الفصل الثالث

كتل البناء البرمجية

يركز هذا الفصل على كتل البناء البرمجية، فلغة فيجول بيسك كحال اللغات الأخرى، تستخدم كتل بناء برمجية: مثل الإجراءات Procedures، والتوابع الوظيفية Functions، وعبارات الشرط If End If، وعبارات الحلقات، والمتحولات، وغير ذلك من مفاهيم البرمجة الهامة الأخرى.

برنامج ضرب عددين

يوضح هذا البرنامج كيف تستخدم الإجراءات والتوابع الوظيفية ضمن البرامج المطوّرة في لغة فيجول بيسك.

التمثيل المرئي لبرنامج ضرب عددين

سننجز الآن التمثيل المرئي لبرنامج ضرب عددين:

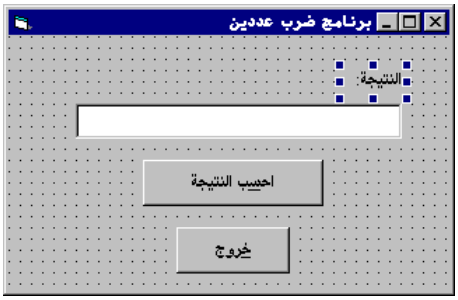
- أنشئ الدليل C:\VB5Prg\Ch03، سنستخدم هذا الدليل لحفظ العمل المنجز.

- افتح مشروعاً جديداً من نوع Standard EXE، واحفظ نموذج المشروع بالاسم Multiply.frm في الدليل C:\VB5Prg\Ch03، واحفظ ملف المشروع بالاسم Multiply.Vbp في ذات الدليل.
- أنشئ النموذج frmMultiply طبقاً للجدول ٣-١.

يفترض أن يظهر النموذج لدى اكتماله كما في الشكل ٣-١.

الجدول ٣-١. جدول خصائص النموذج frmMultiply.

الكائن	الخاصية	القيمة
Form	**Name**	**frmMultiply**
	Caption	برنامج ضرب عددين
	RightToLeft	True
CommandButton	**Name**	**cmdExit**
	Caption	&خروج
	RightToLeft	True
CommandButton	**Name**	**cmdCalculate**
	Caption	احـ&سب النتيجة
	RightToLeft	True
Text Box	**Name**	**txtResult**
	Text	(اجعله فارغاً)
	RightToLeft	True
Label	**Name**	**lblResult**
	Caption	النتيجة:
	RightToLeft	True



الشكل ٣-١

النموذج frmMultiply

بعد انتهاء تصميمه.

إدخال نص برنامج ضرب عددين

سندخل الآن، نص برنامج ضرب عددين:

- ❑ انقر نقراً مزدوجاً على النموذج frmMultiply، لإظهار إطار نص البرنامج. توجد لائحتين عند قمة إطار نص البرنامج. دع اللائحة اليسارية تشير إلى General ، واليمينية إلى Declarations، (وبالتالي فإن مجموع قراءة اللائحتين، سيشير إلى General Declarations، أي قسم التصاريح العامة).
- ❑ اكتب العبارة التالية ضمن قسم التصاريح العامة General Declarations للنموذج frmMultiply:

```
'يجب التصريح عن كل المتحولات
Option Explicit
```

يجب من الآن فصاعداً، التصريح عن المتحول قبل استخدامه في البرنامج.

- ❑ أدخل النص التالي ضمن الإجراء cmdCalculate_Click() للنموذج frmMultiply:

```
Private Sub cmdCalculate_Click()
   Multiply 2, 3
End Sub
```

يُنفذ النص الذي أدخلته ضمن الإجراء cmdCalculate_Click() آلياً، عند نقر الزر **احسب النتيجة**. يُنفذ نص البرنامج هذا، الإجراء المدعو Multiply، سنتعلم بعد قليل كيف نضيف الإجراء Multiply إلى برنامجنا.

- ❑ أدخل النص التالي ضمن الإجراء cmdExit_Click() للنموذج frmMultiply:

```
Private Sub cmdExit_Click()
   End
End Sub
```

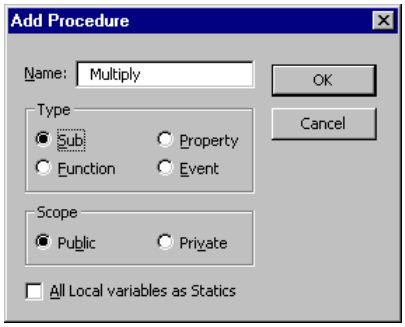
ستضيف إجراء جديداً الآن، إلى النموذج frmMultiply، وسنطلق على هذا الإجراء التسمية Multiply. إليك طريقة إنجاز ذلك:

- ❑ انقر نقراً مزدوجاً على النموذج frmMultiply لإظهار إطار نص البرنامج.
- ❑ انقر **Add Procedure** من القائمة **Tools** للغة فيجول بيسك.

يستجيب فيجول بيسك بإظهار مربع الحوار Add Procedure.

- ❑ أدخل Multiply في مربع النص Name ضمن مربع الحوار Add Procedure.
- ❑ ضع علامة بجانب Sub ضمن الحقل Type. (لأنا نوع الإجراء هو روتين فرعي).
- ❑ ضع علامة بجانب Public ضمن الحقل Scope. (لأن نريد أن يكون مجال الإجراء، مرئياً لجميع وحدات المشروع).

يبين الشكل ٣-٢ كيف يبدو مربع الحوار Add Procedure الآن.



الشكل ٣-٢
إضافة الإجراء
المدعو Multiply.

- ❑ انقر الزر **OK** لإغلاق مربع الحوار Add Procedure.

يستجيب فيجول بيسك بإضافة الإجراء Multiply() إلى المنطقة العامة من النموذج frmMultiply؛ ويظهر نص الإجراء Multiply بالشكل التالي:

```
Public Sub Multiply()
End Sub
```

سنحتاج إلى تعديل السطر الأول من الإجراء Multiply() ليصبح الإجراء بعدها كالتالي:

```
Public Sub Multiply(X As Integer, Y As Integer)
End Sub
```

لأننا سنمرر للإجراء Multiply، وسيطين يمثلان العددين المطلوب معرفة ناتج ضربهما.

- ❑ أدخل الآن النص التالي ضمن الإجراء Multiply:

```
Public Sub Multiply(X As Integer, Y As Integer)
    Dim Z
    Z = X * Y
```

```
    txtResult.Text = Str(Z)
End Sub
```

❑ اختر البند **Save Project** من القائمة **File** لفيجول بيسك لتحفظ عملك.

تنفيذ برنامج ضرب عددين

❑ نفّذ برنامج ضرب عددين، وانقر الزر **احسب النتيجة**.

*يستجيب البرنامج بإظهار الرقم ٦ ضمن مربع النص. (يحسب الإجراء ناتج ضرب ٢ مع ٣، ويظهر الناتج ضمن مربع النص، عند نقر الزر **احسب النتيجة**).*

كيف يعمل برنامج ضرب عددين

يُنفذ برنامج ضرب عددين، الإجراء Multiply() لحساب حاصل ضرب عددين.

نص الإجراء cmdCalculate_Click()

ينفذ هذا الإجراء عند نقر الزر **احسب النتيجة**. تَستدعي العبارة الوحيدة الموجودة ضمنه الإجراء Multiply() مع وسيطين هما العددين ٢ و ٣:

```
Private Sub cmdCalculate_Click()
    Multiply 2, 3
End Sub
```

يحتاج الإجراء لمعرفة العددين اللذين سيحسب حاصل ضربهما، ولهذا قدمنا له العددين ٢ و٣ كوسيطين له.

نص الإجراء Multiply()

يمتلك الإجراء Multiply() وسيطين:

يدعى الوسيط الأول X، ويصرح عنه بأنه من نوع صحيح Integer ، والوسيط الثاني Y من

النوع صحيح Integer أيضاً:

```
Public Sub Multiply(X As Integer, Y As Integer)
```

```
    Dim Z
    Z = X * Y
    txtResult.Text = Str(Z)
End Sub
```

يُصرح الإجراء Multiply() عن متحول يدعى Z ويُسند إليه ناتج حاصل ضرب X مع Y.

ثم يستخدم الإجراء، المتحول Z بعدها، لإسناد قيمة Z إلى الخاصية Text لمربع النص txtResult. وهذا طبعاً بعد تحويل قيمة Z إلى ثابت كتابي بواسطة التابع الوظيفي Str().

يمكن التصريح عن إجراء ما، في فيجول بيسك، دون استخدام أي وسائط، فيكون القوسان بعد اسم الإجراء فارغتان، وبالتالي لا يلزم تحديد وسائط عند استدعاء الإجراء، فمثلاً لدينا الإجراء:

```
    Public Sub Updatelabel()
```

يُستدعى هذا الإجراء بالطريقة التالية:

```
    UpdateLabel
```

أي يكفي ذكر اسمه.

بينما استخدمنا في مثالنا الحالي، هذه العبارة:

```
    Multiply  2,3
```

لاستدعاء (تنفيذ) الإجراءات في لغة فيجول بيسك، فمثلاً نستطيع إعادة كتابة الإجراءات cmdCalculate_Click() بالشكل التالي:

```
Private Sub cmdCalculate_Click()
    Call Multiply (2,3)
End Sub
```

ملاحظة

يجب استخدام الأقواس للإحاطة بوسائط الإجراء، عند استدعائه بواسطة العبارة Call. لا تهم الطريقة التي تُستدعى بها الإجراء، ولهذا يمكنك اختيار الطريقة التي تناسبك. لدى إضافة الإجراء Multiply() اخترنا النطاق Scope بأنه من نوع Public (انظر الشكل ٣-٢). ولهذا صرح فيجول بيسك عن الإجراء Multiply بأنه من النوع العام

:Public
Public Sub Multiply()
لو اخترنا النطاق Private بدلاً من Public، لكان شكل التصريح كما يلي:
Private Sub Multiply()
الفرق بين التصريحين، أن الإجراء في حالة Public يكون عاماً، ويمكن استخدامه من قبل أي إجراء آخر في أي موقع من ملفات المشروع، فحسب ما سيتوضح لك عبر الكتاب يمكن للبرنامج الواحد أن يتألف من عدة ملفات (ربما عدة نماذج) وبالتالي، فعند التصريح عن إجراء بأنه Public، فإنّ بمقدور كل ملفات البرنامج استخدامه.
أما عند استخدام Private، فتنعكس الآية، إذ لا يمكن استخدام الإجراء إلا من قبل الإجراءات الموجودة ضمن نفس الملف. وفي حالتنا هذه يمكن لإجراءات النموذج frmMultiply فقط استخدام الإجراء ()Multiply.

استخدام تابع وظيفي داخل برنامج ضرب عددين

يُطلق على الإجراء ()Multiply عبارة إجراء Procedure، لأنه لا يعيد أي قيمة. وهو يشبه التابع الوظيفي أي Function، باستثناء أنه يعيد قيمة بعد استدعائه.
سنكتب الآن نص برنامج يوضح كيفية استخدام التابع الوظيفي. أولاً، احذف الإجراء ()Multiply من برنامج ضرب عددين باتباع ما يلي:

- أظهر إطار نص البرنامج بالنقر المزدوج على منطقة خالية في النموذج frmMultiply.
- حدد موقع الإجراء ()Multiply في إطار نص البرنامج، ستعثر على هذا الإجراء ضمن المنطقة General Declarations، والتي تستطيع الانتقال إليها بوضع اللائحة اليسارية الموجودة عند قمة إطار نص البرنامج على General ووضع اللائحة المجاورة لها على Multiply.
- ضع إضاءة فوق كامل الإجراء. بما في ذلك عنوانه وآخر سطر فيه، (ضع المؤشر عند أول حرف في الإجراء، ثم اضغط Ctrl+A).
- اضغط المفتاح Delete على لوحة المفاتيح.

هذا كل شيء بالنسبة لحذف الإجراء Multiply.

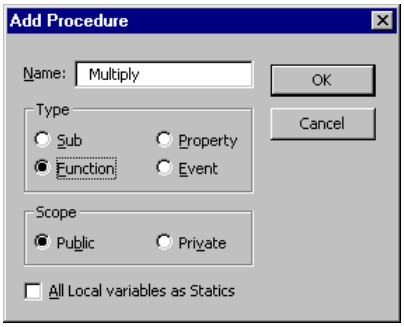
اتبع الآن الخطوات التالية لإضافة تابع وظيفي جديد إلى البرنامج Multiply():

- ❑ أظهر إطار نص برنامج النموذج frmMultiply.
- ❑ اختر **Add Procedure** من القائمة **Tools**.

يستجيب فيجول بيسك بإظهار مربع الحوار Add Procedure.

- ❑ أدخل الاسم Multiply ضمن مربع النص Name.
- ❑ اختر Function ضمن الحقل Type.
- ❑ اختر Public ضمن الحقل Scope.

يفترض أن يبدو مربع الحوار Add Procedure الآن كما في الشكل ٣-٣.



الشكل ٣-٣

مربع الحوار Add Procedure

وإضافة التابع الوظيفي Multiply().

- ❑ انقر الزر **Ok** لإغلاق مربع الحوار Add Procedure.

يستجيب فيجول بيسك بإضافة التابع الوظيفي Multiply() إلى قسم التصاريح العامة General Declarations للنموذج frmMultiply ويظهر إطار نص البرنامج مع التابع الوظيفي Multiply() جاهزاً للتعديل:

```
Public Function Multiply()
End Function
```

- ❑ غير السطر الأول لهذا التابع بحيث يغدو كالتالي:

```
Public Function Multiply(X As Integer, Y As Integer)
End Function
```

والآن يمتلك التابع الوظيفي Multiply() وسيطين (كلاهما من نوع صحيح Integer):

- ❑ أضف النص التالي إلى التابع الوظيفي Multiply():

```
Public Function Multiply(X As Integer, Y As Integer)
```

```
    Dim Z
    Z = X * Y
    Multiply = Z
End Function
```

- غيّر نص الإجراء cmdCalculate_Click() ليصبح كما يلي:

```
Private Sub cmdCalculate_Click()
    txtResult.Text = Str(Multiply(2,3))
End Sub
```

- احفظ المشروع باختيار **Save Project** من القائمة **File**.
- نفّذ برنامج ضرب عددين.
- انقر الزر **احسب النتيجة**.

يتصرف البرنامج كما ترى بذات الطريقة السابقة (في حالة الإجراء Multiply()).

- أنهِ البرنامج بنقر الزر **خروج**.

نص التابع الوظيفي Multiply()

يصرح ضمن هذا التابع عن متحول يدعى Z ويسند إليه ناتج حاصل ضرب الوسيطين X و Y:

```
Public Function Multiply(X As Integer, Y As Integer)
    Dim Z
    Z = X * Y
    Multiply = Z
End Function
```

تُسند آخر عبارة في التابع الوظيفي Multiply() قيمة المتحول Z إلى المتحول Multiply:

```
    Multiply = Z
```

يمثل المتحول Multiply القيمة المعادة من التابع الوظيفي. لنوضح أكثر، التابع الوظيفي اسمه Multiply، ويستعمل اسم التابع الوظيفي ليحتوي على القيمة المعادة، ولهذا لا بد من إسناد القيمة التي سيعيدها التابع الوظيفي إلى متحول (يدعى Multiply) ويحمل اسم ذلك التابع.

```
    Multiply = Z
```

يستطيع كل من يستدعي التابع الوظيفي Multiply() استخدام القيمة المعادة منه حسب ما توضحه الفقرة التالية:

نص الإجراء cmdCalculate_Click()

يُسنِد نص الإجراء القيمة المعادة من التابع الوظيفي Multiply() إلى الخاصية Text لمربع النص txtResult:

```
Private Sub cmdCalculate_Click()
   txtResult.Text = Str(Multiply(2,3))
End Sub
```

كما تلاحظ، يعتبر استخدام تابع وظيفي أكثر تعقيداً بعض الشيء من استخدام الإجراءات. لكنك ستتعود على استخدام التوابع الوظيفية، وستتمكن من تقدير أهميتها، لأنها تساعد بسهولة على قراءة وتوضيح عمل البرامج. لنأخذ العبارات التالية على سبيل المثال:

```
   TtxResult.Text = Str(Multiply(2,3))
```

تقول هذه العبارة ما يلي: " نفّذ التابع الوظيفي Multiply() مع وسيطين هما ٢ و٣، وتحويل القيمة المعادة من Multiply إلى سلسلة نصية (باستخدام التابع الوظيفي Str())، وإسناد السلسلة الناتجة إلى الخاصية Text لمربع النص". ولهذا، فسوف يحوي مربع النص txtResult على السلسلة "٦" داخله (أي ٢×٣ = ٦).

الإجراءات والتوابع الوظيفية والطرائق

حسب ما تقدم في هذا الفصل، فإن الفرق بين الإجراء والتابع الوظيفي أن الإجراء لا يعيد قيمة على عكس التابع الوظيفي. ستصادف عبر الفصول القادمة مصطلحاً جديداً وهو المصطلح Method (طريقة). تعمل الطريقة بشكل مشابه لعمل الإجراءات والتوابع الوظيفية. وعلى العموم. تنجز الطريقة method نوعاً ما من الوظائفية على كائن خاص ما. فمثلاً تمحو العبارة التالية محتويات النموذج الرسومية، والمدعو frmMyForm:

```
frmMyForm.Cls
```

لاحظ أن اسم الطريقة في العبارة السابقة هو Cls. يمكن أن تبدو الطرائق Methods من وجهة نظر المستخدم كالإجراءات والتوابع الوظيفية ولكن بصيغة غريبة. سنتمرن عبر فصول هذا الكتاب على استخدام الطرائق.

عبارات اتخاذ القرار

تصف الفقرات القادمة كتل البناء البرمجية الأساسية التي تتوافر في لغة فيجول بيسك. ويتولى ما تبقى من الفصل شرح كيفية كتابة برامج تستخدم هذه الكتل.

الإشارة إلى عناصر التحكم في نص البرنامج

كما لاحظنا حتى الآن، يمكننا الإشارة إلى خاصية عنصر تحكم ما، بكتابة اسم عنصر التحكم البرمجي، ثم نقطة (.) ثم اسم الخاصية. فمثلاً، توضح العبارة التالية كيف تشير إلى الخاصية Text لعنصر التحكم txtResult:

```
txtResult.Text
```

كما يمكنك الإشارة أيضاً إلى الخاصية، بإضافة اسم النموذج الذي يتوضع عنصر التحكم فيه. فمثلاً، تشير العبارة التالية إلى الخاصية Text لمربع النص txtResult ضمن النموذج frmMultiply:

```
frmMultiply.txtResult.Text
```

تستطيع في معظم الأحوال تجاهل اسم النموذج الحالي، (لتوفير جهد عملية الإدخال)، وعموماً إذا احتوى البرنامج على أكثر من نموذج، فمن الممكن تحديد اسم النموذج، وسنشاهد أمثلة عبر كتابنا على حالات يتوجب ذكر اسم النموذج فيها.

العبارة If

تعرفنا على كتلة عبارات If.End If في الفصل الثاني. وفي كتلة If.End If اللاحقة، تنفذ العبارات الواقعة بين السطر If والسطر If.End فقط إذا كانت قيمة المتحول A تساوي ١:

```
If A = 1 Then
  'ستنفذ العبارات الموجودة هنا
  'A = 1 إذا كانت قيمة المتحول
```

```
End If
```

تُوضح العبارات التالية شكل عبارات If.Else.End If:

```
If A = 1 Then
  'ستنفذ العبارات الموجودة هنا
  'إذا كانت قيمة المتحول A = 1
Else
  'ستنفذ العبارات الموجودة هنا
  'إذا كانت قيمة المتحول A <> 1
End If
```

العبارة Select Case

يكون استخدام Select Case أحياناً أكثر ملاءمة للاستخدام من If.Else.End If. توضح كتلة العبارات التالية كيف تعمل Select Case:

```
Select Case A
   Case 0
     'ستنفذ العبارات الموجودة هنا
     'إذا كانت قيمة المتحول A = 0
   Case 1
     'ستنفذ العبارات الموجودة هنا
     'إذا كانت قيمة المتحول A = 1
   Case 2
     'ستنفذ العبارات الموجودة هنا
     'إذا كانت قيمة المتحول A = 2
End Select
```

كما ترى، تعمل Select.Case بطريقة مشابهة لعمل عبارة If.

طريقة Do While.Loop

تُستخدم العبارة Do.Loop لتنفيذ العبارات ضمنها، مادام الشرط المحدد محققاً. مثال، تعد حلقة Do.Loop التالية من ١ وحتى ١٠٠٠:

```
Dim Counter
```

```
Counter=1
Do While Counter < 1001

   Counter = Counter + 1
Loop
```

يتم تجهيز المتحول Counter بإسناد القيمة الابتدائية ١إليه، ثم تبدأ بعدها حلقة Do While.Loop.

يتحقق السطر الأول من أن قيمة Counter أصغر من ١٠٠١. فإذا كان الأمر كذلك، تنفذ العبارات الواقعة بين السطر Do While والسطر Loop. وفي مثالنا هذا، توجد عبارة واحدة بين هذين السطرين:

```
Counter = Counter +1
```

والتي تزيد قيمة العداد Counter بمقدار ١.

يعود البرنامج بعد ذلك إلى السطر Do While ويتحقق من قيمة العداد Counter، والتي تساوي الآن إلى ٢.

ولهذا فسوف يعاد تنفيذ العبارة الواقعة بين السطر Do While والسطر Loop. تستمر المعالجة حتى تصبح قيمة Counter مساوية إلى ١٠٠١، وعندها يختل الشرط، ويستأنف تنفيذ البرنامج بدأً من العبارة التي تلي السطر Loop.

طريقة Do.Loop While

العبارات ضمن الحلقة Do While.Loop الموصوفة في الفقرة السابقة قد تنفذ أو لا تنفذ، تبعاً لتحقيق الشرط. فمثلاً العبارات في حلقة Do While.Loop التالية لن تنفذ أبداً:

```
Dim Counter
Counter = 2000
Do While Counter < 1001
   Counter = Counter + 1
Loop
```

فعندما يتحقق البرنامج من السطر Do While فإنه سيكتشف أن Counter تساوي ٢٠٠٠، ولهذا فلن تنفذ العبارة الواقعة بين السطر Do While والسطر Loop.

يتطلب البرنامج أحياناً الدخول ضمن الحلقة لمرة واحدة على الأقل دون تحقيق أي شرط.

تُستخدم الحلقة Do.Loop While في مثل هذه الحالة:

```
Dim Counter
Counter = 2000
Do
   txtUserArea.Text = Str(Counter)
   Counter = Counter + 1
Loop While Counter < 1001
```

ينفذ البرنامج العبارات الواقعة بين السطر Do والسطر Loop While مهما تكن الأحوال. وبعد ذلك يحدد البرنامج إذا كان الشرط محققاً (أي Counter أصغر من ١٠٠١)، يعيد البرنامج تكرار تنفيذ الحلقة إذا تحقق الشرط، وبالتالي يعيد تنفيذ العبارات الواقعة بين السطر Do والسطر Loop While.

أما إذا اختل الشرط (أي Counter ليس أصغر من ١٠٠١) فعندها يستأنف التنفيذ من العبارة التي تأتي مباشرة بعد السطر Loop While. تعد الطريقة Do.Loop While التالية من ٥٠ إلى ٣٠٠:

```
Dim Counter
Counter = 50
Do
   Counter = Counter + 1
Loop While Counter < 301
```

الطريقة For.Next

تُعَدُّ الطريقة For.Next وسيلة أخرى لصنع الحلقات في لغة فيجول بيسك. مثلاً، تَعُدُّ الحلقة التالية من ١ و حتى ١٠٠:

```
Dim7 I
For I = 1 to 100 Step 1
   txtMyTextArea.Text = Str(I)
Next
```

أما للتعداد من ١وحتى ١٠٠ وبخطوة (زيادة في كل مرة) قدرها ٢، فتستطيع استخدام الحلقة For.Next التالية:

```
Dim I
For I = 1 to 100 Step 2
   txtMyTextArea.Text = Str(I)
Next
```

تعد هذه الحلقة بالشكل التالي: ١،٢،٣،٥، ... ٩٩.

تساوي قيمة Step في الحالة الافتراضية (أي عند تجاهل كتابتها) إلى ١، وبالتالي فالحلقتين التاليتين متماثلتين:

```
Dim I
For I = 1 to 100 Step 1
   txtMyTextArea.Text = Str(I)
Next
```

و الحلقة:

```
Dim I
For I = 1 to 100
   txtMyTextArea.Text = Str(I)
Next
```

العبارة Exit For

تستطيع الخروج من الحلقة For.Next باستخدام عبارة Exit For كما يلي:

```
Dim I
For I = 1 to 1000
   txtResult.Text = Str(I)
   If I = 500 Then
     Exit For
   End If
Next
```

يعد جزء البرنامج هذا بدءاً من الواحد وبزيادة قدرها ١ للمتحول Z مع كل تكرار للحلقة.

يتحقق شرط عبارة If الداخلية عندما تصبح قيمة I مساوية إلى ٥٠٠ (If I = 500) ونتيجة ذلك تنفذ العبارة Exit For التي تنهي بدورها تنفيذ الحلقة For.Next، قبل انتهاء الحلقة.

أي بكلمة أخرى، تتسبب عبارة If بإنهاء الحلقة عند I = 500 (بينما يفترض أن تنتهي الحلقة بدون عبارة If الداخلية عندما تصبح قيمة I مساوية إلى ١٠٠٠).

العبارة Exit DO

تنتهي الحلقة Do While.Loop باستخدام Exit DO:

```
Dim I
I = 1
Do While I < 10001
   txtResult.Text = Str(I)
   I = I + 2
   If I > 500 Then
     Exit Do
   End If
Loop
```

تعد الحلقة السابقة بدءاً من الواحد وبزيادة قدرها ٢. وينهى تنفيذ الحلقة عندما تصبح قيمة I أكبر من ٥٠٠.

الحلقة اللامنتهية

قد تقع أحياناً في خطأ يشبه ذلك المبين في الحلقة التالية:

```
Dim I
I = 1
Do While I < 10001
   txtResult.Text = Str(I)
   If I > 500 Then
     Exit Do
```

```
   End If
Loop
```

كما تلاحظ، نسينا كتابة العبارة التالية:

```
I = I + 2
```

تبقى قيمة Counter ثابتة في الحلقة Do While.Loop السابقة (I=1)، وهذا بسبب نسيان زيادة قيمته. يبقى البرنامج في هذه الحالة ضمن الحلقة إلى اللانهاية، لأن قيمة I دوماً أصغر من ١٠٠١ ولكن تكون أبداً أكبر من ٥٠٠. بل في الواقع تساوي ١ على الدوام.

العبارة With

تمكّنك العبارة With من إسناد القيم إلى عدة خصائص لكائن ما، دون كتابة اسم الكائن لكل خاصية (أي دفعة واحدة).

خذ مثلاً العبارة With التالية التي تسند مجموعة من خصائص عنصر التحكم cmdMyButton:

```
With cmdMyButton
   .Height = 300   .ClientWidth = 4725
   .Caption = "&My Button"
End With
```

تلعب عبارة With نفس دور العبارات التالية:

```
cmdMyButton.Height = 300cmdMyButton.Width = 4725
cmdMyButton.Caption = "&My Button"
```

توفر عبارة With كما نلاحظ وقت الإدخال (زمن كتابة نص البرنامج). فبدلاً من كتابة اسم عنصر التحكم مع كل خاصية. يكفي كتابة اسم عنصر التحكم لمرة واحدة فقط على سطر العبارة With.

برنامج جمع الأعداد

سنصمم الآن برنامجاً يدعى برنامج جمع الأعداد.

يسمح برنامج جمع الأعداد، للمستخدم باختيار رقم ثم يجمع كل الأرقام الصحيحة من ١ وحتى ذلك الرقم. فمثلاً، يُجري برنامج جمع الأعداد، عملية الجمع التالية: عند اختيار الرقم ٥:

```
1 + 2 + 3 + 4 + 5 = 15
```

ويظهر الناتج.

التمثيل المرئي لبرنامج جمع الأعداد

سننجز الآن التصميم المرئي لبرنامج جمع الأعداد:

- ❑ افتح مشروعاً جديداً من نوع Standard EXE، واحفظ النموذج باسم Sum.frm في الدليل C:\VB5Prg\Ch03 واحفظ المشروع بالاسم Sum.Vbp في ذات الدليل.
- ❑ أنشئ النموذج frmSum طبقاً للجدول ٣-٢.

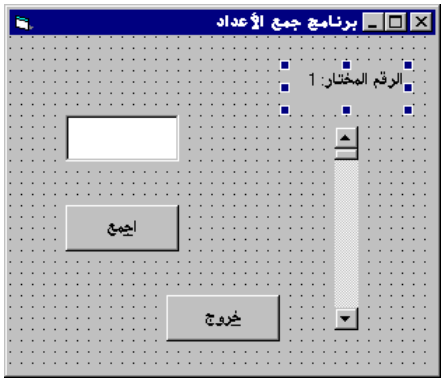
يفترض أن يظهر النموذج المكتمل، كما في الشكل ٣-٤.

الجدول ٣-٢. جدول خصائص النموذج frmSum.

الكائن	الخاصية	القيمة
Form	**Name**	**frmSum**
	Caption	برنامج جمع الأعداد
	RightToLeft	True
CommandButton	**Name**	**cmdExit**
	Caption	&خروج

الكائن	الخاصية	القيمة
	RightToLeft	True
CommandButton	**Name**	**cmdSumIt**
	Caption	ا&جمع
	RightToLeft	True
Text Box	**Name**	**txtResult**

	Text	(اجعله فارغاً)
	Alignment	2-Center
	MultiLine	True
	Enabled	False
	RightToLeft	True
Label	**Name**	**lblResult**
	Caption	الرقم المختار: ١
	RightToLeft	True
Vertical Scroll Bar	**Name**	**vsbNum**
	Min	1
	Max	500
	Value	1



الشكل ٣-٤

النموذج frmSum.

أسندت القيمة False إلى الخاصية Enabled لمربع النص txtResult لمنع المستخدم من الكتابة داخل مربع النص أو تعديله.

إدخال نص برنامج جمع الأعداد

سنكتب الآن نص برنامج جمع الأعداد:

- تحقق بأن العبارة Option Explicit موجودة ضمن قسم التصاريح العامة General Declaration:

```
' يجب التصريح عن كل المتحولات
Option Explicit
```

❑ أدخل النص التالي ضمن الإجراء cmdExit_Click() للنموذج frmSum:

```
Private Sub cmdExit_Click()
    End
End Sub
```

❑ أدخل النص التالي ضمن الإجراء cmdSumIt_Click() للنموذج frmSum:

```
Private Sub cmdSumIt_Click()
    Dim I
    Dim R
    For I = 1 To vsbNum.Value Step 1
      R = R + I
    Next
    txtResult.Text = Str(R)
End Sub
```

❑ أدخل النص التالي ضمن الإجراء vsbNum_Change():

```
Private Sub vsbNum_Change()
    lblNum = " الرقم المختار: " + Str(vsbNum.Value)
End Sub
```

❑ أدخل النص التالي ضمن الإجراء vsbNum_Scroll():

```
Private Sub vsbNum_Scroll()
    vsbNum_Change
End Sub
```

❑ احفظ المشروع باختيار **Save Project** من القائمة **File**.

تنفيذ برنامج جمع الأعداد

❑ نفّذ برنامج جمع الأعداد.

❑ اختر رقماً بالنقر على رمزي الأسهم لشريط التمرير العمودي أو بسحب مؤشر شريط التمرير.

يستجيب البرنامج بإظهار الرقم الذي اخترته.

❑ انقر الزر **اجمع**.

يستجيب البرنامج بإظهار ناتج الجمع في مربع النص (انظر الشكل ٣-٥).

الشكل ٣-٥

برنامج جمع الأعداد.

فمثلاً، يظهر البرنامج الرقم ١٥ عند اختيار الرقم ٥ بواسطة شريط التمرير العمودي ثم النقر على الزر **اجمع** (أي ١ + ٢ + ٣ + ٤ +٥ = ١٥).

❑ أنهِ البرنامج بنقر الزر **خروج**.

كيف يعمل برنامج جمع الأعداد

يستخدم البرنامج حلقة For.Next لإنجاز عملية الجمع.

نص الإجراء cmdSumIt_Click()

ينفذ الإجراء cmdSumIt_Click() عند نقر الزر **اجمع**:

```
Private Sub cmdSumIt_Click()
    Dim I
    Dim R
    For I = 1 To vsbNum.Value Step 1
      R = R + I
    Next
    txtResult.Text = Str(R)
End Sub
```

يُصرح الإجراء عن متحولين، وهما I و R، وتحسب الحلقة For.Next المجموع:

```
1 + 2+ 3+ . . . + vsbNum.Value
```

تساوي القيمة الابتدائية للمتحول R إلى صفر، لأن لغة فيجول بيسك، تُسند القيمة الابتدائية صفر إلى المتحولات آلياً، عند التصريح عنها لأول مرة. لكن يفضل بعض المبرمجين استخدام عبارة زائدة، مثل:

```
R = 0
```

قبل عبارة For وذلك زيادة في الإيضاح.

تُحدِّث آخر عبارة في الإجراء، الخاصية Text لمربع النص بمحتويات المتحول R.

نص الإجراء vsbNum_Change()

يُنفذ هذا الإجراء عند تغيير موضع مؤشر شريط التمرير:

```
Private Sub vsbNum_Change()
    lblNum = " الرقم المختار: " + Str(vsbNum.Value)
End Sub
```

يُحدِّث الإجراء vsbNum_Change() بإسناد الخاصية Value لشريط التمرير، إلى الخاصية Caption للافتة lblNum، وذلك لتمكين المستخدم من أخذ قراءة الوضعية الحالية لشريط التمرير.

لاحظ أن العبارة:

```
lblNum = " الرقم المختار: " + Str(vsbNum.Value)
```

هي ذات العبارة:

```
lblNum.Caption = " الرقم المختار: " + Str(vsbNum.Value)
```

فعند إسناد قيمة إلى عنصر تحكم من نوع لافتة Label بدون ذكر اسم الخاصية، يَفترض فيجول بيسك أنك ترغب بإسناد القيمة إلى الخاصية Caption للافتة، لأن الخاصية Caption هي الخاصية الافتراضية لهذا العنصر (اللافتة).

نمتلك عناصر التحكم الأخرى كحال عنصر اللافتة خصائص افتراضية، فمثلاً الخاصية الافتراضية لعنصر التحكم Text Box (مربع نص) هي الخاصية Text، وهكذا فالعبارة:

```
txtMyTextBox.Text = " Hello "
```

هي نفس العبارة

```
txtMyTextBox = " Hello "
```

ملاحظة

تعوّد دائماً على عدم استخدام الخاصية الافتراضية (إلا بشرط ستعرفه في آخر هذه الملاحظة)، لأنك أحياناً قد تخطئ في كتابة اسم الكائن، الذي لم تسند له الخاصية الافتراضية، ليعتبره فيجول بيسك حينها متحولاً عادياً. مثلاً، افترض وجود مربع نص لديك باسم txtMyText، وأنك تريد إظهار نص معين فيه، (باستخدام ميزة الخاصية الافتراضية)، عندها ستكتب العبارة التالية:

```
txtMyText = "مـرحـبـاً "
```

ولكنك بدلاً من كتابة العبارة السابقة، كتبت ما يلي خطأً:

```
txtMText = "مـرحـبـاً "
```

عندها سيعتبر فيجول بيسك أن txtMText عبارة عن متحول عادي، وسيسند له القيمة "مرحباً"، التي لن تظهر في مربع النص txtMyText.

أما إذا كتبت ما يلي:

```
txtMText.Text = "مـرحـبـاً "
```

عندها سينبهك فيجول بيسك أن txtMText ليس عنصر تحكم، ولا يملك الخاصية Text.

تستطيع حل المشكلة السابقة، بكتابة العبارة Option Explicit في قسم التصاريح العامة، التي تنبهك أن txtMText غير مصرح عنه.

لاحظ مدى أهمية العبارة Option Explicit في برنامجك، وكيف تحل لك الكثير من أخطاء الإدخال، لذلك تعود دائماً على استخدامها.

نص الإجراء vsbNum_Scroll()

ينفذ هذا الإجراء عندما يسحب المستخدم مؤشر شريط التمرير:

```
Private Sub vsbNum_Scroll()
    vsbNum_Change
End Sub
```

ينفذ هذا الإجراء vsbNum_Change()، وهكذا يؤدي تحريك مؤشر شريط التمرير، إلى تنفيذ الإجراء vsbNum_Change() والذي يحدِّث بدوره اللافتة بقيمة الخاصية Value لشريط التمرير.

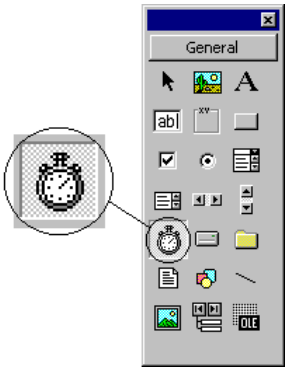
برنامج المؤقِّت

يوضح برنامج المؤقِّت، كيفية استخدام عنصر التحكم Timer.

التمثيل المرئي لبرنامج المؤقِّت

يوضح الشكل ٣-٦ كيف يبدو عنصر التحكم Timer. ضع مؤشر الفأرة فوق هذا العنصر ليظهر مستطيل أصفر مع الرسالة النصية Timer داخله، وبهذه الطريقة تتأكد من سلامة اختيار العنصر.

لاحظ أن مواقع الرموز ضمن مربع الأدوات قد تختلف عن الموجودة لديك بعض الشيء.



الشكل ٣-٦

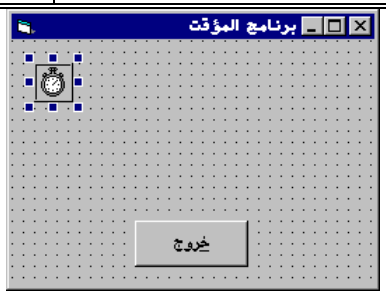
رمز عنصر تحكم الميقاتية Timer داخل إطار مربع الأدوات.

- ❑ أنشئ مشروعاً جديداً من نوع Standard EXE، واحفظ نموذج المشروع بالاسم Timer.frm في الدليل C:\VB5Prg\Ch03 واحفظ ملف المشروع بالاسم Timer.Vbp في ذات الدليل.
- ❑ أنشئ النموذج frmTimer طبقاً للجدول ٣-٣.
- ❑ يفترض أن يبدو النموذج عند الكتابة، كما في الشكل ٣-٧.

الجدول ٣-٣. جدول خصائص النموذج frmTimer.

الكائن	الخاصية	القيمة
Form	**Name**	**frmTimer**
	Caption	برنامج المؤقت
	RightToLeft	True

cmdExit	Name	CommandButton
&خروج	Caption	
True	RightToLeft	
tmrTimer	**Name**	**Timer**
True	Enabled	
2000	Interval	



الشكل ٣-٧

النموذج frmTimer.

إدخال نص برنامج المؤقِّت

سنكتب الآن نص برنامج المؤقِّت:

- تحقق بأن العبارة Option Explicit موجودة ضمن قسم التصاريح العامة frmTimer:

```
' يجب التصريح عن كل المتحولات
Option Explicit
```

- أدخل النص التالي ضمن الإجراء cmdExit_Click() للنموذج frmTimer:

```
Private Sub cmdExit_Click()
    End
End Sub
```

- أدخل النص التالي ضمن الإجراء trmTimer_Timer() للنموذج frmTimer:

```
Private Sub trmTimer_Timer()
    Beep
End Sub
```

- احفظ المشروع باختيار **Save Project** من القائمة **File**.

تنفيذ برنامج المؤقِّت

❑ نفّذ برنامج المؤقِّت.

يطلق برنامج المؤقِّت، صوتاً كل ثانيتين (٢٠٠٠ ملي ثانية)، ولا يظهر عنصر التحكم Timer أثناء تنفيذ البرنامج، لكن يظهر فقط أثناء مرحلة التصميم. ستتعرف عبر الكتاب على بعض عناصر التحكم الأخرى التي لا تشاهد أثناء زمن التنفيذ.

❑ انقر الزر **خروج** لإنهاء البرنامج.

كيف يعمل برنامج المؤقِّت

يستخدم البرنامج عنصر التحكم Timer. والبرنامج ينفذ الإجراء tmrTimer_Timer() آلياً كل ٢٠٠٠ ميلي ثانية (كل ثانيتين).

نص الإجراء tmrTimer_Timer()

تم إسناد القيمة ٢٠٠٠إلى الخاصية Interval للميقاتية tmrTimer أثناء مرحلة التصميم، مما يتسبب بتنفيذ الإجراء tmrTimer_Timer() آلياً كل ٢٠٠٠ ميلي ثانية (ثانيتين).

```
Private Sub trmTimer_Timer()
    Beep
End Sub
```

وهكذا، يعيد الحاسب صوت الرنين كل ثانيتين.

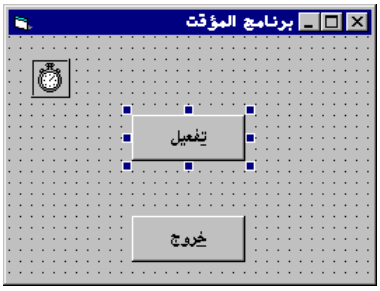
وباعتبار أن رمز الميقاتية Timer لا يظهر عند التنفيذ، وبالتالي فإن موقعه ضمن النموذج ليس هاماً، أي ضعه في أي مكان شئت، بشرط مشاهدته.

تحسين برنامج المؤقِّت

سنحسن الآن برنامج المؤقِّت:

❑ أضف زر آخر Command Button، إلى النموذج frmTimer، حسب الشكل ٣-٨. يفترض أن يحمل زر الأمر هذا الخصائص التالية:

```
Name:        cmdEnableDisable
Caption:     &تفعيل
```



الشكل ٣-٨

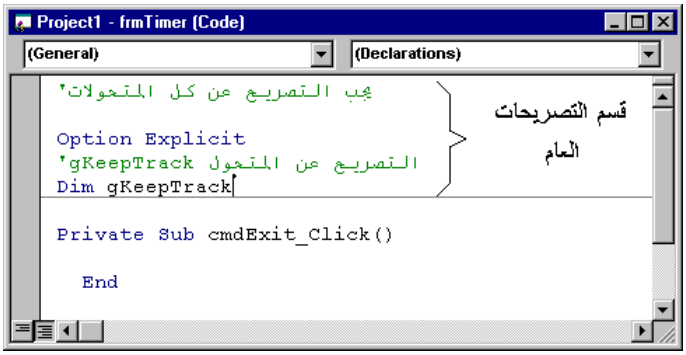
إضافة الزر **تفعيل** إلى النموذج frmTimer.

❑ عدِّل قسم التصاريح العامة General Declarations في النموذج frmTimer بحيث يغدو كالتالي:

```
'يجب التصريح عن كل المتحولات
Option Explicit
'gKeepTrack التصريح عن المتحول
Dim gKeepTrack
```

يبين الشكل ٣-٩ قسم التصاريح العامة General Declarations للنموذج frmTimer.

تستطيع زيادة أو تكبير قسم التصاريح العامة بنقر بداية السطر Option Explicit ثم ضغط المفتاح Enter على لوحة المفاتيح عدة مرات. تتمكن بهذه الطريقة من إضافة عدة أسطر خالية إلى هذا القسم، تستطيع بعد إضافة الأسطر كتابة نص البرنامج المناسب فيها.



الشكل ٣-٩

قسم التصاريح العامة للنموذج frmTimer.

❑ عدِّل الإجراء tmrTimer_Timer().

```
Private Sub Timer_Timer()
    'gKeepTrack أصدر رنينا إذا كانت قيمة المتحول
    'تساوي الواحد
    If gKeepTrack = 1 Then
      Beep
```

```
End If
```

❑ أضف النص التالي إلى الإجراء cmdEnableDisable:

```
Private Sub cmdEnableDisable_Click()
    If gKeepTrack = 1 Then
      gKeepTrack = 0
      cmdEnableDisable.Caption = "&تفعيل"
    Else
      gKeepTrack = 1
      cmdEnableDisable.Caption = "&تعطيل"
    End If
End Sub
```

❑ احفظ المشروع بالطريقة المعتادة.

تنفيذ برنامج المؤقِّت

❑ نفّذ برنامج المؤقِّت. (اضغط المفتاح F5)

لا يصدر البرنامج صوتاً كل ثانيتين.

❑ انقر الزر **تفعيل**، لتمكين عملية إصدار الصوت.

*يتبدل عنوان الزر **تفعيل** إلى **تعطيل**، ويطلق البرنامج صوت رنين Beep كل ثانيتين (انظر الشكل ٣-١٠).*

الشكل ٣-١٠

الزر cmdEnableDisable

وهو يحمل الآن العنوان **تعطيل**.

❑ انقر الزر **تعطيل**.

*يعود عنوان الزر إلى **تفعيل**، ويتوقف البرنامج عن إصدار الصوت Beep.*

❑ كرر عملية النقر على الزر عدة مرات، وبعد انتهائك غادر البرنامج بنقر زر **خروج**.

نص قسم التصاريح العامة للنموذج frmTimer

يتضمن هذا القسم التصريح التالي عن المتحول gKeepTrack:

```
'gKeepTrack التصريح عن المتحول
Dim gKeepTrack
```

السؤال المطروح الآن. ما هو السبب وراء التصريح عن المتحول gKeepTrack هنا ؟!.
يفيد التصريح عن المتحولات ضمن هذا القسم إلى جعله مرئياً لكل إجراءات النموذج frmTimer. فمثلاً، لو صرحنا عن المتحول gKeepTrack ضمن الإجراء tmrTimer_Timer() بدلاً من التصريح عنه ضمن قسم التصاريح العامة، فمن الممكن استخدام هذا المتحول ضمن الإجراء tmrTimer_Timer() فقط، ولا يمكن للإجراءات الأخرى الوصول إليه ومعرفة قيمته. ولكن التصريح عنه ضمن قسم التصاريح العامة General Declarations يجعله بمتناول كل الإجراءات والتوابع الوظيفية الموجودة في النموذج frmTimer.

ملاحظة

يدل الحرف الأول من المتحول gKeepTrack هو g، على أن هذا المتحول مصرح عنه في قسم التصاريح العامة General Declarations.
ولا يعتبر استخدام هذا الحرف إلزاميا أو من متطلبات لغة فيجول بيسك، إلا أنه أسلوب جيد برمجياً. لهذا سنستخدم الحرف g عبر هذا الكتاب عند التصريح عن المتحولات ضمن قسم التصاريح العامة.

نص الإجراء tmrTimer_Timer()

ينفذ هذا الإجراء كل ثانيتين وذلك بسبب إسناد القيمة ٢٠٠٠ إلى الخاصية Interval لعنصر التحكم trmTimer:

```
Private Sub tmeTimer_Timer()
    'gKeepTrack إذا كانت قيمة المتحول
    'تساوي الواحد
    If gKeepTrack = 1 Then
```

```
    Beep
    End If
End Sub
```

تنفذ العبارة Beep إذا تحقق الشرط If. أي إذا كانت قيمة gKeepTrack تساوي الواحد. لاحظ أننا لم نصرح عن gKeepTrack داخل هذا الإجراء، لكن الإجراء يتمكن من التعرف على المتحول بسبب التصريح عنه ضمن قسم التصاريح العامة كما اتفقنا.

كما ذكرنا أيضاً أن فيجول بيسك يعطي المتحولات القيمة الابتدائية "صفر"، لحظة التصريح عنها، (وبالتالي لا يكون الشرط If هذا محققاً منذ بدء تشغيل البرنامج ولهذا لا يصدر صوتاً كل ثانيتين).

نص الإجراء cmdEnableDisable_Click()

ينفذ هذا الإجراء عند نقر الزر cmdEnableDisable:

```
Private Sub cmdEnableDisable_Click()
    If gKeepTrack = 1 Then
      gKeepTrack = 0
      cmdEnableDisable.Caption = "&تفعيل"
    Else
      gKeepTrack = 1
      cmdEnableDisable.Caption = "&تعطيل"
    End If
End Sub
```

تنفذ العبارات الواردة ضمن السطرين If و Else إذا كانت قيمة gKeepTrack تساوي "الواحد"، فتصبح قيمة gKeepTrack مساوية الصفر وتبدل الخاصية Caption إلى &تفعيل.

أما إذا كانت قيمة gKeepTrack مساوية للصفر، فستنفذ العبارات الواقعة بين السطر Else و End If، وتصبح قيمة gKeepTrack مساوية إلى "الواحد" وتبدل الخاصية Caption إلى &تعطيل.

أيضاً، يتعرف الإجراء cmdEnableDisable_Click() على المتحول gKeepTrack وهذا بسبب التصريح عنه ضمن قسم التصاريح العامة.

تعديل برنامج المؤقِّت ثانية

كان الغرض من استخدام المتحول gKeepTrack إيضاح أن المتحول الذي يصرح عنه ضمن القسم General Declarations يصبح مرئياً لكل إجراءات النموذج. تستطيع عموماً كتابة برنامج المؤقِّت بدون استخدام المتحول gKeepTrack. استخدم الخطوات التالية لرؤية كيفية إنجاز ذلك:

- احذف التصريح عن gKeepTrack من قسم التصاريح العامة للنموذج frmTimer، وبالتالي يجب أن يظهر هذا القسم كما يلي:

```
' يجب التصريح عن كل المتحولات
Option Explicit
```

كنا قد طلبنا منك إسناد القيمة True إلى الخاصية Enabled لعنصر التحكم tmrTimer.

- غيّر وضعية الخاصية Enabled إلى False. يؤدي هذا العمل إلى إيقاف تنفيذ الإجراء tmrTimer_Timer().
- غيّر الإجراء cmdEnableDisable بحيث يغدو كالتالي:

```
Private Sub cmdEnableDisable_Click()
   If tmrTimer.Enabled = True Then
     tmrTimer.Enabled = False
     cmdEnableDisable.Caption = "&تفعيل"
   Else
     tmrTimer.Enabled = True
     cmdEnableDisable.Caption = "&تعطيل"
   End If
End Sub
```

- غيّر الإجراء tmrTimer_Timer() بحيث يصبح كالتالي:

```
Private Sub trmTimer_Timer()
   Beep
End Sub
```

- احفظ المشروع.

تنفيذ برنامج المؤقِّت

- نفّذ برنامج المؤقِّت وتحقق بأنه يعمل بنفس طريقة استخدام المتحول gKeepTrack.
- أَنهِ البرنامج بنقر الزر **خروج**.

نص الإجراء tmrTimer_Timer()

ينفذ هذا الإجراء كل ٢٠٠٠ ميلي ثانية (أي كل ٢ ثانية) إذا كانت قيمة الخاصية Enabled تساوي True. (طبعاً الفاصل الزمني يساوي ٢٠٠٠ ميلي ثانية، لأننا أسندنا القيمة ٢٠٠٠ إلى الخاصية Interval أثناء طور التصميم).

لن ينفذ الإجراء tmrTimer_Timer() عند بدء تشغيل البرنامج، وهذا لأننا أسندنا القيمة False إلى الخاصية Enabled، بينما إذا كانت قيمة Enabled تساوي القيمة True فسوف ينفذ الإجراء، ويصدر الحاسب صوتاً Beep كل ثانيتين.

```
Private Sub trmTimer_Timer()
     Beep
End Sub
```

نص الإجراء cmdEnableDisable_Click()

ينفذ هذا الإجراء عندما ننقر الزر cmdEnableDisable:

```
Private Sub cmdEnableDisable_Click()
   If tmrTimer.Enabled = True Then
     tmrTimer.Enabled = False
     cmdEnableDisable.Caption = "&تفعيل"
   Else
     tmrTimer.Enabled = True
     cmdEnableDisable.Caption = "&تعطيل"
   End If
End Sub
```

تتحقق العبارة If من قيمة الخاصية Enabled للعنصر Timer ؛ فإذا كانت القيمة تساوي True، فسوف تُنفذ العبارات الواقعة بين السطرين If و Else، فتصبح قيمة الخاصية Enabled مساوية إلى False وتتغير الخاصية Caption للزر cmdEnableDisable إلى **تفعيل**.

وعلى العكس، تُنفذ العبارات الواقعة بين السطرين Else و End If إذا كانت قيمة الخاصية Enabled تساوي False، فتصبح قيمتها True وتغير الخاصية Caption للزر cmdEnable Disable إلى **تعطيل**.

تستخدم الكثير من البرامج، تقنية تغيير عنوان الزر (الخاصية Caption)، تبعاً للحالة الراهنة للبرنامج.

ملاحظة

يستخدم برنامج المؤقِّت زراً واحداً (الزر cmdEnableDisable)، لإنجاز مهمتي تفعيل وتعطيل (Enable و Disable) عنصر تحكم الميقاتية.

تصبح الخاصية Caption للزر مساوية إلى **تعطيل**، عند تفعيل الميقاتية Timer (وبهذا، يستطيع المستخدم نقر الزر، لتعطيل الميقاتية).

تصبح الخاصية Caption للزر مساوية إلى **تفعيل**، عند تعطيل الميقاتية (ويستطيع المستخدم نقر الزر، لتفعيل الميقاتية).

يُفضل تقليل عدد عناصر التحكم في نموذج البرنامج، قدر الإمكان، واستخدام خاصيتي التفعيل والتعطيل Enable و Disable، وذلك حتى لا نشوش المستخدم. ففي برنامجنا هذا، تمكنا بواسطة زر واحد من إنجاز مهمتين (**تفعيل/تعطيل**).

طبعاً، لا يعني هذا توحيد الأزرار ذات الأدوار المختلفة كلياً. فمثلاً، يجب أن لا نوحّد عمل الزر **خروج** والزر **تفعيل/تعطيل**.

الخلاصة

ناقش هذا الفصل، عبارات اتخاذ القرار في لغة فيجول بيسك وكذلك عبارات الحلقات. تعتبر هذه العبارات أحجار البناء الأساسية للغة البرمجة فيجول بيسك.

كما ركز هذا الفصل على عنصر التحكم Timer (الميقاتيه) ومفهوم مرئية المتحولات وكذلك على التوابع الوظيفية والإجراءات والطرائق.