

```
Imports System.IO
Imports System.Collections
```

ثم تعريف متغير الإشارة Customers لكائن من الصنف ArrayList الذي سيحتوي على متغيرات لكائنات من الصنف Customer. بعد ذلك نعرف متغيراً آخر من النوع String الذي سيحتوي على اسم الملف الذي سيستخدم في المثال التالي. وكما هو واضح أن اسم الملف هو "customerFile.txt" المخزن على القرص المرن A.

```
' Declare a Shared ArrayList called customers
 Shared customers As New ArrayList()
 ' Declare a string and set to file
 Shared cFile As String = "a:\customerFile.txt"
```

الإجراء Initialize
The Initialize Method

يستخدم هذا الإجراء لقراءة جميع سجلات العملاء من الملف التعاقبي ثم إنشاء كائنات من الصنف Customer وتخزين متغيرات إشارة هذه الكائنات داخل مصفوفة ArrayList. ينشئ أول أمر في الإجراء Initialize كائناً من الصنف ArrayList يسمى Customers الذي سوف يخزن متغيرات إشارة لكائنات الصنف Customer.

```
customers = New ArrayList()
```

تنفذ بقية أوامر هذا الإجراء عند التأكد من وجود بيانات عميل، حيث نستخدم الأمر If الذي يستدعي كلاً من الإجراء Exists والخاصية Length لكائن الصنف File المسمى cFile؛ وذلك للتأكد من وجود الملف أولاً وللتأكد من أن طول الملف لم يساوي صفراً قبل محاولة قراءة البيانات (أي التأكد أن الملف يحتوي على سجلات).

```
If File.Exists(cFile) And cFile.Length > 0 Then
```

ثم نعرف متغيرات من النوع String لتخزين قيم صفات العميل. بعد ذلك نضع الأوامر المسؤولة عن إنشاء كائن من الصنف StreamReader وقراءة بيانات عميل من الملف (الاسم، والعنوان، ورقم الهاتف) داخل التركيب Try. وذلك تحسباً لحدوث خطأ I/O وإلقاء استثناء (كائن من الصنف Exception). لاحظ أننا نمرر المتغير cFile لإجراء إنشاء الصنف StreamReader.

```
Dim name, address, phoneNo As String
   Try
        ' Declare a StreamReader and point to file
        Dim CustomerFile As New StreamReader(cFile)
```

ثم نعرف التكرار Do-Loop الذي يستدعي الإجراء ReadLine لقراءة اسم العميل وعنوانه ورقم هاتفه. بعد ذلك يتم إنشاء كائن من الصنف Customer وإضافة متغير إشارة هذا الكائن داخل المصفوفة Customers. ويستمر هذا التكرار إلى أن يعيد الإجراء Peek القيمة 1- والتي تعني الوصول إلى نهاية الملف، ثم نغلق كائن الصنف StreamReader:

```
' Loop over file until end of file-- Peek = -1
Do Until CustomerFile.Peek = -1
   name = CustomerFile.ReadLine
   address = CustomerFile.ReadLine
   phoneNo = CustomerFile.ReadLine
   ' If name returned, then add to customers
   If name <> Nothing Then
      customers.Add(New Customer(name, address, phoneNo))
   End If
Loop
```

ثم يأتي آخر جزء في الإجراء وهو عبارة عن التركيب Catch الذي يتم تنفيذه إذا تم استقبال استثناء وذلك عند حدوث خطأ ناتج من تنفيذ أوامر التركيب Try:

```
Catch e As Exception
   Console.WriteLine(e.ToString)
End Try
```

يوضح الشكل رقم (١٣,١٩) أوامر الإجراء Initialize كاملة.

```
' Initialize Method
   Public Shared Sub Initialize()

      ' Check for file existence that includes data
      If File.Exists(cFile) And cFile.Length > 0 Then
         ' Declare temporary values to hold values returned from file
         Dim name, address, phoneNo As String
         Try
            ' Declare a StreamReader and point to file
            Dim CustomerFile As New StreamReader(cFile)
            ' Loop over file until end of file-- Peek = -1
            Do Until CustomerFile.Peek = -1
               name = CustomerFile.ReadLine
               address = CustomerFile.ReadLine
               phoneNo = CustomerFile.ReadLine
               ' If name returned, then add to customers
               If name <> Nothing Then
                  customers.Add(New Customer(name, address, _
                     phoneNo))
               End If
            Loop
            CustomerFile.Close()
         Catch e As Exception
            Console.WriteLine(e.ToString)
         End Try
      End If
   End Sub
```

الشكل رقم (١٣,١٩)- أوامر الإجراء Initialize كاملة.

الإجراء Terminate
The Terminate Method

يهدف الإجراء Terminate إلى تخزين جميع قيم صفات كائنات الصنف Customer المتواجدة في المصفوفة Customers داخل ملف تعاقبي، حيث يستخدم الإجراء Terminate المتغير cFile الذي يحمل اسم الملف ومكانه في إنشاء كائن من الصنف StreamWriter. ثم يتم إنشاء متغير إشارة لكائن من الصنف Customer يسمى cust الذي يتم استخدامه داخل التكرار For-Each للتجول داخل المصفوفة Customers التي تحتوي على كائنات الصنف Customer. لاحظ أن التكرار For-Each يمكن أن يستخدم مع جميع أنواع المجموعات والذي يمكن استبداله بالتكرار For-Next الذي يستخدم فهرساً للتجول داخل المجموعة، كما يستخدم الصفة Count التي تعيد عدد عناصر المصفوفة. نسترجع قيم صفات كل كائن داخل المصفوفة باستدعاء إجراءات الوصول ثم نكتب هذه القيم داخل الملف التعاقبي باستخدام الإجراء WriteLine. لاحظ أن جميع الأوامر توجد داخل التركيب Try (كما هو واضح في الشكل رقم ١٣,٢٠) وذلك تحسباً لحدوث أخطاء من إجراء عمليات الإدخال والإخراج التي تتسبب في إلقاء استثناءات.

```
' Terminate Method
Public Shared Sub Terminate()

   Try
      ' Instantiate a StreamWriter
      Dim customerFile As New StreamWriter(cFile, True)
      Dim cust As Customer
      ' Define String variables to hold object attributes
      Dim name, address, phoneno As String
      ' Loop over customers ArrayList -- getting and saving attributes
      For Each cust In customers
         name = cust.GetName
         address = cust.GetAddress
         phoneno = cust.GetPhoneNo

         customerFile.WriteLine(name)
         customerFile.WriteLine(address)
         customerFile.WriteLine(phoneno)
      Next
         customerFile.Close()
   Catch e As Exception
         Console.WriteLine(e.ToString)
   End Try
End Sub
```

الشكل رقم (١٣,٢٠). أوامر التركيب Try.

الإجراء Find
The Find Method

تذكر أن المصفوفة Customer تحتوي على متغيرات إشارة لكائنات الصنف Customer ؛ ولذلك فإن الإجراء Find يتجول ببساطة داخل هذه المصفوفة باحثاً عن كائن يملك رقم هاتف يتشابه مع القيمة المدخلة. فإذا تم الحصول على هذا الكائن ، فعندئذ سيعيد الإجراء مؤشراً لهذا الكائن. أما إذا لم يتم الحصول عليه فسيتم إلقاء استثناء (كائن من الصنف NotFoundException). يوضح الشكل رقم (١٣,٢١) أوامر الإجراء Find.

يبدأ الإجراء بتعريف متغيري إشارة لكائنين من الصنف Customer وهما: المتغير aCustomer الذي يأخذ قيمة ابتدائية Nothing في الأمر التالي ، والمتغير cust الذي سيستخدم لاحقاً في التكرار For-Each للبحث عن العميل. ثم نضيف المتغير foundIt من النوع Boolean الذي يأخذ قيمة ابتدائية False.

نتجول داخل المصفوفة Customers باستخدام المتغير cust حيث نسنده في بداية التكرار إلى المتغير aCustomer. بعد ذلك نختبر قيمة رقم الهاتف المستقبلة مع رقم هاتف العميل الحالي والمشار إليه بالمتغير cust. فإذا حدث تساو بينهما ، فعندئذ نغير قيمة المتغير foundIt إلى True ثم ننهي التكرار مباشرة. وهكذا إلى أن نختبر جميع أرقام هواتف عملاء المصفوفة Customers.

```
' Find Function--Throws NotFoundException if Not Found
Public Shared Function Find(ByVal PhoneNo As String) As Customer
   ' Define two object variables and boolean variable--set to False
   Dim cust As New Customer()
   Dim acustomer As New Customer()
   acustomer = Nothing
   Dim foundIt As Boolean = False
   ' Loop ArrayList customers for each customer
   For Each cust In customers
      acustomer = cust
      ' If found, set foundit to true
      If cust.GetPhoneNo = PhoneNo Then
         foundIt = True
            Exit For
      End If
   Next
   If foundIt = True Then
      Return acustomer
   Else
      Throw New NotFoundException(" Not Found ")
   End If
End Function
```

الشكل رقم (١٣,٢١). أوامر الإجراء Find.

وفي نهاية الإجراء يوجد أمر If الذي يختبر قيمة المتغير foundIt. فإذا كانت قيمته تساوي True، عندئذ نعيد متغير الإشارة aCustomer والذي يشير إلى العميل الذي تم الحصول عليه. أما إذا كانت قيمته تساوي False، عندئذ ننشئ كائناً من صنف الاستثناء NotFoundException ثم نلقيه للإجراء الذي استدعى الإجراء Find.

الإجراء AddNew
The AddNew Method

يتم استدعاء هذا الإجراء لإضافة عميل جديد للنظام. ولأن جميع العملاء (كائنات الصنف Customer) تخزن داخل المصفوفة Customers، فإن الإجراء AddNew ينشئ متغير إشارة لكائن الصنف Customer المستقبل ثم يتم اختبار عدم تكرار رقم هاتفه في النظام ثم إضافته إلى المصفوفة Customers كما هو واضح في الشكل رقم (١٣,٢٢).

يبدأ الإجراء AddNew بتعريف المتغير exists من النوع Boolean وإسناده بالقيمة False. ثم تعريف المتغير cust ثانياً للتجول داخل عناصر المصفوفة Customers باستخدام التكرار For-Each وتعريف المتغير existPhoneNo من النوع String لتخزين قيمة رقم هاتف العميل المشار إليه حالياً بالمتغير cust. بعد ذلك يأتي التكرار For-Each للبحث داخل المصفوفة Customers لكي نحدد هل العميل المراد إضافته موجود بالفعل أم لا. فإذا كان موجوداً، فعندئذ سننشئ كائناً من صنف الاستثناء DuplicateException ثم نلقيه إلى الإجراء الذي استدعى الإجراء AddNew.

```
' AddNew Method--Throws DuplicateException if exists
Public Shared Sub AddNew(ByVal aCustomer As Customer)
   ' Set Boolean variable to False, String variable for phoneno
   Dim exists As Boolean = False
   Dim cust As Customer
   Dim existPhoneNo As String
   ' Loop ArrayList customers for each customer
   For Each cust In customers
      existPhoneNo = cust.GetPhoneNo
      ' If already exists, do not add
      If existPhoneNo = aCustomer.GetPhoneNo Then
         exists = True
         Exit For
      End If
   Next
   If exists = True Then
      Throw New DuplicateException("Customer Already Exists")
   Elee
      customers.Add(aCustomer)
   End If
End Sub
```

الشكل رقم (١٣,٢٢). إجراء AddNew المعرف داخل الصنف CustomerDA.

يسترجع الأمر الأول في التكرار قيمة رقم الهاتف لكائن العميل المشار إليه بالمتغير cust. ثم نقارن في الأمر التالي هل تساوي هذه القيمة مع رقم هاتف العميل الجديد. فإذا نجحت المقارنة عندئذ نغير قيمة المتغير exists إلى True ثم ننهي التكرار مباشرة. ثم يأتي آخر أمر في الإجراء وهو الأمر If الذي يختبر قيمة المتغير exists، فإذا كانت قيمته تساوي True، عندئذ ننشئ كائناً من صنف الاستثناء DuplicateException ونلقيه. أما إذا كانت قيمته تساوي False، عندئذ نضيف كائن الصنف Customer الجديد إلى المصفوفة Customers.

الإجراء Update
The Update Method

لا نحتاج أن نكتب أوامر داخل الإجراء Update (انظر الشكل رقم ١٣,٢٣) في حال تخزين بيانات العملاء داخل ملف تعاقبي؛ وذلك لأن التعديل يتم خلال استدعاء إجراءات الوصول Setters (الإجراء setName، والإجراء setAddress، والإجراء setPhoneNo) والتي تقوم بتعديل بيانات العملاء داخل المصفوفة Customers إلى أن يتم إعادة تخزينها داخل الملف بواسطة استدعاء الإجراء Terminate.

```
Public Shared Sub Update(ByVal aCustomer As Customer)
   ' Nothing required--Changes via setter methods
   Return
End Sub
```

الشكل رقم (١٣,٢٣). إجراء Update المعرف داخل الصنف CustomerDA.

الإجراء Delete
The Delete Method

يحذف الإجراء Delete أحد العملاء من النظام وذلك بحذف كائن من الصنف Customer من المصفوفة Customers إلى أن يخزن الإجراء Terminate جميع الكائنات المتواجدة حالياً في المصفوفة Customers إلى الملف التعاقبي. وبذلك فإن الحذف في هذا المثال يعني حذف الكائن من المصفوفة Customers كما هو واضح في الشكل رقم (١٣,٢٤).

```
'Delete Method--Throws NotFoundException if not found
Public Shared Sub Delete(ByVal aCustomer As Customer)
   Dim foundIt As Boolean = False
   Dim phoneNo As String = aCustomer.GetPhoneNo
   Dim i As Integer = 0
   Dim cust As Customer
   ' Loop ArrayList customers for each customer
   For Each cust In customers
      ' If found, set foundit to True and remove from customers
      If cust.GetPhoneNo = phoneNo Then
         foundIt = True
         customers.RemoveAt(i)
         Exit For
      End If
      i = i + 1
   Next
   If Not foundIt Then
      Throw New NotFoundException(" Not Found ")
   End If
End Sub
```

الشكل رقم (١٣,٢٤). إجراء Delete المعرف داخل الصنف CustomerDA.

يبدأ الإجراء Delete بتعريف المتغير foundIt من النوع Boolean الذي يأخذ القيمة الابتدائية False. بعد ذلك يتم تعريف المتغير phoneNo من النوع String الذي يأخذ قيمة رقم هاتف العميل المراد حذفه وذلك بواسطة استدعاء الإجراء GetPhoneNo. ثم تعريف المتغير i من النوع Integer الذي يلعب دور الفهرس للمصفوفة Customers، لأن قيمة الفهرس ضرورية لحذف عنصر من مجموعة. يتجول التكرار For-Each داخل عناصر المصفوفة Customers باحثاً

عن العميل المراد حذفه، وذلك بمقارنة رقم هاتف العميل المستقبل مع أرقام هواتف العملاء في المصفوفة. فإذا تم الوصول إلى كائن العميل سيتم حذفه باستخدام الإجراء RemoveAt الذي يستقبل فهرس العميل المراد حذفه. أما إذا لم يتم الوصول إليه فسيتم تعريف كائن من الاستثناء الخاص NotFoundException وإلقائه.

الإجراء GetAll
The GetAll Method

تحتوي المصفوفة Customers على متغيرات إشارة لجميع كائنات الصف Customer. ولذلك سوف يعيد الإجراء GetAll المصفوفة Customers ثم يستطيع الإجراء المستدعي أن يصل لكائن معين داخل المصفوفة مباشرة (انظر الشكل رقم ١٣,٢٥).

```
Public Shared Function GetAll() As ArrayList
        Return customers
End Function
```

الشكل رقم (١٣,٢٥). إجراء GetAll المعرف داخل الصنف CustomerDA.

الصنف DuplicateException والصنف NotFoundException
The DuplicateException and NotFoundException Classes

يستخدم كل من الصنف DuplicateException والصنف NotFoundException في أصناف التعامل مع البيانات (DA Classes) لإخبار كائنات أصناف مجال المشكلة (PD Classes) بحدوث مشكلة. يوضح الشكل رقم (١٣,٢٦) أوامر تعريف هذين الصنفين.

```
Public Class DuplicateException
    Inherits Exception
    Public Sub New(ByVal message As String)
        MyBase.New(message)
    End Sub
End Class

Public Class NotFoundException
    Inherits Exception
    Public Sub New(ByVal message As String)
        MyBase.New(message)
    End Sub
End Class
```

الشكل رقم (١٣,٢٦). الصنف DuplicateException والصنف NotFoundException.

تجربة الصنف CustomerDA الذي يتعامل مع الملفات التعاقبية
Testing CustomerDA for Sequential File Implementation

سوف ننشئ في هذه الفقرة تطبيقاً لتجربة صنف التعامل مع البيانات CustomerDA الذي تم تطويره خلال ا السابقة. ولعمل ذلك يتم فتح تطبيق ويندوز جديد ثم وضع زر على نموذج التطبيق وإسناد القيمة "Tester for Data Access Class" للخاصية Text لهذا الزر. يستدعي إجراء معالجة هذا الزر الإجراء المعرف الآخر الذي يحتوي على أمر تجربة الصنف CustomerDA. وسوف يتم استخدام المعامل "e" لاستقبال الاستثناءات. وبما أن هذا المعامل معرف لدى جميع عناصر النماذج، فسوف نستخدمه دون إجراء أي تعديلات به. وسنقوم بوصف أوامر اختبار إجراءات صنف التعامل مع المشكلة بالتفصيل لأنها سوف تستخدم لاحقاً دون تغيير في كل من مثال النشر التسلسلي للكائنات (Object Serialization) ومثال قواعد البيانات العلاقية (Relational Databases) في هذا الفصل. سوف تحتوي أوامر الاختبار على:

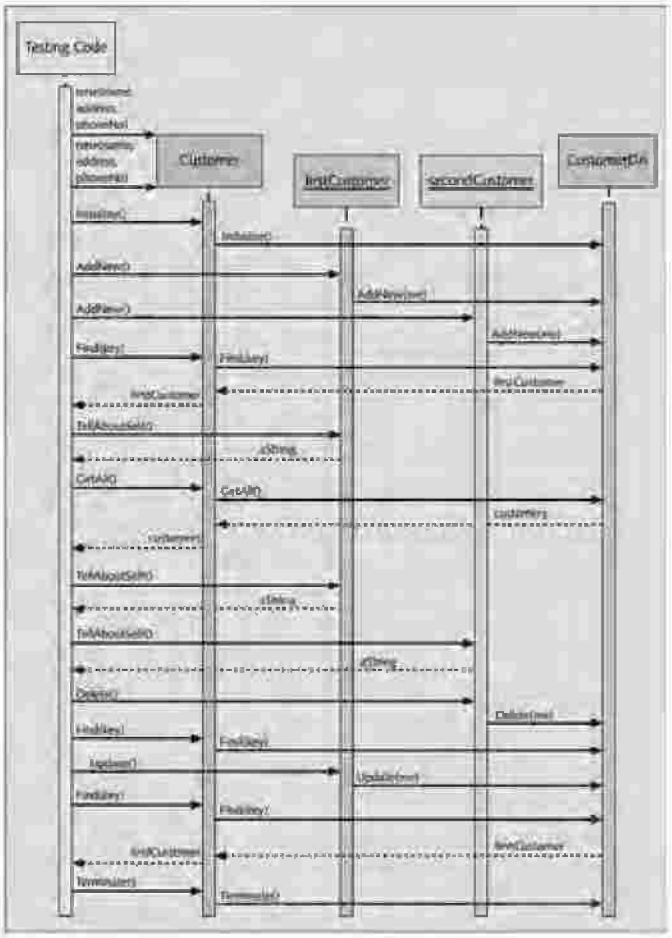
- إنشاء كائنين من الصنف Customer.
- استدعاء الإجراء Initialize.
- استدعاء الإجراء AddNew لإضافة كائني الصنف Customer الذي تم إنشائهما سابقاً.
- استرجاع الكائن الأول باستدعاء الإجراء Find.
- استدعاء الإجراء GetAll لاسترجاع كائني الصنف Customer الذي تم إنشاءهما سابقاً.
- استدعاء الإجراء Delete لحذف الكائن الثاني والتأكد من حذفه.
- تغيير عنوان الكائن الأول والتحقق من التغيير باستدعاء الإجراء Update.
- استدعاء الإجراء Terminate.

يوضح الشكل رقم (١٣,٢٧) نموذج Sequence Diagram الذي يوضح هذه المهام بشكل رسومي. لاحظ تبسيط النموذج وذلك بحذف استدعاء كل من إجراءات الوصول وإجراءات العرض.

ينشئ البرنامج أولاً كائنين من الصنف Customer ثم يشير إليهما متغير الإشارة firstCustomer ومتغير الإشارة secondCustomer. ثم يستدعي الإجراء Initialize للأعداد قبل تنفيذ أوامر المعالجة التالية. ولكن يجب أن تلاحظ أن هذا البرنامج يتعامل فقط مع صنف مجال المشكلة Customer ولم يعلم شيئاً عن صنف التعامل مع البيانات CustomerDA. وبذلك فإن أي مهمة تتطلب استدعاء أحد إجراءات الصنف CustomerDA تتم من خلال الصنف Customer ويجب أن تكون هذه العلاقة واضحة لك.

بعد ذلك يستدعي الإجراء AddNew من كل من المتغير firstCustomer والمتغير secondCustomer لتخزينهما داخل المصفوفة Customers التي تنتقل فقط إلى الملف التعاقبي عند استقبال الإجراء Terminate. وعلى أي حال فإن

برنامج الاختبار ليس لديه أدنى فكرة عن طريقة تخزين بيانات العملاء أو استرجاعها. كما يجب وضع استدعاء الإجراء AddNew داخل التركيب Try وذلك تحسباً لحدوث خطأ عند تخزين العميل واستقبال استثناء من النوع DuplicateException الذي يتم إنشاؤه وإلقاؤه عند محاولة إعادة تخزين عميل موجود (يعرف العميل برقم هاتفه الذي يمثل المفتاح الأساسي له).



الشكل رقم (١٣,٢٧). نموذج Sequence Diagram لاختبار أصناف التعامل مع البيانات.

ثم نستدعي الإجراء Find باستخدام رقم هاتف العميل الأول والذي يجب وضعه أيضاً داخل التركيب Try تحسباً لاستقبال استثناء من النوع NotFoundException. وعندما يتم استقبال متغير إشارة الكائن الأول بعد الحصول عليه يتم استدعاء الإجراء TellAboutSelf لعرض بياناته.

بعد ذلك يتم استدعاء الإجراء GetAll الذي يعيد مصفوفة من متغيرات الإشارة التي تشير إلى كائنات الصنف Customer (في هذا المثال يوجد فقط كائنان: العميل Mike والعميل Eleanor). ثم يتجول داخل هذه

المصفوفة لاسترجاع متغيرات الإشارة واحداً تلو الآخر مع استدعاء الإجراء TellAboutSelf لاسترجاع بيانات كل كائن (بيانات كل عميل) ثم عرضها.

ثم نستدعي الإجراء Delete من متغير إشارة الكائن الثاني (العميل Mike) لحذفه من النظام. تذكر أن البرنامج لم يعلم كيف يتم حذف الكائنات ولكن ببساطة عند استدعاء الإجراء Delete من كائن فلابد أن يحذف من النظام ويختفي. يلي ذلك استدعاء الإجراء Find للتأكد أن العميل Mike قد تم حذفه من النظام. ونؤكد مرة ثانية على ضرورة وضع الإجراء Delete داخل التركيب Try تحسباً لحدوث خطأ واستقبال استثناء من النوع NotFoundException.

بعد ذلك نستدعي الإجراء SetAddress لتغيير عنوان العميل الأول ثم استدعاء الإجراء Update. يغير الإجراء SetAddress قيمة الصفة address للعميل الأول. أما الإجراء Update فيغير الكائنات داخل المصفوفة Customers. ويصبح هذا التعديل نهائياً عند استدعاء الإجراء Terminate الذي يخزن جميع العملاء من المصفوفة Customers إلى الملف التعاقبي. ثم نتأكد من صحة التعديل باستدعاء الإجراء Find من الكائن الأول.

ثم نستدعي الإجراء Terminate الذي يكتب قيم صفات الكائنات المخزنة داخل المصفوفة Customers إلى الملف التعاقبي وذلك لجعلها كائنات مستمرة. ويقوم آخر أمر بعرض رسالة تفيد بأن عمل البرنامج قد انتهى وتطلب منك أن تتفحص مخرجات البرنامج. فيجب عليك الآن البحث عن الملف CustomerFile على القرص الصلب. فإذا كان غير موجود قبل تنفيذ برنامج الاختبار فإنه يجب أن يكون موجوداً الآن، حيث يجب أن يحتوي الملف على العميل Eleanor فقط؛ وذلك لأن البرنامج قد أنشأ عميلين ثم قام بحذف العميل Mike. يوضح الشكل رقم (١٣,٢٨) برنامج الاختبار، ويوضح الشكل رقم (١٣,٢٩) مخرجات هذا البرنامج. لاحظ أن العبارة "Not Found" قد تم كتابتها بالخط الأسود العريض لتمييزها. وقد حدث الاستثناء NotFoundException لأن العميل Mike قد تم حذفه ثم تم البحث عنه باستدعاء الإجراء Find.

تطوير استمرارية الكائنات باستخدام النشر التسلسلي
Implementing Persistence with Object Serialization

عندما نجعل الكائنات مستمرة باستخدام طريقة النشر التسلسلي (Serialization) فإننا نستعمل طريقة تخزين الكائنات (Object Storage) التي تختلف عن طريقة تخزين صفات الكائنات (Attribute Storage) المستخدمة سابقاً، حيث نخزن الكائنات كاملة من وإلى الملف ونسترجعها؛ ولذلك لا نحتاج أن نعيد إنشاء الكائن عند استرجاعه من الملف، ولكن يتطلب الأمر فقط تعريف متغير إشارة ليشير إلى الكائن العائد.

```
' Chapter 13--Example 3
' Object persistence using a sequential file
Imports System.IO
Imports System.Collections
Public Class Form1
    Inherits System.Windows.Forms.Form
Private Sub testOAClasses()
    ' Instantiate two customers and Initialize
    Dim firstCustomer As New Customer("Eleanor", "Atlanta", "123-4567")
    Dim secondCustomer As New Customer("Mike", "Boston", "467-1234")

        Customer.Initialize()

        ' Test AddNew
        Try
            firstCustomer.AddNew()
            secondCustomer.AddNew()
            Console.WriteLine("Added Two customers")
        Catch e As DuplicateException
            Console.WriteLine(e.message.ToString)
        End Try

        ' Test Find Method
        Try
            firstCustomer = Customer.Find("123-4567")
            Console.WriteLine("Find: " & firstCustomer.TellAboutSelf)
        Catch e As NotFoundException
            Console.WriteLine(e.message.ToString)
        End Try

        ' Test GetAll
        Dim allcustomers As ArrayList = Customer.GetAll
        Dim cust As Customer
        For Each cust In allcustomers
            firstCustomer = cust
            Console.WriteLine("Get all: " & firstCustomer.TellAboutSelf)
        Next

        ' Test Delete method
        Try
            secondCustomer.Delete()
            Console.WriteLine("Delete: " & secondCustomer.TellAboutSelf)
            ' Try to find customer just deleted
            secondCustomer = Customer.Find("467-1234")
            Console.WriteLine("Delete: " & secondCustomer.TellAboutSelf)
        Catch e As NotFoundException
            Console.WriteLine(e.message.ToString)
        End Try


        ' Test Update by changing address of Eleanor
        Try
            firstCustomer = Customer.Find("123-4567")
            firstCustomer.SetAddress("Clayton")
            firstCustomer.Update()
            ' Display address after change
            firstCustomer = Customer.Find("123-4567")
            Console.WriteLine("Update: " & firstCustomer.TellAboutSelf)
        Catch e As NotFoundException
            Console.WriteLine(e.message.ToString)
        End Try

        'Test Terminate method--csc.rsp results in file
        Customer.Terminate()
        MessageBox.Show("Test Script Completed--See results")
        Me.Close()
    End Sub

    Private Sub btnTester1_Click(ByVal sender As System.Object, _
        ByVal e As System.EventArgs) Handles btnTester1.Click

        TestOAClasses()
    End Sub

End Class
```

الشكل رقم (١٣,٢٨). برنامج اختبار أصناف التعامل مع البيانات.

```
Added Two customers
First Name = Eleanor, Address = Atlanta, Phone No = 123-4567
Get all: Name = Eleanor, Address = Atlanta, Phone No = 123-4567
Get all: Name = Mike, Address = Boston, Phone No = 467-1234
Delete: Name = Mike, Address = Boston, Phone No = 467-1234
Not Found
Update: Name = Eleanor, Address = Clayton, Phone No = 123-4567
```

الشكل رقم (١٣,٢٩). مخرجات برنامج الاختبار.

يتطلب التغيير من طريقة تخزين صفات الكائنات إلى ملفات تعاقبية إلى طريقة النشر التسلسلي للكائنات إلى إضافة الصفة <Serializable()> إلى رأس تعريف الصنف Customer ، والتي تغير كلاً من الإجراء Initialize والإجراء Terminate داخل الصنف CustomerDA. ولن تتأثر أي أصناف أخرى داخل البرنامج وسوف نستخدم أصناف الاختبار كما هي. لاحظ أن الصفة <Serializable()> يجب أن تضاف إلى الأصناف الفرعية للصنف المراد تخزين كائناته. ولاحظ أيضاً أن النشر التسلسلي يستخدم الصنف FileStream بدل كل من الصنف StreamReader والصنف StreamWriter المشتقين من الصنف Text.

ينشئ الإجراء Initialize المعرف سابقاً داخل الصنف CustomerDA كائناً من الصنف StreamWriter وكائناً من الصنف StreamReader ثم يستدعي كلاً من الإجراء WriteLine والإجراء ReadLine لتخزين صفات الكائنات واسترجاعها. أما في حالة النشر التسلسلي للكائنات فسوف يستخدم الإجراء Initialize الصنف FileStream ثم يستدعي كلاً من الإجراء Serialize والإجراء Deserialize لتخزين الكائنات واسترجاعها.

تقدم لك لغة VB .NET أسلوبين للنشر التسلسلي للكائنات. أولاً، الأسلوب SOAP الذي يستخدم لغة XML في نشر بيانات الكائنات والتي يمكن رؤيتها باستخدام أي برنامج لتحرير النصوص. ثانياً، الأسلوب Binary الذي يستخدم الأرقام الثنائية والذي يحتاج إلى استيراد (باستخدام الأمر Imports) اثنين من الفضاءات المسماة للسماح باستخدام هذا الأسلوب كما هو موضح في الأوامر التالية:

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

إذا أردت استخدام الأسلوب SOAP فيجب عليك أيضاً استيراد هذا الفضاء المسمى ولكن مع استبدال الكلمة "Binary" فقط بالكلمة "SOAP"، مع العلم أنه يجب عليك أولاً إضافة الحاوية System.Runtime.Serialization.Format.Soap وذلك باستخدام الأمر "Add References" من القائمة الفرعية Project.

سوف نناقش فقط في هذه الفقرة التغيرات التي يجب أن تعدل داخل الإجراء Initialize ، حيث ذكرنا سابقاً أنه يجب إضافة الأمر Imports لاستيراد الفضاء المسمى المناسب لأسلوب نشر الكائنات. ثم نحتاج إلى أمر إنشاء كائن من الصنف FileStream (لتعريف كائن الملف الذي سيحتوي على الكائنات) وأمر إنشاء كائن من الصنف BinaryFormatter (لنشر الكائنات باستخدام الأسلوب Binary) كما هو واضح في الأوامر التالية:

```
Dim CustomerFile As New FileStream(cFile, _
   FileMode.OpenOrCreate, FileAccess.Read)
Dim bfCustomer As BinaryFormatter = New BinaryFormatter()
```

يجب استدعاء الإجراء Deserialize داخل التكرار Do-Loop لكي يسترجع كائنات الصنف Customer ثم يضيف مؤشرات لهذه الكائنات داخل المصفوفة Customers. ولكن لاحظ أنه يجب تحويل نوع صنف الكائنات العائدة إلى Customer باستخدام الإجراء CType كما هو واضح في الأمر التالي:

```
cust = CType(bfCustomer.Deserialize(CustomerFile), Customer)
```

ينتهي عمل التكرار Do-Loop عندما نصل إلى نهاية الملف (أي عندما يصبح مؤشر الملف الحالي يساوي قيمة طول الملف الحالي). وإذا حدث خطأ عند استرجاع الكائنات من الملف يظهر استثناء مباشرة على الشاشة باستخدام الإجراء WriteLine المعرف داخل الصنف Console. يوضح الشكل رقم (١٣,٣٠) أوامر الإجراء Initialize المعرف داخل CustomerDA بعد إجراء التعديلات المطلوبة.

```
' Initialize Method
Public Shared Sub Initialize()
   Dim name, address, phoneNo As String
   ' Check for file existence that includes data
   If File.Exists(cFile) And cFile.Length > 0 Then
      Try
         ' Declare a FileStream and point to file
         Dim CustomerFile As New FileStream(cFile, _
         FileMode.OpenOrCreate, FileAccess.Read)
         Dim bfCustomer As BinaryFormatter = New BinaryFormatter()
         Dim cust As Customer
         CustomerFile.Seek(0, SeekOrigin.Begin)
         ' Loop over file until eof
         Do Until CustomerFile.Position = CustomerFile.Length
            ' Must convert type to customer
            cust = CType(bfCustomer.Deserialize(CustomerFile), _
                         Customer)
            name = cust.GetName
            address = cust.GetAddress
            phoneNo = cust.GetPhoneNo
            ' Add customer to ArrayList Customers if Name present
            If name <> Nothing Then
               customers.Add(New Customer(name, address, phoneNo))
            End If
         Loop
         CustomerFile.Close()
      Catch e As Exception
         Console.WriteLine(e.ToString)
      End Try
   End If
End Sub
```

الشكل رقم (١٣,٣٠). إجراء Initialize المعرف داخل الصنف CustomerDA.

بينما كان الإجراء Terminate، المعرف سابقاً داخل الصنف CustomerDA لتخزين كائنات الصنف Customer (طريقة تخزين الملفات)، ينشئ كائناً من الصنف StreamWriter ثم يستدعي الإجراء WriteLine للكتابة داخل الملف التعاقبي، فإن الأمر الآن يتطلب تغيير أوامر هذا الإجراء لاستخدام أسلوب النشر التسلسلي؛ ولذلك سوف نعرف كائناً من الصنف FileStream ثم نستدعي الإجراء Serialize لتخزين كائنات الصنف Customer بدلاً من كتابة قيم

صفات هذه الكائنات (يجب استخدام الكلمة Append في تعريف كائن الصنف FileStream). وكما عرفنا كائناً من الصنف BinaryFormatter لاستدعاء الإجراء Deserialize سابقاً، فنحتاج أن نعرفه ثانية داخل الإجراء Terminate لاستدعاء الإجراء Serialize مع بقاء أوامر Terminate المعرفة سابقاً كما هي دون تغيير. يوضح الشكل رقم (١٣,٣١) أوامر الإجراء Terminate بعد إجراء التعديلات المطلوبة لاستخدام النشر التسلسلي في تخزين بيانات العملاء. لاحظ أنه يجب غلق ملف النشر التسلسلي باستدعاء الإجراء Close من كائن الصنف FileStream بعد تخزين البيانات.

الآن وقد انتهينا من تعديل صنف التعامل مع البيانات CustomerDA لجعل كائنات الصنف Customer (بيانات العملاء) مستمرة عبر الوقت باستخدام أسلوب النشر التسلسلي. ولا تنسى أن تضيف العبارة <Serializable()> إلى تعريف الصنف Customer التي تؤدي إلى إمكانية نشر كائناته داخل ملف. مع العلم أننا سوف نستخدم برنامج الاختبار السابق نفسه لتجربة النشر التسلسلي في تخزين الكائنات.

```
' Terminate Method
Public Shared Sub Terminate()

    Try
        ' Instantiate a FileStream--Append (Note True)
        Dim CustomerFile As New FileStream(cFile, FileMode.Append)
        Dim bfCustomer As BinaryFormatter = New BinaryFormatter()
        Dim cust As Customer
        ' Define string variables to hold object attributes
        Dim name, address, phoneNo As String
        ' Loop over customers ArrayList -- getting and saving attributes
        For Each cust In customers
            '-----------------------------
            name = cust.GetName
            address = cust.GetAddress
            phoneNo = cust.GetPhoneNo
            '-----------------------------
            ' Write (serialize) the object to the file
            bfCustomer.Serialize(CustomerFile, cust)

        Next
        CustomerFile.Close()
    Catch e As Exception
        Console.WriteLine(e.ToString)
    End Try
End Sub
```

الشكل رقم (١٣,٣١). إجراء Terminate المعرف داخل الصنف CustomerDA.

استخدام قواعد البيانات العلائقية مع لغة VB .NET
Using Relational Databases with VB .NET

نفترض في هذا الكتاب أن القارئ لديه فكرة عن البرنامج مايكروسوفت أكسس (Microsoft Access) ويستطيع أن يقوم بعمليات قواعد البيانات الأساسية، حيث نقدم في هذه الفقرة نظرة سريعة عن قواعد البيانات العلائقية (Relational Databases) وكيف نستخدم لغة VB .NET لكي نتعامل مع البيانات ونعالجها داخل قواعد

البيانات. ويشير مصطلح قواعد البيانات العلائقية إلى نظم إدارة قواعد البيانات (Database Management Systems) التي تقدم خدمات الاستعلام والمعالجة والتي تختصر بالاختصار DBMSs.

تقدم قواعد البيانات العلائقية أدوات للمبرمج التي يتمكن من خلالها أن يخزن البيانات وينظمها داخل جداول، حيث يمثل كل عمود داخل الجدول بحقل (Field) ويمثل كل صنف سجلاً (Record). كما هو واضح في الشكل رقم (١٣,٣٢) الذي يعرض جداول تحتوي على بيانات العملاء، يستخدم حقل رقم الهاتف (PhoneNo) للتمييز بين العملاء ولذلك تطلق عليه اسم المفتاح الأساسي (سوف نصف المفاتيح الأساسية بالتفصيل في الفصل الرابع عشر).

سوف نتعلم في الفقرات التالية كيف نستخدم أوامر لغة SQL لكي نتعامل مع قواعد البيانات العلائقية. سوف ننشئ مشروعاً جديداً ثم نطور صنف التعامل مع البيانات CustomerDA اللي سيحتوي على إجراءات استمرارية الكائنات باستخدام قواعد البيانات.

CustomerTable : Table

Name	Address	PhoneNo
Eleanor	Atlanta	123-4567
Mike	Boston	467-1234

الشكل رقم (١٣,٣٢). الجدول Customer.

لغة الاستعلام الهيكلية
Structured Query Language

تعتبر لغة الاستعلام الهيكلية SQL من أشهر اللغات التي تستخدم لإدارة واستعلام قواعد البيانات العلائقية واستعلامها، حيث سنقدم في هذه الفقرة أساسيات لغة SQL ونشرحها، وسنقدم أيضاً أهم أربعة أوامر توجد في هذه اللغة. سوف نناقش تطبيقات قواعد بيانات أكثر تعقيداً خلال الفصل الرابع عشر.

لقد أوضحنا سابقاً أن الصنف CustomerDA يقدم أربع عمليات أساسية وهي: استرجاع بيانات عميل، وتخزين بيانات عميل، وتغيير بيانات عميل، وحذف بيانات عميل. ولقد رأينا أيضاً أن هذه العمليات يتم تطويرها بواسطة الإجراء Find والإجراء AddNew والإجراء Update والإجراء Delete. وعندما نستخدم قواعد البيانات لتخزين بيانات العملاء فسوف نرى أن هذه الإجراءات سوف تقوم بعملها بواسطة تنفيذ أوامر SQL.

يستخدم الإجراء Find الأمر SELECT المعرف داخل لغة SQL لاسترجاع بيانات سجل عميل من قاعدة البيانات. سوف نتبع كتابة الكلمات المحجوزة للغة SQL بالحروف الكبيرة (Capital) التي تميزها عن الكلمات التي

يعرفها المستخدم مع أن لغة SQL ليست حساسة لحالة الأحرف. يوضح الأمر SELECT التالي كيف نسترجع بيانات العميل Eleanor:

```
SELECT Name, Address, PhoneNo
    FROM CustomerTable
    WHERE PhoneNo = '123-4567'
```

يأتي بعد الكلمة المحجوزة SELECT أسماء الحقول التي نريد أن نستخلص بياناتها (الحقل Name، والحقل Address، والحقل PhoneNo). ولأن قاعدة البيانات تحتوي على أكثر من جدول فيمكننا أن نستخلص بيانات من حقول لجداول مختلفة. ويأتي بعد الكلمة FROM أسماء الجداول التي سوف نستخلص منها الحقول. وبما أن الحقول التي نريدها توجد في جدول واحد (الجدول CustomerTable) فقد احتوى هذا الأمر على اسم هذا الجدول فقط. أما الكلمة WHERE فتقيد عملية اختيار السجلات حيث يأتي بعدها شرط أو أكثر يحدد معيار السجلات المراد عرضها. ولأننا نريد عرض بيانات العميل Eleanor، فإننا قيدنا عملية البحث برقم هاتف هذا العميل "٤٥٦٧-١٢٣". تذكر أن رقم الهاتف يمثل المفتاح الأساسي للجدول والذي يجب ألا يتكرر. وبمعنى آخر لقد استخدمنا الأمر SELECT هنا لعرض بيانات عميل واحد فقط (لاحظ استخدام الرمز ' مع القيم النصية الثابتة).

يستخدم الإجراء AddNew الأمر INSERT INTO المعرف في لغة SQL لإضافة سجل جديد داخل جدول. فعلى سبيل المثال يوضح الأمر التالي إضافة سجل لعميل جديد داخل الجدول CustomerTable حيث نتبع أسماء للحقول المراد إضافة قيم لها بعد اسم الجدول. ولكن يجب أن نحددها بالأقواس ثم نكتب قائمة القيم المتناظرة لهذه الحقول بعد الكلمة VALUES بين أقواس أيضاً. أما إذا أردنا إضافة بيانات لجميع الحقول عندئذ نحذف أسماء الحقول ونكتفي بكتابة جميع القيم بالترتيب كما هو واضح في الأمر الذي يلي الأمر التالي. ولاحظ أننا نستخدم قيماً نصية ثابتة (Literals) لبيانات العميل والذي يبدو لنا مثالاً غير عملي، حيث أننا نحتاج أن نستبدل هذه القيم بمتغيرات تحتوي على قيم سجل العميل المراد إضافته كما سوف نوضح في المثال التالي:

```
INSERT INTO CustomerTable (Name, Address, PhoneNo)
    VALUES ("William", "Houston", "444-3333")
                    Or
INSERT INTO CustomerTable
    VALUES ("William" , "Houston" , "444-3333")
```

يوظف الإجراء Update الأمر UPDATE المعرف داخل لغة SQL لتغيير محتويات حقل أو أكثر داخل سجل. فعلى سبيل المثال يستبدل الأمر التالي اسم العميل Eleanor وعنوانه بالبيانات المتواجدة داخل المتغير name والمتغير address:

```
UPDATE CustomerTable Set Name = name, Address = address
WHERE PhoneNo = '756-4321'
```

ينفذ الإجراء Delete الأمر DELETE المعرف داخل لغة SQL بحذف سجل من الجدول حيث يجب أن نحدد قيمة المفتاح الأساسي لهذا السجل. يوضح الأمر التالي حذف سجل العميل الذي يملك رقم الهاتف "٧٥٦-٤٣٢١":

```
DELETE FROM CustomerTable WHERE PhoneNo = '756-4321'
```

التعامل مع نظام إدارة قواعد البيانات من خلال لغة VB .NET

Accessing a DBMS with VB .NET

يوجد لدى شركة مايكروسوفت تاريخ طويل في تقديم تقنيات التعامل مع البيانات، حيث يمكن أن نستخدم تقنية ADO (Active-X Data Objects) أو تقنية ADO .NET لتخزين البيانات ومعالجتها. تتبع تقنية ADO نموذج الخادم والعميل حيث يجب أن يتواجد اتصال دائم بين برنامج VB .NET (العميل) ومصدر البيانات (الخادم). وذلك على عكس تقنية ADO .NET التي لا تتطلب اتصالاً دائماً بل يتم الاتصال بمصدر البيانات عندما نحتاج أن نخزن البيانات أو نعدلها. تعتمد تقنية ADO على مفهوم recordset في تخزين البيانات ومعالجتها، بينما تعتمد تقنية ADO .NET على مفهوم datasets في تخزين البيانات ومعالجتها. وفي الحقيقة إن مفهوم dataset ما هي إلا مجموعة من البيانات تخزن لدى العميل وتعالج لحين إعادة تخزينها إلى مصدر البيانات مرة ثانية. كما تحتوي لغة VB .NET على صنف خاص يسمى DataReader المصمم لقراءة البيانات فقط من مصدر البيانات.

أصناف VB .NET لمعالجة قواعد البيانات

VB .NET Database Access Classes

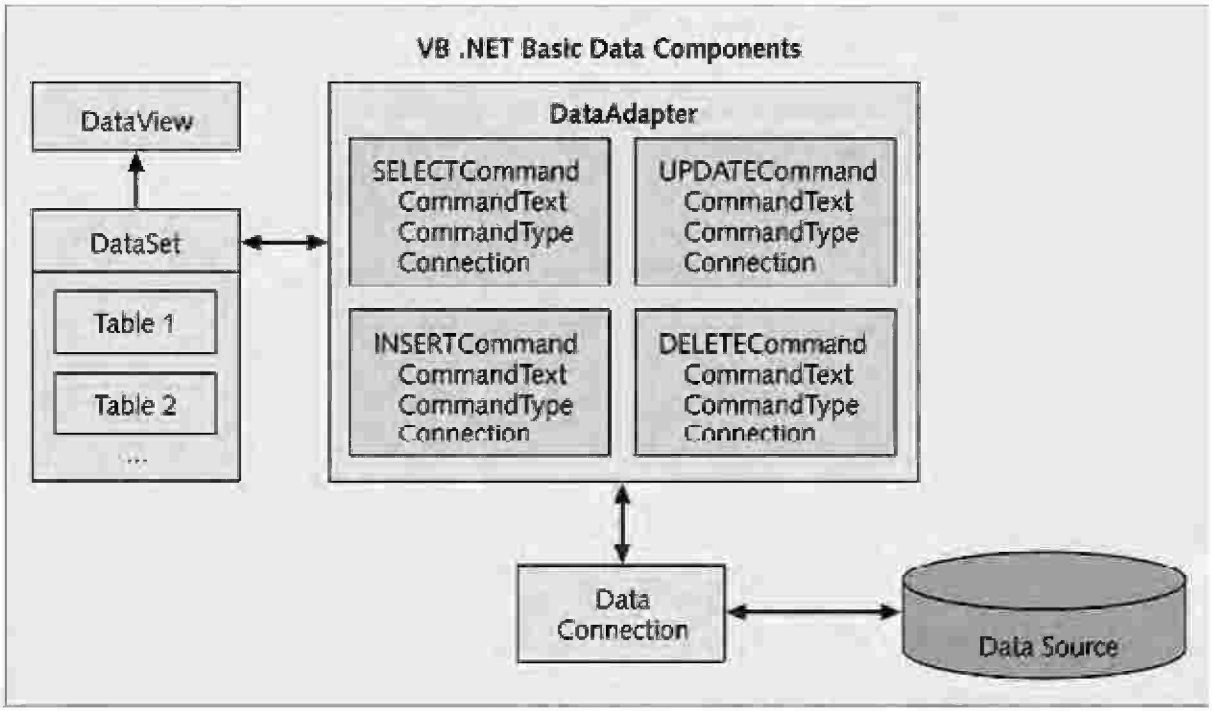
تستخدم جميع أمثلة هذا الكتاب قواعد بيانات مايكروسوفت أكسس، مع العلم أن لغة VB .NET تقدم نوعين من مزودات البيانات (Managed Data Providers) يشار إليها أحياناً بمزودات .نت (.NET Providers)؛ وذلك للتزود بالبيانات من مصدر البيانات والتعامل معها. أولاً، مزود البيانات OleDb (وهو اختصار للعبارة "Object Linking and Embedding DB") والمصمم لقواعد بيانات مايكروسوفت أكسس وأي مصدر آخر للبيانات عدا قاعدة بيانات SQLServer. ثانياً، مزود بيانات SQLServer المصمم خصيصاً لقواعد البيانات SQLServer حيث إنه لا يحتاج أن يمر من خلال طبقة OleDb بل يتعامل مباشرة مع قاعدة بيانات SQLServer. قدمت أيضاً شركة مايكروسوفت مزود بيانات خاص بقاعدة بيانات Oracle. ومن المتوقع إصدار مزودات بيانات أخرى في المستقبل القريب للتعامل مع قواعد بيانات الشركات الأخرى. يقدم المكون ADO .NET خدمة التعامل مع أي مصدر

للبيانات بما فيها قواعد البيانات العلاقية. كما يحتوي هذا المكون على جزء كبير لدعم التعامل بلغة XML (اللغة التي تنمو بشكل مطرد). بالإضافة إلى مزودات البيانات المذكورة أعلاه، فإن المكون ODBC (اختصار للعبارة "Object Database Connectivity") يمكن استخدامه للتعامل مع معظم مصادر البيانات. وأصناف مزود البيانات OleDb يمكن استخدامها أيضاً للتعامل مع قواعد بيانات SQLServer، لكن ليس بكفاءة استخدام أصناف مزود البيانات SQLServer نفسها (الذي يطلق عليه اسم SQLClient) في التعامل معها؛ لذلك سوف نستخدم أصناف مزود البيانات OleDb في جميع أمثلة هذا الكتاب، مع العلم أننا سنحتاج فقط إلى تعديلات طفيفة في هذه البرامج لاستعمال كل من مزود البيانات SQLClient أو استخدام المكون ODBC.

يحتوي كل من الفضاء المسمى OleDb والفضاء المسمى SQLClient على أصناف كل من مزود البيانات OleDb ومزود البيانات SQLServer على التوالي؛ ولذلك يجب استخدام أوامر Imports التالية:

```
Imports System.Data
Imports System.Data.OleDb
Imports System.Data.SqLClient
```

لاحظ أن الفضاء المسمى Data قد تم الإشارة إليه أيضاً باستخدام الأمر Imports في الأوامر السابقة؛ وذلك للحصول على أكبر استفادة ممكنة عند التعامل مع البيانات. ولاحظ أيضاً أن أمر Imports الخاص بمزود البيانات SQLClient لم يستخدم في هذا الكتاب ولكن ذكر فقط لتوضيح كيفية استخدامه. ويوضح الشكل رقم (١٣,٣٣) منظوراً عاماً على كيفية استخدام لغة VB .NET لإنشاء كائنات الصنف DataSet ومعالجتها. نستطيع أن نستخدم أي نوع من أنواع مصادر البيانات، ولكن سنستخدم في هذا الكتاب قاعدة بيانات أكسس فقط. وكما هو واضح من الشكل فإن الصنف Data Connection يعمل على ربط مصدر البيانات (Data Source) بالصنف DataAdapter الذي يعتبر أكثر أصناف مكتبة ADO .NET تعقيداً، حيث يتضح لنا وجود أربعة صناديق تحتوي على أربعة أوامر (الأمر SELECT، والأمر UPDATE، والأمر INSERT، والأمر DELETE) التي تناظر الأوامر الأربعة الأساسية المعرفة داخل لغة SQL. وفي الحقيقة فإنه قد تم تصميم الصنف DataAdapter لكي يحتوي على أربعة متغيرات إشارة لأربعة كائنات من الصنف Command لكي تناظر العمليات الأربعة الأساسية المطلوبة في معالجة البيانات.



الشكل رقم (١٣,٣٣). مكونات بيانات لغة فيجوال بيسك .نت.

وكما ذكرنا سابقاً فإن أوامر الصنف DataAdapter تناظر أربعة كائنات من الصنف Command التي تحتوي على الخاصية CommandText والخاصية CommandType والخاصية Connection التي يجب أن يسند لها القيم المناسبة بناء على الأمر المطلوب. ويجب أن تعلم أنه ربما لا تحتاج أن تستخدم الأوامر الأربعة معاً في جميع التطبيقات. فعلى سبيل المثال، إن معظم التطبيقات تستخلص معلومات وتستعلمها وتعرضها فقط من قواعد البيانات. ففي هذه الحالة تحتاج أن تستخدم فقط الأمر SELECT لأن الأمر لا يتطلب تعديل البيانات الحالية أو الإضافة إليها أو الحذف منها.

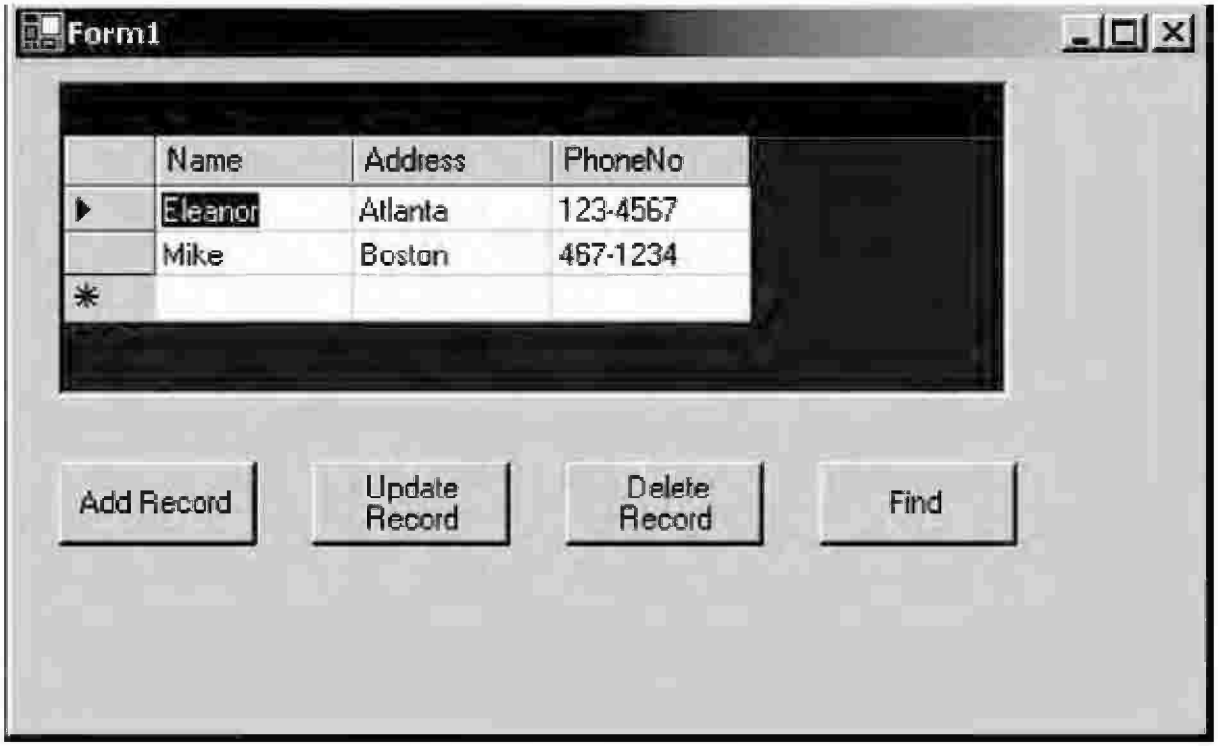
يوجد على يسار النموذج الموضح في الشكل رقم (١٣,٣٣) كلاً من الصنف DataSet والصنف DataView. يمكن إنشاء DataView من أي DataSet والذي يمكن أن يقدم العديد من الإمكانات المفيدة أثناء تطوير النظام. ولكن لا نستطيع أن نعطي هذا الصنف كاملاً في هذا الكتاب. يتكون الصنف DataSet من مجموعة من الجداول (Tables) التي تتكون من أعمدة وصفوف. ومن ثم يمكن معالجتها مثل قواعد البيانات العلاقية تماماً وذلك بغض النظر عن نوع مصدر البيانات. فعلى سبيل المثال، يمكن تكوين جداول داخل DataSet من ملف يحتوي على لغة XML (التحويل إلى بيانات هيكلية – أعمدة وصفوف). ويجب التنويه مرة أخرى على أن كائن الصنف DataSet لا يتصل بمصدر بياناته اتصالاً دائماً، لكن يتم الاتصال فقط عندما توجد حاجة لقراءة البيانات من المصدر أو عندما توجد حاجة لتعديل البيانات داخل المصدر.

لغة VB .NET ومثال نظام إدارة قاعدة بيانات
VB .NET and DBMS Example

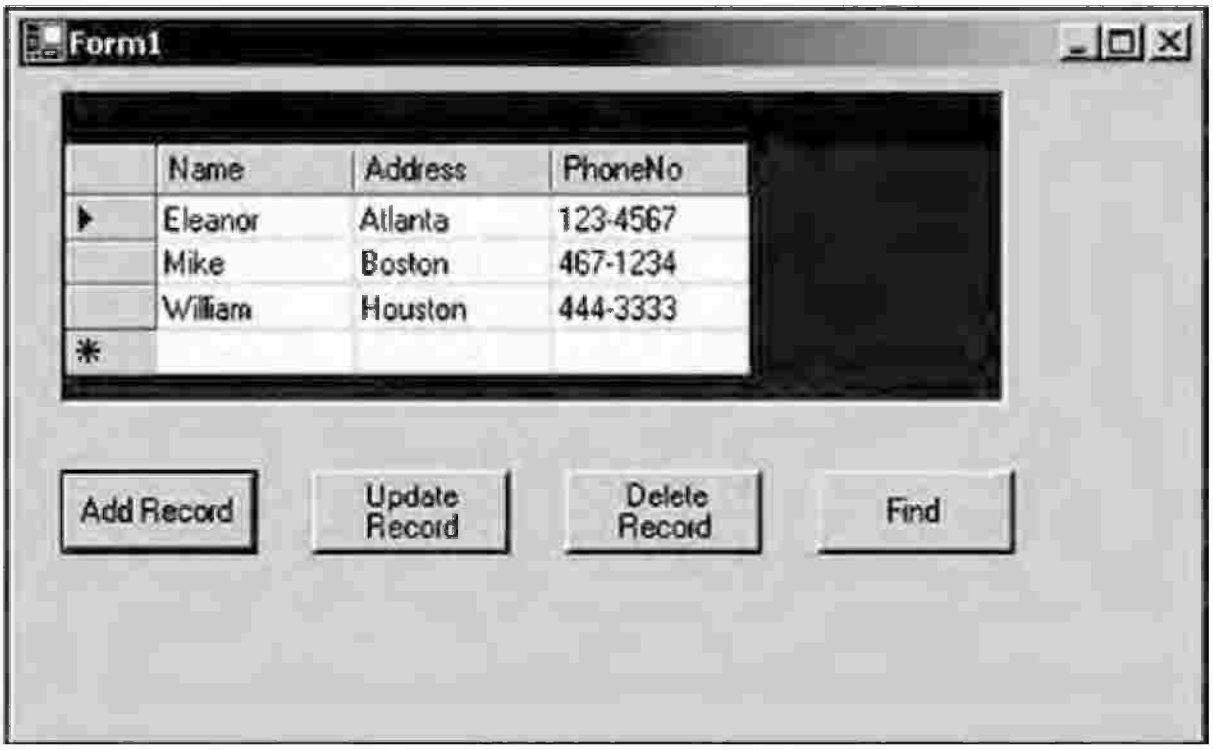
يقدم لك المشروع التالي المفاهيم الأساسية والأوامر الضرورية لكي تتفاعل مع البيانات وتعالجها داخل قاعدة البيانات العلاقية، حيث يحتوي هذا المشروع على وظيفة الاستعلام (الاختيار) والإضافة والتعديل والحذف لسجلات في جدول؛ ولذلك يشمل نموذج واجهة الاستخدام على عنصر DataGrid (جدول لعرض البيانات) وأربعة أزرار وهي: الزر "Add Record"، والزر "Update Record"، والزر "Delete Record"، والزر "Find".

توضح الأشكال بداية من الشكل رقم (١٣,٣٤) حتى الشكل رقم (١٣,٣٩) نماذج واجهة الاستخدام عندما يتم تشغيل البرنامج وعندما يتم الضغط على أزرار نموذج البرنامج بالترتيب من اليسار إلى اليمين، حيث يوضح الشكل رقم (١٣,٣٤) نموذج بداية تشغيل المشروع، ويوضح الشكل رقم (١٣,٣٥) نتيجة الضغط على زر "Add Record"، بينما يوضح الشكل رقم (١٣,٣٦) نتيجة الضغط على زر "Update Record"، حيث تم تغيير عنوان العميل الثالث من "Houston" إلى "Chicago". ويوضح الشكل رقم (١٣,٣٧) نتيجة الضغط على الزر "Delete Record" لحذف سجل العميل "William"، بينما يوضح الشكل رقم (١٣,٣٨) نتيجة الضغط على الزر "Find"، حيث ستظهر رسالة لإدخال رقم هاتف العميل المراد والبحث عنه ثم تظهر رسالة توضح بيانات العميل بعد الحصول عليه كما هو واضح في الشكل رقم (١٣,٣٩).

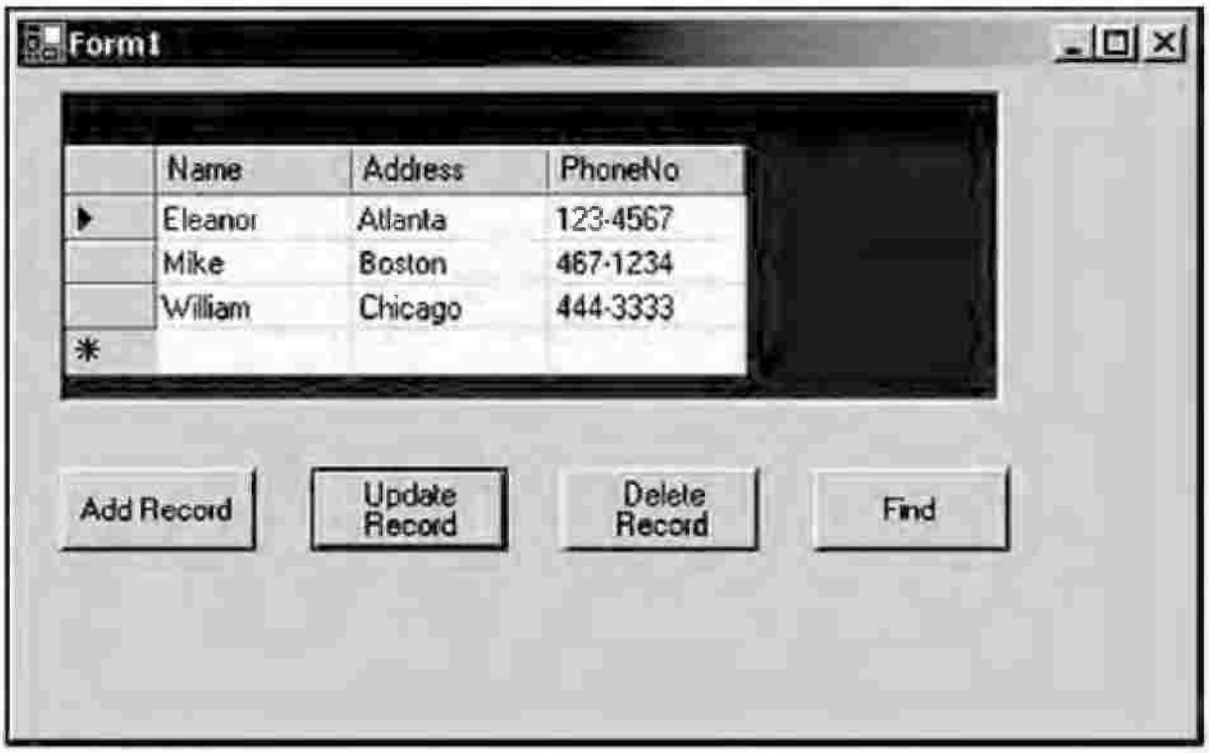
يستخدم هذا البرنامج قاعدة بيانات أكسس تسمى "Customers.mdb" التي تم تخزينها داخل المجلد الافتراضي للمشروع (المجلد bin)؛ ولذلك تم الإشارة إلى الملف بالمسار الحالي، كما يمكن استخدام المسار المطلق (أي بداية من اسم القرص الصلب إلى المجلد الذي يحتوي على الملف). تحتوي قاعدة البيانات على الجدول CustomerTable الذي يحتوي على سجلين فقط. يقوم البرنامج بالاتصال بقاعدة البيانات وذلك بكتابة الأوامر في مكان واحد داخل البرنامج، حيث يمكن استخدام هذا الاتصال في جميع إجراءات البرنامج. وكما هو واضح من الشكل رقم (١٣,٤٠) تم تعريف الاتصال بقاعدة البيانات أسفل تعريف رأس النموذج مباشرة. ثم يأتي الإجراء المسؤول عن إرجاع جميع صفوف جدول قاعدة البيانات وأعمدته داخل كائن من الصنف DataSet ثم ربطها بعنصر واجهة الاستخدام DataGrid (جدول لعرض البيانات على النموذج).



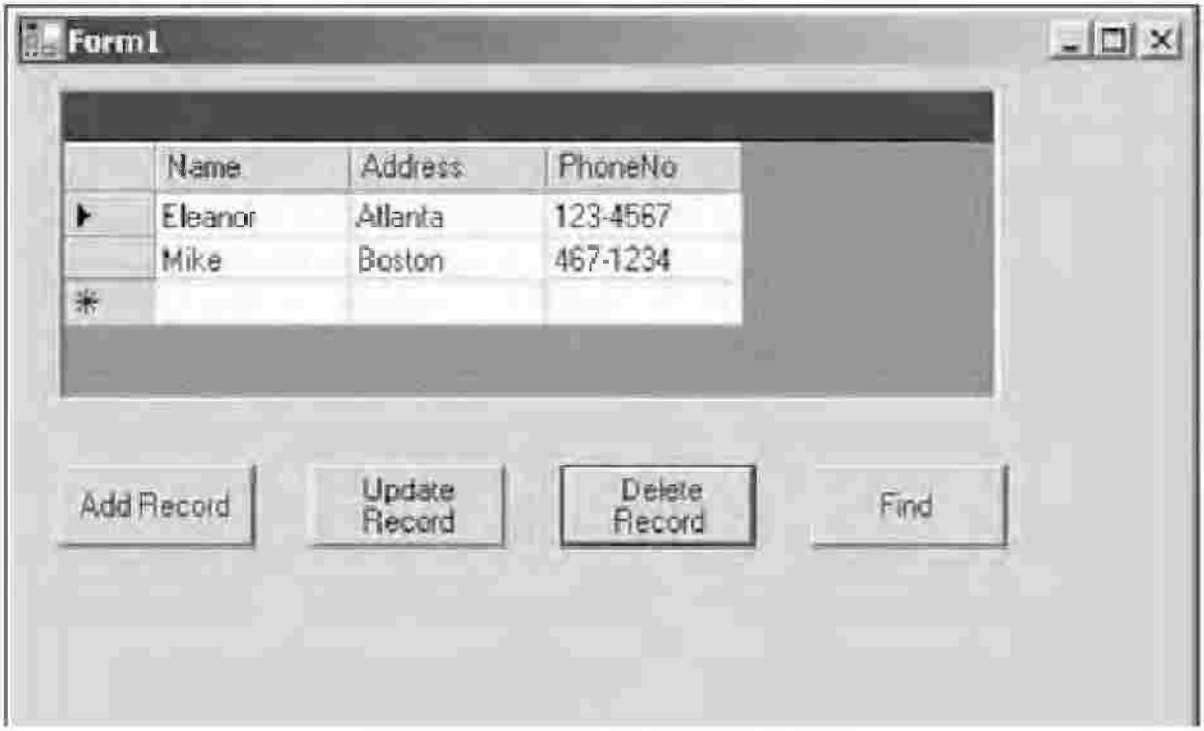
الشكل رقم (١٣,٣٤). شبكة البيانات عند البداية.



الشكل رقم (١٣,٣٥). شبكة البيانات بعد إضافة سجل.



الشكل رقم (١٣,٣٦). شبكة البيانات بعد تعديل سجل.



الشكل رقم (١٣,٣٧). شبكة البيانات بعد حذف سجل.

الشكل رقم (١٣,٣٨). البحث عن سجل.

الشكل رقم (١٣,٣٩). عرض السجل.

```
' Chapter 13--Example 5
' Introduces the basic database processing
Imports System.Data
Imports System.Data.OleDb
Imports System.Runtime.Serialization.Formatters.Soap
Public Class Form1
    Inherits System.Windows.Forms.Form

   Shared cnnCustomer As New _
        OleDbConnection("Provider=Microsoft.Jet.OLEDB.4.0; " & _
        "Data Source=customers.mdb")

   Private Sub ListRecords()
      ' Declare a new instance of a dataset
      Dim dsCustomer As New DataSet()

      ' Declare a string SQL statement
      Dim sql As String = "SELECT * FROM customerTable"

      ' Declare a new instance of an adapter
      ' Select command and connection command
      Dim adpCustomer As New _
         OleDbDataAdapter(sql, cnnCustomer)

      'Fill ds and bind to datagrid
      Try
         adpCustomer.Fill(dsCustomer, "custTable")
         DataGrid1.DataSource =
         dsCustomer.Tables("custTable").DefaultView
      Catch ee As Exception
         MsgBox(ee.ToString)
      End Try
      dsCustomer = Nothing
   End Sub
```

الشكل رقم (١٣،٤٠). أوامر الاتصال وعرض السجلات.

تشير أوامر Imports إلى الأصناف التي نحتاجها لكي تتصل ببيانات قاعدة البيانات "Customers.mdb" وتعالجها. وبما أننا نتعامل مع قاعدة البيانات أكسس؛ لذلك نحتاج أن نستخدم مزود البيانات OleDb. توجد طريقتان للاتصال بأي مصدر للبيانات وهما: استخدام نص يحدد بيانات الاتصال (Connection String)، أو تعريف متغير لاسم مصدر البيانات (Data Source Name) ويختصر بالكلمة DSN. سوف نستخدم أسلوب نص بيانات الاتصال وذلك لسببين. أولاً، ستحدد بنفسك أي نوع من مزودات البيانات سيتم استخدامه وموقع وجود قاعدة البيانات المراد الاتصال بها. ثانياً، إن الاتصال بمصدر البيانات باستخدام تعريف DSN يتطلب استخدام برنامج ODBC الذي يستخدم لتعريف كل من نوع مزود البيانات ومكان مصدر البيانات من خلال مجموعة متتالية من الخطوات. فإذا كانت قاعدة البيانات المراد الاتصال بها توجد على جهاز منفصل عن الجهاز الخاص بك، فستحتاج أن تتصل بمركز الدعم الفني لدى المؤسسة الخاصة بك لتعريف DSN خاصة إذا لم يكن لديك الحق أن تقوم بذلك بنفسك. ولكن على أي حال إن استخدام DSN يجعل أمر الاتصال أقصر، حيث نحتاج أن نسند اسم متغير DSN فقط إلى كائن الاتصال.

يوضح أول جزء في نص بيانات الاتصال نوع مزود البيانات "Provider=Microsoft.Jet.OLEDB.4.0" المستخدم في الاتصال بقاعدة البيانات، حيث يستخدم هذا المزود مع قواعد البيانات أكسس ٢٠٠٠ أو أعلى. بعد ذلك يوضع الجزء الثاني موقع تخزين ملف قاعدة البيانات "Data Source=C:\Chap13\Exercises\customers.mdb" الذي يشير إلى المجلد الحالي (المجلد bin الذي يوجد داخل مجلد المشروع). يبدأ الإجراء ListRecords بإنشاء كائن من الصنف DataSet متبوعاً بتعريف متغير من النوع String الذي أسند إليه أمر SELECT المعرف داخل لغة SQL. لاحظ أن الرمز * يعني في لغة SQL اختيار جميع أعمدة الجدول.

تذكر من الشكل رقم (١٣,٣٣) أن الصنف DataAdapter يحتوي على أربع خواص تشير إلى أربعة كائنات من النوع Command (SELECT و UPDATE و INSERT و DELETE). وإن كل كائن من الصنف Command يحتوي على الخاصية CommandText والخاصية CommandType والخاصية Connection؛ ولذلك فإن الأسلوب المتبع في هذا المثال لتعريف كائن الصنف DataAdapter (كما هو واضح من الأمر التالي) يقلل الإسناد الصريح لهذه الخواص كما سيتضح لنا لاحقاً. وان الإجراء Fill سيكون مسؤولاً عن فتح الاتصال مع قاعدة البيانات وغلقه.

```
Dim adpCustomer As New OleDbDataAdapter(sql, cnnCustomer)
```

إن استدعاء إجراء إنشاء الصنف OleDbDataAdapter بهذا التوقيع يسند المتغير sql إلى الخاصية CommandText للأمر SELECT، كما يسند المتغير cnnCustomer للخاصية Connection للأمر SELECT. ويستخدم الإسناد الافتراضي "CommandType.Text" للخاصية CommandType للأمر SELECT، حيث يوجد ارتباط بين كل من الخاصية CommandType والخاصية CommandText. فمثلاً، إذا أسندنا اسم StoredProcedure (إجراء معرف داخل خادم SQL) داخل الخاصية CommandText، فيجب علينا إسناد القيمة CommandType.StoredProcedure داخل الخاصية CommandType.

وبعد إنشاء كائن من الصنف OleDbDataAdapter نريد أن نسترجع بيانات العملاء الحاليين طبقاً لأمر SELECT وتخزينها داخل جدول في كائن الصنف DataSet عن طريق استدعاء الإجراء Fill، مع العلم أن الاتصال مع قاعدة البيانات سيتم فتحه وغلقه ضمنياً داخل هذا الإجراء. ولقد وضعنا كلاً من استدعاء الإجراء Fill وأمر ربط كائن DataGrid (المسمى DataGrid1) بالجدول الموجود داخل كائن DataSet (الذي يسمى CustomerTable) داخل التركيب Try تحسباً لحدوث خطأ كما هو واضح في الأوامر التالية:

```
Try
   adpCustomer.Fill(dsCustomer, "custTable")
   DataGrid1.DataSource = dsCustomer.Tables("custTable").DefaultView
Catch e As Exception
   MsgBox(e.ToString)
End Try
```

استقبل الإجراء Fill اسم كائن صنف DataSet (المسمى dsCustomer) كمعامل أول واستقبل اسم الجدول الذي سيتم إنشاؤه داخل كائن DataSet (المسمى CustTable) كمعامل ثان. لاحظ أن تعريف اسم الجدول هام جداً ؛ وذلك لكي نتمكن من التعامل مع حقول الجدول وصفوفه. ثم يأتي أمر ربط كائن DataGrid (المسمى DataGrid1) بالرؤية الافتراضية للجداول CustTable، حيث نسند كائن DataView الافتراضي للجدول إلى الكائن DataGrid1.

يحتوي إجراء معالجة حدث Form_Load للنموذج على استدعاء الإجراء ListRecords ؛ وذلك لجلب محتويات جدول العملاء من قاعدة البيانات وإظهارها داخل كائن DataGrid1 المتواجد على النموذج عند بداية تشغيل البرنامج. وذلك كما هو واضح في الأوامر التالية :

```
Private Sub Form1_Load(ByVal sender As System.Object, _
        ByVal e As System.EventArgs) Handles MyBase.Load
        cnnCustomer.Open()
        ListRecords()
End Sub
```

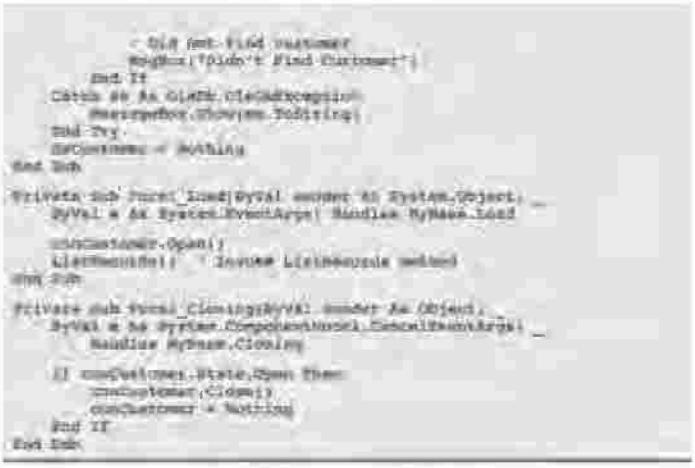
ولتنفيذ مهام كل من الزر "Add Record" والزر "Update Record" والزر "Delete Record"، سنكتب أوامر تشبه أوامر الإجراء ListRecords، مع العلم أن أوامر هذا الإجراء سترجع بيانات من قاعدة البيانات، وهو الأمر الذي يختلف مع أوامر إجراءات هذه الأزرار التي تعتمد على استعلامات تعديل من إضافة وتعديل وحذف وليس على استعلام استرجاع. ومن ثم سنستخدم كلاً من الخاصية InsertCommand والخاصية UpdateCommand والخاصية DeleteCommand المعرفة داخل الكائن OleDbDataAdapter.

يوضح الشكل رقم (١٣,٤١) إجراءات معالجة أحداث الضغط على الزر "Add Record" والزر "Update Record" والزر "Delete Record". كما يظهر هذا الشكل أيضاً إجراء معالجة الضغط على الزر "Find" الذي يعيد محتويات سجل عميل إذا وجد العميل الذي يملك رقم الهاتف المدخل. وكما هو ملاحظ فإن جميع الإجراءات عدا إجراء الزر "Find" يستدعي الإجراء ListRecords.

تتشابه كل من إجراءات الإضافة والحذف والتعديل، حيث ننشأ أمر لغة SQL لتنفيذ العملية المطلوبة أولاً. بعد ذلك ننشئ كائناً من الصنف OleDbDataAdapter وإنشاء كائن من OleDbCommand باستخدام أمر SQL السابق إنشاؤه. ثم إسناده لأحد خواص كائن الصنف OleDbDataAdapter (InsertCommand و UpdateCommand و DeleteCommand) بناء على الوظيفة المراد إنجازها. ثم إسناد كائن الاتصال بخاصية الاتصال للكائن OleDbCommand. وأخيراً يتم استدعاء الإجراء ExecuteNonQuery لتنفيذ الاستعلام. توضح الأوامر التالية هذه الأوامر الثلاثة لإجراء الزر "Delete Record". يحتوي إجراء كل من الزر "Add Record" والزر "Update Record" الأوامر نفسها، إلا أننا سنستخدم الخاصية InsertCommand أو الخاصية UpdateCommand بالترتيب.

```
Private Sub btnAdd_Click(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles btnAdd.Click

    ' Declare a string SQL statement
    Dim name As String = "William"
    Dim address As String = "Houston"
    Dim phoneNo As String = "444-3333"
    Dim sql As String = "INSERT INTO customerTable " & _
        "VALUES (" & name & "', '" & address & "', " & _
        "'" & phoneNo & "')"

    Dim adpCustomer As New OleDbDataAdapter()
    ' Assign Insert Command and Execute
    Try
        adpCustomer.InsertCommand = New OleDbCommand(sql)
        adpCustomer.InsertCommand.Connection = conCustomer
        adpCustomer.InsertCommand.ExecuteNonQuery()

        ListRecords()    ' Invoke ListRecords method
    Catch ee As Exception
        MsgBox(ee.ToString)
    End Try
End Sub

Private Sub btnUpdate_Click(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles btnUpdate.Click

    ' Declare a string SQL statement
    Dim sql As String = "UPDATE customerTable SET Address = " & _
        "'Chicago' WHERE PhoneNo = '444-1111'"

    Dim adpCustomer As New OleDbDataAdapter()
    ' Assign Update Command and Execute
    Try
        adpCustomer.UpdateCommand = New OleDbCommand(sql)
        adpCustomer.UpdateCommand.Connection = conCustomer
        adpCustomer.UpdateCommand.ExecuteNonQuery()
        ListRecords()  ' Invoke ListRecords method
    Catch ee As Exception
        MsgBox(ee.ToString)
    End Try
End Sub

Private Sub btnDelete_Click(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles btnDelete.Click

    ' Declare a string SQL statement
    Dim sql As String = "DELETE FROM customerTable WHERE " & _
        "PhoneNo = '444-3333'"
    Dim adpCustomer As New OleDbDataAdapter()
    ' Assign Delete Command and Execute
    Try
        adpCustomer.DeleteCommand = New OleDbCommand(sql)
        adpCustomer.DeleteCommand.Connection = conCustomer
        adpCustomer.DeleteCommand.ExecuteNonQuery()

        ListRecords()  ' Invoke ListRecords method
    Catch ee As Exception
        MsgBox(ee.ToString)
    End Try

End Sub

Private Sub btnFind_Click(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles btnFind.Click
    Dim sqlQuery As String
    Dim rowRow As DataRow
    Dim dsCustomer As New DataSet()
    dsCustomer.Clear()
    Dim key As String
    key = InputBox("Enter phone number", "Find Dialog", "")
    Try

        sqlQuery = "SELECT Name, Address, PhoneNo " & _
            "FROM CustomerTable " & _
            "WHERE phoneNo = '" & key & "'"
        Dim adpCustomer As New _
            OleDbDataAdapter(sqlQuery, conCustomer)
        adpCustomer.Fill(dsCustomer, "CustTable")

        If dsCustomer.Tables("CustTable").Rows.Count > 0 Then
            ' Found Customer
            Dim name, address, phoneNo, display As String
            name = _
              dsCustomer.Tables("CustTable").Rows(0).Item("Name")
            address = _
              (dsCustomer.Tables("CustTable").Rows(0).Item("address")
            phoneNo = _
              dsCustomer.Tables("CustTable").Rows(0).Item("phoneNo")
              display = "name = " & name & vbCrLf & _
                        "address = " & address & vbCrLf & _
                        "phoneNo = " & phoneNo
            MessageBox.Show(display)
        Else
```

الشكل رقم (١٣,٤١). أوامر الأزرار المعالجة.



تابع الشكل رقم (١٣,٤١).

```
sdpCustomer.DeleteCommand = New OleDbCommand(sql)
sdpCustomer.DeleteCommand.Connection = cnnCustomer
sdpCustomer.DeleteCommand.ExecuteNonQuery()
```

لاحظ أن البيانات المذكورة داخل إجراءات الإضافة والحذف والتعديل بيانات ثابتة (Hard Coded) والتي يجب أن تكون غير ذلك في التطبيقات الحقيقية، حيث يجب أن تستقبل هذه البيانات من المستخدم وذلك من خلال صناديق نص مثلاً. ولكن على أي حال إن كلاً من إجرائي الإضافة والبحث يحتويان على أوامر تحتاج إلى توضيح أكثر، حيث يحتوي أمر SQL داخل إجراء الإضافة على متغيرات بدلاً من بيانات ثابتة. لكن أمر SQL يتطلب أن تكون البيانات نصية ومحاطة بالرمز (') كما هو واضح في الأوامر التالية:

```
Dim sql As String = "INSERT INTO customerTable " & _
   "VALUES ('" & name & "', '" & address & "'," & _
   "'" & phonenum & "')"
```

وبشكل عام، فإن الصيغة العامة لاستخدام المتغير هي " & var1 & "'، حيث إن var1 يمثل متغيراً وقد تم استخدام علامة التخصيص المفردة (') لأن نوع حقل الجدول الذي نريد أن نتعامل معه من النوع النصي، ولذلك يجب أن تكون القيمة نصية وذلك بوضعها بين علامات التخصيص المفردة. أما الحقول الرقمية فتستخدم معها الصيغة " & var1 & ".

يستدعي إجراء البحث الإجراء InputBox لاستقبال رقم الهاتف المستخدم للبحث عن عميل يملك هذا الرقم. ثم ينشئ كائناً من DataSet واسترجاع البيانات من قاعدة البيانات بناء على أمر SELECT الذي يحتوي على رقم الهاتف المدخل. بعد ذلك يأتي الأمر If ليحدد هل الجدول الذي تم استرجاعه يحتوي على سجلات أم لا. فإذا

احتوى الجدول على سجلات، فيعني ذلك أن العميل موجود ومن ثم نستخلص اسمه وعنوانه ورقم هاتفه من السجل الحالي. ثم تظهر جميع هذه المعلومات داخل رسالة كما هو واضح في الأوامر التالية:

```
If dsCustomer.Tables("CustTable").Rows.Count > 0 Then
    ' Found Customer
    Dim name, address, phoneNo, display As String
    name = _
      dsCustomer.Tables("CustTable").Rows(0).Item("Name")
    address = _
      (dsCustomer.Tables("CustTable").Rows(0).Item("address"))
     phoneNo = _
       dsCustomer.Tables("CustTable").Rows(0).Item("phoneNo")
     display = "name = " & name & vbCrLf & _
                 "address = " & address & vbCrLf & _
                 "phoneNo = " & phoneNo
     MessageBox.Show(display)
Else
     ' Did not find customer
     MsgBox("Didn't Find Customer")
End If
```

تطوير استمرارية الكائنات باستخدام قواعد البيانات
Implementing Object Persistence with a Database

لقد رأينا في الأمثلة السابقة لتطوير استمرارية الكائنات باستخدام الملفات التعاقبية أو النشر التسلسلي أنه يتم أولاً استدعاء الإجراء Initialize المسؤول عن قراءة جميع العملاء من الملف وتخزينها داخل مصفوفة. بعد ذلك تتم جميع العمليات من إضافة وتعديل وحذف داخل المصفوفة. ثم يأتي دور الإجراء Terminate المسؤول عن إعادة تخزين العملاء من المصفوفة إلى الملف ثانياً.

وعلى العكس من ذلك فإنه عند تطوير استمرارية الكائنات باستخدام قواعد البيانات يتم اللجوء إلى أسلوب آخر، حيث يستخدم الإجراء Initialize لإتمام الاتصال مع قاعدة البيانات وفتحه، ثم تتم المعالجة من بحث وإضافة وتعديل وحذف على قاعدة البيانات مباشرة. ثم يأتي دور الإجراء Terminate المسؤول عن غلق الاتصال مع قاعدة البيانات لتحرير موارد النظام.

بالإضافة إلى استيراد الفضاء المسمى Collection لاستخدام المصفوفة ArrayList، يجب أيضاً أن نستورد كلاً من الفضاء المسمى Data والفضاء المسمى OleDb داخل الصنف CustomerDA هكذا:

```
Imports System.Data
Imports System.Data.OleDb
Imports System.Collections
```

بعد ذلك تقوم بتعريف متغيرات الصنف CustomerDA، حيث نعرف كلاً من متغير الإشارة cnnCustomer ليشير إلى كائن من الصنف OleDbConnection (يستخدم للاتصال بقاعدة البيانات) ومتغير الإشارة Customers ليشير إلى مصفوفة ArrayList التي ستحتوي على مؤشرات لكائنات من الصنف Customer. ثم نعرف متغير إشارة لكائن من الصنف Customer تحت اسم aCustomer وثلاثة متغيرات من النوع String لتحتوي على قيم صفات العملاء كما هو واضح في الأوامر التالية:

```
Public Class CustomerDA
    ' Declare a connection
    Shared cnnCustomer As New _
        OleDbConnection("Provider=Microsoft.Jet.OLEDB.4.0; " & _
        "Data Source=customers.mdb")
    ' Declare a new ArrayList instance
    Shared customers As New ArrayList()
    ' Declare an instance of Customer
    Shared aCustomer As Customer
    ' Declare variables for Customer attribute value
    Shared name, address, phoneno As String
```

الإجراء Initialize
The Initialize Method

الغرض الرئيس من الإجراء Initialize هو فتح الاتصال بقاعدة بيانات العملاء وذلك عن طريق استدعاء الإجراء Open من الكائن cnnCustomer الذي تم تعريفه سابقاً. ولكن يجب وضع هذا الأمر داخل التركيب Try تحسباً لحدوث أي خطأ عند الاتصال بقاعدة البيانات (انظر الشكل رقم ١٣,٤٢).

```
' Initialize Method
Public Shared Sub Initialize()

   Try
      ' Try to open the connection
      cnnCustomer.Open()
   Catch e As Exception
      Console.WriteLine(e.ToString)
   End Try
End Sub
```

الشكل رقم (١٣,٤٢). الإجراء Initialize للصنف CustomerDA.

الإجراء Terminate
The Terminate Method

من حسن الحظ أيضاً أن تعريف الإجراء Terminate في هذا المثال أسهل بكثير من مثيلاتها في الأمثلة السابقة، حيث نستدعي الإجراء Close فقط عن الكائن cnnCustomer لغلق الاتصال. ولكن تذكر أنه يجب وضع هذا الأمر داخل التركيب Try تحسباً لحدوث خطأ عند غلق قاعدة البيانات (انظر الشكل رقم ١٣,٤٣).

```
' Terminate Method
Public Shared Sub Terminate()

   Try
      cnnCustomer.Close()
      cnnCustomer = Nothing
   Catch e As Exception
      Console.WriteLine(e.Message.ToString)
   End Try
End Sub
```

الشكل رقم (١٣,٤٣). الإجراء Terminate للصنف CustomerDA.

الإجراء Find
The Find Method

إن تعريف الإجراء Find مباشر وسهل حيث يستخدم كائن الصنف OleDbDataAdapter أمر SELECT المعرف داخل لغة SQL ويستخدم كائن الصنف OleDbConnection لاسترجاع بيانات العميل الذي يملك رقم الهاتف المدخل من قاعدة البيانات ثم وضع هذه البيانات داخل كائن DataSet ثم البحث عن العميل داخل هذا الكائن. فإذا تم الحصول عليه، فعندئذ ننشئ كائناً من الصنف Customer باستخدام قيم هذا العميل ثم إرجاعه إلى الإجراء الذي استدعى الإجراء Find. أما إذا لم يتم الحصول على هذا العميل، عندئذ ننشئ كائناً من صنف الاستثناء الخاص NotFoundException ثم نلقيه كما هو واضح في الشكل رقم (١٣,٤٤).

يبدأ الإجراء بتعريف متغير إشارة للصنف Customer (المتغير aCustomer) ويسند له القيمة Nothing ثم يعرف كائناً من الصنف DataSet متبوعاً بتعريف متغير من النوع String ليحتوي على أمر SELECT. بعد ذلك يتم تعريف كائن من الصنف OleDbDataAdapter الذي يتطلب أمر SELECT كمعامل أول (أو اسم إجراء مخزن داخل خادم SQL) ويتطلب كائناً من الصنف OleDbConnection كمعامل ثان، مع العلم أن كائن OleDbDataAdapter سيتولى الاتصال بقاعدة البيانات وتنفيذ أمر SELECT واسترجاع البيانات ووضعها في جدول داخل كائن الصنف DataSet وذلك بواسطة استدعاء الإجراء Fill. تذكر أن كائن الصنف DataSet يمثل بيانات غير دائمة الاتصال.

```
' Find Method--Throws NotFoundException if Not Found
Public Shared Function Find(ByVal key As String) As Customer
   aCustomer = Nothing
   Dim dsCustomer As New DataSet()
      Try

         ' Define the SQL statement using the phoneno key
         Dim sqlQuery As String = "SELECT Name, Address, PhoneNo " & _
            "FROM CustomerTable WHERE phoneNo = '" & key & "'"
         Dim adpCustomer As New _
            OleDbDataAdapter(sqlQuery, cnnCustomer)
         adpCustomer.Fill(dsCustomer, "CustTable")
         If dsCustomer.Tables("CustTable").Rows.Count > 0 Then
            Dim custRow As DataRow
            custRow = dsCustomer.Tables("custTable").Rows(0)
            name = custRow.Item("Name")
            address = custRow.Item("address")
            phoneno = custRow.Item("phoneno")
            aCustomer = New Customer(name, address, phoneno)
         Else
            Throw New NotFoundException("Not Found")
         End If
         dsCustomer = Nothing
      Catch e As OleDb.OleDbException
         Console.WriteLine(e.Message.ToString)
      End Try
      Return aCustomer
End Function
```

الشكل رقم (١٣,٤٤). الإجراء Find للصنف CustomerDA.

بعد ذلك يأتي دور أمر If الذي يتحقق من صحة وجود العميل أم لا، حيث يستكشف هل الجدول الموجود داخل كائن الصنف DataSet يحتوي على سجلات أم لا. فإذا احتوى على سجل، فهذا يعني أن العميل موجود وعندئذ يتم استرجاع قيم العميل وإنشاء كائن من الصنف Customer يتم إرجاعه إلى الإجراء الداعي للإجراء Find. أما إذا لم يجد سجلات، فعندئذ ينشئ كائناً من صنف الاستثناء الخاص NotFoundException وإلقائه.

الإجراء AddNew
The AddNew Method

يتم استدعاء الإجراء AddNew لإضافة عميل جديد (كائن من الصنف Customer) إلى النظام، حيث يتم استخدام الأمر Insert المعرف داخل لغة SQL كما هو واضح في الشكل رقم (١٣,٤٥). يبدأ تعريف هذا الإجراء باستدعاء إجراءات الوصول Getters لاسترجاع قيم صفات كائن الصنف Customer الذي تم استقباله. ثم يستدعي الإجراء Find داخل التركيب Try ليرى هل العميل موجود بالفعل أم لا. فإذا كان العميل موجوداً، عندئذ سينشئ

كائناً من صنف الاستثناء الخاص DuplicateException ثم يلقيه. أما إذا تلقى الاستثناء NotFoundException نتيجة استدعاء الإجراء Find (أي أن العميل ليس موجود)، فعندئذ نعالج هذا الاستثناء بإضافة بيانات العميل الجديد داخل قاعدة البيانات بكتابة الأوامر المطلوبة بعد الكلمة المحجوزة Catch. لاحظ أن كلاً من عملية الإضافة والحذف والتعديل لا تعيد بيانات من قاعدة البيانات نتيجة تنفيذ أمر SQL ؛ ولذلك لا نحتاج أن نعرف كائناً من الصنف DataSet ومن ثم نستخدم خواص كائن الصنف OleDbDataAdapter مباشرة (الخاصية InsertCommand والخاصية UpdateCommand والخاصية DeleteCommand). وبالعودة إلى الإجراء AddNew نحتاج أن نكتب ثلاثة أوامر. أولاً، إنشاء كائن من الصنف OleDbCommand باستخدام المتغير SQL INSERT (الذي يحتوي على أمر Insert المعرف سابقاً) داخل الخاصية InsertCommand. ثانياً، إسناد المتغير cnnCustomer للخاصية InsertCommand التي تعيد لنا كائناً من الصنف OleDbCommand. ثالثاً، استدعاء الإجراء ExecuteNonQuery من الخاصية InsertCommand التي تعيد كائناً من الصنف oleDbCommand لتنفيذ أمر Insert المعرف سابقاً.

```
' AddNew Method--Throws DuplicateException if exists
Public Shared Sub AddNew(ByVal aCustomer As Customer)
    ' Get customer information
   name = aCustomer.GetName
   address = aCustomer.GetAddress
   phoneno = aCustomer.GetPhoneNo
   ' Declare a string SQL statement
   Dim sqlInsert As String = "INSERT INTO customerTable " & _
      "VALUES ('" & name & "', '" & address & "'," & _
      "'" & phoneno & " ')"

   Try
      Dim c As Customer = Customer.Find(phoneno)
      Throw New DuplicateException(" Customer Exists ")
   Catch e As NotFoundException
      Try
         Dim adpcustomer As New OleDbDataAdapter()
         adpcustomer.InsertCommand = New OleDbCommand(sqlInsert)
         adpcustomer.InsertCommand.Connection = cnnCustomer
         adpcustomer.InsertCommand.ExecuteNonQuery()
      Catch sqle As OleDb.OleDbException
         Console.WriteLine(sqle.ToString)
      End Try
   End Try
End Sub
```

الشكل رقم (١٣,٤٥). الإجراء AddNew للصنف CustomerDA.

الإجراء Update

The Update Method

يبدأ الإجراء Update باستدعاء إجراءات الوصول Getters لاسترجاع قيم صفات كائن الصنف Customer (العميل المراد تعديل بياناته). بعد ذلك يتم استدعاء الإجراء Find للتأكد من وجود العميل في قاعدة البيانات. فإذا

لم يتم الحصول على العميل، فعندئذ سيرسل الإجراء Find الاستثناء NotFoundException، وبذلك سوف يعاد إرساله بواسطة الإجراء Update للإجراء الداعي له. أما إذا تم الحصول على العميل، عندئذ سيتم استدعاء الإجراء ExecuteNonQuery لتنفيذ أمر Update، المعرف داخل لغة SQL، لتعديل بيانات العميل داخل قاعدة البيانات كما هو موضح في الشكل رقم (١٣,٤٦).

```
'Delete Method--Throws NotFoundException if not found
Public Shared Sub Delete(ByVal aCustomer As Customer)
   Dim phoneno, sqlDelete As String
   phoneno = aCustomer.GetPhoneNo
   sqlDelete = "DELETE FROM CustomerTable " & _
      "WHERE PhoneNo = '" & phoneno & "'"
   Try
      Dim c As Customer = Customer.Find(phoneno)
      Dim adpCustomer As New OleDbDataAdapter()
      adpCustomer.DeleteCommand = New OleDbCommand(sqlDelete)
      adpCustomer.DeleteCommand.Connection = cnnCustomer
      adpCustomer.DeleteCommand.ExecuteNonQuery()
   Catch e As Exception
      Console.WriteLine(e.ToString)
   End Try
End Sub
```

الشكل رقم (١٣,٤٦). الإجراء Update للصنف CustomerDA.

الإجراء Delete
The Delete Method

يتشابه الإجراء Delete مع الإجراء Update حيث يستدعي أولاً إجراء الوصول Getters للحصول على رقم هاتف العميل المراد حذفه من قاعدة البيانات. بعد ذلك يتم استدعاء الإجراء Find لمعرفة هل هذا العميل موجود في قاعدة البيانات أم لا. إذا لم يوجد العميل سيتم إلقاء الاستثناء NotFoundException إلى الإجراء الداعي للإجراء Delete. أما إذا وجد العميل، فعندئذ سيتم تنفيذ الإجراء ExecuteNonQuery الذي ينفذ الأمر Delete، المعرف داخل لغة SQL، لحذف العميل من قاعدة البيانات كما هو واضح في الشكل رقم (١٣,٤٧).

```
'Delete Method--Throws NotFoundException if not found
Public Shared Sub Delete(ByVal aCustomer As Customer)
   Dim phoneno, sqlDelete As String
   phoneno = aCustomer.GetPhoneNo
   sqlDelete = "DELETE FROM CustomerTable " & _
      "WHERE PhoneNo = '" & phoneno & "'"
   Try
      Dim c As Customer = Customer.Find(phoneno)
      Dim adpCustomer As New OleDbDataAdapter()
      adpCustomer.DeleteCommand = New OleDbCommand(sqlDelete)
      adpCustomer.DeleteCommand.Connection = cnnCustomer
      adpCustomer.DeleteCommand.ExecuteNonQuery()
   Catch e As Exception
      Console.WriteLine(e.ToString)
   End Try
End Sub
```

الشكل رقم (١٣,٤٧). الإجراء Delete للصنف CustomerDA.

الإجراء GetAll
The GetAll Method

يعيد الإجراء GetAll مصفوفة تحتوي على جميع العملاء المخزنين في قاعدة البيانات (كائنات من الصنف Customer) كما هو واضح في الشكل رقم (١٣,٤٨). مثل الإجراء Find فإن الإجراء GetAll يستخدم الخاصية SelectCommand المعرفة داخل الصنف OleDbDataAdapter ويستدعي الإجراء Fill الذي ينفذ الأمر SELECT لاسترجاع جميع حقول الجدول CustomerTable. وجميع سجلات الجدول يتم نقلها داخل الجدول CustTable في كائن الصنف DataSet. بعد ذلك يتجول التكرار For داخل سجلات الجدول CustTable حيث تسترجع قيم سجل في كل مرة ثم إنشاء كائن من الصنف Customer وإضافته إلى المصفوفة Customers. في النهاية بعد تكوين المصفوفة يتم إرجاعها إلى الإجراء الداعي للإجراء GetAll.

```
' GetAll Method
Public Shared Function GetAll() As ArrayList
   Dim dsCustomer As New DataSet()
   Dim sqlQuery As String = "SELECT Name, Address, PhoneNo " & _
      "FROM CustomerTable"
   Try
      Dim adpCustomer As New _
         OleDbDataAdapter(sqlQuery, cnnCustomer)
      adpCustomer.Fill(dsCustomer, "CustTable")
      If dsCustomer.Tables("CustTable").Rows.Count > 0 Then
         Dim dsRow As DataRow
         'Clear the array list customers.clear()
         For Each dsRow In dsCustomer.Tables("CustTable").Rows
            name = dsRow("Name")
            address = dsRow("Address")
            phoneno = dsRow("PhoneNo")
            Dim aCustomer As New _
               Customer(name, address, phoneno)
            customers.Add(aCustomer)
         Next
      Else
      ' No records in database
      End If
      deCustomer = Nothing
   Catch e As Exception
      Console.WriteLine(e.ToString)
   End Try
   Return customers
End Function
```

الشكل رقم (١٣,٤٨). الإجراء GetAll للصنف CustomerDA.

اختبار الصنف CustomerDA بعد تطوير التعامل مع قواعد البيانات

Testing CustomerDA for a Database Implementation

يمكنك اختبار الصنف CustomerDA بعد تطوير التعامل مع قاعدة البيانات العلائقية باستخدام الصنف Customer نفسه برامج الاختبار المستخدمة في الأمثلة السابقة نفسها. وعلى أي حال، إن الجدول CustomerTable يحتوي بالفعل على كل من العميل Eleanor والعميل Mike؛ ولذلك ربما نحتاج أن نعرف كائنين من الصنف Customer هكذا:

```
Dim firstCustomer As New Customer("Heather", "Miami", "321-7654")
Dim secondCustomer As New Customer("George", "Oakland", "444-3333")
```

سوف يغير برنامج الاختبار عنوان العميل الأول (العميل Heather) وحذف العميل الثاني (العميل George). وبعد حذف العميل الثاني نحتاج أن نعيد قاعدة البيانات إلى حالتها الأصلية؛ لذلك نحتاج أن نضيف عميلاً آخر بدلاً من العميل الذي تم حذفه. ويجب أن تكون مخرجات البرنامج مثل الأمثلة السابقة؛ ولذلك تأكد أن كل شيء يعمل بشكل صحيح.

ملخص الفصل

Chapter Summary

- تتطلب التطبيقات الواقعية إمكانية تخزين الكائنات واسترجاعها (لتكون كائنات نظام مستمرة) حيث توجد طريقتين للاحتفاظ بالكائنات وهما: تخزين قيم صفات الكائنات (Attribute Storage)، أو تخزين الكائنات نفسها (Object Storage).
- إن الغرض الرئيس من أصناف التعامل مع البيانات (Data Access Classes) هو تقديم مجموعة من الإجراءات التي تستخدم لتخزين البيانات واسترجاعها (صفات الكائن)، ومن ثم تجعل كائنات أصناف مجال المشكلة كائنات مستمرة (Persistent Objects).
- ترجع أسباب فصل مهام تخزين البيانات واسترجاعها ووضعها في أصناف التعامل مع البيانات إلى عزل هذه البيانات بعيداً عن الأصناف الأخرى مما يقلل من مشكلات عملية الصيانة، حيث إن تغيير طريقة تخزين البيانات لا تتطلب أي تغيير في كل من أصناف مجال المشكلة وأصناف واجهة الاستخدام، بل تؤثر فقط على أصناف التعامل مع البيانات.
- سوف يقوم صنف التعامل مع البيانات بأربع مهام وهي: استرجاع بيانات كائن، وتخزين بيانات كائن، وتغيير بيانات كائن، وحذف بيانات كائن. وسنطور أربعة إجراءات تناظر هذه الأربع مهام وهي: الإجراء Find، والإجراء AddNew، والإجراء Update، والإجراء Delete.
- لقد تعلمنا كيف نخزن كائنات ونسترجعها باستخدام أكثر من أسلوب منها: الاحتفاظ بقيم صفات الكائنات واسترجاعها باستخدام كل من الصنف StreamWriter والصنف StreamReader، ونشر الكائنات إلى القرص الصلب (Object Serialization)، والاحتفاظ بقيم صفات الكائنات باستخدام قواعد البيانات (Databases) لتخزين قيم صفات كائنات أصناف مجال المشكلة واسترجاعها باستخدام أصناف التعامل مع البيانات.

المصطلحات الأساسية

Key Terms

التعامل مع البيانات (Data Access Classes)	الإجراء Find
كائنات مستمرة (Persistent Objects)	الإجراء AddNew
الصنف StreamWriter	الإجراء Update
الصنف StreamReader	الإجراء Delete
نشر الكائنات إلى القرص الصلب (Object Serialization)	

أسئلة المراجعة

Review Questions

١- SQL هو .....:

(أ) نظام إدارة قواعد البيانات.

(ب) كلمة محجوزة.

(ج) صنف.

(د) لغة.

٢- OleDb مزود من أجل .....:

(أ) IBM DB2.

(ب) مايكروسوفت أكسس.

(ج) مايكروسوفت SQL server.

(د) "ب" و "ج".

٣- عميل SQL مزود من أجل .....:

(أ) IBM DB2.

(ب) مايكروسوفت أكسس.

(ج) مايكروسوفت SQL server.

(د) "ب" و "ج".

٤- فضاء الأسماء الذي يحتوي على الأصناف التي تحتاجها للوصول لقاعدة بيانات هو .....:

(أ) System.Data.OleDb.

(ب) System.Runtimes.Serialization.

(ج) System.Data.SQL.

(د) System.Runtime.Formatters.

٥- يحتوي صنف OleDbDataAdapter على الأمر الذي يتناسب مع جملة SQL المقابلة:

(أ) SELECT.

(ب) UPDATE.

(ج) DELETE.

(د) INSERT.

(هـ) جميع ما سبق.

٦- جدول قاعدة البيانات العلاقية دائماً له .....:

(أ) جداول فرعية.

(ب) صفوف وأعمدة.

(ج) مفتاح حقل.

(د) علاقة مع جداول أخرى.

٧- الكلمة SQL المحجوزة ..... تستخدم في إرجاع البيانات من قاعدة البيانات العلاقية :

(أ) DELETE.

(ب) INSERT.

(ج) FIND.

(د) SELECT.

٨- أصناف الوصول للبيانات تكون مختفية عن .....:

(أ) أصناف واجهة المستخدم الرسومية.

(ب) أصناف مجال المشكلة.

(ج) أصناف واجهة المستخدم الرسومية وأصناف مجال المشكلة.

(د) جميع الأصناف ماعدا أصناف الوصول للبيانات.

٩- عندما تحتاج البيانات أن يحصل عليها من مصدر البيانات فإن بيانات صنف Adapter ..... تختار إجراءات البيانات والأماكن في .....:

(أ) Select, Place.

(ب) Get, recordset.

(ج) Fetch, Dataset.

(د) Fill, Dataset.

١٠- تعالج سلسلة المدخلات والمخرجات :

(أ) مثل اللغات الأخرى.

(ب) مثل تدفق الحروف.

(ج) مثل السجلات المهيكلة.

(د) حسب الجهاز المستخدم.

١١- تسلسل الكائن هو.....:

(أ) أداة RDBMS لتخزين الصفات.

(ب) أداة الفيجوال بيسك .نت لتخزين الكائن.

(ج) صنف.

(د) لغة.

١٢- ODBC هو .....:

(أ) نظام إدارة قاعدة البيانات.

(ب) الكلمة المحجوزة.

(ج) بروتوكول.

(د) لغة.

١٣- المفتاح الأساسي هو الذي .....:

(أ) يستعمل لتعريف السجل.

(ب) حقل خاص للبيانات المتتالية.

(ج) صفة لنسخة DBMS.

(د) جزء من SQL.

١٤- في هذا الفصل صنف Customer.....:

(أ) له إجراءات مرور البيانات DA.

(ب) هو الصنف الوحيد الذي مسموح له استدعاء إجراءات CustomerDA.

(ج) هو صنف مجال المشكلة PD.

(د) جميع ما سبق.

١٥- أي من الجمل البرمجية التالية صحيحة بالنسبة لصنف dataset .....:

(أ) تعبأ داخل الإجراء Fill المعرف داخل الصنف Adapter.

(ب) تتكون من واحد أو أكثر من الجداول.

(ج) يمكن أن تشير إلى الصفوف والأعمدة للجدول في الصنف dataset.

(د) جميع ما سبق.

١٦- NotFoundException هو .....:

(أ) صنف استثناء مجهز.

(ب) عضو من فضاء الأسماء الاستثناءات داخل الفيجوال بيسك.نت.

(ج) استخدامه نادر.

(د) لا شيء مما سبق.

١٧- العنصر "OleDb.OleDbException" هو:

(أ) صنف معرف مسبقاً من النوع Exception.

(ب) عنصر داخل الفضاء المسمى Exception للغة VB .NET.

(ج) نادراً ما يستخدم.

(د) جميع ما سبق.

أسئلة المناقشة

Discussion Questions

١- ما هي المميزات والعيوب في حذف إجراءات التعامل مع البيانات من الصنف CustomerPD والسماح لصنف واجهة الاستخدام استدعائها مباشرة؟

٢- لماذا تعتقد أن الإجراء GetAll لا يمكن استخدامه كبديل للإجراء Find؟ بمعنى آخر، لماذا لا نستطيع حذف الإجراء Find وأستخدام الإجراء GetAll لاسترجاع جميع الكائنات داخل المصفوفة ArrayList والبحث بها؟ وفي أي ظروف يمكن اتباع هذه الطريقة؟

٣- لم يتعرض هذا الفصل إلى قواعد البيانات المعتمدة على الكائنات، ولكن بخبرتك عن قواعد البيانات قم بوصف الاختلاف بينها وبين قواعد البيانات العلاقية.

٤- أي من التطبيقات المنفذة داخل هذا الفصل يمثل التطبيق الأمثل لبرنامج برادشو مارينا؟ ولماذا؟

مشاريع الفصل

Projects

احتوت أمثلة هذا الفصل على الصنف Customer الذي يحتوى على ثلاث صفات وهي: الاسم، والعنوان، ورقم الهاتف. افترض أن شركة برادشو مارينا قررت أن تتواصل مع العملاء من خلال عناوينهم البريدية الإلكترونية (e-mail). استخدم قاعدة البيانات خلال هذا المشروع.

١- قم بإضافة هذه الصفة على قاعدة البيانات المستخدمة داخل الفصل باستخدام برنامج MS Access.

٢- قم بإجراء التعديلات المطلوبة التالية على كل من الصنف Customer والصنف CustomerDA وبرنامج الاختبار للتعامل مع هذه الصفة:

(أ) أضف الصنف e-mail وكل من إجراءات المرور للتعامل مع هذه الصفحة وكذلك تعديل كل من إجراء الإنشاء وإجراء TellAboutSelf داخل الصنف.Customer.

(ب) قم بتعديل برنامج الاختبار لإضافة e-mail عند إنشاء كائنات من الصنف.Customer.

(ج) قم بتعديل كل من الإجراء AddNew والإجراء Find والإجراء Update داخل الصنف CustomerDA.

٣- قم باختبار البرنامج.