الفصل السابع

إضافة مهام جديدة لأصناف مجال المشكلة

ADDING RESPONSIBILITIES TO PROBLEM DOMAIN CLASSES

أهداف الفصل:

- تعريف صنف مجال مشكلة جديد (PD Class).
- إنشاء إجراءات خاصة (Custom Methods).
- تعريف إجراءات ومتغيرات أصناف.
- تعريف إجراءات متعددة الاستخدام (Overloaded Methods).
- التعامل مع الاستثناءات (Exceptions).

لقد تعلمت في الفصل السابق كيف تعرف صنف مجال المشكلة Customer حيث تم تعريف صفات هذا الصنف، وتعريف إجراءات الوصول التي تستخدم لتخزين قيم من الصفات واسترجاعها، وسوف نكمل في هذا الفصل التعامل مع أصناف مجال المشكلة حيث نعرف صنف مجال جديد لمشكلة برادشو اسمه Slip (مرسى المركب)، وبدلاً من تعريف إجراءات الوصول فقط، سوف نُعرِّف إجراءاً خاصاً (Custom Method) اسمه LeaseSlip والذي يحسب قيمة إيجار العقد للمرسى.

ويجب أن تعلم أن الإجراءات والصفات التي تم تطويرها إلى الآن هي إجراءات وصفات للكائنات، وهذا يعني أن لكل كائن الصفات الخاصة به، وأن الإجراءات لا تستدعى إلا من الكائن. أما في هذا الفصل فسوف نعرف متغيرات وإجراءات مشتركة (Shared) بين جميع كائنات الصنف Slip.

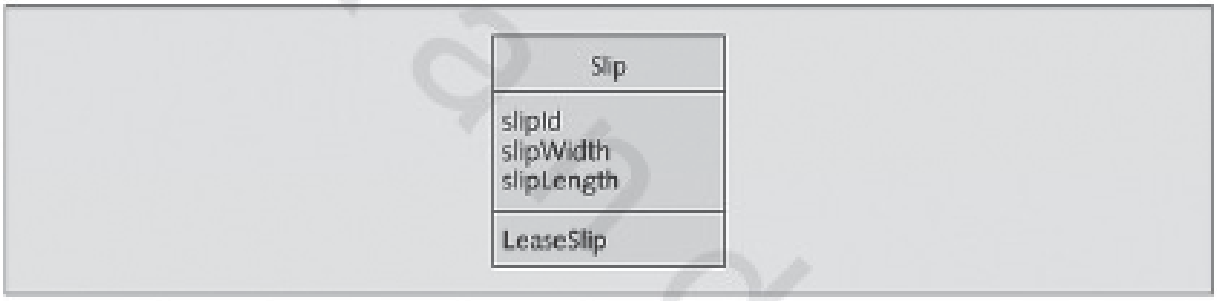
وسوف نشرح مفهوم الإجراءات متعددة الاستخدام (Overloading Methods) من خلال إجراءات إنشاء الصنف Slip ومن خلال الإجراء LeaseSlip. وأخيراً سوف نضيف الأوامر المنطقية اللازمة للتحقق من صحة

البيانات الممررة لإجراءات الوصول قبل إسنادها للصفات، وسوف تتعلم كيفية تعريف الثوابت (Constants) وكيفية تعريف استثناء (Exception) لكائن عميل وذلك لإخباره أن البيانات المستقبلة خطأ وتم رفضها.

تعريف صنف مجال مشكلة جديد

Writing a New Problem Domain Class Definition

كما رأينا في الفصل الخامس أن نظام معلومات شركة برادشو مارينا يحتوي على العديد من أصناف مجال المشكلة مثل الصنف Customer والصنف Slip، ولقد طورنا الصنف Customer في الفصل السادس، وسوف نطور الصنف Slip في هذا الفصل لمعرفة المزيد عن أصناف مجال المشكلة. وبالرجوع إلى الفصل الخامس نجد أن الصنف Slip يحتوي على ثلاثة صفات (الصفة slipId، والصفة slipWidth، والصفة slipLength) بالإضافة إلى الإجراء الخاص LeaseSlip كما هو واضح في نموذج Class Diagram في الشكل رقم (٧.١).



الشكل رقم (٧.١). نموذج Class Diagram للصنف Slip.

يتبع تعريف الصنف Slip التركيب البسيط نفسه المستخدم لتعريف الصنف Customer في الفصل السادس حيث نبدأ برأس الصنف (Class Header) الذي يحدد اسم الصنف هكذا:

```
Public Class Slip
```

ثم نكتب أوامر تعريف صفات الصنف حيث نخصص نوع البيانات Integer للصفات الثلاثة، ولاحظ أن درجة الوصول لهذه الصفات أخذت الدرجة Private وذلك لحماية البيانات من التعامل العشوائي حيث تمنع الكائنات الأخرى من استرجاع هذه الصفات مباشرة أو تغييرها إلا من خلال إجراءات الوصول.

```
Private slipId As Integer
Private slipWidth As Integer
Private slipLength As Integer
```

ثم نكتب أوامر إجراء الإنشاء التي تستقبل البيانات (Parameterized Constructor) وذلك لإسناد هذه البيانات مباشرة لصفات الكائن بعد إنشائه، وتذكر أن هذا الإجراء يستدعى مباشرة وذلك عند تعريف كائن جديد مستخدماً الكلمة المحجوزة New وإرسال قيم للصفات الثلاثة (slipId و slipWidth و slipLength)؛ ولذلك يجب تعريف ثلاث معاملات (Parameters) عند تعريف رأس إجراء الإنشاء لاستقبال قيم الصفات الثلاثة وهكذا.

```
Public Sub New(ByVal aSlipId As Integer, ByVal aSlipWidth As Integer, _
      ByVal aSlipLength As Integer)
```

لاحظ أن نوع بيانات هذه المعاملات هو نوع بيانات صفات الصنف نفسه، وبشكل عام إن نوع بيانات المعاملات لابد أن تتوافق مع نوع بيانات الصفات، وذلك يعني أن نوع بيانات المعاملات يشترط أن يسند إلى نوع بيانات الصفات، فمثلاً لا يجوز أن تسند معاملاً من النوع Double إلى صفة من النوع Integer ولكن العكس صحيح.

بعد ذلك نكتب أوامر إسناد المعاملات إلى الصفات وذلك باستدعاء إجراءات الوصول Setters كما فعلنا في إجراء إنشاء الصنف Customer في الفصل السابق، ولتجنب إعادة كتابة الأوامر فيجب على إجراء الإنشاء استدعاء إجراءات الوصول Setters لإسناد القيم إلى الصفات بدلاً من كتابة أوامر الإسناد، وسوف نضيف في هذا الفصل أوامر التحقق من صحة البيانات لإجراءي الوصول Setters، وإذا لم يستدع إجراء الإنشاء إجراءات الوصول Setters، فننتظر إعادة كتابة هذه الأوامر ثانية، ومن ثم يُعد استدعاء إجراءات الوصول Setters داخل إجراء الإنشاء تصميماً جيداً، وهذا يعني أن إجراء الوصول هو الإجراء الوحيد الذي له الحق أن يسند قيمة إلى صفة وذلك بعد التحقق من صحة هذه القيمة.

```
'invoke setter methods to populate attributes
SetSlipId(aSlipId)
SetSlipWidth(aSlipWidth)
SetSlipLength(aSlipLength)
```

بعد ذلك نكتب إجراء TellAboutSelf للصنف Slip كما عرفناه للصنف Customer في الفصل السادس والذي يستدعي إجراءات الوصول Getters (الإجراء GetSlipId، والإجراء GetSlipWidth، والإجراء GetSlipLength)، ثم نلصق القيم العائدة منها مع كلمات ذات معنى لتعطي جملة مفيدة ثم نعيدها.

وبالمثل فإن الإجراء TellAboutSelf للصنف Slip سوف يستدعي إجراءات الوصول Getters ثم تلصق القيم العائدة منها مع كلمات ذات معنى وإعادة النتيجة هكذا.

```
Public Function TellAboutSelf() As String
      Dim info As String
      info = "Slip: Id = " & GetSlipId() & ", Width = " & GetSlipWidth() _
            & ", Length = " & GetSlipLength()
      Return info
End Function
```

تعطي إجراءات TellAboutSelf مثالاً رائعاً على مفهوم Polymorphism حيث يوجد إجراءين بنفس الاسم (TellAboutSelf) داخل صنفين مختلفين (الصنف Customer والصنف Slip) ولكن لهما سلوك مختلف. عندما يستدعى الإجراء TellAboutSelf من كائن من النوع Customer سوف تحصل على نص يحتوي على الاسم والعنوان ورقم الهاتف، ولكن عندما يستدعى إجراء TellAboutSelf من كائن من النوع Slip سوف تحصل على نص يحتوي على رقم المرسى، وعرض المرسى، وطول المرسى، وفي هذه الحالة نطلق على هذه الإجراءات اسم Polymorphic Methods.

إن آخر خطوة في تعريف الصنف Slip هو تعريف إجراءات الوصول Setters وإجراءات الوصول Getters هكذا:

```
'get accessor methods
Public Function GetSlipId() As Integer
      Return slipId
End Function

Public Function GetSlipWidth() As Integer
      Return slipWidth
End Function

Public Function GetSlipLength() As Integer
      Return slipLength
End Function


'set acces sor methods
Public Sub SetSlipId(ByVal aSlipId As Integer)
      slipId = aSlipId
End Sub

Public Sub SetSlipWidth(ByVal aSlipWidth As Integer)
      slipWidth = aSlipWidth
End Sub

Public Sub SetSlipLength(ByVal aSlipLength As Integer)
      slipLength = aSlipLength
End Sub
```

يوضح الشكل رقم (٧.٢) تعريف الصنف Slip كاملاً.

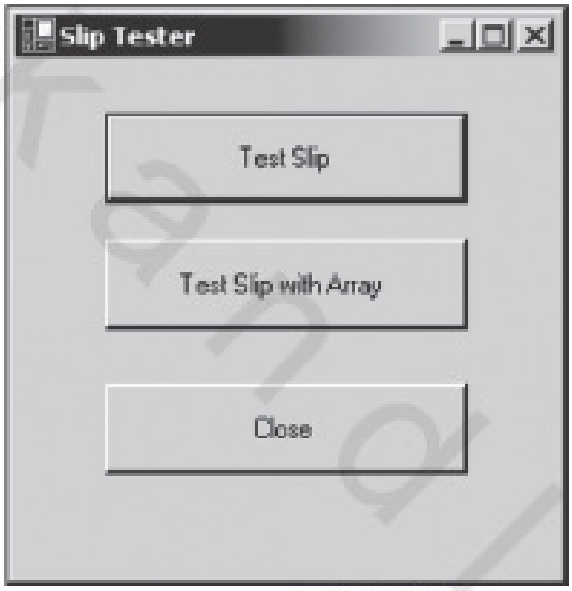
```
' Chapter 7 Slip class Example 1

Public Class Slip

   'attributes
   Private slipId As Integer
   Private slipWidth As Integer
   Private slipLength As Integer

   'constructor (three parameters)
   Public Sub New(ByVal aSlipId As Integer, ByVal aSlipWidth As
Integer, ByVal aSlipLength As Integer)
      'invoke setter methods to populate attributes
      SetSlipId(aSlipId)
      SetSlipWidth(aSlipWidth)
      SetSlipLength(aSlipLength)
   End Sub

      'custom method TellAboutSelf
   Public Function TellAboutSelf() As String
      Dim info As String
      info = "Slip: Id = " & GetSlipId() & ", Width = " & _
         GetSlipWidth() & ", Length = " & GetSlipLength()
      Return info
   End Function

   'get accessor methods
   Public Function GetSlipId() As Integer
      Return slipId
   End Function
   Public Function GetSlipWidth() As Integer
      Return slipWidth
   End Function
   Public Function GetSlipLength() As Integer
      Return slipLength
   End Function

   'set accessor methods
   Public Sub SetSlipId(ByVal aSlipId As Integer)
      slipId = aSlipId
   End Sub
   Public Sub SetSlipWidth(ByVal aSlipWidth As Integer)
      slipWidth = aSlipWidth
   End Sub
   Public Sub SetSlipLength(ByVal aSlipLength As Integer)
      slipLength = aSlipLength
   End Sub

End Class
```

الشكل رقم (٧,٢). تعريف الصنف Slip.

لقد كتبنا العديد من أصناف الاختبار في الفصل السادس وذلك لاستدعاء إجراءات من كائنات الصنف Customer للتأكد من صحة أوامر هذه الإجراءات، وبالمثل سوف نكتب هنا صنف الاختبار TesterOne ولكن

للتأكد من صحة أوامر إجراءات الصنف Slip وسوف نستخدم نموذج (Form) كما تعلمنا في آخر مثال في الفصل السادس بدلاً من استخدام الملفات البرمجية (Modules). عزيزي القارئ ربما تحتاج إلى مراجعة الموضوع الخاص بتطوير النماذج والتعامل مع الأزرار (Buttons) في الفصل الثاني.

يوضح الشكل رقم (٧.٣) نموذج الصنف TesterOne والذي يحتوي على ثلاث أزرار بعنوان "Test Slip" وعنوان "Test Slip with Array" وعنوان "Close"، ولقد أسند لها الأسماء btnTestSlip و btnTestSlipArray و btnClose. يوضح الفصل الثاني أن الضغط على زر يؤدي إلى وقوع حدث ومن ثم إلى تنفيذ إجراء؛ ولذلك لابد من تعريف إجراء لكل زر من هذه الأزرار والتي سيتم استدعاؤه عند الضغط على هذا الزر.



الشكل رقم (٧.٣). نموذج TesterOne.

سوف نعرف في المثال التالي الإجراءات الثلاثة المصاحبة للأزرار الثلاثة وسوف نعطي لكل إجراء اسماً خاصاً به وهو عبارة عن اسم الزر ثم الرمز "_" متبوعاً بالكلمة "Click".

يوضح الشكل رقم (٧.٤) إجراء معالجة الحدث المصاحب للزر Test Slip والذي يحتوي على أوامر إنشاء ثلاث كائنات من الصنف Slip ثم استدعاء الإجراء TellAboutSelf من كل كائن لاسترجاع قيم صفاته وعرضها. ويوضح الشكل رقم (٧.٥) مخرجات هذا الإجراء.

```
Private Sub btnTestSlip_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles btnTestSlip.Click
      'create three instances of Slip
      Dim slip1 As Slip = New Slip(1, 10, 20)
      Dim slip2 As Slip = New Slip(2, 12, 25)
      Dim slip3 As Slip = New Slip(3, 14, 30)

      Console.WriteLine("Begin Slip Test")
      Console.WriteLine(slip1.TellAboutSelf())
      Console.WriteLine(slip2.TellAboutSelf())
      Console.WriteLine(slip3.TellAboutSelf())
   End Sub
```

الشكل رقم (٧,٤). إجراء الحدث المصاحب للزر Test Slip.

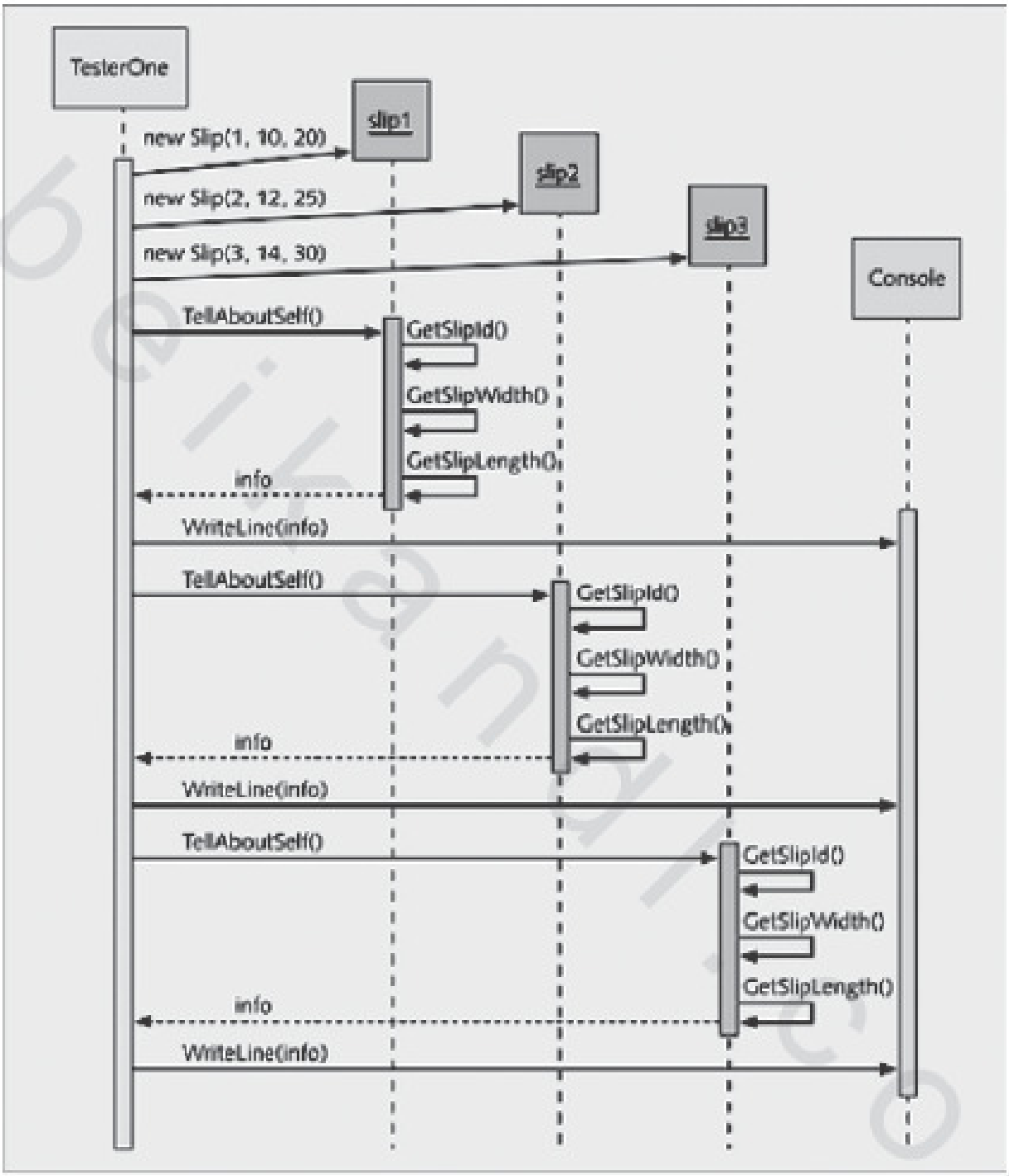
```
Begin Slip Test
Slip: Id = 1, Width = 10, Length = 20
Slip: Id = 2, Width = 12, Length = 25
Slip: Id = 3, Width = 14, Length = 30
```

الشكل رقم (٧,٥). مخرجات إجراء الحدث المصاحب للزر Test Slip.

قدم الفصل الخامس نماذج Sequence Diagram التي تصف التفاعل بين كائنات النظام لإنجاز المهام. إن أوامر إجراء الزر Test Slip تجعل الصنف TesterOne يتفاعل مع الكائنات الثلاثة المعرفة من الصنف Slip ويتفاعل أيضاً مع كائن Console. ويوضح الشكل رقم (٧,٦) نموذج Sequence Diagram لهذا التفاعل.

كما هو واضح في الشكل رقم (٧,٦) توجد خمسة مستطيلات في قمة النموذج لتمثيل الكائنات التي تتفاعل مع بعضها وهي TesterOne والكائنات الثلاثة المعرفة من الصنف Slip والمشار إليها بالمتغيرات الثلاث slip1 و slip2 و slip3 والصنف Console. ويبدأ التفاعل عندما ينشئ الصنف TesterOne ثلاث كائنات من الصنف Slip وإسنادها للمتغيرات الثلاث slip1 و slip2 و slip3 ؛ ولذلك فإن المتغير slip1 يشير إلى الكائن الأول من الصنف Slip والمتغير slip2 يشير إلى الكائن الثاني من الصنف Slip والمتغير slip3 يشير إلى الكائن الثالث من الصنف Slip.

ثم يستدعي الصنف TesterOne الإجراء TellAboutSelf من الكائن الأول والمشار إليه بالمتغير slip1، ثم يستدعي الإجراء TellAboutSelf إجراءات الوصول Getters الثلاثة من الكائن الأول والمشار إليه بالمتغير slip1، ثم يعيد النص الذي يتكون في كائن من النوع String والمشار إليه بالمتغير info إلى الصنف TesterOne. وبعد ذلك يستدعي TesterOne الإجراء WriteLine من الصنف Console وإرسال المتغير info له وذلك لطباعته على الشاشة. وبالمثل يستدعي الصنف TesterOne من كل من الكائن الثاني والثالث ثم يمرر القيم العائدة من هذه الإجراءات إلى الإجراء WtiteLine لطباعتها.



الشكل رقم (٧,٦). نموذج Sequence Diagram للصنفين TesterOne و Slip.

وعندما نضغط على الزر "Test Slip with Array" يتم استدعاء الإجراء btnTestSlipArray_Click. يعمل هذا الإجراء مثل الإجراء السابق حيث ينشئ أولاً ثلاث كائنات من الصنف Slip ثم يستدعي الإجراء TellAboutSelf

من كل واحد منها وطباعة النتيجة العائدة، ولكن بدلاً من استخدام ثلاثة متغيرات منفصلة يستخدم هذا الإجراء مصفوفة ذات ثلاثة عناصر لكي يشاور على الكائنات الثلاثة. يوضح الشكل رقم (٧.٧) أوامر هذا الإجراء كاملة.

```
Private Sub btnTestSlipArray_Click(ByVal sender As System.Object, ByVal
    e As System.EventArgs) Handles btnTestSlipArray.Click
        'test Slip class using an array of Slip references
        Dim slips(2) As Slip
        slips(0) = New Slip(1, 10, 20)
        slips(1) = New Slip(2, 12, 25)
        slips(2) = New Slip(3, 14, 30)

        ' For-Next loop to invoke TellAboutSelf
        Console.WriteLine("Begin Slip Test using Array")
        Dim i As Integer
        For i = 0 To 2
            Console.WriteLine(slips(i).TellAboutSelf())
        Next i

    End Sub
```

الشكل رقم (٧.٧). إجراء الحدث المصاحب للزر Test Slip with Array.

يعرف الأمر الأول مصفوفة ذات ثلاثة عناصر من النوع Slip، ولقد حددنا Slip كنوع بيانات لأن عناصر المصفوفة ستشير إلى كائنات من الصنف Slip، ومن الملاحظ أيضاً أننا وضعنا رقم ٢ بين الأقواس والذي يشير إلى فهرس آخر عنصر حيث يبدأ فهرس عناصر المصفوفة دائماً بالرقم صفر في لغة VB .NET؛ ولذلك فإن العنصر ذا الفهرس رقم ٢ هو العنصر الثالث وليس العنصر الثاني. ثم كتبنا ثلاثة أوامر لإنشاء ثلاثة كائنات من الصنف Slip مستدعين إجراء الإنشاء (Constructor) الذي يستقبل المعاملات (رقم المرسى، وعرض المرسى، وطول المرسى)، وأسندنا تلك الكائنات إلى عناصر المصفوفة بالترتيب (بداية من العنصر رقم صفر إلى العنصر رقم ٢).

إن آخر جزء في هذا الإجراء يحتوي على تكرار For-Next ليتجول داخل عناصر المصفوفة مستدعياً الإجراء TellAboutSelf من كل كائن من الصنف Slip ثم يعرض المعلومات العائدة من هذا الإجراء. وسوف ينفذ هذا التكرار ثلاثة مرات مستخدماً العداد i، وتذكر أن التكرار For-Next يسند قيمة ابتدائية للعداد ويزيد قيمة العداد، وفي هذا المثال نعطي قيمة ابتدائية صفر للعداد i، ويزيد قيمته بواحد في نهاية كل دورة. وبذلك فإن الأمر الذي يوجد داخل التكرار سينفذ ثلاثة مرات. عندما يكون العداد صفراً سينفذ للمرة الأولى، وعندما يحتوي على القيمة ١ سينفذ للمرة الثانية، وعندما يحتوي على القيمة ٢ سينفذ للمرة الثالثة، وهذا يعني أن الإجراء TellAboutSelf سيستدعى ثلاثة مرات من عناصر المصفوفة الثلاثة: slips(0) و slips(1) و slips(2). ولاحظ أن هذا التكرار لا يفيد إلا مع مصفوفة ذات ثلاثة عناصر لأن نهاية التكرار محددة بالرقم ٢، ولكن كان من الأفضل استخدام الصفة Length لجعل التكرار يعمل مع أي حجم بدون تعديل.

يوضح الشكل رقم (٧,٨) مخرجات الإجراء btnTestSlipArray_Click.

```
Begin Slip Test using Array
Slip: Id = 1, Width = 10, Length = 20
Slip: Id = 2, Width = 12, Length = 25
Slip: Id = 3, Width = 14, Length = 30
```

الشكل رقم (٧,٨). مخرجات الإجراء المصاحب للزر Test Slip with Array.

أما الزر الثالث (الزر Close) الموجود على النموذج TesterOne فيهدف إلى إنهاء التطبيق ويسمى هذا الزر بالاسم "btnClose"، والإجراء المناظر له موضح في الأوامر التالية، حيث يحتوي هذا الإجراء على أمر واحد فقط وهو استدعاء الإجراء الموروث Dispose وهو المسؤول عن إيقاف تشغيل البرنامج وحذف النموذج من الذاكرة. وتشير الكلمة المحجوزة Me إلى أن الإجراء المستدعى سوف ينفذ من الكائن الذي يحتوي على تعريف هذا الإجراء (وفي هذا المثال الكلمة Me تشير إلى النموذج). سوف تتعامل مع النماذج بكثرة في كل من الفصل العاشر والحادي عشر.

```
Private Sub btnClose_Click(ByVal sender As System.Object, ByVal
      e As System.EventArgs) Handles btnClose.Click
            Me.Dispose()
End Sub
```

إنشاء إجراءات خاصة
Creating Custom Methods

إلى الآن لم ننشئ إلا إجراءات الوصول (Accessor Methods) مثل GetSlipId ومثل SetSlipId وهكذا. ويطلق على هذه الإجراءات اسم الإجراءات القياسية أو الاعتيادية (Standard Methods) لأنها من المعتاد تواجدها في معظم أصناف مجال المشكلة، ونقدم في هذا الفقرة الإجراءات الخاصة (Custom Methods) وهي المسؤولة عن معالجة البيانات على عكس الإجراءات الاعتيادية المسؤولة عن استرجاع قيم الصفات وتخزينها.

سوف نعرف إجراءاً خاصاً اسمه LeaseSlip وهو المسؤول في هذا الفصل عن حساب قيمة إيجار المرسى، وسوف نزيد أوامر هذا الإجراء في الفصول القادمة وذلك لربط كل من المرسى والعميل بالعقد.

تحسب شركة برادشو مارينا قيمة إيجار المرسى بناء على عرض المرسى، وكما هو واضح في الجدول رقم (٧,١) فإن قيمة إيجار المرسى تتراوح بين ٨٠٠ دولار إلى ١٥٠٠ دولار بناء على عرض المرسى، مع العلم أن طول المرسى لا يدخل في حساب قيمة الإيجار لأن الطول يتناسب مع العرض، وبمعنى آخر أن المرسى الأكثر عرضاً هو المرسى الأكثر طولاً.

الجدول رقم (٧.١). جدول حساب قيمة إيجار المرسى.

Slip Width	Annual Lease Fee
10	$800
12	$900
14	$1,100
16	$1,500

يجب أن نعرّف هذا الإجراء من النوع "إجراء دالة" لأنه سوف يعيد قيمة (قيمة إيجار المرسى)، وتذكر أن إجراء الدالة (Function Procedure) يعيد قيمة على عكس الإجراء الفرعي (Sub Procedure) الذي لا يعيد قيمة؛ ولذلك سوف تستخدم الإجراء الفرعي عندما لا تعيد قيمة. ومثل جميع الإجراءات فإن الإجراء الخاص يبدأ برأس الإجراء، وسوف نأخذ درجة وصول عام (Public) لتمكين جميع الكائنات من استدعائه، ولأنه سوف يعيد قيمة الإيجار والذي سوف يكون بالدولار ولذلك نوع البيانات سيكون Single لتعريف نوع البيانات العائدة.

```
Public Function LeaseSlip() As Single
```

لكي تحسب قيمة الإيجار، يجب عليك أن تستخدم سلسلة من أوامر If الشرطية أو تستخدم أمر Select، وتستخدم الأوامر التالية أمر Select لأنها الأسهل في هذه الحالة. ويعرّف الأمر الأول متغيراً من النوع Single اسمه fee لتخزين قيمة الإيجار، يلي ذلك أمر Select الذي يختبر محتويات الصفة slipWidth وإسناد قيمة إيجار تناسبه بناء على قيمة هذه الصفة.

```
Dim fee As Single
Select Case slipWidth
      Case 10
            fee = 800
      Case 12
            fee = 900
      Case 14
            fee = 1100
      Case 16
            fee = 1500
      Case Else
            fee = 0
End Select
Return fee
```

ينتهي الأمر Select بالجملة End Select، ولاحظ أن الأمر السابق له يحتوي على Case Else والذي يسند القيمة صفر للمتغير fee وذلك إذا لم يكن عرض المرسى يساوي ١٠ أو ١٢ أو ١٤ أو ١٦، وآخر أمر في الإجراء يعيد القيمة المستنتجة إلى الإجراء المستدعي لهذا الإجراء، ولاحظ أن هذا الإجراء في هذه الحالة يلعب دور الخادم داخل الصنف Slip وأن الكائنات التي تستدعي ذلك الإجراء تلعب دور العملاء. يسرد الشكل رقم (٧.٩) تعريف الصنف Slip بعد تعديله وذلك بإضافة الإجراء LeaseSlip.

```
' Chapter 7 Slip class Example 2
' Illustrate custom method LeaseSlip

Public Class Slip

    'attributes
    Private slipId As Integer
    Private slipWidth As Integer
    Private slipLength As Integer

    'constructor (three parameters)
    Public Sub New(ByVal aSlipId As Integer, ByVal aSlipWidth As _
      Integer, ByVal aSlipLength As Integer)
        'invoke setter methods to populate attributes
        SetSlipId(aSlipId)
        SetSlipWidth(aSlipWidth)
        SetSlipLength(aSlipLength)
    End Sub

    'custom method LeaseSlip calculates and returns fee
    Public Function LeaseSlip() As Single
        Dim fee As Single
        Select Case slipWidth
            Case 10
                fee = 800
            Case 12
                fee = 900
            Case 14
                fee = 1100
            Case 16
                fee = 1500
            Case Else
                fee = 0
        End Select
        Return fee
    End Function

    'custom method TellAboutSelf
  Public Function TellAboutSelf() As String
    Dim info As String
    info = "Slip: Id = " & GetSlipId() & ", Width = " _
      & GetSlipWidth() & ", Length = " & GetSlipLength()
    Return info
  End Function

  'get accessor methods
  Public Function GetSlipId() As Integer
    Return slipId
  End Function
  Public Function GetSlipWidth() As Integer
    Return slipWidth
  End Function
```

الشكل رقم (٧,٩). تعريف الصنف Slip بعد إضافة الإجراء LeaseSlip.

```
    Public Function GetSlipLength() As Integer
       Return slipLength
    End Function

    'set accessor methods
    Public Sub SetSlipId(ByVal aSlipId As Integer)
       slipId = aSlipId
    End Sub
    Public Sub SetSlipWidth(ByVal aSlipWidth As Integer)
       slipWidth = aSlipWidth
    End Sub
    Public Sub SetSlipLength(ByVal aSlipLength As Integer)
       slipLength = aSlipLength
    End Sub

End Class
```

تابع الشكل رقم (٧,٩).

والآن سوف نكتب صنف اختبار آخر اسمه TesterTwo على شكل ملف برمجي (Module) بدلاً من نموذج (Form)، ومثلما فعلنا مع الصنف TesterOne سوف نعرف مصفوفة ذات ثلاثة عناصر مستخدمين نوع البيانات Slip، وبعد ذلك ننشئ ثلاثة كائنات من النوع Slip.

```
' define an array of Slip references
Dim slips(2) As Slip
slips(0) = New Slip(1, 10, 20)
slips(1) = New Slip(2, 12, 25)
slips(2) = New Slip(3, 14, 30)
```

بعد ذلك، سوف نكتب تكرار For-Next مثل تكرار الصنف TesterOne، ولكن سوف نستدعي الإجراء LeaseSlip بدلاً من الإجراء TellAboutSelf من كل كائن ونعرض قيمة الإيجار العائدة من الإجراء LeaseSlip. سوف يتم تنفيذ جسم التكرار ثلاثة مرات. الأولى، يكون عداد التكرار i يساوي صفراً، والثانية يكون عداد التكرار i يساوي ١، والثالثة يكون عداد التكرار i يساوي ٢. والأمر الوحيد المتواجد داخل التكرار ينجز المهام التالية:

١- استدعاء الإجراء LeaseSlip من كائن الصنف Slip والمشار إليه بعنصر المصفوفة المفهرس بعداد التكرار i والذي يأخذ القيمة صفر في أول تكرار، ويأخذ القيمة ١ في التكرار الثاني، وهكذا.

٢- استدعاء الإجراء GetSlipId من كائن الصنف Slip والمشار إليه بعنصر المصفوفة المفهرس بعداد التكرار i.

٣- لصق المقطع "Fee is " مع القيمة العائدة من الإجراء LeaseSlip، ثم لصق الناتج بالمقطع " for slip"، وأخيراً لصق القيمة العائدة من الإجراء GetSlipId.

٤- استدعاء الإجراء WriteLine من الصنف Console ممرراً إليه النص الناتج من الخطوة الثالثة.

لاحظ أن هذا المثال يستخدم الصفة Length للمصفوفة وذلك لإنهاء التكرار بدلاً من استخدام قيمة ثابتة وذلك بكتابة slips.Length - 1 وهو ما يلغي مشكلة صيانة التكرار عند تغيير حجم المصفوفة.

```
'compute & display lease fee for all three slips
Console.WriteLine("Computing lease for all three slips:")
Dim i As Integer
For i = 0 To slips.Length - 1
      Console.WriteLine("Fee is " & slips(i).LeaseSlip() _
      & " for Slip " & slips(i).GetSlipId())
Next i
```

يسرد الشكل رقم (٧.١٠) أوامر الصنف TesterTwo، ويعرض الشكل رقم (٧.١١) المخرجات.

```
' Chapter 7 TesterTwo Example 2
Module TesterTwo
   Sub Main()

      ' define an array of Slip references
      Dim slips(2) As Slip
      slips(0) = New Slip(1, 10, 20)
      slips(1) = New Slip(2, 12, 25)
      slips(2) = New Slip(3, 14, 30)

      'compute & display lease fee for all three slips
      Console.WriteLine("Computing lease for all three slips:")
      Dim i As Integer
      For i = 0 To slips.Length - 1
         Console.WriteLine("Fee is " & slips(i).LeaseSlip() _
         & " for Slip " & slips(i).GetSlipId())
      Next i
   End Sub
End Module
```

الشكل رقم (٧.١٠). أوامر الصنف TesterTwo.vb.

```
Computing lease for all three slips:
Fee is 800 for Slip 1
Fee is 900 for Slip 2
Fee is 1100 for Slip 3
```

الشكل رقم (٧.١١). مخرجات الصنف TesterTwo.

تعريف متغيرات وإجراءات الأصناف
Writing Class Variables and Methods

لقد رأينا في الفصل السادس كيفية توصيف متغيرات الكائن (Instance Variables) وإجراءات الكائن (Instance Methods) واستخدامها، فعندما نعرِّف كائناً جديداً من صنف ما يخصص لهذا الكائن نسخة من متغيرات وإجراءات الكائن المعرفة داخل هذا الصنف. فعلى سبيل المثال إن كل كائن تم إنشاؤه في الفقرة السابقة من الصنف Slip يمتلك نسخة خاصة به من متغيرات الكائن (الصفات) الثلاثة وهي SlipId و SlipWidth و SlipLength لأن المراسي المختلفة لها أرقام تعريفية مختلفة وربما يكون لها عرض مختلف وطول مختلف، وبالمثل فإن إجراءات الوصول والإجراءات الخاصة مثل الإجراء LeaseSlip يتم استخدامها لكل كائن على حده، ولتجنب التكرار فإن لغة VB .NET لا تخصص نسخاً مختلفة من إجراءات الكائن لكل كائن بل يبدو الأمر هكذا. سوف نتعلم في هذه الفقرة عن متغيرات الصنف (Class Variables) وإجراءات الصنف (Class Methods) والتي تكون مشتركة (Shared) على مستوى جميع الكائنات لصنف ما. أي لا توجد نسخة خاصة من هذه المتغيرات والإجراءات لكل كائن بل توجد نسخة واحدة على مستوى الصنف وكل كائن له الحق أن يتعامل معها. وتستخدم الكلمة المحجوزة Shared لتوصيف متغيرات الصنف وإجراءاته، وإذا حذفنا هذه الكلمة فإن لغة VB .NET تعتبرها متغيرات وصفات غير مشتركة، ومن ثم فإن كل كائن يتم إنشاؤه يأخذ نسخة منها، ولذلك فمن المهم وضع الكلمة Shared عند تعريف متغيرات الصنف وإجراءاته.

ولشرح فائدة متغيرات الصنف وإجراءاته افترض معي أن شركة برادشو تريد أن تعلم العدد الكلي للمراسي، فيمكنك فعل ذلك بتعريف متغير صنف مشترك داخل الصنف Slip مستخدماً الأمر التالي الذي يعرف المتغير numberOfSlips ويسند له القيمة صفر، ولقد حددنا درجة الوصول Private لهذا المتغير لتمكين الإجراءات المعرفة داخل الصنف فقط من التعامل معه.

```
Private Shared numberOfSlips As Integer = 0
```

ولأن شركة برادشو يمكنها بناء مراسي جديدة للعملاء الجدد، فإن إضافة الأمر التالي داخل إجراء الإنشاء للصنف Slip والذي يزيد المتغير numberOfSlips بمقدار واحد عند إنشاء كائن جديد من الصنف Slip، يجعل هذا المتغير يحتفظ بالعدد الكلي للمراسي.

```
'increment shared attribute
numberOfSlips += 1
```

والآن يمكننا تعريف إجراء وصول Getters لاسترجاع قيمة المتغير numberOfSlips، وسوف نعرف هذا الإجراء من النوع إجراء دالة لأنه يعيد قيمة المتغير numberOfSlips والمعرف من النوع Integer، وتستخدم الكلمة المحجوزة Shared مع هذا الإجراء لمنع إمكانية استدعائه من كائنات الصنف Slip ولكن تستدعى فقط من الصنف Slip، وإعطاء درجة وصول Public له لاستدعائه من أي مكان في البرنامج هكذا.

```
'shared (class) method
Public Shared Function GetNumberOfSlips() As Integer
      Return numberOfSlips
End Function
```

يسرد الشكل رقم (٧,١٢) شفرة الصنف Slip بعد تعديله.

```
' Chapter 7 Slip class Example 3
' Illustrate shared (class) attributes and methods

Public Class Slip

      'attributes
      Private slipId As Integer
      Private slipWidth As Integer
      Private slipLength As Integer

      'shared (class) attribute
      Private Shared numberOfSlips As Integer = 0

   Public Sub New(ByVal aSlipId As Integer, ByVal aSlipWidth _
     As Integer, ByVal aSlipLength As Integer)
      'invoke setter methods to populate attributes
      SetSlipId(aSlipId)
      SetSlipWidth(aSlipWidth)
      SetSlipLength(aSlipLength)
      'increment shared attribute
      numberOfSlips += 1
   End Sub

   'custom method LeaseSlip calculates and returns fee
   Public Function LeaseSlip() As Single
      Dim fee As Single
      Select Case slipWidth
         Case 10
            fee = 800
         Case 12
            fee = 900
         Case 14
            fee = 1100
         Case 16
            fee = 1500
         Case Else
            fee = 0
```

الشكل رقم (٧,١٢). تعريف الصنف Slip بعد التعديل.

```
        End Select
        Return fee
    End Function

    'custom method TellAboutSelf
    Public Function TellAboutSelf() As String
        Dim info As String
        info = "Slip: Id = " & GetSlipId() & ", Width = "
          & GetSlipWidth() & ", Length = " & GetSlipLength()
        Return info
    End Function

    'shared (class) method
    Public Shared Function GetNumberOfSlips() As Integer
        Return numberOfSlips
    End Function

    'get accessor methods
    Public Function GetSlipId() As Integer
        Return slipId
    End Function
    Public Function GetSlipWidth() As Integer
        Return slipWidth
    End Function
    Public Function GetSlipLength() As Integer
        Return slipLength
    End Function

    'set accessor methods
    Public Sub SetSlipId(ByVal aSlipId As Integer)
        slipId = aSlipId
    End Sub
    Public Sub SetSlipWidth(ByVal aSlipWidth As Integer)
        slipWidth = aSlipWidth
    End Sub
    Public Sub SetSlipLength(ByVal aSlipLength As Integer)
        slipLength = aSlipLength
    End Sub

End Class
```

تابع الشكل رقم (٧,١٢).

نبدأ صنف الاختبار الثالث TesterThree بتعريف ثلاثة متغيرات إشارة من النوع Slip.

```
' define 3 Slip reference variables
Dim slip1, slip2, slip3 As Slip
```

ثم ننشئ أول كائن من الصنف Slip، ثم نسترجع قيمة المتغير numberOfSlips ونظهرها باستدعاء الإجراء GetNumberOfSlips بكتابة اسم الصنف متبوعاً بنقطة يليها اسم الإجراء هكذا:

```
' create slip instances & display numberOfSlips for each
slip1 = New Slip(1, 10, 20)
Console.WriteLine("Number of slips: " & _ Slip.GetNumberOfSlips())
```

يستدعي هذا الأمر الإجراء GetNumberOfSlips من الصنف Slip، ثم يلصق على القيمة العائدة جملة مفيدة وتمريرها إلى الأجراء WriteLine من الصنف Console.

يمكنك أيضاً استدعاء إجراءات الصنف مستخدماً اسم الكائن لأن لغة VB .NET تعلم إلى أي صنف ينتمي إليه هذا الكائن، وبذلك سوف تستبدل لغة VB .NET اسم الكائن باسم الصنف الخاص به عند ترجمة البرنامج هكذا:

```
'retrieve & display number of slips using slip1 instead of Slip
Console.WriteLine("Number of slips using slip1: " & _
      slip1.GetNumberOfSlips())
```

يؤدي هذا الأمر إلى نتيجة الأمر السابق نفسه مع أنه يستخدم اسم متغير إشارة slip1 بدلاً من اسم الصنف Slip، وفي الحقيقة يمكنك أيضاً استخدام slip2 أو slip3 لأن لغة VB .NET تعلم أنها جميعاً كائنات من الصنف Slip، ولذلك سوف تستبدل أسمائها باسم الصنف Slip عند استدعاء الإجراء GetNumberOfSlips. يوضح الشكل رقم (٧,١٣) محتويات الصنف TesterThree، ويوضح الشكل رقم (٧,١٤) المخرجات.

```
' Chapter 7 TesterThree Example 3

Module TesterThree
   Sub Main()

      ' define 3 Slip reference variables
      Dim slip1, slip2, slip3 As Slip
      ' create slip instances & display numberOfSlips for each
      slip1 = New Slip(1, 10, 20)
      Console.WriteLine("Number of slips: " & Slip.GetNumberOfSlips())
      slip2 = New Slip(2, 12, 25)
      Console.WriteLine("Number of slips: " & Slip.GetNumberOfSlips())
      slip3 = New Slip(3, 14, 30)
      Console.WriteLine("Number of slips: " & Slip.GetNumberOfSlips())

      'retrieve and display number of slips using slip1 instead of Slip
      Console.WriteLine("Number of slips using slip1: " & _
        slip1.GetNumberOfSlips())
   End Sub
End Module
```

الشكل رقم (٧,١٣). أوامر الصنف TesterThree.vb.

```
Number of slips: 1
Number of slips: 2
Number of slips: 3
Number of slips using slip1: 3
```

الشكل رقم (٧,١٤). مخرجات الصنف TesterThree.

تعريف الإجراءات متعددة الاستخدام
Writing Overloaded Methods

لقد تحدثنا في الباب الأول من هذا الكتاب عن توقيع الإجراء (Method Signature) والتي يتكون من اسم الإجراء وقائمة المعاملات (Parameter List) المقررة له حيث تميز لغة VB .NET الإجراء بتوقيعه لا باسمه؛ ولذلك يمكن تعريف أكثر من إجراء بنفس الاسم ولكن بقائمة معاملات مختلفة، عندئذ ترى لغة VB .NET هذه الإجراءات مختلفة، ويسمى الإجراء الذي له الاسم نفسه مع إجراء آخر في نفس الصنف ولكن بقائمة معاملات مختلفة اسم Overloaded Method أي إجراء متعدد الاستخدام.

أرجو ألا يختلط عليك الأمر عزيزي القارئ بين الإجراءات المتعددة الاستخدام والإجراءات المعاد تعريفها (Overridden Methods) حيث يمكنك إعادة تعريف إجراء موروث من الصنف الأب في الصنف الابن بنفس توقيع الإجراء، وعندما تفعل ذلك فإنك تستبدل الإجراء الموروث بالإجراء الجديد ويسمى هذا بمفهوم Overriding أي إعادة التعريف، ويوضح الفصل الثامن كيفية إعادة تعريف الإجراءات.

أما الإجراءات متعددة التعريفات (Polymorphic Methods) فهي الإجراءات المعرفة في أكثر من صنف بالتوقيع نفسه، ولذلك فإن مفهوم Polymorphism يسمح لكائنات مختلفة أن تستجيب بطرق مختلفة لنفس الرسالة (Message)، ومن أشهر أمثلة الإجراءات متعددة التعريفات هو الإجراء TellAboutSelf الذي عرفناه في الصنف Customer وعرفناه أيضاً في الصنف Slip بالتوقيع نفسه حيث إن الإجراء TellAboutSelf المعرف في الصنف Customer يعيد معلومات حول العميل (الاسم والعنوان ورقم الهاتف)، بينما الإجراء TellAboutSelf المعرف في الصنف Slip يعيد معلومات حول المرسى (رقم التعريف والعرض والطول).

وعلى العكس فإن الإجراءات متعددة الاستخدام (Overloaded Methods) تعرف بالاسم نفسه وفي الصنف نفسه ولكن بقائمة معاملات مختلفة. سوف نطور إجراء إنشاء متعدد الاستخدام وكذلك إجراءاً خاصاً متعدد الاستخدام في الفقرتين القادمتين، أما مفهوم كل من الإجراءات متعددة التعريفات (Polymorphic Methods) والإجراءات المعاد تعريفها (Overridden Methods) فسوف يتم التعرض لها في الفصول القادمة.

تعريف إجراء إنشاء متعدد الاستخدام
Overloading a Constructor

في بعض الأحيان نحتاج إلى تعريف أكثر من إجراء إنشاء بقوائم معاملات مختلفة، فعلى سبيل المثال إن عرض معظم مراسي شركة برادشو يساوي ١٢ قدماً، وطول معظم المراسي أيضاً يساوي ٢٥ قدماً، ولكن يوجد القليل من المراسي لها طول وعرض مختلف. على أي حال إن إجراء الإنشاء الحالي المعرف داخل الصنف Slip يستقبل ثلاث معاملات وهي: SlipId و SlipWidth و SlipLength.

يمكنك في هذا الموقف تعريف إجراء إنشاء آخر يستقبل فقط معاملاً واحداً فقط وهو SlipId، أما المعاملان الآخران فيأخذان القيم الافتراضية (١٢ قدماً للعرض، و٢٥ قدماً للطول)، وفي حال وجود هذا الإجراء الجديد، لا يحتاج الكائن العميل أن يمرر قيم الطول والعرض عند تعريف كائن جديد من الصنف Slip.

ولتبسيط شفرة تعريف الصنف Slip، يمكن تعريف متغيرات ثابتة لقيم العرض والطول الافتراضية كما هو واضح في أوامر الشفرة التالية، ولتعريف متغير ثابت في لغة VB .NET، نستخدم الكلمة المحجوزة Const ثم نسند له قيمة، ولاحظ أيضاً أن هذا المتغير أخذ درجة الوصول Private لجعله مرئياً فقط لإجراءات الصنف Slip فقط. إن المتغيرات الثابتة تكون مشتركة على مستوى جميع كائنات الصنف حتى ولو لم نكتب الكلمة Shared صراحة. ويقال على هذه المتغيرات أنها ثوابت لأن قيمها لا تتغير أثناء تنفيذ البرنامج، ولاحظ أيضاً أنك تستخدم طريقة تسمية لغة VB .NET نفسها حيث تستخدم حروفاً كبيرة ونفصل بين الكلمات بالرمز "_".

```
'constants
Private Const DEFAULT_SLIP_WIDTH As Integer = 12
Private Const DEFAULT_SLIP_LENGTH As Integer = 25
```

إن رأس إجراء الإنشاء الجديد يتشابه مع إجراء الإنشاء السابق عدا إنه يحتوي على معامل واحد فقط (رقم التعريف) بدلاً من ثلاثة معاملات، ويحتوي هذا الإجراء على أمر واحد فقط وهو استدعاء إجراء الإنشاء الأصلي على أن يمرر إليه المعاملات الثلاثة: رقم التعريف، والقيم الافتراضية للطول والعرض، ولاحظ أننا استخدمنا الكلمة المحجوزة Me واسم إجراء الإنشاء لاستدعاء إجراء الإنشاء الأصلي.

```
'1-parameter constructor
'Overloads keyword not used with constructors
Public Sub New(ByVal aSlipId As Integer)
      'invoke 3-parameter constructor, passing default values
      Me.New(aSlipId, DEFAULT_SLIP_WIDTH, DEFAULT_SLIP_LENGTH)
End Sub
```

يوضح صنف الاختبار الرابع TesterFour كيفية إنشاء كائنات الصنف Slip مستدعياً إجراءات الإنشاء المختلفة، ولكن تذكر أن لغة VB .NET تميز الإجراءات بتوقيعها (عدد المعاملات الممررة لها) أي إذا مررنا ثلاثة معاملات عند إنشاء كائن الصنف Slip فسوف تستدعي إجراء الإنشاء الأصلي، أما إذا مررنا معاملاً واحداً فقط، فإن إجراء الإنشاء الجديد هو الذي سيستدعى، بمعنى آخر إن عدد المعاملات يحدد الإجراء المطلوب تنفيذه.

ينشئ الأمر الأول كائناً من الصنف Slip بواسطة تمرير ثلاثة معاملات. وعندئذ سيستدعى إجراء الإنشاء الأصلي الذي يستقبل ثلاث معاملات.

```
'create slip using 3-parameter constructor
slip1 = New Slip(1, 10, 20)
```

أما الأمر الثاني فيمرر معاملاً واحداً فقط (رقم التعريف) عند إنشاء كائن الصنف Slip ؛ ولذلك سوف يستدعي إجراء الإنشاء الثاني، ومن ثم سوف يسند لهذا الكائن القيمة الافتراضية ١٢ للعرض والقيمة الافتراضية ٢٥ للطول.

```
'create slip using 1-parameter constructor
slip2 = New Slip(2)
```

وأخيراً، الأمران الآخران يستدعيان الإجراء TellAboutSelf من كل كائن وعرض المعلومات العائدة.

```
'retrieve & display info for both slips
Console.WriteLine(slip1.TellAboutSelf())
Console.WriteLine(slip2.TellAboutSelf())
```

وتكون المخرجات هكذا:

```
Slip: Id = 1, Width = 10, Length = 20
Slip: Id = 2, Width = 12, Length = 25
```

تعريف إجراء خاص متعدد الاستخدام
Overloading a Custom Method

بالإضافة إلى تعريف إجراء الإنشاء المتعدد الاستخدام يمكنك أيضاً تعريف إجراء خاص متعدد الاستخدام (Overloaded Custom Method). افترض أن شركة برادشو تريد أن تعطي خصماً على قيمة إيجار المرسى تحت ظروف معينة، ولعمل ذلك يمكن إضافة إجراء ثانٍ من الإجراء الخاص LeaseSlip والذي يمكن أن يستقبل نسبة الخصم، وبذلك سيصبح لدينا إجراءين بنفس الاسم ولكن بتوقيع مختلف واستخدام مختلف لأن الإجراء الأصلي لا يستقبل أي معاملات بينما الإجراء الثاني يستقبل نسبة الخصم كمعامل (aDiscountPercent)، وفي هذه الحالة لابد أن يحتوي رأس الإجراء الثاني على الكلمة المحجوزة Overloads، وأيضاً يجب أن تضاف هذه الكلمة إلى الإجراء الأصلي، ومعنى ذلك أنه يجب إدراج كلمة Overloads في رؤوس الإجراءات المتعددة الاستخدام ؛ ولذلك يبدو رأس الإجراء الثاني هكذا:

```
Public Overloads Function LeaseSlip(ByVal aDiscountPercent As Single) _
      As Single
```

يحتوي الإجراء الثاني على ثلاثة أوامر لحساب قيمة الإيجار بعد حساب نسبة الخصم حيث يستدعي الأمر الأول الإجراء الأصلي لحساب قيمة الإيجار كاملاً دون خصم، ويحسب الأمر الثاني قيمة الإيجار بعد خصم نسبة الخصم، ويعيد الأمر الثالث قيمة الإيجار المحسوبة إلى الإجراء الذي استدعاه. لاحظ أننا استخدمنا الكلمة المحجوزة Me والتي تعني استدعاء الإجراء LeaseSlip من الكائن الحالي للصنف Slip مع العلم أننا نستطيع حذف هذه الكلمة لأنها تمثل الكائن الافتراضي، ولكن على أي حال إن وجودها يجعل أوامر البرنامج أكثر وضوحاً.

```
'overload custom method LeaseSlip if discount requested
Public Overloads Function LeaseSlip(ByVal aDiscountPercent As Single) _
      As Single
      'invoke LeaseSlip() to get fee
      Dim fee As Single = Me.LeaseSlip()
      'calculate and return discount fee
      Dim discountFee As Single = fee * (100 - aDiscountPercent) / 100
      Return discountFee
End Function
```

يمكنك الآن إضافة أوامر إلى صنف الاختبار TesterFour وذلك لاستدعاء كل من الإجراء الأصلي LeaseSlip لحساب الإيجار دون خصم، واستدعاء الإجراء الثاني LeaseSlip لحساب قيمة الإيجار بعد طرح نسبة الخصم. وسوف تميز لغة VB .NET بينهما بناء على قائمة المعاملات الممررة لهما، فإذا لم يمرر أي معاملات سوف تستدعي الإجراء الأصلي، وإذا مرر معامل واحد، فسوف تستدعي الإجراء الثاني.

```
'compute lease for slip1 without a discount, then with a 10% discount
Console.WriteLine("slip1 fee is " & slip1.LeaseSlip())
Console.WriteLine("With a 10% discount it's " & slip1.LeaseSlip(10))

'lease slip2 without a discount, then with a 20% discount
Console.WriteLine("slip2 fee is " & slip2.LeaseSlip())
Console.WriteLine("With a 20% discount it's " & slip2.LeaseSlip(20))
```

يسرد الشكل رقم (٧.١٥) تعريف الصنف Slip بعد إضافة إجراء لإنشاء متعدد الاستخدام والإجراء LeaseSlip، بينما يسرد الشكل رقم (٧.١٦) تعريف الصنف TesterFour، وتظهر مخرجات الصنف TesterFour في الشكل رقم (٧.١٧).

```
' Chapter 7 Slip class Example 4
' Illustrate overloading methods

Public Class Slip

   'attributes
   Private slipId As Integer
   Private slipWidth As Integer
   Private slipLength As Integer

   'shared (class) attribute
   Private Shared numberOfSlips As Integer = 0

   'constants
   Private Const DEFAULT_SLIP_WIDTH As Integer = 12
   Private Const DEFAULT_SLIP_LENGTH As Integer = 25

   '1-parameter constructor
   'Overloads keyword not used with constructors
   Public Sub New(ByVal aSlipId As Integer)
      'invoke 3-parameter constructor, passing default values
      Me.New(aSlipId, DEFAULT_SLIP_WIDTH, DEFAULT_SLIP_LENGTH)
   End Sub
  '3-parameter constructor
  Public Sub New(ByVal aSlipId As Integer, ByVal aSlipWidth As _
    Integer, ByVal aSlipLength As Integer)
     'invoke setter methods to populate attributes
     SetSlipId(aSlipId)
     SetSlipWidth(aSlipWidth)
     SetSlipLength(aSlipLength)
     'increment shared attribute
     numberOfSlips += 1

  End Sub

  'custom method LeaseSlip calculates and returns fee
  Public Overloads Function LeaseSlip() As Single
     Dim fee As Single
     Select Case slipWidth
           Case 10
                 fee = 800
           Case 12
                 fee = 900
           Case 14
                 fee = 1100
           Case 16
                 fee = 1300
           Case Else
                 fee = 0
     End Select
     Return fee
  End Function

  'overloaded custom method LeaseSlip if discount requested
  Public Overloads Function LeaseSlip(ByVal aDiscountPercent As _
     Single) As Single
     'invoke LeaseSlip() to get fee
     Dim fee As Single = Me.LeaseSlip()
     'calculate and return discount fee
     Dim discountFee As Single = fee * (100 - aDiscountPercent) / 100
     Return discountFee
  End Function

  'custom method TellAboutSelf
  Public Function TellAboutSelf() As String
     Dim info As String
     info = "Slip: Id = " & GetSlipId() & ", Width = " & _
        GetSlipWidth() & ", Length = " & GetSlipLength()
     Return info
  End Function

  'shared (class) accessor method
  Public Shared Function GetNumberOfSlips() As Integer
     Return numberOfSlips
  End Function

  'get accessor methods
```

الشكل رقم (٧,١٥). تعريف الصنف Slip بعد إضافة الإجراءات متعددة الاستخدام.

```
    Public Function GetSlipId() As Integer
       Return slipId
    End Function
    Public Function GetSlipWidth() As Integer
       Return slipWidth
    End Function
    Public Function GetSlipLength() As Integer
       Return slipLength
    End Function

    'set accessor methods
    Public Sub SetSlipId(ByVal aSlipId As Integer)
       slipId = aSlipId
    End Sub
    Public Sub SetSlipWidth(ByVal aSlipWidth As Integer)
       slipWidth = aSlipWidth
    End Sub
    Public Sub SetSlipLength(ByVal aSlipLength As Integer)
       slipLength = aSlipLength
    End Sub

End Class
```

تابع الشكل رقم (٧,١٥).

```
' Chapter 7 TesterFour Example 4

Module TesterFour
   Sub Main()
      Dim slip1, slip2 As Slip

      'create slip using 3-parameter constructor
      slip1 = New Slip(1, 10, 20)

      'create slip using 1-parameter constructor
      slip2 = New Slip(2)

      ' retrieve & display info for both slips
      Console.WriteLine(slip1.TellAboutSelf())
      Console.WriteLine(slip2.TellAboutSelf())

      'compute lease for slip1 without discount, then with a 10% discount
      Console.WriteLine("slip1 fee is " & slip1.LeaseSlip())
      Console.WriteLine("With a 10% discount it's " & _
         slip1.LeaseSlip(10))
      'lease slip2 without a discount, then with a 20% discount
      Console.WriteLine("slip2 fee is " & slip2.LeaseSlip())
      Console.WriteLine("With a 20% discount it's " & -
        slip2.LeaseSlip(20))

   End Sub
End Module
```

الشكل رقم (٧,١٦). أوامر الصنف TesterFour.vb.

```
Slip: Id = 1, Width = 10, Length = 20
Slip: Id = 2, Width = 12, Length = 25
slip1 fee is 800
With a 10% discount it's 720
slip2 fee is 900
With a 20% discount it's 720
```

الشكل رقم (٧,١٧). مخرجات الصنف TesterFour.

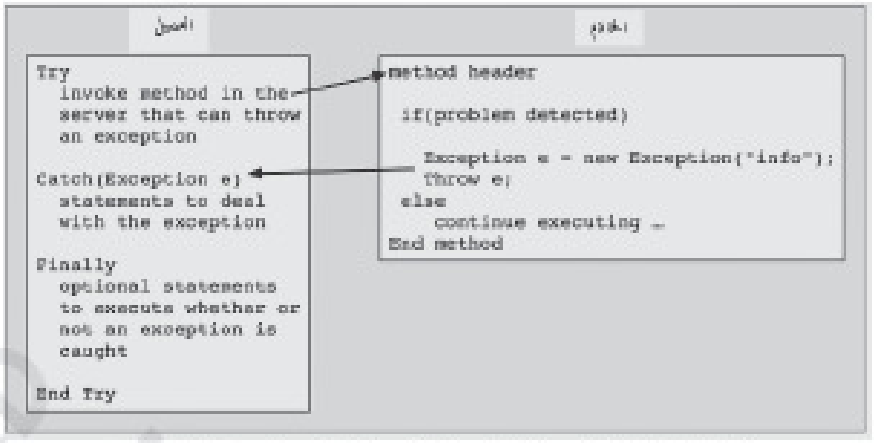
التعامل مع الاستثناءات
Working with Exceptions

تستخدم لغة VB .NET الاستثناءات (Exceptions) لكي تعلمك عن حدوث أخطاء أو مشاكل أو ظروف غير عادية أثناء تشغيل البرنامج، والاستثناء هو كائن مثله مثل أكثر الأشياء في لغة VB .NET، ولكي نكون أكثر دقة، هو كائن من الصنف Exception المعرف داخل اللغة أو من أحد الأصناف المشتقة منه.

ولقد أوضحنا في الباب الأول أن المعالجة المعتمدة على الكائنات (OO Processing) تستخدم نموذج العميل–الخادم حيث يستدعي العميل (كائن) إجراءاً من الخادم (كائن آخر) ممرراً إليه معاملات (وربما لا يمرر)، عندئذ ينجز الخادم المهمة وربما يعيد نتيجة المعالجة للعميل أو لاّ. ويستخدم الخادم الاستثناءات لكي يعلم العميل عن حدوث مشكلة، فربما تكون المعاملات التي يتم تمريرها إليه غير مناسبة، وربما لا يستطيع الخادم إتمام عملية المعالجة لوجود ظروف أخرى.

وعند حدوث مثل ذلك الموقف، ينشئ الخادم كائناً من الصنف Exception الذي يحتوي على معلومات حول المشكلة، ثم يرسل هذا الكائن إلى العميل. عندئذ لابد أن يكون العميل جاهزاً لاستقبال هذا الكائن ويأخذ الحدث المناسب.

تستخدم لغة VB .NET خمس كلمات محجوزة للتعامل مع الاستثناءات وهي الكلمة Try، والكلمة Catch، والكلمة Finally، والكلمة End Try، والكلمة Throw. يستخدم العميل أربع كلمات من هؤلاء الكلمات ويستخدم الخادم الكلمة Throw فقط. وسوف نشرح خلال هذه الفقرة استخدام هذه الكلمات لإضافة الأوامر المنطقية للصنف Slip وذلك للتأكد من صحة البيانات. يوضح الشكل رقم (٧,١٨) التفاعل الذي يحدث بين كل من العميل والخادم عند حدوث استثناء.



الشكل رقم (٧,١٨). التفاعل بين الخادم و العميل عند حدوث استثناء.

عندما يستدعي العميل إجراءاً من الخادم ويوجد هناك احتمال حدوث مشكلة ومن ثم استقبال استثناء من هذا الإجراء، فعندئذ لابد من كتابة استدعاء لهذا الإجراء داخل التركيب الذي يبدأ بالكلمة Try وينتهي بالكلمة End Try. فإذا حدثت مشكلة ما عند تنفيذ هذا الإجراء، فإن الخادم ينشئ كائناً من الصنف Exception ثم يرسله إلى العميل مستخدماً الكلمة Throw. عندئذ يتلقى العميل هذا الكائن داخل التركيب Catch وينفذ الأوامر اللازمة للتعامل مع الاستثناء. أما التركيب Finally فهو اختياري (أي يمكن أن يلغى) وسوف تنفذ أوامره سواء حدث استثناء أو لم يحدث.

التحقق من صحة البيانات للصفة slipId

Data Validation for slipId

إن المراسي في شركة برادشو ترتبط برصيف ولا يمكن أن يسع الرصيف لأكثر من ٥٠ مرسى. وعلى هذا فإن رقم المرسى (slipId) يجب ألا يزيد عن ٥٠؛ لذلك يجب علينا إضافة الأوامر المنطقية للتحقق من صحة الرقم الممرر للإجراء SetSlipId والمسؤول عن إسناد رقم المرسى والذي يجب أن يتراوح بين ١ إلى ٥٠، حيث يمكن إنشاء كائن من الصنف Exception وإرساله إلى كائن العميل (الكائن المستدعي لإجراء SetSlipId) مستخدماً الكلمة Throw عند استقبال رقم يقع خارج هذه الحدود. يستخدم المثال التالي الصنف Exception المعرف داخل لغة VB .NET، ولكن يمكنك تعريف أصناف خاصة بك مشتقة من هذا الصنف واستخدامها عند حدوث المشكلة كما سنتعلم خلال الفصول التالية.

أولاً، سوف نعرّف ثابتاً باسم MAXIMUM_NUMBER_OF_SLIPS داخل الصنف Slip يحتوي على القيمة العظمى لعدد المراسي المسموح بها داخل الرصيف (في هذه الحالة ٥٠)، وهذا الأسلوب مفيد جداً لتبسيط عملية

صيانة البرنامج، فمثلاً إذا أرادت الشركة تغيير القيمة العظمى لعدد المراسي فهذا يتطلب ببساطة تغيير الرقم المسند للثابت دون التأثير على بقية البرنامج.

```
Private Const MAXIMUM_NUMBER_OF_SLIPS As Integer = 50
```

ثم نكتب أمر If الذي يتحقق إذا كانت القيمة المستقبلة aSlipId تقع داخل الحدود أم لا، فإذا وقعت القيمة خارج الحدود المقبولة، سوف ننشئ كائناً من الصنف Exception ثم نرسله للكائن العميل. ولإنشاء كائن من الصنف Exception نستخدم الكلمة New مع تمرير نص يصف الخطأ الذي حدث لإجراء إنشاء الكائن.

ولاحظ أننا استخدمنا اسم الثابت MAXIMUM_NUMBER_OF_SLIPS بدلاً من القيمة ٥٠ وذلك في كل من أمر If وأمر توصيف الخطأ، حيث تم لصق اسم الثابت مع النص الذي يصف الخطأ. وعلى الجانب الآخر إذا وقعت القيمة المستقبلة داخل الحدود المقبولة، فسوف يسند الأمر Else القيمة المستقبلة داخل الصفة SlipId.

```
Public Sub SetSlipId(ByVal aSlipId As Integer)
      'reject slipId if < 0 or > maximum
      If aSlipId < 1 Or aSlipId > MAXIMUM_NUMBER_OF_SLIPS Then
            Throw New Exception("Slip Id not between 1 and " _
                  & MAXIMUM_NUMBER_OF_SLIPS)
      Else
            slipId = aSlipId
      End If
End Sub
```

التحقق من صحة بيانات الصنف slipWidth
Data Validation for slipWidth

إن عرض مراسي شركة برادشو يأخذ قيمة من عدة قيم (١٠ أو ١٢ أو ١٤ أ. ١٦)، ولا يمكن أن يخرج عن هذه القيم؛ ولذلك سيتم إضافة الأوامر المنطقية للتحقق من القيمة المستقبلة للإجراء SetSlipWidth والمسؤولة عن إسناد عرض المرسى (تذكر أن حساب قيمة الإيجار يعتمد على عرض المرسى الذي يجب أن يكون صحيحاً). سوف نقوم بتعريف مصفوفة من النوع Integer تسمى VALID_SLIP_WIDTHS داخل الصنف Slip لتحتوي على القيم الصحيحة لعرض المرسى، وبدلاً من استخدام الكلمة Const، سوف تستخدم الكلمات Shared و ReadOnly. والكلمة Shared تعني أن هذه المصفوفة مشتركة بين جميع كائنات الصنف، والكلمة ReadOnly تعني أنه لا يمكن تغيير محتويات هذه المصفوفة (أي مصفوفة ثابتة المحتويات).

```
Private Shared ReadOnly VALID_SLIP_WIDTHS As Integer() = _
      {10, 12, 14, 16}
```

سوف نتجول داخل عناصر المصفوفة للبحث على القيمة المستقبلة aSlipWidth، فإذا حصلنا عليها تكون القيمة المستقبلة صحيحة، وإذا وصلنا إلى نهاية المصفوفة ولم نجدها، تكون القيمة المستقبلة غير صحيحة وعندئذ ننشئ كائناً من الصنف Exception ونرسله إلى كائن العميل.

مثل تكرارات التجوال داخل المصفوفات التي درسناها في كل من الفصل الثالث والرابع، سوف نعرّف متغيراً من النوع Bolean اسمه ValidWidth ونسند له القيمة False، ثم نعرف التكرار For-Next الذي يستمر إلى أن يجد القيمة المستقبلة داخل المصفوفة أو يصل إلى نهاية المصفوفة دون أن يحصل عليها. ويحتوي جسم التكرار على أمر If الذي يقوم بالمقارنة بين القيمة المستقبلة aSlipWidth وأحد عناصر المصفوفة VALID_SLIP_WIDTHS(i). فإذا نجحت عملية المقارنة، يغير قيمة المتغير ValidWidth إلى True الذي يؤدي إلى إنهاء التكرار. لاحظ أن هذا التكرار يستخدم لتحديد القيمة العليا التعبير VALID_SLIP_WIDTHS.Length - 1.

```
Dim validWidth As Boolean = False
'search for a valid width
Dim i As Integer
For i = 0 To VALID_SLIP_WIDTHS.Length - 1
      If aSlipWidth = VALID_SLIP_WIDTHS(i) Then validWidth = True
Next i
```

ثم نكتب أمر if الذي سيتحقق من قيمة المتغير validWidth، فإذا كان المتغير يساوي القيمة True سوف نسند القيمة المستقبلة aSlipWidth إلى الصفة slipWidth، أما إذا كان المتغير مازال يساوي القيمة False، سوف ننشئ كائناً من النوع Exception الذي يحتوي على وصف المشكلة إلى كائن العميل ونرسله.

```
'if a valid width found, set value
If validWidth Then
      slipWidth = aSlipWidth
Else 'else throw exception
      Throw New Exception("Invalid Slip Width")
End If
```

لقد رأينا في هذا المثال أن إجراءات الوصول Setters سوف تستدعى بواسطة إجراء الإنشاء Constructor، والذي إذا استقبل كائناً من النوع Exception سوف يتم إرساله بشكل تلقائي إلى كائن العميل. ولكن بالطبع يمكنك إضافة التركيب Try إلى إجراء الإنشاء لعمل ذلك صراحة، ولكن لا داعي لإضافة أوامر ليس لها احتياج أو تجعل الصنف Slip أكثر تعقيداً. يوضح الشكل رقم (٧.١٩) تعريف الصنف Slip بعد تعديله مشتملاً على إجراءات الوصول Setters الجديدة.

```
' Chapter 7 Slip class Example 5
' Illustrate Validation and Exceptions

Public Class Slip

       'attributes
       Private slipId As Integer
       Private slipWidth As Integer
       Private slipLength As Integer

       'shared attribute ("static" or "class variable")
       Private Shared numberOfSlips As Integer = 0
'constants
Private Const DEFAULT_SLIP_WIDTH As Integer = 12
Private Const DEFAULT_SLIP_LENGTH As Integer = 25

'constants for validation
Private Const MAXIMUM_NUMBER_OF_SLIPS As Integer = 50
Private Shared ReadOnly VALID_SLIP_WIDTHS As
Integer() = {10, 12, 14, 16}

    'first constructor (one parameter)
    'Overloads keyword not used with constructors
    Public Sub New(ByVal aSlipId As Integer)
           'invoke other constructor, passing default values
           Me.New(aSlipId, DEFAULT_SLIP_WIDTH, DEFAULT_SLIP_LENGTH)
    End Sub

    'second constructor (three parameters)
    Public Sub New(ByVal aSlipId As Integer, ByVal aSlipWidth As
    Integer, _
    ByVal aSlipLength As Integer)
           'invoke setter methods to populate attributes
           SetSlipId(aSlipId)
           SetSlipWidth(aSlipWidth)
           SetSlipLength(aSlipLength)
           'increment shared attribute
   numberOfSlips += 1
    End Sub

    'custom method LeaseSlip calculates and returns fee
    Public Overloads Function LeaseSlip() As Single
           Dim fee As Single
           Select Case slipWidth
                  Case 10
                         fee = 800
                  Case 12
                         fee = 900
                  Case 14
                         fee = 1100
                  Case 16
                         fee = 1500
                  Case Else
                         fee = 0
           End Select
           Return fee
    End Function

    'overloaded custom method LeaseSlip if discount requested
    Public Overloads Function LeaseSlip(ByVal aDiscountPercent As
      Single) _
     As Single
           'invoke LeaseSlip() to get fee
           Dim fee As Single = Me.LeaseSlip()
           'calculate and return discount fee
           Dim discountFee As Single = fee *(100 - aDiscountPercent)
             / 100
```

الشكل رقم (٧,١٩). تعريف الصنف Slip بعد إضافة الاستثناءات.

```
        Return discountFee
    End Function

    'custom method TellAboutSelf
    Public Function TellAboutSelf() As String
        Dim info As String
        info = "Slip: Id = " & GetSlipId() & ", Width = " &
        GetSlipWidth()
         & ", Length = " & GetSlipLength()
        Return info
    End Function

    'shared accessor method (static method)
    Public Shared Function GetNumberOfSlips() As Integer
        Return numberOfSlips
    End Function

    'get accessor methods
    Public Function GetSlipId() As Integer
        Return slipId
    End Function
    Public Function GetSlipWidth() As Integer
        Return slipWidth
    End Function
    Public Function GetSlipLength() As Integer
        Return slipLength
    End Function

    'set accessor methods with validation
    Public Sub SetSlipId(ByVal aSlipId As Integer)
        'reject slipId if < 0 or > maximum
        If aSlipId < 1 Or aSlipId > MAXIMUM_NUMBER_OF_SLIPS Then
            Throw New Exception("Slip Id not between 1 and " _
            & MAXIMUM_NUMBER_OF_SLIPS)
        Else
            slipId = aSlipId
        End If
    End Sub
    Public Sub SetSlipWidth(ByVal aSlipWidth As Integer)
        'reject slipWidth if not in list of valid values
        Dim validWidth As Boolean = False
        'search for a valid width
        Dim i As Integer
        For i = 0 To VALID_SLIP_WIDTHS.Length - 1
            If aSlipWidth = VALID_SLIP_WIDTHS(i) Then validWidth = True
        Next i

        'if  a validwidth found, set value
        If validWidth Then
            slipWidth = aSlipWidth
        Else 'else throw exception
            Throw New Exception("Invalid Slip Width")
        End If
    End Sub

    Public Sub SetSlipLength(ByVal aSlipLength As Integer)
        'write validation for length as exercise
        slipLength = aSlipLength
    End Sub

End Class
```

تابع الشكل رقم (٧,١٩).

استقبال الاستثناءات
Catching Exceptions

ترسل الاستثناءات بواسطة إجراءات الخادم إلى العميل الذي قام باستدعائها لكي تعلمه بحدوث مشكلة ما. يلعب الصنف Slip دور الخادم في هذا المثال لأنه يحتوي على إجراءات لديها القدرة على إنشاء كائن من النوع Exception عند استقبال بيانات غير صحيحة وإلقائه، وسوف يلعب صنف الاختبار الخامس TesterFive دور العميل الذي سيختبر شفرة التحقق من صحة البيانات المعرفة في الصنف Slip. إذا كان من المتوقع أن أوامر البرنامج الخاص بك ربما تستقبل استثناء، فلابد من إعداد هذه الأوامر لاستقبال هذا الاستثناء. وإذا لم تفعل ذلك فإن محرك اللغة VB .NET المشترك سوف ينهي المعالجة في حالة إلقاء استثناء ولم يتم استقباله. إن أول خطوة لاستقبال الاستثناء هي وضع الأوامر داخل تركيب Try كما هو واضح في الشكل رقم (٧.٢٠) حيث يبدأ التركيب بالكلمة المحجوزة Try متبوعاً بأوامر البرنامج المتوقعة أن تعيد استثناء ثم تنتهي بالكلمة End Try. تعرِّف الأوامر التالية متغير إشارة aSlip من الصنف Slip متبوعاً بالتركيب Try الذي يحتوي على أمر إنشاء كائن من الصنف Slip.

```
Dim aSlip As Slip

Try 'force an exception with invalid slipID (150)
      aSlip = New Slip(150, 10, 25)
      Console.WriteLine(aSlip.TellAboutSelf())
Catch theException As Exception
      Console.WriteLine("An exception was caught: " & _
            theException.ToString())
End Try
```

لاحظ أن التركيب Try يحتوي على أمرين. فإذا تسبب الأمر الأول في إصدار استثناء، عندئذ لا يتم تنفيذ الأمر الثاني (استدعاء الإجراء TellAboutSelf)، ثم يبدأ تنفيذ أوامر التركيب Catch مباشرة.

يستدعي الأمر الأول (داخل التركيب Try) إجراء إنشاء كائن من الصنف Slip ممرراً إليه ثلاثة معاملات (رقم المرسى SlipId، وعرض المرسى SlipWidth، وطول المرسى SlipLength). وتذكر أن إجراء الإنشاء يستدعي بدوره إجراءات الوصول Setters لإسناد المعاملات إلى الصفات. ولذلك سيتم استدعاء الإجراء SetSlipId الذي يحتوي على إصدار استثناء وإلقائه إذا تم استقبال رقم مرسى غير صحيح. المثال السابق يقوم متعمداً بتمرير رقم مرسى غير صحيح (١٥٠) لإجراء إنشاء كائن من الصنف Slip والذي يجعل إجراء الوصول Setters ينشئ كائناً من الصنف Exception ويلقيه.

```
' Chapter 7 TesterFive Example 5
Module TesterFive
   Sub Main()
      Dim aSlip As Slip
      Try 'force an exception with invalid slipID (150)
         aSlip = New Slip(150, 10, 25)
         Console.WriteLine(aSlip.TellAboutSelf())
      Catch theException As Exception
         Console.WriteLine("An exception was caught: " & _
            theException.ToString())
      End Try

      Try 'force an exception with invalid width (15)
         aSlip = New Slip(1, 15, 25)
         Console.WriteLine(aSlip.TellAboutSelf())
      Catch theException As Exception
         Console.WriteLine("An exception was caught: " & _
           theException.ToString())
      Finally
         Console.WriteLine("Finally block always executes")
      End Try

      'create a Slip instance using valid id & width
      Try
         aSlip = New Slip(2, 10, 25)
         Console.WriteLine(aSlip.TellAboutSelf())
      Catch theException As Exception
         Console.WriteLine("An exception was caught: " & _
           theException.ToString())
      Finally
         Console.WriteLine("Finally block always executes")
      End Try

   End Sub
End Module
```

الشكل رقم (٧,٢٠). أوامر الصنف TesterFive.vb.

وكما ذكرنا سابقاً أن إجراء الإنشاء يعيد إلقاء الاستثناء بشكل تلقائي؛ ولذلك فإنه ليس من الضروري إعداد إجراء الإنشاء لكي يستقبل استثناء ويلقيه صراحة. وإذا كنت تفضل فعل ذلك صراحة، فيمكنك إضافة التركيب Catch لاستقبال الاستثناء، ثم استخدام الأمر Throw لإعادة إرسال الاستثناء إلى العميل الأصلي الذي استدعى إجراء الإنشاء.

وبعد التركيب Try داخل الصنف TesterFive يجب إضافة التركيب Catch لاستقبال الاستثناء المرسل من الإجراء SetSlipId ومعالجته. مثلما نفعل عند كتابة رأس الإجراء، فيجب تعريف متغير إشارة لاستقبال كائن الصنف Exception، ثم نكتب أوامر التركيب Catch حيث نستدعي في هذا المثال الإجراء WriteLine لعرض وصف

الاستثناء الذي تم استقباله. لاحظ أنه قد تم استخدام الإجراء ToString لاستخلاص وصف الاستثناء وتمريره إلى الإجراء WriteLine.

```
Catch theException As Exception
      Console.WriteLine("An exception was caught: " & _
            theException.ToString())
```

والآن يمكن إضافة شفرة اختبار التحقق من صحة بيانات عرض المرسى إلى الصنف TesterFive حيث نعرف كائناً من الصنف Slip داخل التركيب Try ونمرر لإجراء الاستثناء قيمة غير صحيحة (١٥) لعرض المرسى.

```
Try 'force an exception with invalid width (15)
      aSlip = New Slip(1, 15, 25)
      Console.WriteLine(aSlip.TellAboutSelf())
```

ثم نكتب أوامر التركيب Catch لاستقبال الاستثناء ومعالجته.

```
Catch theException As Exception
      Console.WriteLine("An exception was caught: " & _
            theException.ToString())
```

يستخدم تركيب Finally لإضافة أوامر يتم تنفيذها سواء تم استقبال استثناء أم لا، حيث يحتوي الصنف TesterFive على التركيب Finally هكذا:

```
Finally
      Console.WriteLine("Finally block always executes")
```

يوضح الشكل رقم (٧.٢٠) شفرة الصنف TesterFive كاملة، ويوضح الشكل رقم (٧.٢١) مخرجات هذا الصنف (لاحظ أن مخرجات شفرة البرنامج الخاص بك ستكون أكثر تفصيلاً).

```
An exception was caught: System.Exception: Slip Id not between 1 and 50
An exception was caught: System.Exception: Invalid Slip Width
Finally block always executes
Slip: Id = 2, Width = 10, Length = 25
Finally block always executes
```

الشكل رقم (٧.٢١). مخرجات الصنف TesterFive.

ملخص الفصل
Chapter Summary

- يطلق على إجراءات الوصول (Accessor Methods) اسم الإجراءات القياسية أو الاعتيادية (Standard Metods) لأنها من المعتاد أن تتواجد في معظم أصناف مجال المشكلة، أما الإجراءات الخاصة (Custom Methods) فهي الإجراءات المسؤولة عن معالجة البيانات على عكس الإجراءات الاعتيادية المسؤولة عن استرجاع قيم الصفات وتخزينها.
- تستخدم الكلمة المحجوزة Shared لتوصيف متغيرات الصنف (Class Variables) وإجراءات الصنف (Class Methods) والتي تكون مشتركة على مستوى جميع الكائنات لصنف ما، أي لا توجد نسخة خاصة من هذه المتغيرات والإجراءات لكل كائن بل توجد نسخة واحدة على مستوى الصنف.
- يتم استدعاء إجراءات الصنف بكتابة اسم الصنف متبوعاً بنقطة ثم اسم الإجراء وذلك على عكس إجراءات الكائنات التي يتم استدعاؤها بواسطة اسم الكائن، كما يمكنك أيضاً استدعاء إجراءات الصنف مستخدماً اسم الكائن.
- يتكون توقيع الإجراء (Method Signature) من اسم الإجراء وقائمة المعاملات (Parameter List) المقررة له حيث تميز لغة VB .NET الإجراء بتوقيعه لا باسمه؛ ولذلك يمكن تعريف أكثر من إجراء بنفس الاسم ولكن بقائمة معاملات مختلفة، عندئذ ترى لغة VB .NET هذه الإجراءات مختلفة، ويسمى الإجراء الذي له الاسم نفسه مع إجراء آخر في الصنف نفسه ولكن بقائمة معاملات مختلفة اسم Overloaded Method أي إجراء متعدد الاستخدام.
- تستخدم لغة VB .NET الاستثناءات (Exceptions) لكي تعلمك عن حدوث أخطاء أو مشاكل أو ظروف غير عادية أثناء تشغيل البرنامج، والاستثناء هو كائن مثله مثل كل شيء في لغة VB .NET، ولكي نكون أكثر دقة، هو كائن من الصنف Exception المعرف داخل اللغة أو من أحد الأصناف المشتقة منه.
- تستخدم لغة VB .NET خمس كلمات محجوزة للتعامل مع الاستثناءات وهي الكلمة Try، والكلمة Catch، والكلمة Finally، والكلمة End Try، والكلمة Throw. يستخدم العميل أربع كلمات من هؤلاء الكلمات ويستخدم الخادم الكلمة Throw فقط.
- إذا كان من المتوقع أن أوامر البرنامج الخاص بك ربما تستقبل استثناء، فلابد من إعداد هذه الأوامر لاستقبال هذا الاستثناء. وإذا لم تفعل ذلك فإن محرك اللغة VB .NET المشترك سوف ينهي المعالجة في حالة إلقاء استثناء ولم يتم استقباله.

المصطلحات الأساسية

Key Terms

الاستثناء (Exception)	إجراءا الوصول (Accessor Method)
إجراء متعدد الاستخدام (Overloaded Method)	إجراء خاص (Custom Method)
توقيع الإجراء (Method Signature)	إجراء معاد تعريفه (Overridden Method)
متغيرا الصنف (Class Variable)	إجراءا الصنف (Class Method)

أسئلة المراجعة

Review Questions

١- معنى التسلسل:

(أ) فصل العناصر في السلاسل النصية.

(ب) طرح العناصر الرقمية.

(ج) دمج العناصر مع بعضها البعض.

(د) ظهور العناصر.

٢- المتغيرات المشتركة:

(أ) لا تكون متفقة

(ب) نادراً ما تكون مكررة.

(ج) نسخة واحدة فقط.

(د) على كل منها نسخة متغيرات.

٣- الإجراء المشترك:

(أ) من الممكن حدوثه بالإشارة إلى اسم الصنف.

(ب) من الممكن حدوثه بالإشارة إلى اسم الكائن.

(ج) كل من (أ) و (ب).

(د) لا (أ) ولا (ب).

٤- متغيرات القراءة فقط المشتركة يطلق عليها:

(أ) الثوابت.

(ب) كائن الثوابت.

(ج) المتغيرات الحرة.

(د) جميع الإجابات خاطئة.

٥- الكلمة المحجوزة Me:

(أ) لا أساس لها من الصحة.

(ب) ترجع إلى الصنف وكل قيمه الثابتة.

(ج) ترجع إلى صفة الكائن.

(د) ترجع إلى الكائن الذي ينفذ الإجراء.

٦- توقيع الإجراء هو:

(أ) اسم الإجراء.

(ب) رأس الإجراء.

(ج) اسم الإجراء وقائمة المعاملات.

(د) معطيات الإجراء.

٧- الإجراء متعدد الاستخدام:

(أ) لا يمكن استدعاؤه.

(ب) إجراء له العديد من الخطوات.

(ج) له نفس قائمة معاملات إجراء آخر ولكن باسم مختلف.

(د) له نفس اسم إجراء آخر ولكن بقائمة معاملات مختلفة.

٨- أي من الآتي غير صحيح عن الإجراء متعدد الاستخدام:

(أ) من الممكن أن يكون إجراء الإنشاء متعدد الاستخدام.

(ب) ليس من الممكن أن تكون إجراءات الوصول متعددة الاستخدام.

(ج) يمكن أن تكون الإجراءات الخاصة متعددة الاستخدام.

(د) يمكن أن تكون إجراءات الوصول متعددة الاستخدام.

٩- الاستثناء هو:

(أ) كائن.

(ب) صفة.

(ج) إجراء.

(د) لا إجابة مما سبق.

١٠- الكلمة المحجوزة Throw :

(أ) ترسل الاستثناء إلى الإجراء المستدعي.

(ب) يجب أن تكون في رأس الإجراء المرسل للاستثناء.

(ج) تستخدم مع الكلمة المحجوزة Return.

(د) ليس لها قيمة.

١١- الإجراء الذي يستقبل الاستدعاء :

(أ) ينتهي بشكل عام.

(ب) يجب أن يمسك الاستثناء.

(ج) كل من (أ) و (ب).

(د) لا (أ) ولا (ب).

١٢- الكلمة المحجوزة Finally :

(أ) جزء من Select Case.

(ب) تنفذ بشكل دائم.

(ج) تنفذ فقط عند اكتشاف استثناء.

(د) ليس لها قيمة.

١٣- صفة طول المصفوفة :

(أ) يحتوي على رقم العناصر المتفقة.

(ب) يحتوي على رقم العناصر.

(ج) يستخدم فقط مع تكرار For-Next.

(د) ليس لها قيمة.

١٤- عند تحميل الإجراء يجب :

(أ) تغيير اسم الإجراء.

(ب) تحديد رأس الإجراء بالكلمة المحجوزة Overloads ما لم يكن إجراء إنشاء.

(ج) تحديد رأس جميع الإجراءات المعاد استخدامها بالكلمة Overloads ما لم يكن إجراء إنشاء.

(د) لا تستطيع تحميل الإجراء بالكلمة المحجوزة Override.

١٥- توافق الإسناد يعني :

(أ) أن المتغير من الممكن أن يسند لمتغير آخر بنوع بيانات مختلف.

(ب) متغير من الممكن أن يسند لمتغير آخر بنفس نوع البيانات.

(ج) أن متغيراً من الممكن أن يستخدم فقط مع نوع البيانات Single.

(د) يجب أن يحول قيمة السلاسل النصية ثم يقوم بعمل تحديد المهام.

أسئلة المناقشة

Discussion Questions

١- لقد رأينا في هذا الفصل أن إجراءات إنشاء الصنف Slip يستدعي إجراءات المرور Setters لإسناد القيم الممررة له إلى الصفات. ما هي المشاكل المتوقعة عند استبدال هذه الإجراءات بأوامر إسناد مباشرة إلى الصفات؟

٢- لقد رأينا أيضاً أن إجراء إنشاء الصنف Slip ذي المعامل الواحد يستدعي إجراء إنشاء الصنف Slip ذي المعاملات الثلاثة. ما فائدة ذلك؟ وما التغييرات المطلوبة لكي تجعل إجراء إنشاء الصنف Slip ذي المعامل لا يستدعي إجراء إنشاء الصنف Slip ذي المعاملات الثلاثة؟

٣- ما هي المشاكل المتوقعة إذا جعلنا جميع صفات الصنف Slip من النوع Shared؟

مشاريع الفصل

Projects

١- افترض أن شركة برادشو أضافت رصيفاً جديداً يحتوي على ٣٠ مرسى بنفس الحجم (١٢ قدماً للعرض، ٣٠ قدماً للطول)، وأخذ المرسى الأول الرقم ١ وأخذ المرسى الثاني الرقم ٢، وهكذا.

(أ) قم بنسخ محتويات المجلد Ex04 من المجلد Chap07\Examples المتواجد داخل ملفات بيانات الكتاب إلى المجلد Chap07\Projects\Proj01\Project1 داخل مجلد العمل الخاص بك.

(ب) اكتب الأوامر المطلوبة داخل الملف TestFour.vb لإنشاء ٣٠ مرسى كما هو موضح أعلاه داخل مصفوفة ذات بعد واحد.

(ج) أضف تكراراً داخل الملف TestFour.vb الذي يستدعى الإجراء TellAboutSelf من جميع المراسي الثلاثون.

٢- استمر في استخدام المثال ٤ لتطبيق مفهوم Overloaded Methods كمرشد لك في المشاريع التالية :

(أ) قم بنسخ محتويات المجلد Ex04 من المجلد Chap07\Examples المتواجد داخل ملفات بيانات الكتاب إلى المجلد Chap07\Projects\Proj02\Project2 داخل مجلد العمل الخاص بك.

(ب) أضف الإجراء GetFormattedFee إلى الصنف Slip الذي لا يستقبل معاملات ويستدعي الإجراء LeaseSlip لحساب قيمة الإيجار، ثم يعيد تشكيله في صيغة عملة وإعادتها كمتغير من النوع String.

(ج) أضف إجراء آخر بنفس اسم الإجراء الذي أضفته في الخطوة (ب) إلى الصنف Slip الذي يستقبل معامل نسبة خصم ويستدعي الإجراء LeaseSlip الذي يستقبل معامل نسبة خصم لحساب قيمة الإيجار، ثم يعيد تشكيله في صيغة عملة وإعادتها كمتغير من النوع String.

(د) أضف الأوامر المطلوبة إلى الملف TesterFour لتجربة الإجراءات التي عرفتها في كل من الخطوة (ب) والخطوة (ج).

٣- استمر في استخدام المثال ٤ لتطبيق مفهوم Overloading كمرشد لك في المشاريع التالية:

(أ) قم بنسخ محتويات المجلد Ex04 من المجلد Chap07\Examples المتواجد داخل ملفات بيانات الكتاب إلى المجلد Chap07\Projects\Proj03\Project3 داخل مجلد العمل الخاص بك.

(ب) أضف الإجراء FormatFee إلى الصنف Slip من النوع Shared الذي يستقبل قيمة الإيجار، ثم يعيد تشكيله في صيغة عملة وإعادتها كمتغير من النوع String.

(ج) أضف الأوامر المطلوبة إلى الملف TesterFour لتجربة الإجراءات التي عرفتها في كل الخطوة (ب).