الفصل الثانى
التعامل مع الإجراءات والدوال

يتناول هذا الفصل الإجراءات **Procedures** والـــدوال **Functions** كوسائل أساسية لاستخدام نفس الكود في أكثر من موقع داخل نفس البرنامج . ويشرح كيف نضيف هذه الإجراءات والدوال إلى البرنامج.

بانتهاء هذا الفصل ستتعرف على :

- استخدام الإجراءات والدوال
- تحديد مجال الإجراءات والدوال
- استخدام نفس الكود في أكثر من موقع في نفس البرنامج
- التعامل مع أكثر من نموذج بالبرنامج.

إن فهمك للدوال والإجراءات يعتبر العمود الفقرى لفهم البرمجة بلغة **Visual Basic** ، وذلك لأن التعليمات يجب أن تكتب داخل إجراءات أو دوال، ولذلك يمكنك اعتبار هذا الفصل مدخلا إلى البرمجة. حيث من الضرورى أن تفهم كيف تستخدم الإجراء أو الدالة وما هو الفرق بينهما وكيف تتعامل مع الوحدة النمطية (**Module**) التى تكتب فيها الإجراءات والدوال.

استخدام الإجراءات والدوال

عند العمل في مشروعات البرمجة وخاصةً الكبيرة منها تجد نفسك تكرر جزء ما من الكود كثيرا وفي أماكن متفرقة من البرنامج، وإن كان المشروع من عدة برامج فإن التكرار يمتد إلى البرامج المختلفة أيضا.

يمكننا تجنب هذا التكرار باستخدام الإجراءات، وهي مجموعة من التعليمات يتم تنفيذها عند نداء الإجراء ثم يعود البرنامج إلى تنفيذه العادي. لقد تعاملت من قبل مع الإجراءات، وذلك من خلال الإجراءات الحدثية، وهي التي يناديها البرنامج عند وقوع حدث معين لأحد الكائنات. بالطبع يمكننا إنشاء إجراءاتنا الخاصة، وتمييزاً لها عن الإجراءات الحدثية فهي تسمى الإجراءات الفرعية **Sub procedures** أو الروتينات الفرعية **sub routines**.

استخدام الإجراءات

الفكرة الرئيسية وراء استخدام الإجراءات بكفاءة تكمن في تقسيم مهمة البرنامج إلى مهام صغيرة يمكن احتوائها على مجموعة إجراءات أو دوال أو كائنات عند الحاجة (يتم تناول الكائنات في فصول تالية)، ويحقق هذا الأسلوب العديد من المزايا منها :

- سهولة اختبار كل إجراء على حده للتأكد من أنه يعمل بصورة صحيحة.
- تجنب تكرار الكود بلا داع باستدعاء الإجراء عندما نحتاج إليه بدلا من إعادة كتابة كل محتوياته.

- يمكننا توسيع هذا الأسلوب بإنشاء مكتبة من الإجراءات الخاصة والتي يمكن استخدامها عبر البرامج المختلفة.
- صيانة البرامج تصبح أيسر من خلال شيئين : التعديل يتم في مكان واحد (الإجراء) بدلا من الأماكن التى تنادي عليه والتي قد تكون كثيرة داخل البرنامج الواحد، أيضا الفصل بين العناصر الرئيسية في البرنامج تتيح التعديل المنفصل لإحداها بعيدا عن الآخر.

الملاحظات في متن البرنامج تساعد كثيرا في فهمك وصيانتك للكود بعد كتابته.

إنشاء الإجراءات الفرعية

يتم إنشاء الإجراءات الفرعية بالكتابة المباشرة في الوحدة النمطية أو نافذة الكود. لأداء ذلك، ضع مؤشر الإدخال في نافذة الكود في مكان خارج الإجراءات (ليس داخل أي من الإجراءات أي قبل كلمة **Sub** أو بعد كلمة **End Sub** كما يوضح شكل ٢-١) ثم قم بهذه الخطوات:

١. اكتب **Sub** واتبعها بمسافة.

٢. اكتب اسم الإجراء وليكن **testProc**.

٣. اضغط **Enter** لإنشاء الإجراء.

شكل ٢-١ الأماكن المختلفة الصالحة لبدء إنشاء إجراء بالكتابة المباشرة في نافذة الكود.

بمجرد الضغط على **Enter** يقوم **Visual Basic** بالآتي (كما يظهر في شكل ٢-٢):

- وضع أقواس المعاملات (arguments) بعد اسم الإجراء مباشرة.
- إضافة عبارة End Sub في السطر التالي.
- كتابة اسم الإجراء الجديد في مربع الأحداث.

شكل ٢-٢ نافذة الكود عند إضافة إجراء جديد

الصيغة العامة لتعريف الإجراء الفرعي كالتالي:

```
[ Public | Private ] [ Static ] Sub procname ( [arguments] )

        statements

End sub
```

كلمتي Public و Private تحددان مجال استخدام الإجراء تماماً كمجال المتغير، أيضا كلمة Static تحدد خصائص المتغيرات المحلية في هذا الإجراء.

Arguments هي المعاملات أو الوسيطات المطلوبة للإجراء، مثلا إجراء يقوم بالبحث عن اسم في ملف لابد أن نمرر له هذا الاسم.

نداء الإجراء

بعد إنشاء الإجراء وكتابة الكود المطلوب بداخله يبقى أن تستطيع تنفيذ هذا الكود من أي جزء في البرنامج. يتم ذلك عن طريق نداء الإجراء ويمكننا نداء الإجراء بطريقتين:

الأولى : باستخدام الأمر Call

```
Call procname ([arguments])
```

الثانية : بذكر اسم الإجراء مباشرة

```
procname ([arguments])
```

في كلتا الطريقتين **procname** هو اسم الإجراء المراد استدعاؤه و**arguments** هي المعاملات أو الوسيطات المراد تمريرها إلى الإجراء ويمكن بوجه عام أن تكون قيم مباشرة أو متغيرات أو تعبيرات ولكن لابد أن يكون عددها وأنماطها متطابقة مع المعاملات المعلن عنها في تعريف الإجراء (أي في عبارة **Sub**) وإلا سيتم إظهار رسالة خطأ أثناء التشغيل.

المثال التالي يبين كيفية تعريف إجراء وظيفته الكتابة على عنوان النافذة:

```
Sub Banner(F As Form, stLabel As String)
   F.Text = stLabel
End Sub
```

الإجراء يستخدم الخاصية **Text** الخاصة بالنموذج، ونلاحظ أن هناك معاملين يمرران إليه هما النموذج المراد الكتابة عليه، والعنوان المراد كتابته.

يمكنك استدعاء هذا الإجراء بالأمر التالي:

```
Banner (Form1, "Hello")
```

تمرير البيانات من وإلى الإجراء

هناك طريقتان لتبادل البيانات مع الإجراء:

الطريقة الأولى: استخدام المتغيرات العامة **Public** التي تظهر في أي مكان من الكود، ومن ثم يمكن قراءتها وتغييرها من خلال الإجراء. ولكن هذه طريقة غير محبذة لأنها تتضمن خطورة أن يقوم إجراء آخر بالخطأ بتعديل هذا المتغير مسببا أخطاء غير متوقعة، ويجب أن يتم تقليل استخدامها إلا في حالات قليلة كحفظ اسم قاعدة البيانات التي سيتعامل معها كافة الإجراءات في البرنامج، حيث يُسمح بتمرير الاسم لكل إجراء.

الطريقة الثانية: هي استخدام المعاملات **parameters** والتي يمكن تمريرها من وإلى الإجراء دون الحاجة إلى المتغيرات العامة. وهذه الطريقة ذات فائدة أخرى وهي إمكانية استخدامها في برامج متعددة دون الحاجة إلى تغييرها مما يوفر كثيرا من وقت البرمجة كما رأينا في المثال السابق.

تبادل البيانات بالمعاملات كما قلنا ثنائي الاتجاه حيث يمكن نقل بيانات من الإجراء إلى البرنامج الرئيسي بسهولة كما في المثال الآتي:

```
Sub Geometry(ByVal Radius As Single, ByRef Perimeter As
Single, ByRef Area As Single)
   Const PI = 3.14
   Perimeter = 2 * PI * Radius
   Area = PI * Radius ^ 2
End Sub
```

المثال السابق يحسب محيط ومساحة دائرة بمعلومية نصف قطرها ويستخدم المعاملان الثاني والثالث لتمرير النتائج.

يمكننا نداء هذا الإجراء كالتالي:

```
Dim P, A, R As Single
R = 5
Geometry (R , P, A )
```

إذا ما اختبرنا **P** و**A** سنجد قيمتي المحيط ونصف القطر.

في المثال السابق تمكننا من تمرير البيانات إلى البرنامج لأن المعاملات تم تمريرهـــا بـــالعنوان (**Passing by reference**) وذلك يعني أن البرنامج يمرر عنوان المتغير في الذاكرة إلى الإجراء وعليه يكون المتغير في الإجراء هو نفسه المتغير خارجه وأي تغيير فيـــه في مكـــان ينعكس عليه في الآخر. يمكننا أيضا منع الإجراء من تغيير القيمة الأصلية للمعامل وذلـــك بالتمرير بالقيمة (**Passing by Value**)، في هذه الحالة يتم عمل نسخة من المـــتغير في مكان آخر في الذاكرة ليستخدمها الإجراء، حتى لو تم تغييرها فلن يتأثر المتغير الأصلي.

يتم تعريف طريقة تمرير المعامل في تعريف الإجراء وذلك من خـــلال الكلمـــة **ByVal** أو **ByRef**. كما في السطر التالي:

```
Sub Geometry(ByVal Radius As Single, ByRef Perimeter As
Single, ByRef Area As _ Single)
```

في السطر السابق قمنا بتعريف نصف القطر على أنه يمرر بالقيمة، وهذا منطقي لأن الإجراء سيقرأ هذه القيمة فقط وليس من وظيفته أن يغيرها.

عند استخدام التمرير بالعنوان لابد من تمرير متغير، ولا يمكن استخدام تعبير أو ثوابت أو قيم مباشرة.

الخروج من الإجراء

لسبب أو لآخر قد تحتاج إلى مغادرة الإجراء دون إكمال تنفيذ بقية أوامره ، يتم ذلك من خلال العبارة **Exit Sub**.

مثال لهذا الاستخدام، إجراء يقوم بالتحقق من قيم بعض المدخلات، لو أن أحد هذه القيم غير صالح فلا داعي لإكمال التحقق من الباقي. سيتم الخروج وإظهار رسالة خطأ على سبيل المثال كما يوضح الإجراء التالي:

```
Sub Geometry(ByVal Radius As Single, ByRef Perimeter As
Single, ByRef Area As_ Single)
   If Radius<0 Then Exit Sub
   Const PI = 3.14
   Perimeter = 2 * PI * Radius
   Area = PI * Radius ^ 2
End Sub
```

في الإجراء السابق يتم اختبار قيمة نصف القطر أولاً فإن كانت سالبة، يتم الخروج من الإجراء من خلال عبارة **Exit Sub** وذلك لتلافي النتائج الخاطئة التي تنتج من مدخلات غير صالحة.

استخدام الدوال

تشبه الدوال الإجراءات تماما، الفرق الوحيد هو أن الدالة ترجع قيمة عند استدعائها، حيث يمكن تخزين هذه القيمة داخل متغير أو استخدامها في تعبير مباشرة.

يحتوي **Visual Basic** على العديد من الدوال المبنية داخل اللغة . منها الدوال الحسابية أو الحرفية مثل **Sin()** ، **Len()** ، **Instr()** أو خلاف ذلك. وقد تعرضنا لبعض هذه الدوال فى الفصول السابقة وكل هذه الدوال يمكن استخدامها مباشرةً وبدون تعريف، كما يمكنك أيضاً تعريف دوالك الخاصة.

إنشاء الدوال يتبع نفس قواعد إنشاء الإجراءات، حيث يمكنك إنشاء الدالة بكتابتها مباشرة في نافذة الكود باستخدام الكلمة **Function** بدلاً من كلمة **Sub** وفى هذه الحالة يتم إنشاء الهيكل الآتي للدالة (المسماة **Sum** على سبيل المثال):

```
Public Function Sum()

End Function
```

هذه الصيغة رغم أنها مقبولة إلا أنها تفتقر إلى تحديد النمط الذي سترجعه الدالة (الذى يكون افتراضياً من النوع Object)، والذي يتم تعريفه كما في حالة المتغيرات باستخدام كلمة As بعد أقواس المعاملات، كما يوضح التعريف التالي:

```
Public Function Sum ( A As single, B As Single) As Single
End Function
```

المثال السابق يوضح تعريف المعاملات ونمط القيمة المرجعة.

و لكن كيف سنرجع القيمة للبرنامج بانتهاء الدالة ؟ يتم ذلك من خلال عبارة تخصيص داخل الإجراء، تضع القيمة المطلوبة في متغير باسم الدالة كالتالي:

```
Public Function Sum ( A As single, B As Single) As Single
        Sum = A + B
End Function
```

لاحظ في المثال السابق أن Sum هو اسم الدالة وليس متغير، وأن هذه العبارة تجعل قيمة مجموع المعاملين ترجع للبرنامج عند استخدام الدالة، فمثلا عند كتابة الكود التالي في البرنامج:

```
Dim N As Single, M As Single, x As Single
N = 3
M = 4
x = Sum ( N , M )
```

تصبح قيمة x بعد كتابة الكود هي 7.

و كذلك لو تم استخدام الدالة في تعبير كالتالي:

```
If Sum ( N , M ) < 10 Then MessageBox.Show( "Small Sum")
```

يمكنك استخدام عبارة التخصيص لاسم الدالة عدة مرات في الدالة ولكن آخر هذه العبارات تنفيذا قبل العودة للبرنامج هي التي تكون ذات تأثير. والجدير بالذكر أيضا أننا يمكننا استخدام الدوال كالإجراءات تماما بتجاهل القيمة المرجعة.

تحديد مجال الإجراءات والدوال

الإجراءات والدوال كالمتغيرات تماما لها مجال رؤية أو مدى Scope وهي الأماكن التي يمكن نداء الإجراء أو الدالة منها. وتنقسم إلى نوعين هما الإجراءات العامة Public Procedures والتي تتم على مستوى المشروع بالكامل والإجراءات الخاصة Private Procedures التي تتم على مستوى التصنيف Class.

الإجراءات العامة

إذا أردت أن يرى برنامجك الإجراء أو الدالة من أي مكان في البرنامج استخدم الكلمة Public، ولكن عليك تسمية كافة الإجراءات العامة بأسماء مميزة غير متشابهة. لو تجاهلت كلمة Public فإن الإجراء يعتبر عاما بصورة افتراضية.

الإجراءات الخاصة

تستخدم الكلمة Private لتحصر رؤية الإجراء أو الدالة في التصنيف الذي تم تعريفه فيه داخل النموذج أو الوحدة النمطية. ربما لاحظت أن الإجراءات الحدثية كلها خاصة بطبيعتها، ذلك لأن الكائنات لا يتم ندائها عادة خارج النموذج الذي توجد فيه هذه الكائنات. يرجع ذلك إلى مبدأ "حجب البيانات" Information Hiding والذي يتم تطبيقه أثناء البرمجة بالكائنات Object Oriented Programming.

المتغيرات الساكنة في الذاكرة

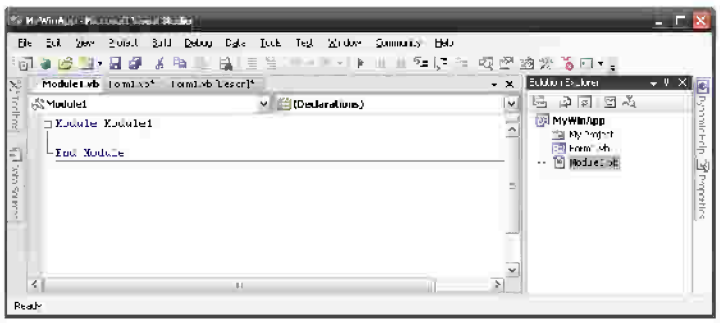
قلنا فيما سبق أن المتغيرات المحلية التي لا تريد محوها عند انتهاء الإجراء يجب الإعلان عنها بالكلمة Static، يمتد هذا ليشمل كافة المتغيرات المحلية في إجراء إذا تم الإعلان عنه بالكلمة Static.

لأغراض تتعلق بكفاءة تنفيذ البرنامج والحفاظ على موارد النظام System resources، حاول دائما أن تتجنب استخدام الإجراءات العامة Public والمتغيرات الساكنة في الذاكرة إلا عند الحاجة إليها.

إعادة استخدام الإجراءات والدوال

عند تعريفك لإجراء أو دالة في ملف نموذج نافذة فإنك تستطيع استخدامها في أنحاء البرنامج إن كانت معرفة على أنها دالة عامة، ولكن إذا قررت أن تستخدم هذه الدالة في مشروع آخر فيما بعد فيستحسن أن تقوم بتعريف هذه الدالة في ملف وحدة نمطية **Module**. يمكنك إن كنت قد كتبت الإجراء بالفعل أن تنقله بالقص ثم اللصق في ملف آخر.

إن لم يكن مشروعك يحتوي على ملف وحدة نمطية يمكنك إضافة واحد إلى مشروعك بأمر **Project ≺ Add Module** أو بنقر الزر الأيمن على نافذة المشروع واختيار القائمة **Add** ثم اختيار **Module** من القائمة الفرعية، وفى هذه الحالة تظهر الوحدة النمطية داخل نافذة مستكشف الحل كما فى شكل ٢-٣.



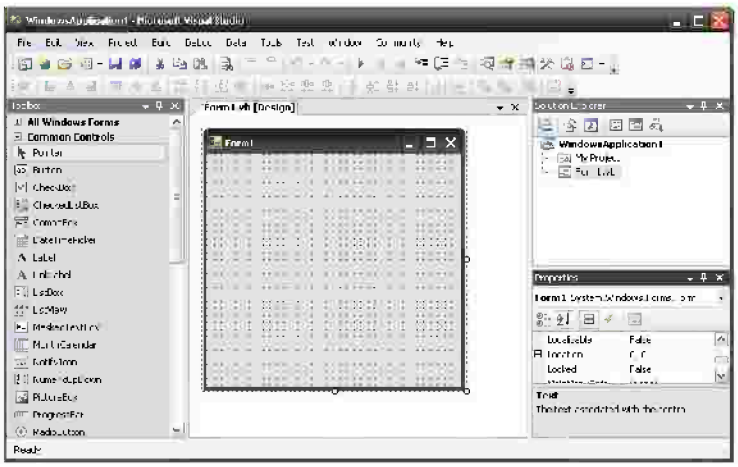
شكل ٢-٣ نافذة المشروع بعد إضافة ملف وحدة نمطية **module**.

التعامل مع أكثر من نموذج في البرنامج

إن كان برنامجك بسيط قليل الوظائف فإنه غالبا يتكون من نموذج واحد فقط، أما إذا كان البرنامج كبيراً نوعاً ما فإنه غالبا ما يحتوي على عدة نماذج إما لأن مساحة نافذة واحدة لا تكفي وإما لأن الوظائف تنقسم لمجموعات فيكون من المنطقي جعل كل مجموعة في نموذج ، وأخيرا للمحافظة علي موارد النظام، حيث أن البرنامج الذي يقوم بتحميل النماذج عند الحاجة أكثر توفيرا للذاكرة من البرنامج الذي به نافذة واحدة ممتلئة بالعناصر.

إضافة نموذج جديد للبرنامج

يبدأ **Visual Basic** دائما بنموذج فارغ كما ترى في شكل ٢-٤، حيث تستطيع إضافة العناصر وكتابة الإجراءات ومعالجة الأحداث. إذا أردت أن تضيف نموذجا آخر للبرنامج، فيمكنك ببساطة اختيار أمر **Project ≺Add Windows Form** لإضافة نموذج جديد أو **Project ≺Add Existing Item** لإضافة نموذج موجود مسبقاً داخل مشروع آخر. عندئذ يظهر نموذج جديد في المشروع يشبه الأول تماما، إلا أن اسمه يكون (إن لم تغير اسم الأول) **Form2** وهكذا سيستمر في تسمية النماذج **Form3, Form4**، أما إذا كنت قد غيرت اسم النموذج الأول فإن النموذج الثاني يأتي باسم **Form1** وهكذا.



شكل ٢-٤ يبدأ **Visual Basic** المشروع الجديد دائما بنموذج فارغ

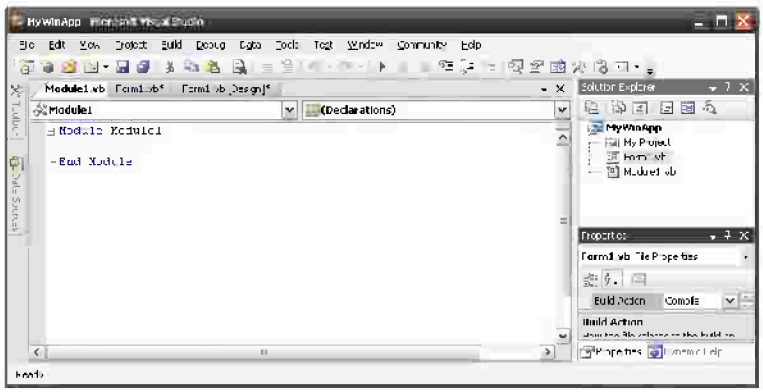
تذكر أنك تستطيع إضافة الملفات إلى مشروعك بالنقر الأيمن في نافذة المستكشف.

النموذج الثاني كالأول تماماً، إلا أنه لن يظهر من نفسه عند تشغيل البرنامج، وإنما يمكنك إظهاره من خلال النموذج الأول. أو بدلاً من ذلك يمكنك اختياره ليكون نموذج بدء المشروع كما عرفت من قبل.

إضافة وحدات نمطية إلى المشروع

إذا كان هناك إجراء يستعمل من قِبَل عدة نماذج، فمن الأفضل أن يتم وضع الإجـــراء في وحدة نمطية **Module** وليس في نموذج.

ملف الوحدات النمطية يحتوي على كود **Visual Basic** فقط، ولا يحتوي على أدوات تحكم أو صور أو أي مكونات رسومية أخرى، يمكنك إضافة وحدة نمطية إلى مشـــروعك كما سبق وأوضحنا بأمر **Project ≺ Add Module** أو بالنقر الأيمـــن علـــى نافـــذة المشروع ليظهر ملف الوحدة كما بالشكل ٢-٥.



شكل ٢-٥ ملف الوحدة النمطية فارغا كما يظهر عند إنشاؤه

للوحدة النمطية أو **Module** خاصية تسمى **Name** تعمل كما يعمل اســـم النمـــوذج، ويستحسن تحديدها أثناء إنشاء الوحدة، حيث يخصـــص لهـــا **Visual Basic** الاســـم الافتراضي **Module1**.

إظهار النماذج والوحدات النمطية في المشروع

كلما أضفت نموذجاً أو وحدة نمطية يظهر اسمها في نافذة المستكشف، والتي تمكنـــك مـــن الوصول بسهولة إلى الملف الذي تريد رؤيته أو تعديله. عند اختيارك لنمـــوذج في نافـــذة المستكشف يمكنك رؤيته أو رؤية الكود الخاص به وذلك من خلال الزرين الموجودين أعلى نافذة المستكشف كما في شكل ٢-٦.

شكل ٢-٦ نافذة المستكشف وبها أزرار الرؤية

أما إذا كان الملف المختار وحدة نمطية فإن زر رؤية الكود فقط هو الذي يكـــون متاحـــا. لاحظ أن النقر المزدوج على الوحدات النمطية يفتح نافذة الكود الخاصة بها، بينما النقـــر المزدوج على اسم النموذج يظهر شكل النافذة في طور التصميم.