الجامعة الافتراضية السورية
SYRIAN VIRTUAL UNIVERSITY

البرمجة المقادة بالأحداث

الدكتور باسل الخطيب

ISSN: 2617-989X

Books

البرمجة المقادة بالأحداث

الدكتور باسل الخطيب

من منشورات الجامعة الافتراضية السورية

الجمهورية العربية السورية 2018

Events driven programming

Bassel Al Khatib

Publications of the Syrian Virtual University (SVU)

Syrian Arab Republic, 2018

ISSN: 2617-989X

الفهرس

- أساسيات بناء واجهات ويندوز باستخدامVisual Studio 2013.....1
  - مقدمة.....3
  - البدء بإستخدام Visual studio 2013.....5
  - نماذج النوافذ Windows form.....7
  - معالجة الأحداث.....10
    - واجهة بسيطة مقادة بالأحداث.....10
    - الكود المولد من قبل Visual studio.....12
    - المفوضات(Delegates) وألية معالجة الأحداث.....14
    - ايجاد معلومات عن الأحداث.....17
  - خصائص وتوضع عنصر التحكم.....19
  - استخدام Visual studio لتحرير توضع عناصر الواجهة.....23
- عناصر التحكم (1).....24
  - اللصاقات, صناديق النص , الأزرار 26
  - صناديق المجموعات , اللوحات.....30
  - صناديق الاختيار , أزرار الخيار.....34
- عناصر التحكم (2).....43
  - صناديق الصور.....45
  - التلميحات.....49
  - القرائم الرقمية.....52
  - معالجة أحداث الفأرة.....55
  - معالجة أحداث لوحة المفاتيح.....60
- عناصر التحكم(3).....64
  - القوائم.....66
  - عناصر تحكم التاريخ والوقت.....79
    - عنصر التحكم Month Calendar.....79
    - عنصر التحكم Date Time Picker 80
  - عنصر التحكمLink Label.....84
- عناصر التحكم(4).....89
  - عنصر التحكم List Box.....91
  - عنصر التحكم checked List Box.....98
  - عنصر التحكم Combo Box.....102
- عناصر التحكم(5).....108
  - عنصر التحكم Tree View.....110
  - عنصر التحكم List View.....118
  - عنصر التحكم Tab Control.....124
- مواضيع متقدمة.....131
  - نوافذ الواجهة المتعددة المستندات MDI.....133
  - الورائة المرئية Visual inheritance.....144
  - العناصر المخصصة User-defined controls.....148
- الرسوميات.....153
  - صفوف الرسم ونظام الاحدائيات.....155
    - سياقات وأغراض الرسم.....157
    - التحكم باللون.....159
    - معالجة الألوان.....161
    - استعمال مربع حوار الألوان لاختيار لون.....165
  - الخطوط.....169
    - التحكم بخط النص.....169
    - رسم سلاسل محرفية باستعمال أكثر من خط نص.....170

ISSN: 2617-989X

الفهرس

- معايير خط النص.....172
  - رسم الأشكال.....176
    - رسم الخطوط والمستطيلات و الأشكال البيضوية.....176
    - رسم الأقواس.....179
    - رسم المضلعات ومتعددات الخطوط.....183
- أساسيات LINQ.....189
  - الاستعلام في مصفوفة من الأعداد 191
  - الكلمة المفتاحية var.....193
  - الاستعلام في مصفوفة من الأغراض باستخدام LINQ.....194
  - انشاء نمط جديد في العبارة select.....197
- التعامل مع قواعد البيانات(1).....198
  - مكتبة صف نموذج كائن البيانات.....200
    - اضافة نموذج كائن البيانات.....203
    - اختيار محتويات النموذج.....204
    - اختيار الاتصال مع البيانات.....205
    - اختيار كائنات قاعدة البيانات المطلوب تضمينها في النموذج.....206
    - معاينة مخطط نموذج كائن البيانات.....207
    - بناء صف المكتبة.....208
  - انشاء تطبيق مرتبط مع نموذج كائن البيانات.....209
    - بناء تطبيق ويندوز وربطه مع نموذج كائن البيانات.....209
    - الربط بين عناصر التحكم ونموذج كائن البيانات.....211
      - ✓ انشاء الكائن DBContext.....215
      - ✓ كتابة حدث التحميل.....215
      - ✓ اجرائية الحفظ.....217
- التعامل مع قواعد البيانات (2).....218
  - الربط الديناميكي مع نتائج الاستعلام.....220
  - استرجاع البيانات من عدة جداول باستخدام LINQ.....225
- التعامل مع قواعد البيانات(3).....230
  - انشاء نموذج رئيسي/فرعي.....232
  - مثال تطبيقي دفتر عناوين.....235

ISSN: 2617-989X

الفصل الأول: أساسيات بناء واجهات ويندوز باستخدام Visual Studio 2013

ISSN: 2617-989X

عنوان الموضوع:

أساسيات بناء واجهات ويندوز باستخدام Visual Studio 2013

الكلمات المفتاحية:

GUI ، عناصر التحكم، النماذج، الخصائص، الأحداث، الطرائق، التطبيقات المقادة بالأحداث.

ملخص:

نُبين في هذا الفصل أساسيات بناء تطبيقات ويندوز باستخدام Visual Studio 2013.

أهداف تعليمية:

يتعرف الطالب في هذا الفصل على:

- النماذج.
- عناصر التحكم.
- الخصائص.
- الطرق.
- الأحداث.

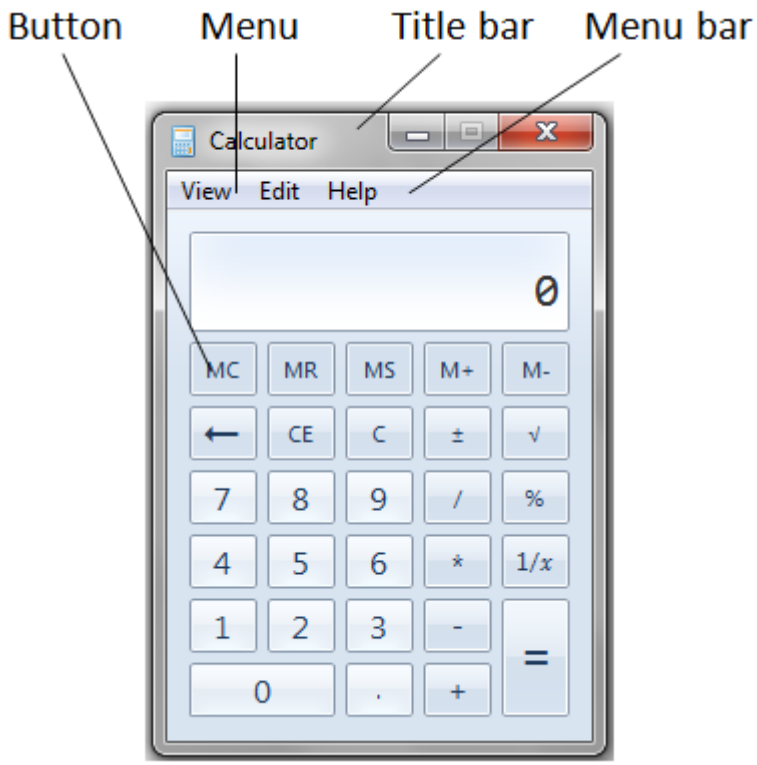
المخطط:

أساسيات واجهات ويندوز باستخدام Visual Studio 2013

4 وحدات (Learning Objects)

مقدمة

- تكوّن واجهة المستخدم البيانية (GUI = Graphical User Interface) الجزء من البرنامج الذي يشاهده المستخدم ويتعامل معه بشكل تفاعلي سهل. كما أنها تُعطي البرنامج مظهره المعين.
- ولذا، فإن توفير مجموعة من عناصر التحكم سهلة الاستخدام في أي تطبيق يُعد عاملاً أساسياً في نجاح التطبيق وانتشاره. كما أن حسن تصميم واجهات التطبيق واختيار عناصر التحكم المناسبة يُمكّن المستخدمين من زيادة إنتاجيتهم وتفاعلهم مع التطبيق بشكل بسيط و سريع.
- يُبين الشكل التالي مثالاً عن واجهة مستخدم بيانية لتطبيق شهير تستخدم مجموعة من عناصر التحكم.



- يُظهر الشكل السابق واجهة برنامج الآلة الحاسبة (Calculator) والتي تحوي العديد من عناصر التحكم:
- ففي أعلى النافذة، نجد شريط القوائم (Menu Bar) والذي يحوي القوائم (Menus): عرض (View)، تحرير (Edit)، ومساعدة (Help).
- تحوي واجهة هذا البرنامج مجموعة من الأزرار (Buttons) لكل منها مهمة يُحدّدها البرنامج. مثل كتابة الأرقام والعمليات الحسابية.
- تُشكّل عناصر التحكم هذه مكونات الواجهة البيانية للتطبيق، والتي يتفاعل المستخدم من خلالها مع مختلف وظائف البرنامج.

- تُبنى الواجهات البيانية باستخدام مجموعة من عناصر التحكم، والتي تدعى أحياناً (كائنات) (Components). يكون دور هذه العناصر عرض معلومات على الشاشة، أو تمكين المستخدم من التفاعل مع البرنامج عن طريق الفأرة ولوحة المفاتيح أو بعض أساليب الإدخال الأخرى (كالأوامر الصوتية).

- يُظهر الجدول التالي مجموعة من أكثر عناصر التحكم استخداماً:

عنصر التحكم		شرح
لصاقة	Label	يعرض صور ونصوص غير قابلة للتحرير من قبل المستخدم.
صندوق نص	TextBox	يُمكّن المستخدم من إدخال البيانات باستخدام لوحة المفاتيح، كما يُستخدم لعرض نصوص قابلة أو غير قابلة للتحرير.
زر	Button	يرفع حدثاً عند النقر عليه بالفأرة.
مربع اختيار	CheckBox	يُمكن المستخدم من اختيار أو عدم اختيار بند معين.
زر الخيار	RadioButton	يكون عادةً ضمن مجموعة من أزرار الخيار حيث يُمكن اختيار خيار واحد منها فقط.
قائمة منسدلة	ComboBox	يوفر هذا العنصر قائمة منسدلة من البنود التي يستطيع المستخدم اختيار أحدها.
صندوق قائمة	ListBox	يوفر هذا العنصر قائمة من البنود التي يستطيع المستخدم اختيار واحد منها أو أكثر.
قائمة رقمية	NumericUpDown	تُمكّن المستخدم من اختيار قيمة رقمية ضمن مجال محدد.
صندوق صورة	PictureBox	صندوق صورة.
صندوق مجموعة	GroupBox	صندوق يضم مجموعة من العناصر.
لوحة	Panel	حاوية لمجموعة من العناصر.
تلميح	ToolTip	يُستخدم لإظهار تلميحات مساعدة.

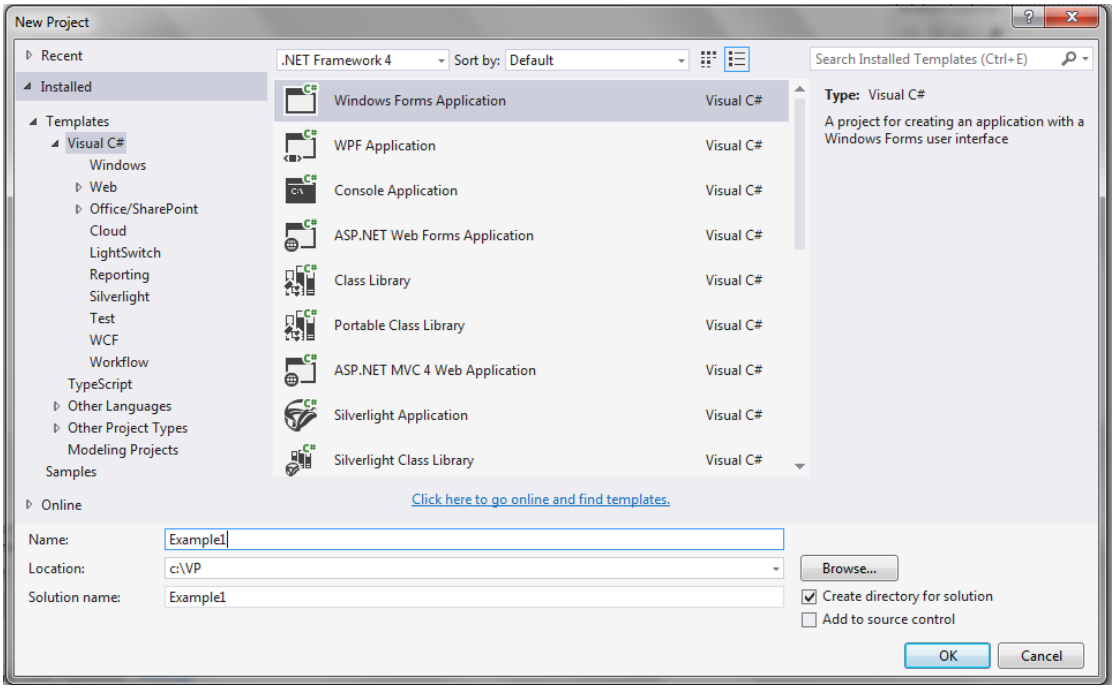
البدء باستخدام Visual Studio 2013

يسمح محيط العمل Visual Studio 2013 بإنشاء تطبيقات ويندوز Windows Applications بشكل بسيط وسريع:

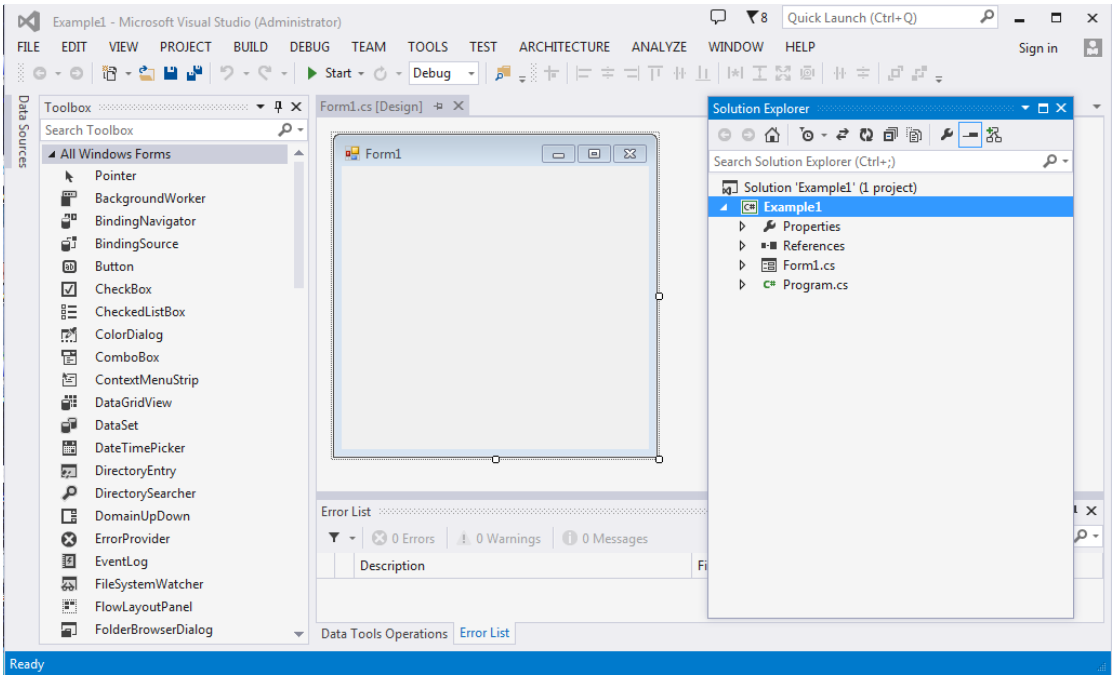
1. افتح محيط العمل Visual Studio 2013.
2. أنشئ مشروع جديد:

File → New → Project

3. اختر Visual C#/Windows Forms Application، وقم بإدخال اسم المشروع ومسار التخزين:



4. يتم فتح مشروع جديد يحوي نموذج واحد Form1:

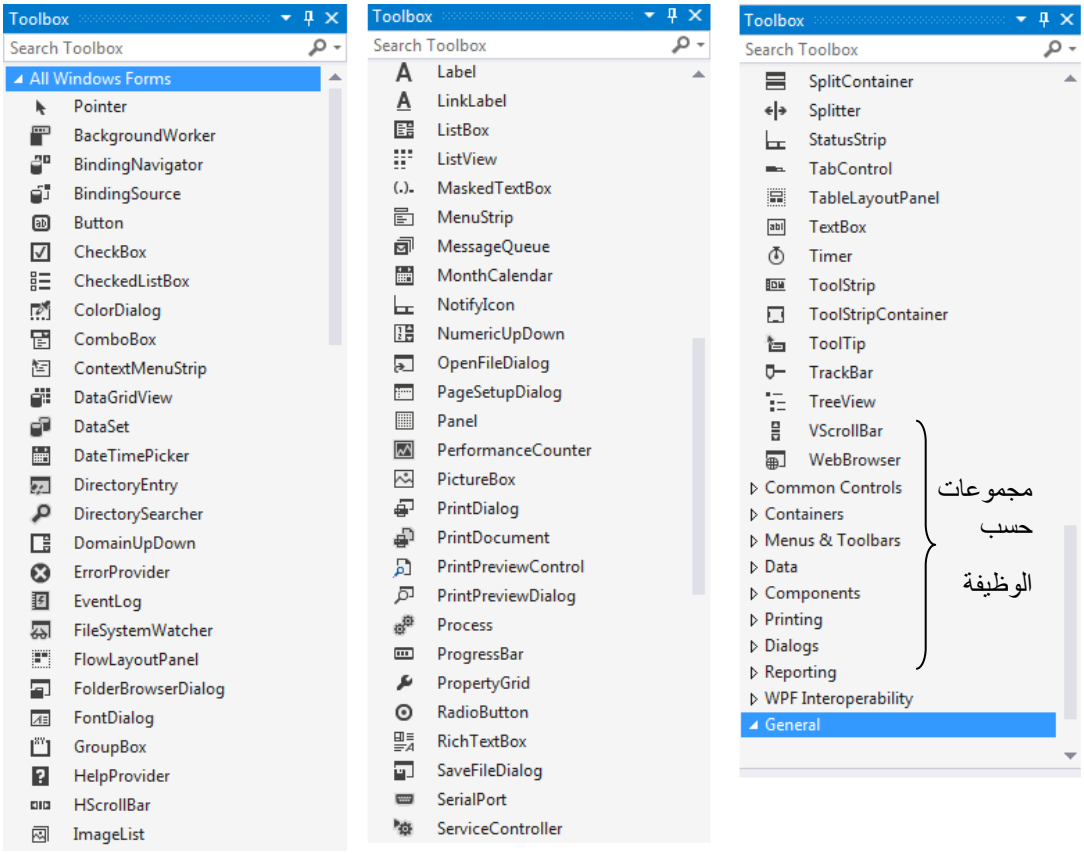


5. يُمكنك الآن إضافة عناصر تحكم من شريط الأدوات Toolbox وكتابة الكود البرمجي اللازم لمعالجة أحداث هذه العناصر.

نماذج النوافذ Windows Forms

- تُستخدم نماذج النوافذ (Windows Forms) لإنشاء واجهات بيانية للبرامج. حيث يكون النموذج (Form) عبارة عن عنصر بياني (نافذة) يظهر على الشاشة. كما يُمكن أن يكون مربع حوار، أو نافذة من النوع MDI (نافذة واجهة متعددة المستندات) (Multiple Document Interface Window) والتي سنقوم بدراستها لاحقاً.
- يكون الكائن (Component) نسخة من صف يحقق الواجهة IComponent والتي تُعرّف التصرفات (Behaviors) التي يجب على الكائن سلوكها، مثل كيفية تحميل (Load) المكون.
- تملك عناصر التحكم مثل الزر أو اللصاقة تمثيلاً بيانياً في وضع التنفيذ (Runtime)، بينما نجد كائنات أخرى ليس لها تمثيل بياني (مثل المؤقت Timer).
- نعرض فيما يلي شكلاً يُظهر عناصر التحكم والكائنات الموجودة في صندوق أدوات C# (ToolBox)، وهي تُصنّف بحسب وظائفها.
- عند اختيار التصنيف (All Windows Forms) في أعلى صندوق الأدوات، يُمكن استعراض جميع عناصر التحكم والكائنات الموجودة ضمن باقي التصنيفات وذلك ضمن قائمة واحدة.
- لإضافة عنصر تحكم أو كائن إلى نموذج (Form)، يجب اختياره وسحبه من صندوق الأدوات إلى النموذج. لإلغاء التحديد، يُمكن اختيار البند مؤشر (Pointer) من صندوق الأدوات، والذي يظهر في أول كل قائمة من الأعلى. (سيجنبك ذلك أن تضيف عنصراً جديداً إلى النموذج عن طريق الخطأ).

- عند وجود أكثر من نافذة على الشاشة، تكون النافذة الفعالة Active Window هي الواقعة في المقدمة والتي لها شريط عنوان Title Bar مميز Highlighted. عادةً، يكون لونه أكثر قتامة من باقي النوافذ.
- تصبح نافذة ما فعالة عندما ينقر المستخدم في مكان ما داخلها، ونقول عن النافذة الفعالة أنها "تملك التركيز" Focus. على سبيل المثال، يكون صندوق الأدوات في Visual Studio هو النافذة الفعالة عندما تقوم باختيار عنصر منه، وتكون نافذة الخصائص هي النافذة الفعالة عندما تقوم بتحرير خصائص عنصر ما.
- يُشكّل النموذج Form حاوية Container لعناصر التحكم والكائنات الموجودة عليه. فعندما تضيف عنصر تحكم أو كائن إلى النموذج من صندوق الأدوات، يقوم Visual Studio بتوليد كود دوره إنشاء نسخة من الغرض وتحديد خصائصه الأساسية.



- عندما تُعدّل خصائص الكائن أو عنصر التحكم في بيئة التطوير، يتم تلقائياً تعديل الكود المتولد من قبل NET.. وعند حذفه، يُحذف الكود المرتبط به. تقوم بيئة التطوير بحفظ الكود المتولد في ملف منفصل باستخدام الصفوف الجزئية Partial Classes.

- ومع أنه بإمكاننا أن نقوم بكتابة هذا الكود بأنفسنا، إلا أنه من الأسهل أن نستعمل صندوق الأدوات ونافذة الخصائص وأن نترك Visual Studio يهتم بالتفاصيل.

- توجد جميع عنصر التحكم والكائنات التي سنقوم بعرضها في هذا الفصل والذي يليه في فضاء الأسماء System.Windows.Forms.

- نقوم عادةً عند بناء واجهة تطبيق بإنشاء نموذج (Windows Form)، و من ثم تعديل خصائصه، ثم نضيف إليه عناصر تحكم ونعدل خصائصهم ونكتب معالجات لأحداثهم (Event Handlers) وهي طرائق تستجيب للأحداث التي يولدها عنصر تحكم.
- نُبين في الجدول التالي الخصائص والأحداث والطرائق الأكثر استخداماً في النموذج Form:

الخاصية أو الحدث أو الطريقة	الوصف
الخصائص	
AcceptButton	الزر الذي يتم نقره تلقائياً عند ضغط محرف الإدخال Enter. وتنفيذ إجرائية النقر على الزر إن وجدت.
AutoScroll	قيمة منطقية تُحدّد فيما إذا أردنا أن تظهر أشرطة التمرير عند لزومها.
CancelButton	الزر الذي يتم نقره عند ضغط محرف الهروب Escape. وتنفيذ إجرائية النقر على الزر إن وجدت.
FormBorderStyle	نمط إطار النموذج (بدون إطار، إطار مفرد، إطار ثلاثي الأبعاد ...).
Font	خط النصوص التي تظهر على النموذج، وهو الخط الافتراضي لعناصر التحكم المضافة عليه.
Text	النص الذي يظهر في شريط العنوان للنموذج.
الطرائق	
Close	تُغلق النموذج وتحرر جميع الموارد المرتبطة به، كالذاكرة المستخدمة لتخزين عناصره ومكوناته.
Hide	تقوم بإخفاء النموذج دون تحرير الموارد.
Show	تقوم بإظهار نموذج مخفي.
الأحداث	
Load	يُرفع هذا الحدث قبل أن يتم إظهار النموذج للمستخدم. يقوم Visual Studio بإظهار معالج هذا الحدث عند النقر المزدوج على النموذج في مصمم Visual Studio.

- عندما نقوم بإنشاء عناصر التحكم، يقوم Visual Studio بتوليد الكثير من الكود المتعلق بالواجهة البيانية. في البرمجة المرئية، يتولى Visual Studio إدارة الكود المرتبط بالواجهه. ويبقى على المبرمج كتابة أجسام معالجات الأحداث ليحدد ما الذي يجب على البرنامج القيام به عندما يحدث حدث ما.

معالجة الأحداث Events Handling

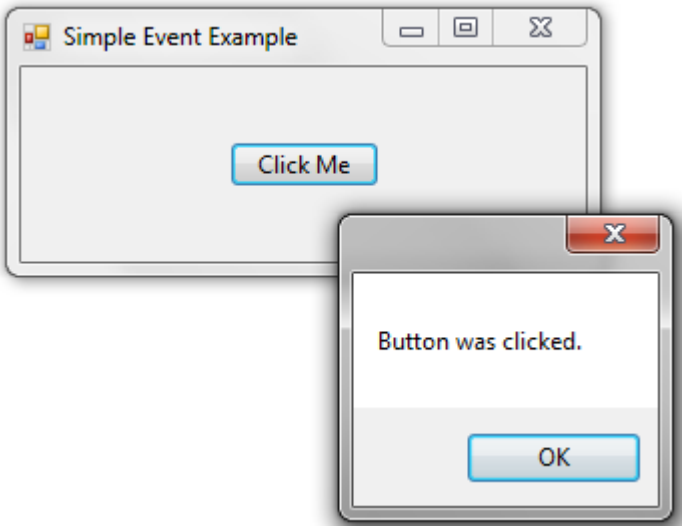
- يتفاعل المستخدم عادةً مع البرنامج ليحدد له ما المهمة التي يجب عليه فعلها. فمثلاً، في برنامج بريد إلكتروني E Mail وبعد أن تكتب رسالة ما، فإن ضغط زر (إرسال) يطلب من البرنامج أن يرسل الرسالة إلى العنوان المحدد.

- توصف التطبيقات ذات الواجهات البيانية بأنها تطبيقات مقادة بالأحداث (Event Driven)، حيث أن تفاعل المستخدم مع مكون ما في الواجهة يدفع البرنامج إلى تنفيذ مهمة محددة. نسمي هذ التفاعل حدث Event. ومن هذه الأحداث: النقر على زر، والكتابة في مربع نص، واختيار بند من قائمة، وإغلاق نموذج، وتحريك الفأرة.
- ندعو الطريقة Method التي تُنفذ مهمة مرتبطة بحدث ما بمعالج الحدث Event Handler، وتدعى عملية كتابة هذه الطرائق بمعالجة الأحداث Event Handling.

واجهة بسيطة مقادة بالأحداث A Simple Event-Driven GUI

نقوم في المثال التالي، بإنشاء نموذج Form يحوي زر Button يُظهر رسالة للمستخدم عند النقر عليه.

```
1  // SimpleEventExampleForm.cs
2  // Simple event handling example.
3  using System;
4  using System.Windows.Forms;
5  namespace SimpleEventExample
6  {
7  // Form that shows a simple event handler
8  public partial class SimpleEventExampleForm : Form
9  {
10 // default constructor
11 public SimpleEventExampleForm()
12 {
13  InitializeComponent();
14 } // end constructor
15 // handles click event of Button clickButton
16 private void clickButton_Click( object sender, EventArgs e )
17 {
18              MessageBox.Show( "Button was clicked." );
19 } // end method clickButton_Click
20 } // end class SimpleEventExampleForm
21 } // end namespace SimpleEventExample
```



- لتنفيذ هذا المثال، نقوم أولاً بإنشاء تطبيق Windows جديد ونضيف زراً إلى النموذج.
- في نافذة الخصائص للزر، نُعدّل اسم الزر إلى clickButton، ونعدّل الخاصية Text له لتصبح Click Me. يصطلح بعض المبرمجون عادةً على إضافة نمط الغرض في نهاية اسمه مثل clickButton الذي له النمط Button، وذلك للدلالة داخل الكود على نمط الغرض مما يُساعد المبرمج على تذكر الغرض.
- وبما أننا نريد من البرنامج أن يستجيب بإظهار صندوق رسالة MessageBox عندما يقوم المستخدم بنقر الزر السابق. فيجب إنشاء معالج حدث Event Handler لمعالجة حدث النقر على الزر. يُمكن إنشاء هذا المعالج بالنقر المزدوج على الزر، سيؤدي ذلك إلى تعريف معالج الحدث التالي (لاحظ أنه يكون بدايةً فارغاً):

```
private void clickButton_Click( object sender, EventArgs e )

{


}
```

- تصطلح NET. على تسمية طريقة معالج الحدث على الشكل التالي:

  controlName_eventName مثل clickButton_Click
- سيتم تنفيذ هذه الطريقة عندما يقوم المستخدم في وضع التنفيذ بالنقر على الزر clickButton.
- يتلقى كل معالج حدث وسيطين عند استدعائه:

الأول – واسمه sender من النمط object – يدل على الغرض الذي أنشأ الحدث. والثاني يدل على وسائط الحدث وهو من النمط EventArgs (أو أحد أبنائه)، ويسمى عادةً e. يحتوي هذا المتحول عادةً على معلومات إضافية عن الحدث.

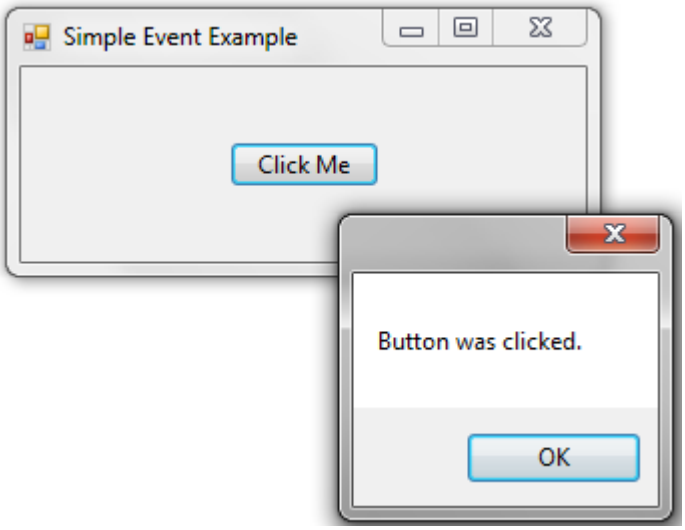
للاستجابة على حدث نقر الزر بإظهار رسالة، نكتب الكود التالي في جسم معالج الحدث:

```
MessageBox.Show( "Button was clicked." );
```

- يكون الشكل النهائي لمعالج الحدث:

```
private void clickButton_Click( object sender, EventArgs e )
{
   MessageBox.Show( "Button was clicked." );
}
```

- عند تشغيل البرنامج والنقر على الزر تظهر الرسالة الموافقة:



الكود المولد من قبل Visual Studio

- عندما نقوم ببناء واجهة بيانية في وضع التصميم، يقوم Visual Studio بتوليد الكود الذي يقوم ببناء هذه الواجهة وتهيئتها برمجياً.
- يُخزّن هذا الكود في ملف خاص لكل نموذج يدعى Designer.cs (وفي مثالنا SimpleEventExampleForm.Designer.cs)
- يمكن فتح هذا الملف عن طريق توسيع عقدة الملف الذي نعمل عليه (SimpleEventExampleForm.cs) ثم النقر المزدوج على الملف Designer.cs المطلوب، يظهر الشكل التالي والذي يليه محتوى الملف في مثالنا، يقوم Visual Studio بطي الأسطر من 23 إلى 58 بشكل افتراضي.

SimpleEventExampleForm.Designer.cs

SimpleEventExample.SimpleEventExampleForm | clickButton

```
1  namespace SimpleEventExample
2  {
       3 references
3      partial class SimpleEventExampleForm
4      {
5          /// <summary>
6          /// Required designer variable.
7          /// </summary>
8          private System.ComponentModel.IContainer components = null;
9
10         /// <summary>
11         /// Clean up any resources being used.
12         /// </summary>
13         /// <param name="disposing">true if managed resources should be disposed; otherwise, false.</param>
               1 reference
14         protected override void Dispose(bool disposing)
15         {
16             if (disposing && (components != null))
17             {
18                 components.Dispose();
19             }
20             base.Dispose(disposing);
21         }
22
23         Windows Form Designer generated code
58
59         private System.Windows.Forms.Button clickButton;
60     }
61 }
```

- يمكنك فهم هذا الكود! إلا أنه، وبما أن Visual Studio يولد ويدير هذا الكود، فلا داعي لمطالعته! في الحقيقة، لست بحاجة لفهم معظم الكود المكتوب لبناء الواجهات البيانية، إلا أننا الآن نحاول شرح كيفية عمل التطبيقات ذات الواجهة البيانية.
- إن الكود المولد آلياً والذي يُعرّف الواجهة البيانية هو في الواقع جزء من الصف الذي يُعرّف النموذج (في حالتنا SimpleEventExampleForm).
- تمّ استعمال الكلمة المفتاحية partial في السطر الأول من الشكل السابق والتي تسمح بتوزيع تعريف الصف على أكثر من ملف.

```
23 #region Windows Form Designer generated code
24 /// <summary>
25 /// Required method for Designer support - do not modify
26 /// the contents of this method with the code editor.
27 /// </summary>
    1 reference
28 private void InitializeComponent()
29 {
30     this.clickButton = new System.Windows.Forms.Button();
31     this.SuspendLayout();
32     //
33     // clickButton
34     //
35     this.clickButton.Location = new System.Drawing.Point(104, 37);
36     this.clickButton.Name = "clickButton";
37     this.clickButton.Size = new System.Drawing.Size(75, 23);
38     this.clickButton.TabIndex = 0;
39     this.clickButton.Text = "Click Me";
40     this.clickButton.UseVisualStyleBackColor = true;
41     this.clickButton.Click += new System.EventHandler(this.clickButton_Click);
42     //
43     // SimpleEventExampleForm
44     //
45     this.AutoScaleDimensions = new System.Drawing.SizeF(7F, 15F);
46     this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font;
47     this.ClientSize = new System.Drawing.Size(282, 97);
48     this.Controls.Add(this.clickButton);
49     this.Font = new System.Drawing.Font("Segoe UI", 9F, System.Drawing.FontStyle.Regular, System.Drawing.GraphicsUnit.Point, ((byte)(0)));
50     this.Name = "SimpleEventExampleForm";
51     this.Text = "Simple Event Example";
52     this.ResumeLayout(false);}
53 #endregion
```

- نلاحظ في السطر 30 تعريف الزر clickButton والذي قمنا بإنشائه في وضع التصميم، والذي هو متحول منتَسخ Instance Variable ضمن الصف SimpleEventExampleForm.
- يقوم C# بتعريف متحول لكل عنصر تحكم نضيفه في وضع التصميم. ويكون هذا المتحول خاص private بشكل افتراضي.
- كما يحوي الكود أيضاً على الطريقة Dispose لتحرير الموارد. والطريقة InitializeComponent (في السطور من 28 إلى 52) والتي تحوي الكود الذي ينشئ الزر ويحدد خصائصه وخصائص النموذج الموافقة لما تم إدخاله في نافذة الخصائص في وضع التصميم.
- لاحظ أن Visual Studio يضيف تعليقات (الأسطر من 32 إلى 34)، كما أنه تمّ توليد السطر رقم 41 عندما أنشأنا حدث النقر على الزر.
- يتمّ استدعاء الطريقة InitializeComponent عند إنشاء النموذج ويتم فيها تحديد الخصائص كعنوان النموذج وحجمه وأحجام عناصر التحكم ونصوصها.
- كما أن Visual Studio يستعمل هذا الكود لإظهار النموذج في وضع التصميم. إن تعديل كود الطريقة InitializeComponent من قبل المبرمج بشكل خاطئ قد يحول دون إمكانية عرض الواجهة بشكل صحيح!

المفوضات (Delegates) وآلية معالجة الأحداث

- يُدعى عنصر التحكم الذي يولد الحدث بمرسل الحدث (Event Sender). مثلاً، يؤدي نقر زر أمر من قبل المستخدم إلى توليد حدث النقر وبصبح الزر مرسلاً لهذا الحدث.
- بينما تُدعى الطريقة المسؤولة عن معالجة الحدث بمستقبل الحدث (Event Receiver). وعندما يحصل الحدث يقوم مرسل الحدث باستدعاء مستقبل الحدث ليعالج الحدث.
- إن آلية معالجة الأحداث في .NET. تسمح باختيار أي اسم للطريقة المسؤولة عن معالجة حدث ما، إلا أنه يجب للطريقة أن تمتلك وسائطاً مناسبة (توقيع Signature معين) لتلقي المعلومات المرتبطة بالحدث. وبما أنه يُمكن للمبرمج اختيار اسم الطريقة بنفسه فلا بد من وجود آلية لتحديد شكل الطريقة المستقبلة للحدث.

المفوضات Delegates

- يتم ربط معالجات الأحداث إلى أحداث عنصر التحكم باستعمال أغراض خاصة تدعى "المفوضات" Delegates. يُحدّد غرض مفوض مرجع لطريقة لها ترويسة (توقيع Signature) يُحدّدها التصريح عن نمط المفوض. تتوفر مجموعة من المفوضات مسبقة التعريف لتغطي جميع أحداث عناصر تحكم الواجهة البيانية.

- فمثلاً، يوجد نمط مفوض مُعرّف لحدث نقر زر واسمه EventHandler في فضاء الأسماء System، وإذا بحثت عن هذا النمط في ملفات المساعدة سترى السطر التالي:

```
public delegate void EventHandler(object sender, EventArgs e);
```

- لاحظ استعمال الكلمة المفتاحية delegate لتعريف نمط مفوض اسمه EventHandler، والذي يُحدد توقيع لطريقة تُرجع void وتتلقى وسيطين، الأول هو مرسل الحدث وهو من النمط object، والثاني من النمط EventArgs، إذا قارنت تعريف المفوض السابق مع السطر 16 من المثال السابق سترى أن معالج الحدث يُطابق تعريف نمط المفوض:

```
private void clickButton_Click( object sender, EventArgs e )
```

تحديد الطريقة التي يجب على المفوض استدعاؤها

- يستدعي مرسل الحدث غرض مفوض وكأنه طريقة. أي أنه وبما أن معالج الحدث يُعرّف كغرض مفوض، يستطيع مرسل الحدث استدعاءه ببساطة عندما يحصل الحدث. فالزر يستدعي غرض المفوض الخاص به –وهو من النمط EventHandler– ليستجيب للنقر.
- ولكي يتمّ تعيين أن clickButton_Click هي الطريقة المطلوب استدعاؤها يقوم Visual Studio بالربط بين الطريقة والزر. لاحظ السطر 41 من الشكل السابق حيث يضيف Visual Studio هذا الكود عند النقر المزدوج على الزر في وضع التصميم.
- تقوم العبارة:

```
this.clickButton_Click.Click+ = new System.EventHandler(this.clickButton_Click);
```

بإنشاء غرض مفوض من النمط EventHandler وتهيئته بالطريقة clickButton_Click.

- يستعمل السطر 41 المعامل =+ لإضافة غرض المفوض إلى حدث نقر الزر. مما يعني أن الطريقة clickButton_Click ستُنفذ عندما يقوم المستخدم بالنقر على الزر.
- في الواقع، يُمكن أن نحدد أكثر من طريقة ليتم استدعاؤهم كرد على الحدث وذلك عن طريق إضافة مفوضات أخرى إلى حدث نقر الزر بكتابة عبارات شبيهة بالمكتوبة في السطر 41.

- توصف مفوضات الأحداث بأنها متعددة Multicast، أي أنها تمثل مجموعة من الطرائق (المفوضات) تحمل جميعها نفس الترويسة (التوقيع)، تُمكننا هذه المفوضات المتعددة من استدعاء أكثر من طريقة للرد على حدث محدد، فعندما يحصل حدث ما يستدعي مرسل الحدث جميع الطرائق المشار إليها من المفوض المتعدد، يدعى هذا بتعدد الأحداث Event Multicasting.

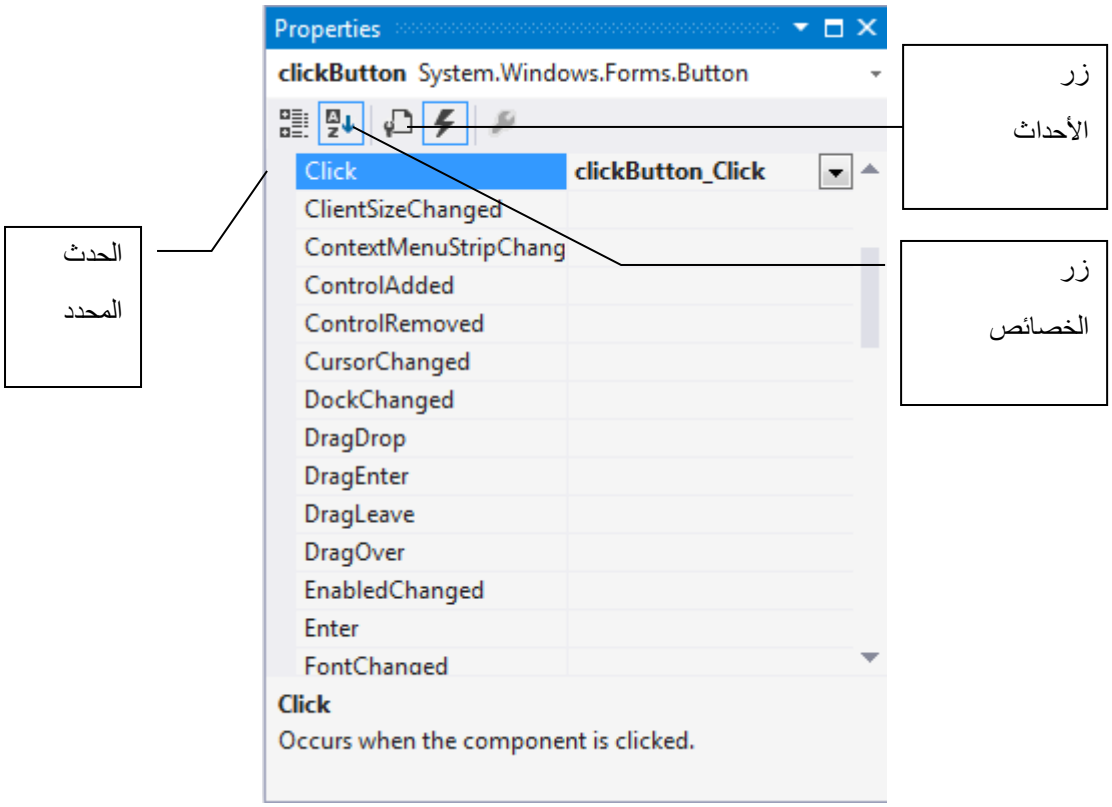
- تُشتق مفوضات الأحداث من الصف MulticastDelegate، والذي يُشتق بدوره من الصف Delegate، وكلاهما في فضاء الأسماء System.

طرق أخرى لإنشاء معالجات أحداث

- قمنا في المثال السابق بالنقر المزدوج على الزر في وضع التصميم لنكتب له معالج حدث. يُنشئ Visual Studio بهذه الطريقة معالج حدث للحدث الافتراضي Default Event لهذا العنصر، أي الحدث الأكثر استخداماً عادةً لهذا العنصر (النقر للزر مثلاً).
- تملك عناصر التحكم أنواعاً عديدة من الأحداث، وكل منها يملك معالج حدث خاص به.

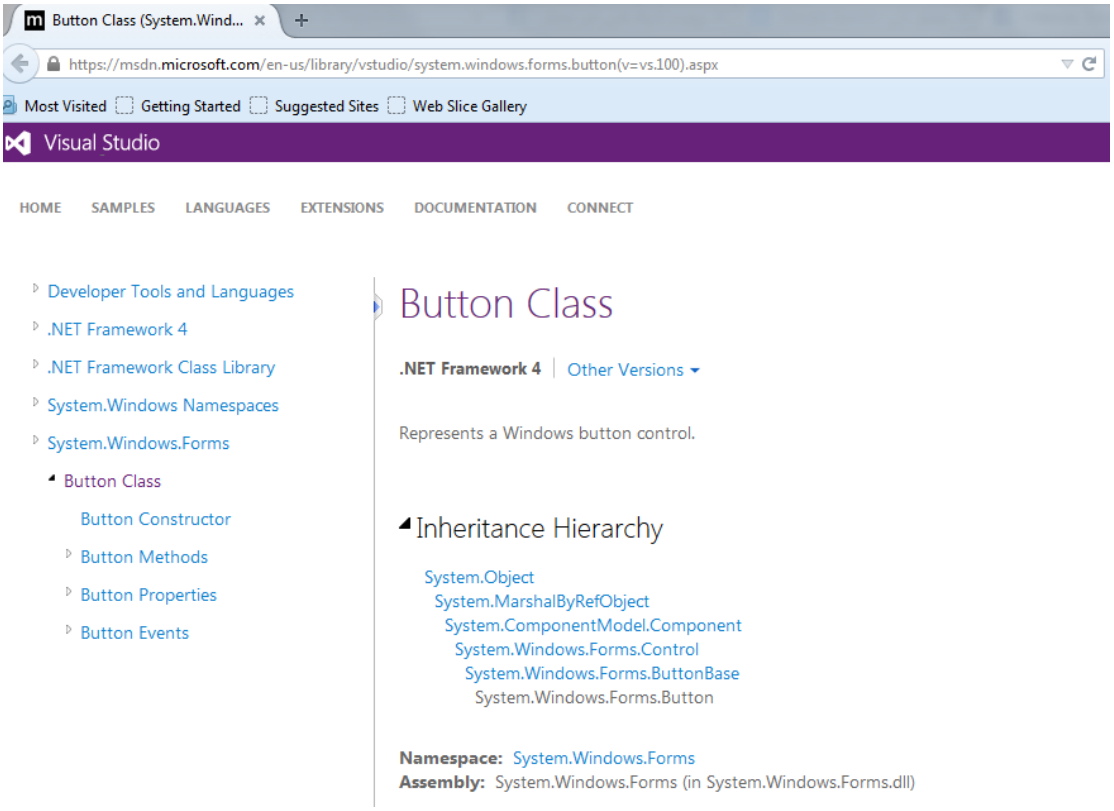
استعمال نافذة الخصائص لإنشاء معالجات أحداث

- يُمكن إنشاء معالجات أحداث إضافية عن طريق نافذة الخصائص Properties Window للعنصر.
- عند اختيار عنصر تحكم ثم النقر على زر Events في نافذة الخصائص والذي يبدو في الشكل التالي وعليه رمز يشبه البرق، ستظهر قائمة بجميع أحداث هذا العنصر. يُمكن إظهار كود معالج حدث موجود أو إنشاء معالج حدث جديد بالنقر المزدوج على اسم الحدث، كما يُمكن استخدام القائمة المنسدلة بجانب اسم الحدث لربطه إلى طريقة موجودة، حيث تظهر قائمة بالطرائق التي تحقق ترويسة المفوض المرتبط بالحدث المحدد.
- يُمكن العودة إلى خصائص العنصر بالنقر على زر Properties (خصائص) الموضح في الشكل التالي.

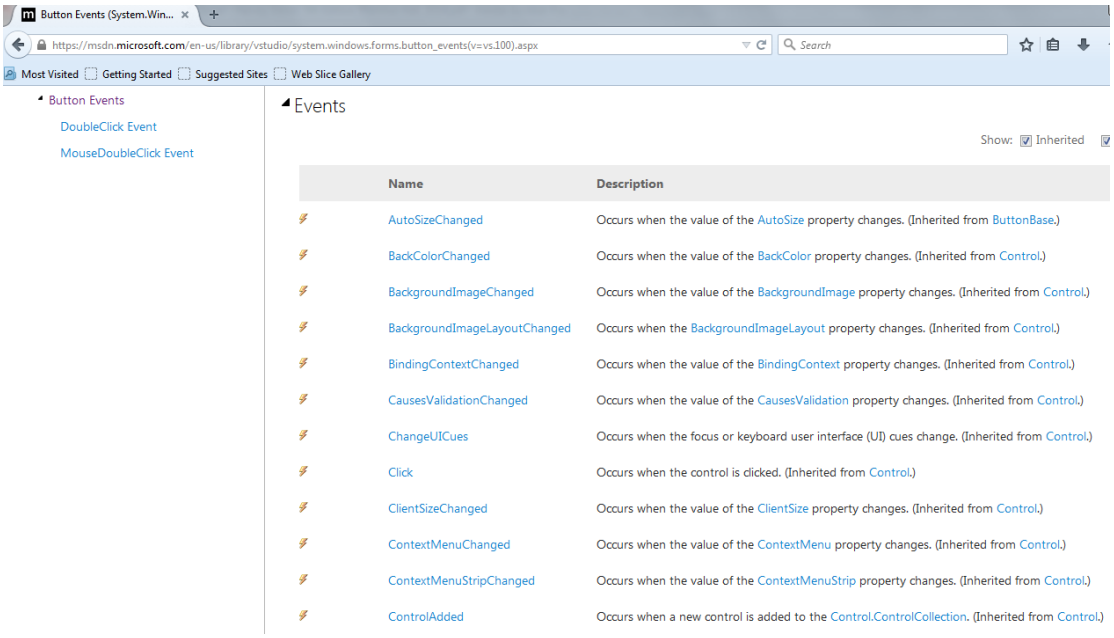


إيجاد معلومات عن الحدث

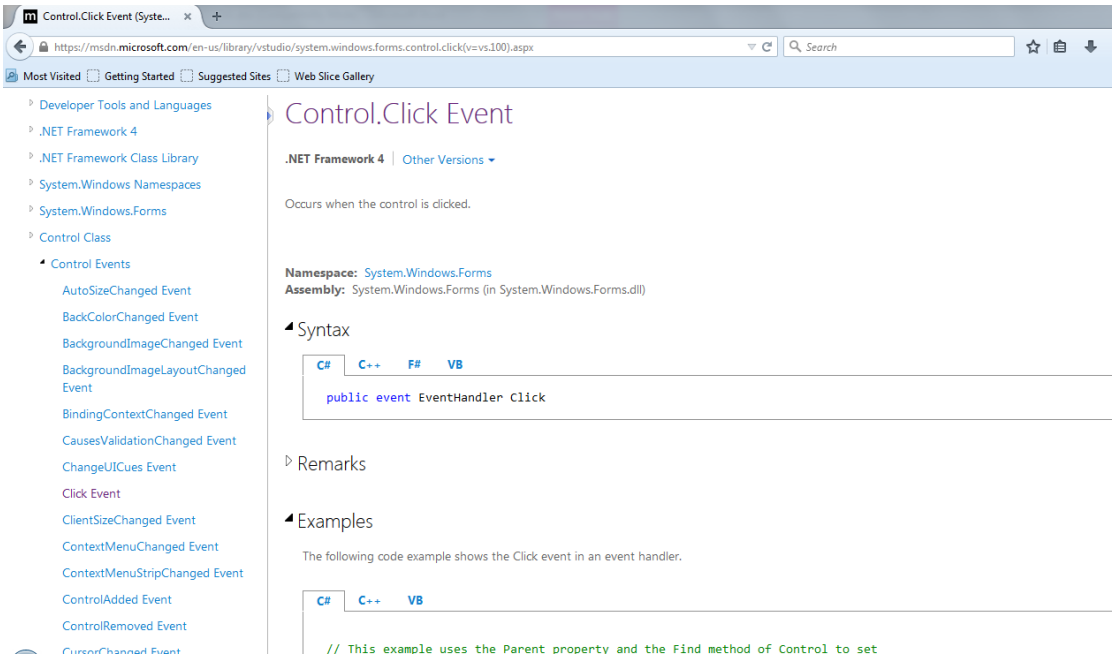
- يُمكن استخدام توثيق Visual Studio للحصول على المزيد من المعلومات حول الأحداث المختلفة لعنصر تحكم ما.
- اختر مثلاً زر الأمر ثم اضغط F1 لفتح صفحة المساعدة عن أزرار الأمر:



- انقر على الرابط Button Events مثلاً للحصول على شرح عن أحداث الزر:



- ثم انقر الرابط Click للحصول على شرح ومثال مساعد عن حدث النقر للزر:



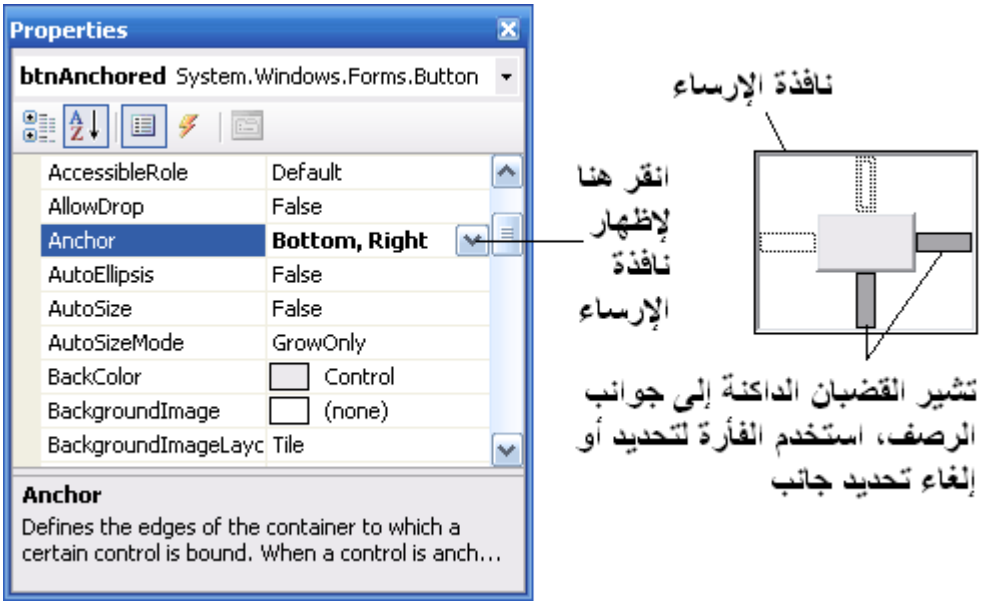
خصائص وتوضع عنصر التحكم

- نعرض فيما يلي خصائص شائعة للعديد من عناصر التحكم.
- تُشتق عناصر التحكم من الصف Control في فضاء الأسماء System.Windows.Froms.
- يُظهر الجدول التالي بعض خصائص وطرائق الصف Control:

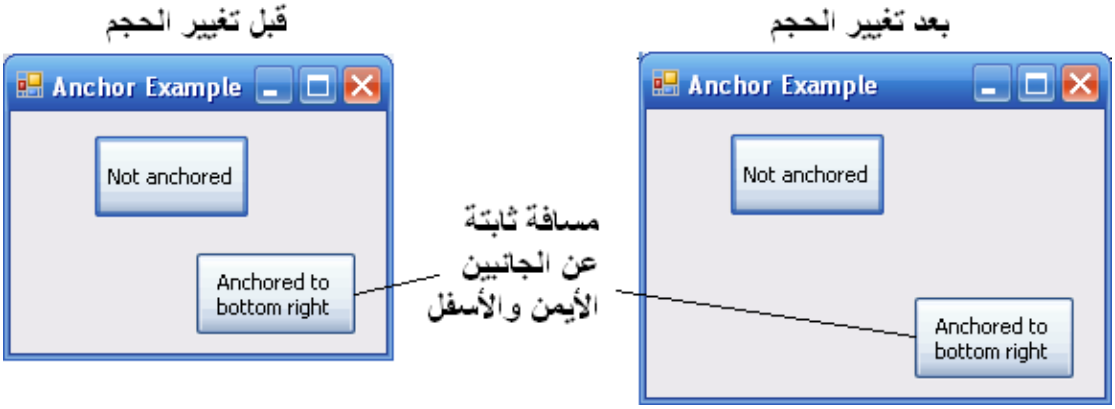
خصائص وطرائق الصف	الوصف
Control	
الخصائص	
BackColor	لون خلفية عنصر التحكم.
BackgroundImage	صورة خلفية عنصر التحكم.
Enabled	تُحدّد فيما إذا كان العنصر فعالاً (أي فيما إذا كان يُتاح للمستخدم التفاعل معه) وعادة تظهر عناصر التحكم غير الفعالة بلون باهت كدليل على عدم فعاليتها.
Focused	تُحدّد فيما إذا كان التركيز الحالي على العنصر.
Font	الخط المستعمل لإظهار نص عنصر التحكم.
ForeColor	اللون الأمامي للعنصر، ويُحّدد عادةً لون النص في الخاصية Text.
TabIndex	الترتيب الجدولي للعنصر، فعندما يتم ضغط محرف الجدولة Tab ينتقل التركيز بين العناصر وفق ترتيبها الجدولي، يُمكنك تحديد هذا الترتيب باستخدام هذه الخاصية.
TabStop	إذا كانت true يُمكن للمستخدم الوصول للعنصر باستخدام الزر Tab وإلا فلن نصل له باستخدام الزر Tab.
Text	النص المرتبط بالعنصر، يختلف موقعه ومظهره بحسب نمط العنصر.
Visible	تُحدّد فيما إذا كان العنصر مرئياً أو لا.
الطرائق	
Focus	وضع التركيز على العنصر.
Hide	إخفاء العنصر (يُسند false إلى Visible).
Show	إظهار العنصر (يُسند true إلى Visible).

- يُمكن تحديد قيم هذه الخصائص للعديد من عناصر التحكم. فمثلاً، تُحدّد الخاصية Text النص الذي يظهر على عنصر التحكم. يختلف موقع النص ومظهره بحسب نوع العنصر، ففي النموذج يظهر النص في شريط العنوان، أما في حالة الزر يظهر على سطح الزر.
- تنقل الطريقة Focus التركيز إلى عنصر التحكم وتجعله العنصر الفعال الحالي Active Control.

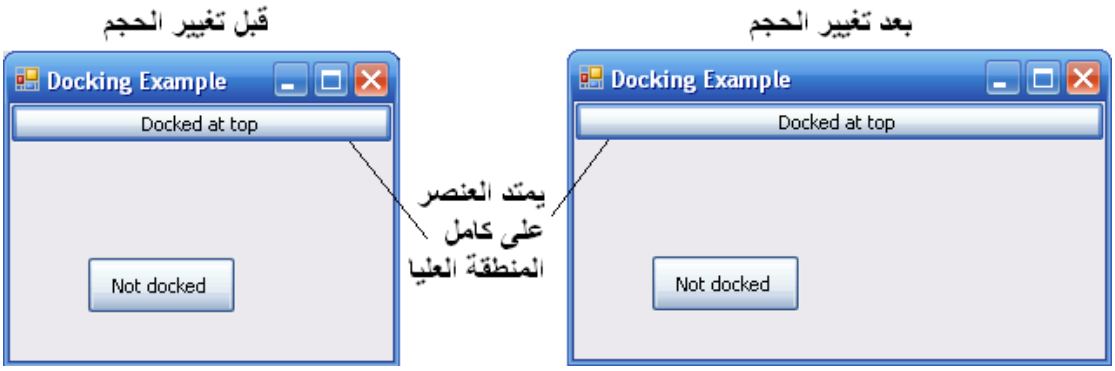
- عند الضغط على محرف الجدولة Tab في تطبيقات Windows، ينتقل التركيز من عنصر لآخر وفق الترتيب المحدد بالخاصية TabIndex. يقوم Visual Studio بإعطاء قيمة لهذه الخاصية بحسب ترتيب إضافة العناصر إلى النموذج في وضع التصميم. إلا أنه يُمكن تغيير ترتيب الجدولة باستخدام الخاصية TabIndex.
- إن ترتيب الجدولة مفيد للمستخدمين الذين يقومون بإدخال المعلومات في عدة عناصر، مثل مجموعة من صناديق النصوص التي تُمثل اسم المستخدم وعنوانه ورقم هاتفه، يُمكن للمستخدم إدخال المعلومات في عنصر ثم الانتقال بسرعة إلى العنصر التالي باستخدام الزر Tab.
- تُحدّد الخاصية Enabled فيما إذا كان العنصر متاحاً للمستخدم أم لا. فمثلاً، يكون زر اللصق في محرر نصوص غير متاح حتى يقوم المستخدم بنسخ نص ما.
- يظهر العنصر غير الفعال في معظم الحالات بلون رمادي باهت (يكون العنصر الفعال أقرب إلى الأسود من غير الفعال).
- يُمكن إخفاء عنصر تحكم بإسناد القيمة false إلى الخاصية Visible له، أو باستدعاء الطريقة Hide. في كلا الحالتين يبقى العنصر موجوداً لكن بدون أن يظهر على النموذج في وضع التنفيذ.
- يُمكن استعمال الإرساء والرصف لتحديد كيفية توضع العناصر في حاوية (مثل نموذج).
- يُبقي الإرساء Anchoring العنصر الموجود في حاوية على مسافة ثابتة من جوانب الحاوية عندما يتغير حجمها. مما يُحسن من تعامل المستخدم مع البرنامج. فمثلاً، يتوقع المستخدم وجود عنصر معين في زاوية محددة، يضمن الإرساء بقاء العنصر في تلك الزاوية حتى بعد تغيير حجم النموذج.
- يربط الرصف Docking عنصر ما بأحد جوانب الحاوية بحيث يتمدد على كامل الجانب، مثلاً، إذا كان هناك زر مرصوف على الجانب الأعلى للنموذج، يتمدد هذا الزر على طول الجانب الأعلى بغض النظر عن عرض النموذج.
- عند تغيير حجم حاوية ما تتحرك العناصر المرساة (وقد يتغير حجمها) بحيث يبقى بعدها عن جوانب الإرساء ثابتاً. تُرسى أغلب العناصر بشكل افتراضي على الجانبين الأيسر والعلوي. لمعاينة تأثير الإرساء، قم بإنشاء تطبيق يحوي زرين، قم بإرساء أحد الزرين على الجانبين الأيمن والسفلي كما يبدو في الشكل التالي:



واترك الزر الثاني بدون إرساء، شغل البرنامج وكبر النموذج، لاحظ أن بعد الزر ذو الإرساء عن الزاوية السفلى اليمنى يبقى ثابتاً، بينما يبقى بعد الزر الآخر عن الزاوية العليا اليسرى ثابتاً (الوضع الافتراضي).



- يُمكن أن نجعل عنصر يحتل ناحية كاملة من النموذج حتى عند تغيير حجم النموذج، مثل شريط الحالة StatusBar الذي يجب أن يبقى عادة في أسفل النموذج. يسمح الرصف Docking لعنصر أن يحتل ناحية كاملة من حاويته (الناحية اليسرى أو اليمنى أو العليا أو السفلى) أو أن يملأ كل الحاوية، حيث يتغير حجم العنصر المرصوف عند تغيير حجم حاويته. وضعنا في المثال التالي زر مرصوف على الجانب العلوي، عند تغيير حجم النموذج، يتغير حجم الزر ليوافق عرض النموذج الجديد.



- تُحدد الخاصية Padding (الحشوة)، المسافات الداخلية للعنصر، وتملك هذه الخاصية أربع قيم (قيمة لكل جانب)، وتكون قيمها صفر افتراضياً.
- يوضح الجدول التالي ملخصاً لبعض خصائص توضع العناصر:

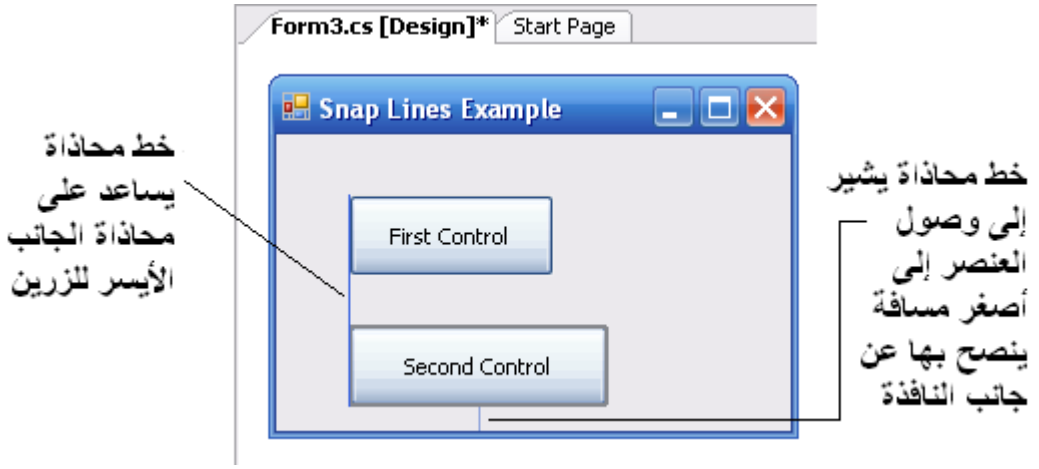
خصائص توضع عنصر Control	الوصف
Anchor	تُبقي العنصر على بعد ثابت عن جانب (أو جوانب) حاويته عند تغيير حجمها.
Dock	تسمح للعنصر أن يحتل ناحية كاملة من حاويته أو أن يملأ كل الحاوية.
Padding	تُحدّد المسافة بين العنصر والعناصر المرصوفة داخله، القيمة الافتراضية هي الصفر.
Location	تُحدّد موقع (إحداثيات) الزاوية العليا اليسرى للعنصر بالنسبة لحاويته.
Size	يُحدّد حجم العنصر بالبيكسل (Pixel) وهو غرض من النمط Size الذي له خاصيتان: العرض Width والارتفاع Height.
MinimumSize, MaximumSize	تُحدّد القياس الأصغري والأعظمي للعنصر على الترتيب.

- تتحدد خاصيتي الإرساء والرصف لعنصر ما بحسب حاويته Container (العنصر الحاوي له) والتي يمكن أن تكون نموذجاً أو لوحة Panel (نتعرض لها في فقرة قادمة).
- يُمكن تحديد القياس الأصغري والأعظمي لنموذج أو عنصر ما باستعمال الخاصيتين MaximumSize, MinimumSize على الترتيب. لكلتا هاتين الخاصيتين: العرض Width والارتفاع Height لتحديد حجم العنصر. تسمح هاتان الخاصيتان بتصميم واجهة ضمن مجال حجم معين، حيث لا يُسمح للمستخدم تصغير النموذج دون قياسه الأصغري، ولا يسمح تكبيره أكثر من القياس الأعظمي.

كما يُمكن جعل حجم النموذج ثابتاً بإعطائه قياساً أعظمياً وأصغرياً بنفس القيمة، أو تغيير الخاصية FormBorderStyle لتصبح FixedSingle.

استعمال Visual Studio لتحرير توضع عناصر الواجهة

- يؤمن Visual Studio أدوات تساعد على توضع العناصر في الواجهة. لاحظ أنه أثناء تحريك عنصر في نموذج تظهر الخطوط الزرقاء (التي تُعرف بخطوط المحاذاة Snap Lines) لتساعد في تحديد موقع عنصر مع مراعاة باقي العناصر وحواف النموذج، وهذه ميزة جديدة في Visual Studio.



- كما يوفر Visual Studio القائمة تنسيق Format التي تحوي العديد من الخيارات لتحرير توضع العناصر. عند تحديد أكثر من عنصر يمكنك استخدام القائمة الفرعية محاذاة Align لضبط محاذاة العناصر مع بعضها (يسار، مركز، يمين، فوق، وسط، تحت).
- كما يُمكن تحديد قيمة المسافات الأفقية Horizental Spacing والعمودية Vertical Spacing بين العناصر.
- يُمكن أيضاً إعطاء العناصر نفس القياس Make Same Size الطول Height أو العرض Width أو كليهما.

الفصل الثاني: عناصر التحكم (1)

عنوان الموضوع:

عناصر التحكم (1)

الكلمات المفتاحية:

Label، Box Text، Button، Group Box، Panel، Check Box، Radio Button.

ملخص:

نعرض في هذا الفصل بعض عناصر التحكم الأساسية وطريقة استخدام خصائصها وأحداثها.

أهداف تعليمية:

يتعرف الطالب في هذا الفصل على:

- اللصاقة Label.
- صندوق النص Text Box.
- الزر Button.
- صندوق المجموعة Group Box.
- اللوحة Panel.
- صندوق الاختيار Check Box.
- زر الخيار Radio Button.

المخطط:

عناصر التحكم (1)

- 3 وحدات (Learning Objects)

اللصاقات (Labels)، صناديق النص (TextBoxes) والأزرار (Buttons)

- تُظهر اللصاقات معلومات نصية، كما يُمكن أن تُظهر صوراً. وتُعرّف اللصاقات في الصف Label الذي يرث من الصف Control.
- تُظهر اللصاقات نصوصاً لا يُسمح للمستخدم أن يُعدّلها بشكل مباشر. إلا أنه يُمكن تحريره برمجياً عن طريق الخاصية Text للصاقة.
- يُظهر الجدول التالي قائمة بأهم خصائص اللصاقة:

أهم خصائص الصف Label	الوصف
Font	خيارات خط النص ضمن اللصاقة.
Text	النص الذي تظهره اللصاقة.
TextAlign	محاذاة النص ضمن اللصاقة، أفقياً (يمين، وسط، يسار) وعمودياً (أعلى، وسط، أسفل)

- أما صندوق النص (الصف TextBox) فهو صندوق يُظهر نصاً، كما يُتيح للمستخدم إدخال نص باستعمال لوحة المفاتيح.
- يُمكن الحصول على صندوق كلمة سر بإعطاء قيمة للخاصية PasswordChar لمربع النص TextBox. نحتاج عادةً لاستعمال كلا النوعين: صندوق النص وصندوق كلمة السر. فمثلاً، عند تسجيل الدخول يُطلب من المستخدم إدخال اسمه وكلمة سره.
- نعرض في الجدول التالي أهم خصائص وأحداث صندوق النص.

خصائص وأحداث الصف TextBox	الوصف
الخصائص	
AcceptsReturn	إذا كانت هذه الخاصية true وكان الصندوق متعدد الأسطر Multiline، سيؤدي ضغط محرف الإدخال Enter إلى إنشاء سطر جديد، وإلا فسيكون ضغط Enter مكافئاً للنقر على الزر الافتراضي للنموذج، وهو المحدد في الخاصية AcceptButton للنموذج Form.
Multiline	إذا كانت قيمتها true يمكن أن يحوي صندوق النص أكثر من سطر، القيمة الافتراضية هي false.
PasswordChar	إذا أسند محرف ما إلى هذه القيمة، يصبح صندوق النص صندوق كلمة

	سر، ويظهر هذا المحرف عوضاً عن كل محرف يدخله المستخدم.
ReadOnly	إذا كانت true لا يمكن تحرير النص في صندوق النص، القيمة الافتراضية هي false.
ScrollBars	في صناديق النصوص ذات الأسطر المتعددة، تُحدّد هذه الخاصية أشرطة التمرير التي تظهر (بدون None، أفقي Horizontal، عمودي Vertical، كلاهما Both)
Text	محتوى النص في صندوق النص.
الأحداث	
TextChanged	يتولد هذا الحدث عندما يتغير النص في صندوق النص (أي عندما يضيف أو يحذف المستخدم محارفاً منه). يتولد معالج حدث فارغ لهذا الحدث عند النقر المزدوج على صندوق النص في وضع التصميم.

- أما الزر Button، فهو عنصر التحكم الذي ينفذ مهمة ما عندما ينقر عليه المستخدم.
- يُظهر الجدول التالي قائمة بأهم خصائص وأحداث الصف Button:

خصائص وأحداث الصف Button	الوصف
الخصائص	
Text	تُحدد النص الذي يظهر على الزر.
FlatStyle	تُعدّل مظهر الزر، وتأخذ القيم Flat (بدون مؤثرات ثلاثية الأبعاد)، وPopup (بدون مؤثرات حتى يستقر مؤشر الفارة فوق الزر)، وStandard (ثلاثي أبعاد)، وSystem (يتحكم نظام التشغيل بالمظهر)، تكون القيمة الافتراضية هي Standard.
الأحداث	
Click	يتم توليد هذا الحدث عندما ينقر المستخدم على الزر. يتولد معالج حدث فارغ لهذا الحدث عند النقر المزدوج على الزر في وضع التصميم.

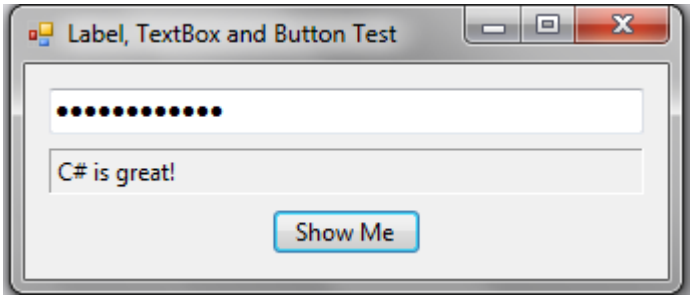
- نستخدم في المثال التالي صندوق نص وزر ولصاقة. حيث يُدخل المستخدم نصاً في صندوق كلمة سر، ثم يضغط الزر ليتم عرض النص الذي أدخله في اللصاقة.
- عندما ينقر المستخدم الزر Show Me سيتم عرض النص المدخل في صندوق النص في اللصاقة.
- لإنشاء هذا المثال، نقوم أولاً ببناء الواجهة عن طريق سحب عناصر التحكم (صندوق النص والزر واللصاقة) إلى النموذج. ومن ثم تعديل أسماء العناصر بعد توزيعها (من textBox1,button1,label1) إلى أسماء أكثر وضوحاً، ولتكن:

displayPasswordLabel, displayPasswordButton, displayPasswordTextBox

- تسمح الخاصية (اسم) (Name) في نافذة الخصائص بتغيير اسم المتغير المُعرّف لعنصر التحكم. يقوم Visual Studio بكتابة الكود اللازم ووضعه في الوظيفة IntializeComponent في الصف الجزئي Parial Class الموجود في الملف LabelTextBoxButtonTestForm.Designer.cs.

```
1  // LabelTextBoxButtonTestForm.cs
2  // Using a TextBox, Label and Button to display
3  // the hidden text in a password TextBox.
4  using System;
5  using System.Windows.Forms;
6  namespace LabelTextBoxButtonTest
7  {
8  // Form that creates a password TextBox and
9  // a Label to display TextBox contents
10 public partial class LabelTextBoxButtonTestForm : Form
11 {
12 // default constructor
13 public LabelTextBoxButtonTestForm()
14 {
15  InitializeComponent();
16 } // end constructor
17 // display user input in Label
18 private void displayPasswordButton_Click(
19    object sender, EventArgs e )
20 {  // display the text that the user typed
21   displayPasswordLabel.Text = inputPasswordTextBox.Text;
22 } // end method displayPasswordButton_Click
23 } // end class LabelTextBoxButtonTestForm
24 } // end namespace LabelTextBoxButtonTest
```

- نقوم أيضاً بتغيير الخاصية Text للزر لتصبح Show Me. ومن ثم مسح كل من النص الخاص باللصاقة والنص الخاص بصندوق النص كي يكونا خاليين عند بداية تشغيل البرنامج.
- نقوم بعدها بتعديل الخاصية BorderStyle للصاقة لتصبح Fixed3D لتعطي لها مظهراً ثلاثي الأبعاد. لاحظ أن الخاصية BorderStyle لصندوق النص هي Fixed3D بشكل افتراضي.
- ثم نقوم بإعطاء الخاصية PasswordChar القيمة *. لاحظ أن هذه الخاصية لا تأخذ سوى محرف واحد.
- نقوم بإنشاء معالج حدث للنقر على الزر وذلك بالنقر المزدوج عليه في وضع التصميم، ثم نكتب السطر 21 في جسم معالج الحدث:
  - `displayPasswordLabel.Text = inputPasswordTextBox.Text;`
- عندما ينقر المستخدم الزر ShowMe أثناء تنفيذ البرنامج، يقوم السطر 21 بعرض النص الذي أدخله المستخدم في صندوق النص ضمن اللصاقة.


Label, TextBox and Button Test
C# is great!
Show Me

صناديق المجموعات (GroupBoxes) واللوحات (Panels)

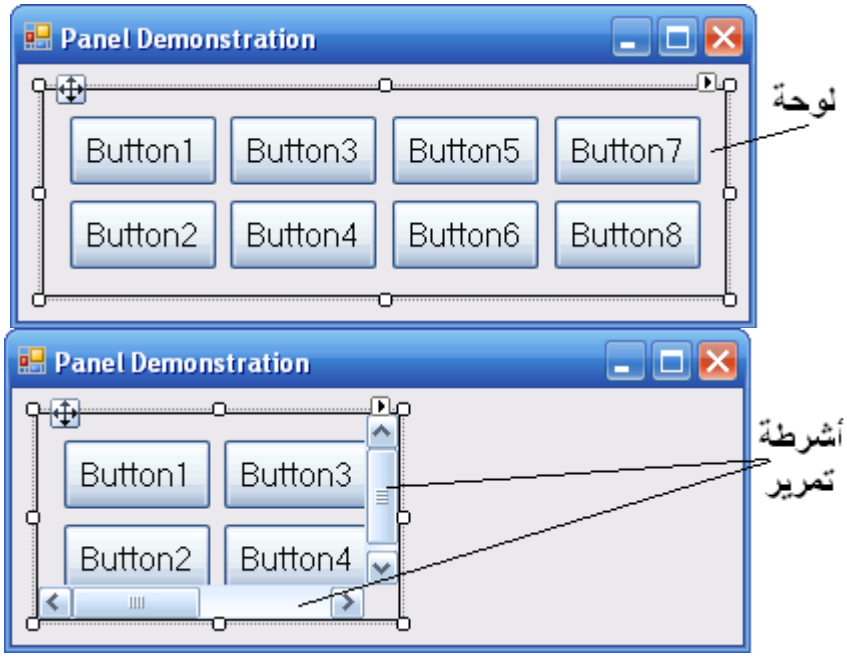
- يكون دور صناديق المجموعات واللوحات هو ترتيب العناصر ضمن الواجهة البيانية، وتُستخدم عادةً لتجميع عدة عناصر لها وظائف متقاربة أو مرتبطة فيما بينها. كما تتحرك كل هذه العناصر معاً عند تحريك مربع المجموعة أو اللوحة.
- يكمن الفرق الأساسي بين هذين العنصرين في أن صندوق المجموعة يُمكن أن يعرض نصاً ولا يمكن أن يحوي شريط تمرير، في حين يُمكن للوحة أن تحوي أشرطة تمرير لكنها لا تملك نصاً.
- يكون لصندوق المجموعة حافة ضيقة بشكل افتراضي، ويمكن تعديل حواف اللوحة عن طريق الخاصية BorderStyle، يُبين الجدولان التاليان أهم خصائص مربع المجموعة واللوحة.

خصائص GroupBox الصف	الوصف
Controls	مجموعة العناصر التي يحويها صندوق المجموعة.
Text	يحدد النص الذي سيظهر في أعلى صندوق المجموعة.

خصائص Panel الصف	الوصف
AutoScroll	تحدد فيما إذا كانت أشرطة التمرير ستظهر إذا أصبح حجم اللوحة غير كافياً، القيمة الافتراضية false.
BorderStyle	تحدد مظهر إطار اللوحة، القيمة الافتراضية هي None والقيم الأخرى هي Fixed3D و FixedSingle.
Controls	مجموعة العناصر التي تحويها اللوحة.

- لإنشاء صندوق مجموعة، نقوم بسحب الرمز الموافق في مربع الأدوات إلى النموذج ثم نسحب عناصر جديدة من مربع الأدوات إليه. تُضاف هذه العناصر إلى الخاصية Controls لتصبح جزءاً من صندوق المجموعة، كما يُمكن تحرير النص الخاص بصندوق المجموعة باستخدام الخاصية Text.
- لإنشاء لوحة، نقوم بسحب الرمز الموافق في مربع الأدوات إلى النموذج، يُمكن بعد ذلك إضافة عناصر على اللوحة بسحبها من مربع الأدوات إلى اللوحة.
- لتفعيل أشرطة التمرير للوحة نجعل قيمة الخاصية AutoScroll مساوية true، فإذا تغير حجم اللوحة بحيث يصبح من غير الممكن عرض جميع العناصر المحتواة فيها –سواء في زمن التنفيذ أو التصميم– تظهر أشرطة التمرير.

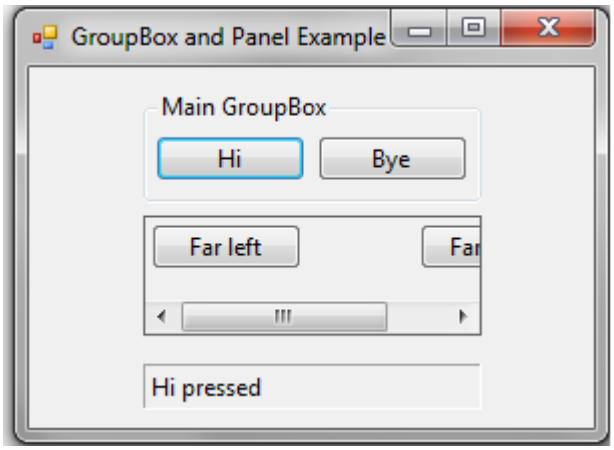
- قمنا في الشكل التالي بتعديل الخاصية BorderStyle للوحة لتصبح FixedSingle لتمييزها عن النموذج.

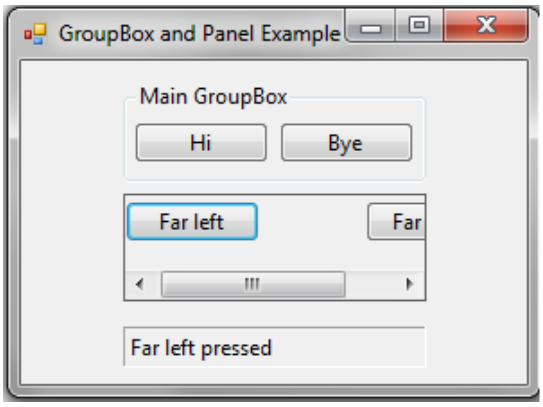


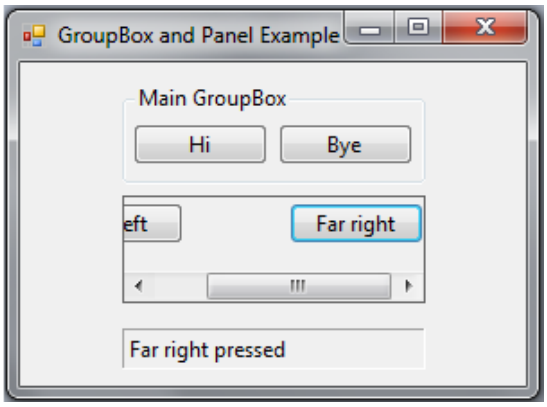
- نستعمل في المثال التالي صندوق مجموعة ولوحة لترتيب مجموعة أزرار، عند النقر على أحد هذه الأزرار يقوم معالج الحدث الموافق بتغيير النص في اللصاقة messageLabel.
- يحوي صندوق المجموعة (mainGroupBox) زرين: hiButton الذي يعرض النص Hi، والزر byeButton الذي يعرض النص Bye.
- تحوي اللوحة (mainPanel) زرين أيضاً: leftButton والذي يعرض النص Far Left والزر rightButton والذي يعرض النص Far Right، كما أن الخاصية AutoScroll للوحة تحمل القيمة true مما يسمح لأشرطة التمرير بالظهور عندما لا تكفي مساحة اللوحة.

```
1  // GroupboxPanelExampleForm.cs
2  // Using GroupBoxes and Panels to arrange Buttons.
3  using System;
4  using System.Windows.Forms;
5  namespace GroupBoxPanelExample
6  {
7  // Form that displays a GroupBox and a Panel
8  public partial class GroupBoxPanelExampleForm : Form
9  {
10 // default constructor
11 public GroupBoxPanelExampleForm()
12 {
13    InitializeComponent();
14 } // end constructor
15 // event handler for Hi Button
16 private void hiButton_Click( object sender, EventArgs e )
17 {
18    messageLabel.Text = "Hi pressed"; // change text in Label
19 } // end method hiButton_Click
20 // event handler for Bye Button
```

```
21 private void byeButton_Click( object sender, EventArgs e )
22 {
23    messageLabel.Text = "Bye pressed"; // change text in Label
24 } // end method byeButton_Click
25 // event handler for Far Left Button
26 private void leftButton_Click( object sender, EventArgs e )
27 {
28   messageLabel.Text = "Far left pressed"; // change text in Label
29 } // end method leftButton_Click
30 // event handler for Far Right Button
31 private void rightButton_Click( object sender, EventArgs e )
32 {
33   messageLabel.Text = "Far right pressed"; // change text in Label
34 } // end method rightButton_Click
35 } // end class GroupBoxPanelExampleForm
36 } // end namespace GroupBoxPanelExample
```







- لإضافة عناصر إلى صندوق المجموعة أو اللوحة يستدعي Visual Studio الطريقة Add ضمن الخاصية Control لكل حاوية، ويكتب هذا الكود في الملف LabelTextBoxButtonTestForm.Designer.cs.
- توجد معالجات الأحداث للأزرار الأربعة في الأسطر من 16 إلى 34، وقد أضفنا سطراً لكل معالج حدث (الأسطر 18 و23 و28 و33) لتغيير النص المعروض في اللصاقة للدلالة على الزر الذي تم نقره.

صناديق الاختيار (CheckBoxes) وأزرار الخيار (RadioButtons)

- تملك NET. نوعين من الأزرار التي تحمل معنى الحالة (محدد – غير محدد أو صح–خطأ) وهما: صندوق الاختيار CheckBox وزر الخيار RadioButton. يُشتق كل من هذين الصفين من الصف ButtonBase (مثل صف الزر Button).

صندوق الاختيار CheckBox

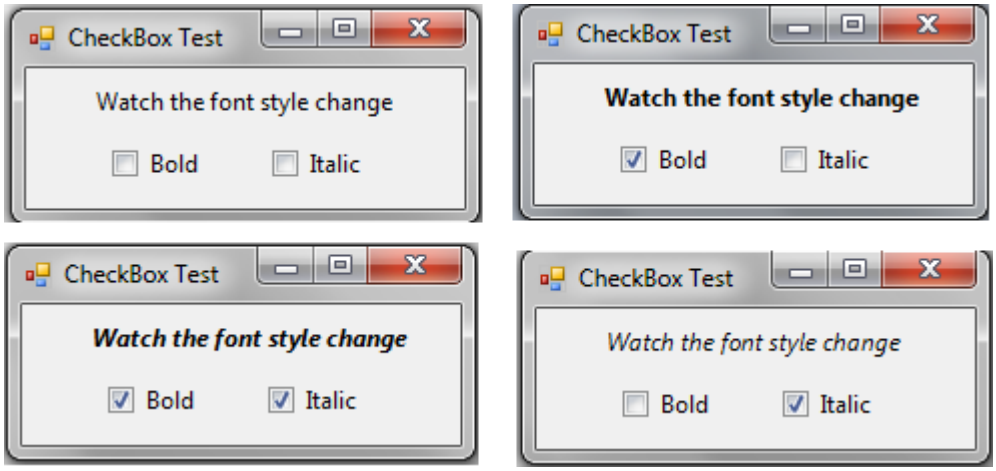
- يظهر صندوق الاختيار CheckBox كمربع صغير يكون خالياً أو يحمل علامة اختيار، عندما يقوم المستخدم بتحديد الخيار الذي يُمثله صندوق الاختيار تظهر علامة الاختيار في المربع، وإذا نقر عليه مرة أخرى ليُلغي اختياره تزول العلامة.
- يعرض الجدول التالي أهم خصائص وأحداث صندوق الاختيار:

خصائص وأحداث الصف CheckBox	الوصف
الخصائص	
Appearance	تأخذ قيمتين: Normal (الافتراضي) Button يظهر مربع الاختيار كزر
Checked	تُحدّد فيما إذا كان صندوق الاختيار محدداً (يحوي علامة اختيار) أو غير محدد (لا يحوي علامة اختيار)، وتُعيد قيمة منطقية True أو False.
CheckState	تُحدّد حالة اختيار صندوق الاختيار بقيمة من نمط التعداد (enum) CheckState والذي يأخذ إحدى القيم Checked,Unchecked,Indetermined. يُستعمل Indetermined عندما يكون من غير الواضح فيما إذا كان الصندوق الخيار محدداً أو غير محدد، ويكون الصندوق في هذه الحالة مظللاً.
ThreeState	تأخذ قيمتين: False (القيمة الافتراضية) True يكون في هذه الحالة لمربع الاختيار ثلاث قيم: checked مُحدّد unchecked غير مُحدّد indeterminate حالة عدم تعيين

Text	تُحدّد النص الذي يظهر على يمين الصندوق.
الأحداث	
CheckedChanged	يتولد عندما تتغير الخاصية Checked، وهو الحدث الافتراضي لصندوق الاختيار. يتولد معالج حدث فارغ لهذا الحدث عند النقر المزدوج على صندوق الاختيار في وضع التصميم.
CheckStateChanged	يتولد عندما تتغير الخاصية CheckState. يتولد معالج حدث فارغ لهذا الحدث عند النقر المزدوج على صندوق الاختيار في وضع التصميم.

- يسمح المثال التالي للمستخدم بتحديد صندوقي اختيار لتغيير نمط الخط في لصاقة، حيث يُحدّد معالج الحدث للصندوق الأول فيما إذا كان الخط عريضاً Bold. كما يُحدد الآخر فيما إذا كان الخط مائلاً Italic.

```
1  // CheckBoxTestForm.cs
2  // Using CheckBoxes to toggle italic and bold styles.
3  using System;
4  using System.Drawing;
5  using System.Windows.Forms;
6  namespace CheckBoxTest
7  {
8  // Form contains CheckBoxes to allow the user to modify sample text
9  public partial class CheckBoxTestForm : Form
10 {
11 // default constructor
12 public CheckBoxTestForm()
13 {
14  InitializeComponent();
15 } // end constructor
16 // toggle the font style between bold and
17 // not bold based on the current setting
18 private void boldCheckBox_CheckedChanged(
19 object sender, EventArgs e )
20 {
21  outputLabel.Font = new Font( outputLabel.Font,
22        outputLabel.Font.Style ^ FontStyle.Bold );
23 } // end metod boldCheckBox_CheckedChanged
24 // toggle the font setting between italic and
25 // not italic based on the current setting
26 private void italicCheckBox_CheckedChanged(object sender, EventArgs e)
27 {
28 outputLabel.Font = new Font( outputLabel.Font,
29          outputLabel.Font.Style ^ FontStyle.Italic );
30 } // end method italicCheckBox_CheckedChanged
31 } // end class CheckBoxTestForm
32 } //end namespace CheckBoxTest
```



- يحوي نموذج المثال صندوق الاختيار boldCheckBox والذي يحمل النص Bold، وصندوق الاختيار italicCheckBox والذي يحمل النص Italic، واللصاقة outputLabel والتي تحمل النص Watch the font style change.
- بعد أن نقوم بإنشاء العناصر، نقوم بكتابة معالجات الأحداث. إذا قمنا بالنقر المزدوج على صندوق الاختيار في زمن التصميم سيتم إنشاء معالج للحدث CheckedChanged.
- لتغيير نمط الخط في اللصاقة، يجب أن نُسند إلى الخاصية Font غرضاً جديداً من النمط Font. استعملنا هنا بانياً Constructor للنمط Font يأخذ كوسطاء اسم الصف وحجمه ونمطه. حيث قمنا بتمرير القيمتين outputLabel.Font.Name و outputLabel.Font.Size كأول وسيطين لنبقي اسم الخط وقياسه السابق. كما نُحدّد النمط Style باستعمال نمط التعداد FontStyle Enumeration، والذي يحوي القيم: Regular, Bold, Italic, Strikeout,Underline.
- إن الخاصية Style للخط هي للقراءة فقط read-only لذا فلا يمكن تحديد قيمتها إلا عند إنشاء غرض من الصف Font.
- يُمكن تركيب أنماط الخط باستعمال العمليات على البتات bitwise operators. حيث أن جميع المعطيات تُمثّل ضمن الحاسب على شكل تراكيب من 0 و 1 وكل منهما يُمثّل ضمن بت bit. يكون للنمط FontStyle الصفة System.FlagAttribue مما يعني أن قيم البتات المستعملة لتخزين النمط FontStyle تُختار بطريقة تسمح بتركيب قيم متعددة من النمط FontStyle لتكوين نمط مركب، وذلك باستعمال العمليات على البتات. أي أنه يُمكن تركيب أنماط مختلفة وإزالتها بحيث أنه لا يؤثر أحد هذه القيم في الأخرى.
- يُمكن تركيب أنماط مختلفة باستخدام العمليتين: الأو OR (|) والأو المقصورة (^) exclusive OR.
- عند تطبيق العملية OR على قيمتين تحمل إحداهما القيمة 1 ستكون النتيجة 1، ولفهم نتيجة استعمال هذه العملية لتركيب نمطين للخط لنفرض أن القيمة FontStyle.Bold تُمثّل بالقيمة 01 والقيمة FontStyle.Italic تُمثّل بالقيمة 10، لذا فعند تركيبهما باستخدام العملية OR نحصل على القيمة 11

01 = Bold

10 = Italic

--

11 = Bold and Italic

فالعملية OR تساعد في تركيب أنماط مختلفة، ولكن ما الذي سيحصل لو تراجع المستخدم عن تركيب النمط؟

- تُساعد العملية exclusive OR في تركيب أنماط مختلفة وإلغاء هذا التركيب بسهولة. فعند تطبيق العملية exclusive OR على بتين متساويتين تكون النتيجة 0، أما إذا كانا غير متساويين تكون النتيجة 1. ولفهم نتيجة استعمال هذه العملية لتركيب نمطين للخط لنفرض أن القيمة FontStyle.Bold تُمثل بالقيمة 01 والقيمة FontStyle.Italic تُمثل بالقيمة 10، لذا فعند تركيبهما نحصل على القيمة 11

01 = Bold

10 = Italic

--

11 = Bold and Italic

- لنفترض الآن أننا نريد إلغاء تركيب النمط Bold، فتكون أسهل طريقة هي إعادة تطبيق العملية exclusive OR بين القيمة Bold and Italic والقيمة Bold

11 = Bold and Italic

01 = Bold

--

10 = Italic

ولنوضح فائدة الاعتماد على هذه العمليات على البتات نذكر أن هناك 5 أنماط مختلفة للخط (Regular, Bold, Italic, Strikeout,Underline) تؤدي إلى وجود 16 حالة مختلفة، مما يجعل الاعتماد على العمليات على البتات أكثر سهولة وأقل تطلباً للكود.

زر الخيار RadioButton

- تُشبه أزرار الخيار –والتي يُمثلها الصف RadioButton– صناديق الاختيار، فلها حالتان فقط: محددة selected وغير محددة not selected، إلا أن أزرار الخيار تكون عادة ضمن مجموعة group بحيث يكون واحد منها فقط محدد، كما أن تحديد أحدها يجعل الباقين غير محددين حتماً، لذا تُستعمل أزرار الخيار لتمثيل مجموعة خيارات متنافية.
- تُشكّل جميع أزرار الخيار المضافة إلى حاوية واحدة مجموعة واحدة متنافية.
- يُبين الجدول التالي أهم خصائص وأحداث زر الخيار:

خصائص وأحداث الصف RadioButton	الوصف
الخصائص	
Checked	تحدد إذا كان زر الخيار محدداً.
Text	تحدد نص الزر.
الأحداث	
CheckedChanged	يتولد كلما يتم تحديد أو إلغاء تحديد زر الخيار، وهو الحدث الافتراضي لزر الخيار. يتولد معالج حدث فارغ لهذا الحدث عند النقر المزدوج على زر الخيار في وضع التصميم.

- يستعمل البرنامج في المثال التالي أزرار خيار ليتيح للمستخدم تحديد خيارات رسالة MessgeBox. يضغط المستخدم –بعد تحديد الخيارات– الزر Display لتظهر الرسالة، وتوجد لصاقة في الزاوية اليسرى السفلى تعرض نتيجة الرسالة (نعم، لا، إلغاء الأمر ...).
- قمنا لتخزين خيارات المستخدم بإنشاء وتهيئة الغرضين iconType و buttonType.
- الغرض iconType هو غرض من النمط MessageBoxIcon ويُمكن أن يحوي إحدى القيم: Asterisk, Error, Exclamation, Hand, Information,Question.
- أما الغرض buttonType فهو غرض من النمط MessageBoxButtons ويأخذ أحد القيم: AbortRetryIgnore, OK, OKCancel, RetryCancel, YesNo, YesNoCancel. يوضح اسم القيمة الخيارات المتاحة للمستخدم في الرسالة، وقد عرضنا في الأمثلة كل الحالات المختلفة للأزرار.

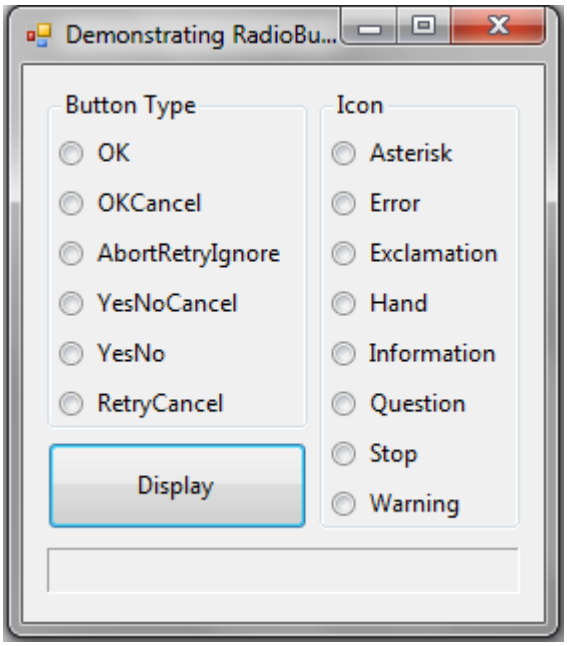
قمنا بإنشاء مربعي مجموعة، واحد لكل نمط تعداد، ولهما النصين Button Type و Icon، ويحوي كل منهما أزرار خيارات مناسبة لنمط التعداد الموافق، وبما أنه تم تجميع أزرار الخيار في مجموعتين فيمكننا اختيار زر واحد من كل مجموعة.

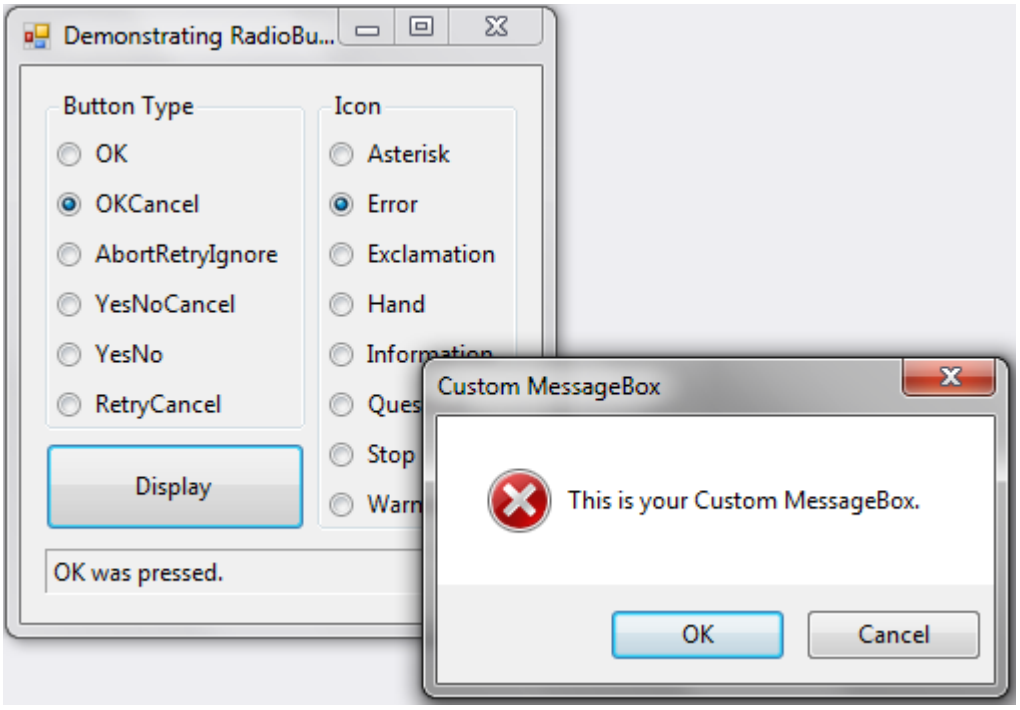
قمنا بكتابة معالج للحدث CheckedChanged لأزرار المجموعة الأولى Button Type والذي يقوم بتحديد القيمة المناسبة للمتحول buttonType (الأسطر من 22 إلى 38)، وقمنا بذلك بشكل مشابه من أجل المجموعة الثانية (الأسطر من 43 إلى 65) لتحديد قيمة المتحول iconType.

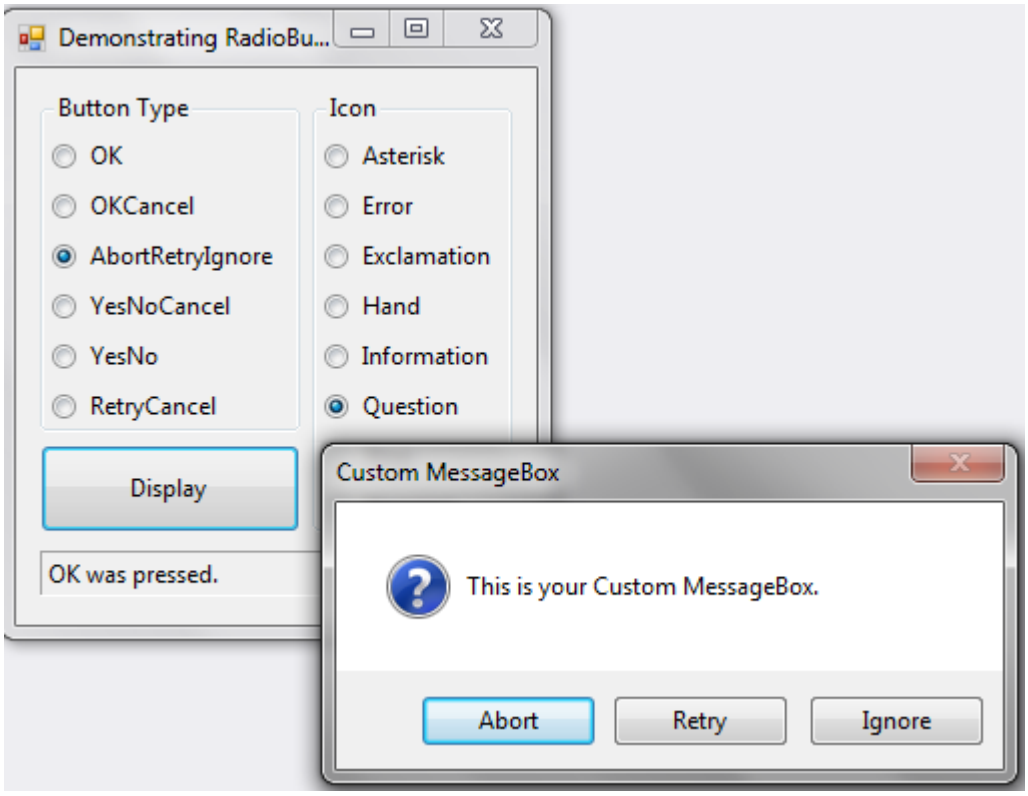
```
1  // RadioButtonTestForm.cs
2  // Using RadioButtons to set message window options.
3  using System;
4  using System.Windows.Forms;
5  namespace RadioButtonTest
6  {
7  // Form contains several RadioButtons--user chooses one
8  // from each group to create a custom MessageBox
9  public partial class RadioButtonTestForm : Form
```

```
{
// create variables that store the user's choice of options
private MessageBoxIcon iconType;
private MessageBoxButtons buttonType;
// default constructor
public RadioButtonTestForm()
{
InitializeComponent();
} // end constructor
// change Buttons based on option chosen by sender
private void buttonType_CheckedChanged( object sender, EventArgs e )
{
if ( sender == okRadioButton ) // display OK Button
      buttonType = MessageBoxButtons.OK;
  // display OK and Cancel Buttons
else if ( sender == okCancelRadioButton )
      buttonType = MessageBoxButtons.OKCancel;
 // display Abort, Retry and Ignore Buttons
else if ( sender == abortRetryIgnoreRadioButton )
      buttonType = MessageBoxButtons.AbortRetryIgnore;
// display Yes, No and Cancel Buttons
else if ( sender == yesNoCancelRadioButton )
      buttonType = MessageBoxButtons.YesNoCancel;
// display Yes and No Buttons
else if ( sender == yesNoRadioButton )
      buttonType = MessageBoxButtons.YesNo;
// only one option left--display Retry and Cancel Buttons
else
buttonType = MessageBoxButtons.RetryCancel;
} // end method buttonType_Changed
// change Icon based on option chosen by sender
private void iconType_CheckedChanged( object sender, EventArgs e )
{
 if ( sender == asteriskRadioButton ) // display asterisk Icon
  iconType = MessageBoxIcon.Asterisk;
 // display error Icon
 else if ( sender == errorRadioButton )
 iconType = MessageBoxIcon.Error;
// display exclamation point Icon
 else if ( sender == exclamationRadioButton )
 iconType = MessageBoxIcon.Exclamation;
// display hand Icon
else if ( sender == handRadioButton )
iconType = MessageBoxIcon.Hand;
// display information Icon
else if ( sender == informationRadioButton )
iconType = MessageBoxIcon.Information;
// display question mark Icon
else if ( sender == questionRadioButton )
iconType = MessageBoxIcon.Question;
// display stop Icon
else if ( sender == stopRadioButton )
iconType = MessageBoxIcon.Stop;
// only one option left--display warning Icon
Else
iconType = MessageBoxIcon.Warning;
} // end method iconType_CheckChanged
// display MessageBox and Button user pressed
private void displayButton_Click( object sender, EventArgs e )
{
// display MessageBox and store
// the value of the Button that was pressed
```

```
72  DialogResult result = MessageBox.Show("This is your Custom MessageBox.",
    "Custom MessageBox",buttonType, iconType);
73  // check to see which Button was pressed in the MessageBox
74  // change text displayed accordingly
75  switch ( result )
76  {
77  case DialogResult.OK:
78  displayLabel.Text = "OK was pressed.";
79  break;
80  case DialogResult.Cancel:
81  displayLabel.Text = "Cancel was pressed.";
82  break;
83  case DialogResult.Abort:
84  displayLabel.Text = "Abort was pressed.";
85  break;
86  case DialogResult.Retry:
87  displayLabel.Text = "Retry was pressed.";
88  break;
89  case DialogResult.Ignore:
90  displayLabel.Text = "Ignore was pressed.";
91  break;
92  case DialogResult.Yes:
93  displayLabel.Text = "Yes was pressed.";
94  break;
95  case DialogResult.No:
96  displayLabel.Text = "No was pressed.";
97  break;
98  } // end switch
99  } // end method displayButton_Click
100 } // end class RadioButtonsTestForm
101 } // end namespace RadioButtonsTest
```







- يقوم معالج حدث النقر على الزر displayButton بإظهار رسالة (السطر 72)، تُحدد خيارات الرسالة بالمتحولات المخزنة سابقاً buttonType و iconType. وعندما يقوم المستخدم بالنقر على أحد الأزرار في الرسالة، تُرد هذه القيمة إلى البرنامج، وهي من نمط التعداد DialogResult والذي يحمل أحد القيم Abort, Cancel, Ignore, No, None, OK, Retry, Yes.

- تقوم تعليمة switch في الأسطر من 75 إلى 98 بفحص القيمة المعادة من الرسالة وتحدد القيمة المناسبة للخاصية displayLabel.Text.

الفصل الثالث: عناصر التحكم (2)

عنوان الموضوع:

عناصر التحكم (2)

الكلمات المفتاحية:

Picture Boxe، Tootip، NumericUpDown، أحداث الفأرة، أحداث لوحة المفاتيح.

ملخص:

نتابع في هذا الفصل استعراض عناصر التحكم.

أهداف تعليمية:

يتعرف الطالب في هذا الفصل على:

- صناديق الصور Picture Boxes.
- التلميحات Tooltips.
- القوائم الرقمية NumericUpDown.
- أحداث الفأرة.
- أحداث لوحة المفاتيح.

المخطط:

عناصر التحكم (2)

- 4 وحدات (Learning Objects)

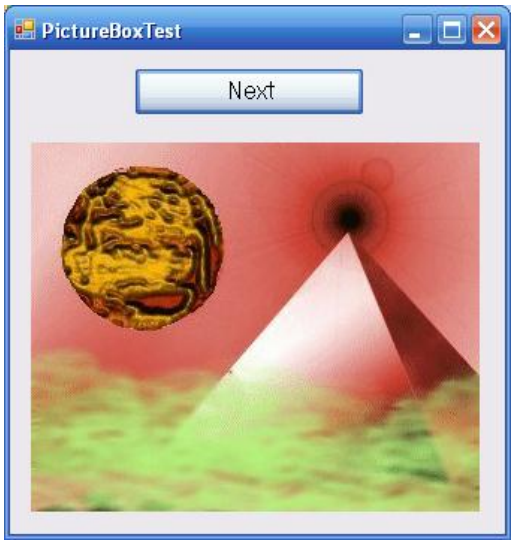
صناديق الصور (Picture Boxes)

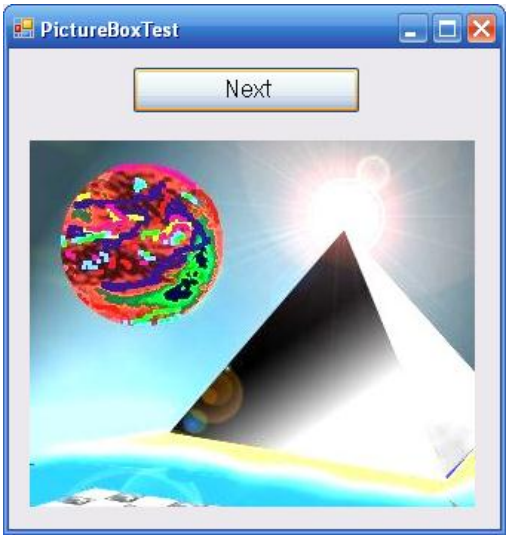
- يكون دور صندوق الصورة هو عرض صورة. يُمكن أن تكون الصور المعروضة من عدة أنواع:bitmap أو GIF أو JPEG.
- تُحدّد الصورة عن طريق الخاصية Image لصندوق الصورة.
- يعرض الجدول التالي أهم الخصائص والأحداث لصندوق الصورة:

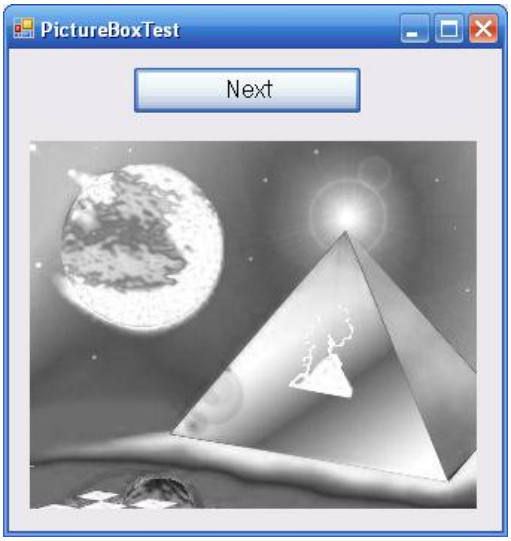
خصائص وأحداث الصف PictureBox	الوصف
الخصائص	
Image	تحدد الصورة التي سيتم عرضها في مربع الصورة.
SizeMode	وهي من نمط تعداد يحدد كيفية تحديد موقع وقياس الصورة، ويمكن أن يأخذ إحدى القيم: Normal والذي يعني وضع الصورة في الزاوية العليا اليسرى من مربع الصورة، CenterImage والذي يعني وضع الصورة في وسط مربع الصورة، وكلا القيمتين السابقتين يقتطع جزءاً من الصورة إذا كانت كبيرة، StretchImage يقوم بتغيير قياس الصورة لتناسب قياس مربع الصورة، AutoSize يقوم بتغيير قياس مربع الصورة ليناسب قياس الصورة.
الأحداث	
Click	يحصل عندما ينقر المستخدم على الصورة. يتولد معالج حدث فارغ لهذا الحدث عند النقر المزدوج على مربع الصورة في وضع التصميم.

- يستعمل المثال التالي مربع صورة يحمل الاسم imagePictureBox لعرض أحد ثلاث صور: image0,image1,image2 والموجودة في مصادر المشروع **resources**.
- عندما ينقر المستخدم الزر Next Image تظهر الصورة التالية حسب التسلسل، وعندما نصل إلى الصورة الأخيرة ثم ننقر الزر Next Image نعود لعرض الصورة الأولى.
- قمنا ضمن معالج الحدث nextButton_Click (الأسطر من 20 إلى 24) باستعمال متغير عدد صحيح int اسمه imageNum لنخزن رقم الصورة التي نقوم بعرضها.

```
1  //PictureBoxTestForm.cs
2  // Using a PictureBox to display images.
3  using System;
4  using System.Drawing;
5  using System.Windows.Forms;
6  namespace PictureBoxTest
7  {
8  // Form to display different images when PictureBox is clicked
9  public partial class PictureBoxTestForm : Form
10 {
11 private int imageNum = -1; // determines which image is displayed
12 // default constructor
13 public PictureBoxTestForm()
14 {
15 InitializeComponent();
16 } // end constructor
17 // change image whenever Next Button is clicked
18 private void nextButton_Click( object sender, EventArgs e )
19 {
20 imageNum = ( imageNum + 1 ) % 3; // imageNum cycles from 0 to 2
21 // retrieve image from resources and load into PictureBox
22 imagePictureBox.Image = ( Image )
23 ( Properties.Resources.ResourceManager.GetObject(
24           string.Format( "image{0}", imageNum ) ) );
25 } // end method nextButton_Click
26 } // end class PictureBoxTestForm
27 } // end namespace PictureBoxTest
```



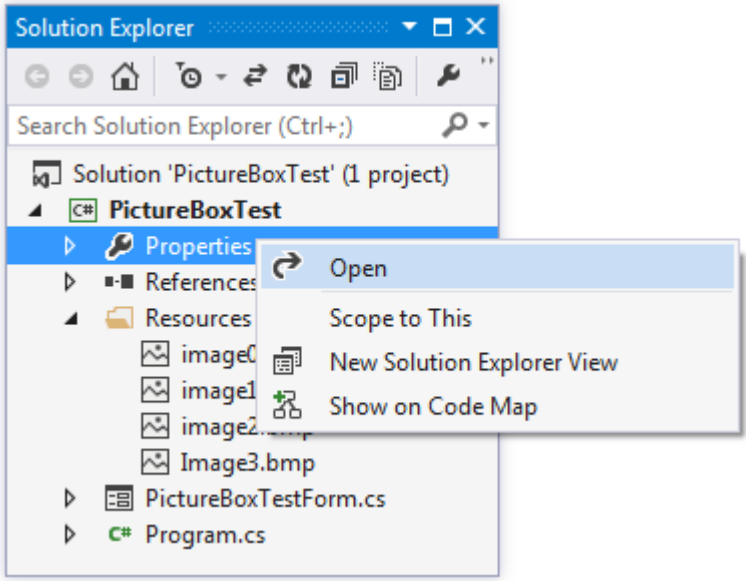




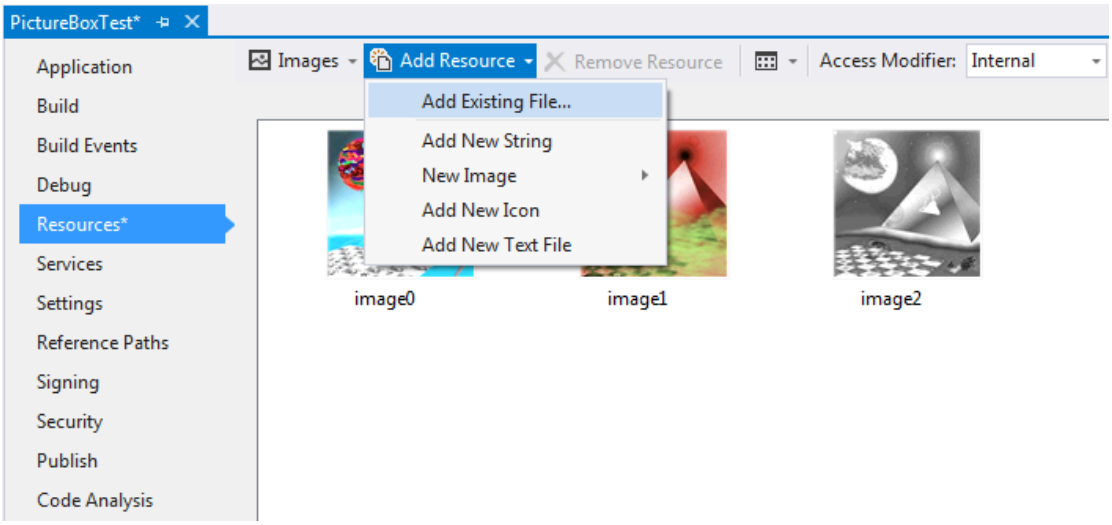
استخدام المصادر برمجياً

- نقوم في هذا المثال بإضافة الصور كمصادر **resources**. سيقوم المترجم بتضمين هذه الصور في الملف التنفيذي والسماح للتطبيق بالوصول إلى الصور من خلال فضاء الأسماء Properties.
- يُمكنك اتباع الخطوات التالية لإضافة الصور كمصادر للمشروع:

1. انقر بالزر الأيمن على عقدة الخصائص **Properties** في مستعرض الحل **Solution Explorer** واختر فتح **Open** لفتح خصائص المشروع.



2. افتح التبويب مصادر Resources من اليسار.
3. انقر السهم جانب Add Resource ثم اختر Add Existing File... لإظهار صندوق حوار إضافة ملف Add existing file to resources.



4. أضف الصور المطلوبة ثم قم بالحفظ.

- ستظهر الملفات التي تُضيفها في المجلد **Resources** في مستعرض الحل **Solution Explorer**.
- تُخزّن مصادر المشروع في الصف **Resources** من فضاء الأسماء Properties. يحوي الصف Resources الغرض **ResourceManager** للتعامل مع المصادر برمجياً.
- للوصول إلى صورة، استخدم الطريقة **GetObject** والتي لها معامل هو اسم المصدر كما يظهر في تبويب المصادر.

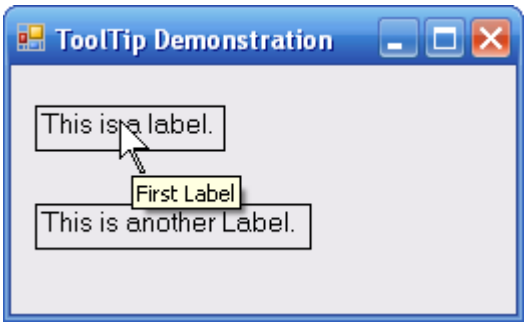
التلميحات (ToolTips)

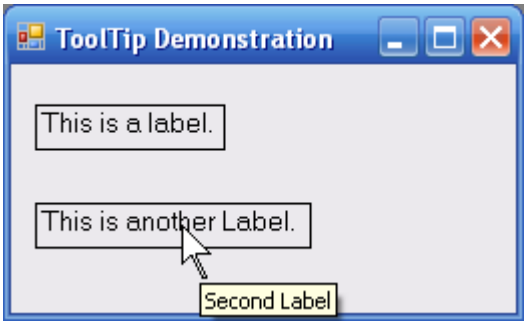
- وهي النصوص التي تظهر عندما تستقر الفأرة فوق عنصر في الواجهة. تساعدك مثلاً التلميحات التي يوفرها Visual Studio على أن تألف بيئة التطوير وتذكرك بدور كل بند في الواجهة. كما أن العديد من البرامج توفر تلمحيات لتساعد المستخدم في تذكر مهام العناصر. نعرض فيما يلي كيفية استخدام الكائن ToolTip لإضافة التلميحات.
- يعرض الجدول التالي أهم خصائص وأحداث هذا الكائن:

خصائص وأحداث الصف ToolTip	الوصف
الخصائص	
AutoPopDelay	الفترة الزمنية (بالميللي ثانية ms) التي يظهر فيها التلميح عندما تكون الفأرة فوق عنصر التحكم.
InitialDelay	الفترة الزمنية (بالميللي ثانية ms) التي يجب أن تستقر فيها الفأرة فوق عنصر التحكم قبل أن يظهر التلميح.
ReshowDelay	الفترة الزمنية (بالميللي ثانية ms) التي تفصل بين ظهور تلميحين عند نقل الفأرة من عنصر تحكم لآخر.
الأحداث	
Draw	يحصل عندما يظهر التعليق، يتيح هذا الحدث للمبرمج تغيير مظهر التلميح.

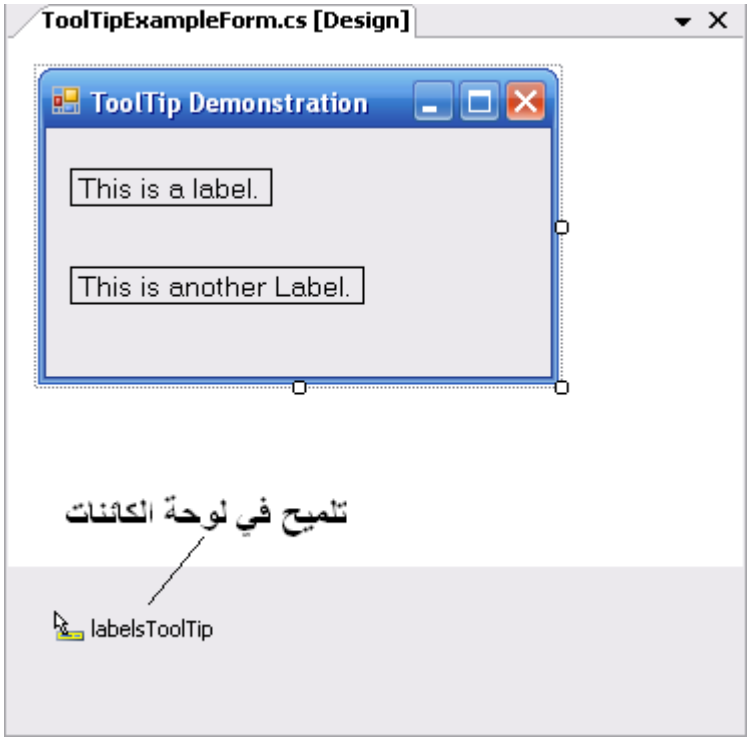
- عند إضافة كائن تلميح ToolTip Component إلى النموذج يظهر في لوحة الكائنات Component Tray وهي المساحة الرمادية تحت النموذج في وضع التصميم. وبعدها، ستظهر خاصية جديدة لجميع عناصر التحكم الأخرى وهي الخاصية ToolTip on متبوعة باسم كائن التلميح المضاف. فلو كان اسمه مثلاً helpfulToolTip، لكان اسم الخاصية helpfulToolTip on ToolTip، تُحدّد هذه الخاصية نص التلميح لكل عنصر.
- قمنا في المثال التالي بإنشاء واجهة تحوي لصاقتين لنوضح كيفية تحديد نصي تلميح مختلفين لهما. كما قمنا بتعديل الخاصية BorderStyle للصاقتين لتصبح FixedSingle. وبما أنه لا يوجد لدينا معالجات أحداث في الكود فهو يحوي فقط بانياً Constructor.

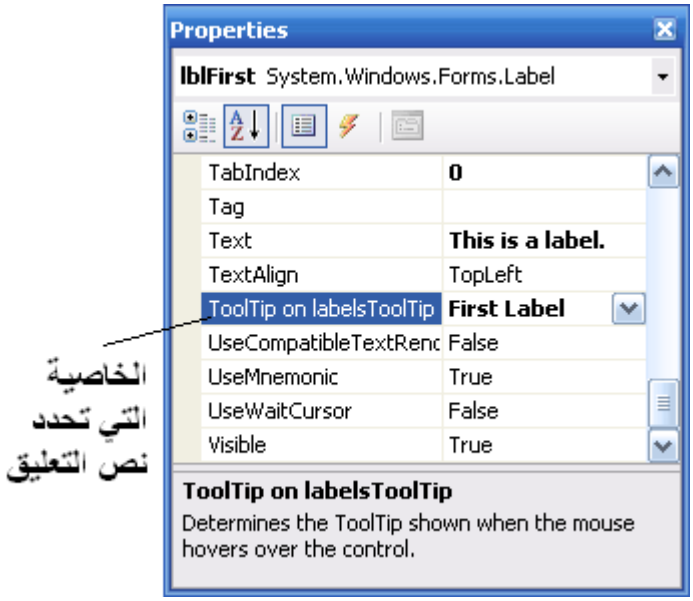
```
1  // ToolTipDemonstrationForm.cs
2  // Demonstrating the ToolTip component.
3  using System;
4  using System.Windows.Forms;
5  namespace ToolTipDemonstration
6  {
7  public partial class ToolTipDemonstrationForm : Form
8  {
9  // default constructor
10 public ToolTipDemonstrationForm()
11 {
12 InitializeComponent();
13 } // end constructor
14 // no event handlers needed for this example
15 } // end class ToolTipDemonstrationForm
16 } // end namespace ToolTipDemonstration
```





- أضفنا في هذا المثال كائن تلميح اسمه labelsToolTip ولصاقتين، وغيرنا قيمة نص التلميح للصاقة الأولى لتصبح First Label، وللثانية لتصبح Second Label، توضح الأشكال التالية مكان كائن التلميح في لوحة الكائنات وكيفية تحديد نص التلميح للصاقة لأولى.


ToolTipExampleForm.cs [Design]
ToolTip Demonstration
This is a label.
This is another Label.
تلميح في لوحة الكائنات
labelsToolTip


Properties
lblFirst System.Windows.Forms.Label
TabIndex 0
Tag
Text This is a label.
TextAlign TopLeft
ToolTip on labelsToolTip First Label
UseCompatibleTextRenc False
UseMnemonic True
UseWaitCursor False
Visible True
ToolTip on labelsToolTip
Determines the ToolTip shown when the mouse hovers over the control.
الخاصية
التي تحدد
نص التعليق

القوائم الرقمية (NumericUpDown)

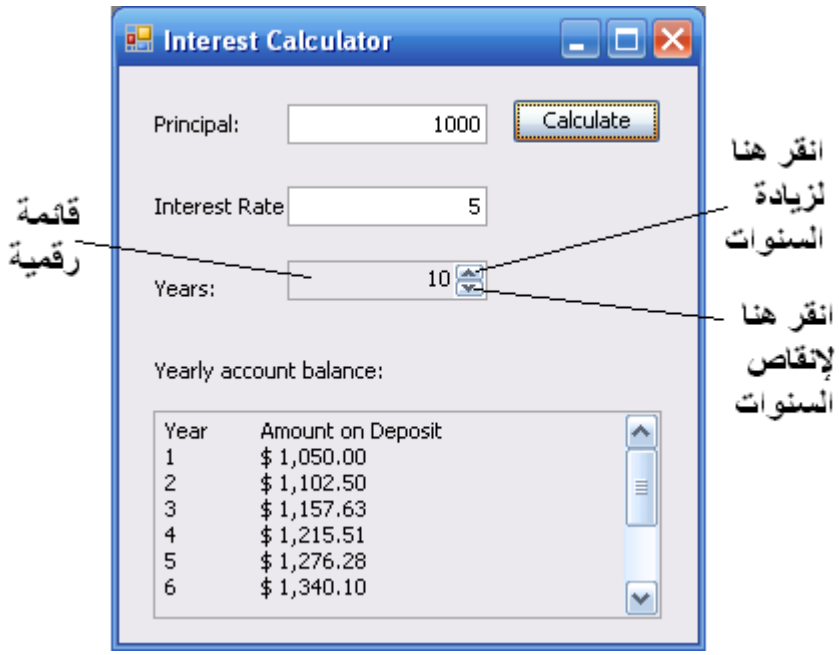
- تُستخدم القائمة الرقمية NumericUpDown لحصر القيمة المدخلة من المستخدم بقيمة رقمية في مجال محدد. وتظهر بشكل مربع نص له في الجانب الأيمن زران صغيران يحملان صورة سهمين أحدهما للأعلى والآخر للأسفل لزيادة أو إنقاص القيمة في مربع النص. تتحدد القيمة العليا والدنيا للقائمة الرقمية بالخاصيتين Maximum و Minimum على الترتيب، وكلاهما من النمط قيمة عشرية decimal. كما تُحدد الخاصية Increment مقدار تغير القيمة عند النقر على أحد الزرين (للأعلى أو الأسفل)، وهي أيضاً من النمط decimal.
- يوضح الجدول التالي أهم خصائص وأحداث القائمة الرقمية:

خصائص وأحداث الصف NumericUpDown	الوصف
الخصائص	
Increment	تحدد مقدار زيادة أو انقاص القيمة الحالية إذا نقر المستخدم أحد الزرين (للأعلى أو الأسفل).
Maximum	أكبر قيمة يمكن إدخالها.
Minimum	أصغر قيمة يمكن إدخالها.
UpDownAlign	تحدد محاذاة الزرين على اليمين أو اليسار.
Value	القيمة الحالية المعروضة.
الأحداث	
ValueChanged	يحصل عندما تتغير القيمة، وهو الحدث الافتراضي للقائمة الرقمية.

- يوضح المثال التالي استعمال القائمة الرقمية في برنامج يقوم بحساب معدل الفائدة. نستخدم مربعي نص لإدخال المبلغ الأولي ومعدل الفائدة، كما نستخدم قائمة رقمية اسمها yearUpDown لتحديد عدد السنوات. تكون قيمة الخاصية Minimum هي 1، والخاصية Maximum هي 10، كما تركنا الخاصية Increment على قيمتها الافتراضية وهي 1. مما يعني أن المستخدم يستطيع إدخال عدد من السنوات بين 1 و10 بتزايد 1، أما لو قمنا بجعل الزيادة مساوية 0.5 لكان بإمكان المستخدم إدخال قيم مثل 1.5 أو 2.5.
- كما أننا جعلنا للخاصية Readonly القيمة true كي لا يتمكن المستخدم من إدخال رقم بتحريره في القائمة، فهو مضطر إذاً إلى استعمال الزرين (للأعلى والأسفل) لتحديد عدد السنوات. القيمة الافتراضية لهذه الخاصية هي false.

- قمنا في النهاية باستعمال مربع نص متعدد الأسطر للقراءة فقط لعرض النتيجة، وزودناه بشريط تمرير عمودي ليتمكن المستخدم من استعراض النتيجة.

```
1  // interestCalculatorForm.cs
2  // Demonstrating the NumericUpDown control.
3  using System;
4  using System.Windows.Forms;
5  namespace NumericUpDownTest
6  {
7  public partial class InterestCalculatorForm : Form
8  {
9  // default constructor
10 public InterestCalculatorForm()
11 {
12  InitializeComponent();
13 } // end constructor
14 private void calculateButton_Click(object sender, EventArgs e )
15 {
16 // declare variables to store user input
17 decimal principal; // store principal
18 double rate; // store interest rate
19 int year; // store number of years
20 decimal amount; // store amount
21 string output; // store output
22 // retrieve user input
23 principal = Convert.ToDecimal( principalTextBox.Text );
24 rate = Convert.ToDouble( interestTextBox.Text );
25 year = Convert.ToInt32( yearUpDown.Value );
26 // set output header
27 output = "Year\tAmount on Deposit\r\n";
28 // calculate amount after each year and append to output
29 for ( int yearCounter = 1; yearCounter <= year; ++yearCounter )
30 {
31 amount = principal *
32 ( ( decimal ) Math.Pow( ( 1 + rate / 100 ), yearCounter ) );
33 output += ( yearCounter + "\t" +
34 String.Format( "{0:C}", amount ) + "\r\n" );
35 } // end for
36 displayTextBox.Text = output; // display result
37 } // end method calculateButton_Click
38 } // end class interestCalculatorForm
39 } // end namespace NumericUpDownTest
```


Interest Calculator
Principal:
1000
Calculate
Interest Rate
5
Years:
10
Yearly account balance:
Year
Amount on Deposit
1
$ 1,050.00
2
$ 1,102.50
3
$ 1,157.63
4
$ 1,215.51
5
$ 1,276.28
6
$ 1,340.10
قائمة رقمية
انقر هنا لزيادة السنوات
انقر هنا لإنقاص السنوات

معالجة أحداث الفأرة (Mouse-Events)

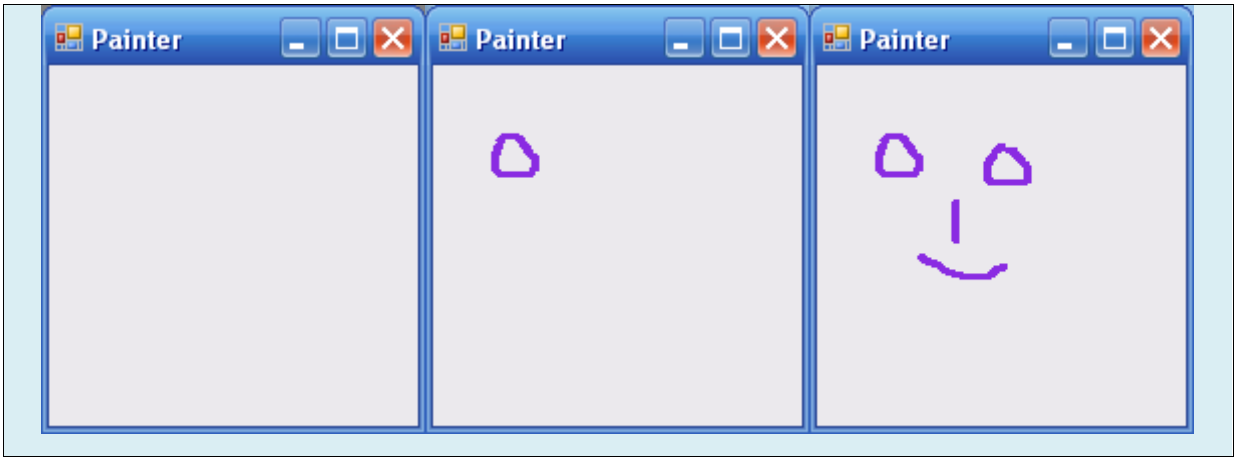
- تتولد أحداث الفأرة مثل النقر والتحريك عندما يتفاعل المستخدم مع عنصر التحكم بواسطة الفأرة. يُمكن لأي عنصر يرث من الصف System.Windows.Forms.Control أن يعالج أحداث الفأرة.
- يتم، في معظم أحداث الفأرة، تمرير معلومات الحدث إلى معالج الحدث عن طريق غرض من الصف MouseEventArgs، كما يُستخدم نمط المفوض MouseEventHandler لإنشاء معالج لأحداث الفأرة. تملك كل طريقة تمثل معالج حدث وسيط من النمط object وآخر من النمط MouseEventArgs.
- يحوي الصف MouseEventArgs معلومات تتعلق بحدث الفأرة، مثل إحداثيات الفأرة، وزر الفأرة الذي تم ضغطه (الأيمن Right أو الأيسر Left أو الأوسط Middle) وعدد مرات نقر الفأرة، مع الانتباه إلى أن إحداثيات الفأرة هي بالنسبة للعنصر، أي أن الإحداثيات (0،0) تمثل الزاوية اليسرى العليا لعنصر التحكم.
- نعرض في الجدول التالي أهم أحداث الفأرة:

أحداث الفأرة	الوصف
أحداث لها وسيط من النوع EventArgs	
MouseEnter	يحصل عندما يدخل مؤشر الفأرة حدود العنصر.
MouseLeave	يحصل عندما يغادر مؤشر الفأرة حدود العنصر.
MouseHover	يحصل عندما يتحرك مؤشر الفأرة ضمن حدود العنصر.
أحداث لها وسيط من النوع MouseEventArgs	
MouseDown	يحصل عندما يتم ضغط زر الفأرة عندما تكون ضمن حدود العنصر.
MouseMove	يحصل عندما يتحرك مؤشر الفأرة عندما تكون ضمن حدود العنصر.
MouseUp	يحصل عندما يتم تحرير زر الفأرة عندما تكون ضمن حدود العنصر.
MouseWheel	يحصل عندما يتم تحريك دولاب الفأرة عندما يكون التركيز على العنصر.
خصائص الصف MouseEventArgs	
Button	زر الفأرة الذي تم ضغطه (الأيمن Right أو الأيسر Left أو الأوسط Middle).
Clickes	عدد مرات النقر.

X	الإحداثية X للفأرة عند وقوع الحدث.
Y	الإحداثية Y للفأرة عند وقوع الحدث.

- يستعمل المثال التالي أحداث الفأرة في الرسم على نموذج، فكلما قام المستخدم بسحب الفأرة (أي تحريك الفأرة وهو يضغط زرها) نقوم برسم دائرة صغيرة في مكان الفأرة.
- نُعرّف في السطر 11 المتحول shouldPaint الذي يحدد أنه يجب الرسم على النموذج. نريد من البرنامج أن يرسم على النموذج فقط إذا كان زر الفأرة مضغوطاً أثناء تحريكها. لذا، وبما أن المستخدم عندما يضغط زر الفأرة يتولد الحدث MouseDown فسنقوم بإسناد القيمة true للمتحول shouldPaint في معالج الحدث الموافق (في الأسطر من 18 إلى 22)، وعندما يحرر المستخدم زر الفأرة يتولد الحدث MouseUp فسنقوم بإسناد القيمة false للمتحول shouldPaint في معالج الحدث الموافق (في الأسطر من 24 إلى 28) وسيتوقف البرنامج عن الرسم حينها. وعلى العكس من الحدث MouseMove الذي يتولد بشكل مستمر، يتولد الحدث MouseDown فقط لحظة ضغط زر الفأرة، ويتولد الحدث MouseUp فقط عند تحريره.
- كلما تحركت الفأرة فوق عنصر تحكم يتولد الحدث MouseMove الخاص بالعنصر. ضمن معالج الحدث PainerForm_MouseMove (الأسطر من 30 إلى 40) يقوم البرنامج بالرسم إذا كانت قيمة shouldPaint هي true (أي أن زر الفأرة مضغوط)، يستدعي السطر 35 طريقة موروثة ضمن الصف Form وهي CreateGraphics لإنشاء غرض من الصف Graphics لنتمكن من الرسم على النموذج. يوفر الصف Graphics العديد من الطرائق لرسم عدة أشكال. نستعمل في السطر 37 الطريقة FillEllipse لنرسم دائرة، يكون الوسيط الأول لهذه الطريقة غرض من الصف SolidBrush والذي يحدد لوناً لرسم الدائرة يمرر في البناء، يحوي النمط Color العديد من الألوان مسبقة التعريف وقد اخترنا BlueViolet (بنفسجي)، تقوم الطريقة FillEllipse برسم قطع ناقص ضمن مستطيل نحدد إحداثيات زاويته العليا اليسرى وعرضه وارتفاعه، وهي الوسطاء الأربعة المتبقية للطريقة FillEllipse. لقد استعملنا معلومات الحدث لنستخرج إحداثيات الفأرة (e.X,e.Y) ونمررها على أنها زاوية المستطيل، ولنرسم دائرة نحدد طولاً وعرضاً متساويين (4Pixel). توجد الصفوف Graphics و SolidBrush و Color في فضاء الأسماء System.Drawing.

```
1  // PainterForm.cs
2  // Using the mouse to draw on a Form.
3  using System;
4  using System.Drawing;
5  using System.Windows.Forms;
6  namespace Painter
7  {
8  // creates a Form that is a drawing surface
9  public partial class PainterForm : Form
10 {
11  bool shouldPaint = false; // determines whether to paint
12 // default constructor
13 public PainterForm()
14 {
15 InitializeComponent();
16 } // end constructor
17 // should paint when mouse button is pressed down
18 private void PainterForm_MouseDown( object sender, MouseEventArgs e )
19 {
20 // indicate that user is dragging the mouse
21    shouldPaint = true;
22 } // end method Painter_MouseDown
23 // stop painting when mouse button is released
24 private void PainterForm_MouseUp( object sender, MouseEventArgs e )
25 {
26 // indicate that user released the mouse button
27 shouldPaint = false;
28 } // end method Painter_MouseUp
29 // draw circle whenever mouse moves with its button held down
30 private void PainterForm_MouseMove( object sender, MouseEventArgs e )
31 {
32 if ( shouldPaint ) // check if mouse button is being pressed
33 {
34 // draw a circle where the mouse pointer is present
35 using ( Graphics graphics = CreateGraphics() )
36 {
37 graphics.FillEllipse(
38 new SolidBrush( Color.BlueViolet ), e.X, e.Y, 4, 4 );
39 } // end using; calls graphics.Dispose()
40 } // end if
41 } // end method Painter_MouseMove
42 } // end class PainterForm
43 } // end namespace Painter
```


Painter
Painter
Painter

أحداث لوحة المفاتيح

الأهداف التعليمية:

- أحداث لوحة المفاتيح

ISSN: 2617-989X

معالجة أحداث لوحة المفاتيح (Keyboard-Events)

- تحصل أحداث لوحة المفاتيح عند ضغط و تحرير مفتاح ما في لوحة المفاتيح. ويُمكن معالجة هذه الأحداث من عناصر التحكم التي تشتق من الصف System.Windows.Forms.Control.
- يوجد ثلاثة أحداث للمفاتيح: KeyPress, KeyUp, KeyDown.
- يحصل الحدث KeyPress عندما يقوم المستخدم بضغط مفتاح يُمثل محرف ASCII والذي يُمكن أن يُعرف من الخاصية KeyChar للوسيط من نمط KeyPressEventArgs لحدث الضغط.
- لايوفر الحدث KeyPress معلومات عن حالة مفاتيح التبديل(Ctrl, Shift, Alt). أما إذا كانت هذه المعلومات هامة للمبرمج، فيُمكن استعمال الحدث KeyDown أو KeyUp، واللذان يملكان وسيطاً من النمط KeyEventArgs يحوي معلومات عن حالة مفاتيح التبديل. عادةً، نستعمل حالة مفاتيح التبديل مع الفأرة لتحديد معلومات معينة.
- يعرض الجدول التالي قائمة بأهم أحداث المفاتيح، تُعيد العديد من الخصائص قيمة من نمط التعداد Keys والذي يوفر قيمة ثابتة لكل مفتاح في لوحة المفاتيح. يحمل هذا النمط الصفة System.FlagAttribute لذا يمكن تركيب عدة قيم للدلالة على ضغط أكثر من مفتاح في نفس الوقت.

أحداث المفاتيح ووسائطها	الوصف
أحداث لها وسيط من النوع KeyEventArgs	
KeyDown	يتولد عند بداية ضغط المفتاح.
KeyUp	يتولد عند تحرير المفتاح.
أحداث لها وسيط من النوع KeyPressEventArgs	
KeyPress	يتولد عند ضغط المفتاح.
خصائص الصف KeyPressEventArgs	
KeyChar	يعيد محرف ASCII الموافق للمفتاح الذي ضغط.
Handled	يحدد فيما إذا تمت معالجة الحدث KeyPress.
خصائص الصف KeyEventArgs	
Alt	يحدد فيما إذا كان المفتاح Alt.
Control	يحدد فيما إذا كان المفتاح Ctrl.
Shift	يحدد فيما إذا كان المفتاح Shift.
Handled	يحدد فيما إذا تمت معالجة الحدث.
KeyCode	يعيد رمز المفتاح على شكل قيمة من نمط التعداد Keys وهذا لا يحتوي معلومات مفاتيح التبديل، يستعمل للتحقق من مفتاح معين.
KeyData	يعيد رمز المفتاح على شكل قيمة من نمط التعداد Keys ويحتوي

	معلومات مفاتيح التبديل، ويحمل كل المعلومات عن المفتاح الذي تم ضغطه.
KeyValue	يعيد رمز المفتاح على شكل عدد صحيح وليس من نمط التعداد Keys، ويستعمل للحصول على الرقم المعبر عن المفتاح والذي يسمى رمز ويندوز الافتراضي للمفتاح Windows virtual key code.
Modifiers	يعيد قيمة من نمط التعداد Keys تحوي معلومات مفاتيح التبديل.

- نوضح في المثال التالي كيفية معالجة ضغط المستخدم لمفتاح، حيث يتألف البرنامج من نموذج يحوي لصاقتين، استخدمنا إحداهما لعرض المفتاح الذي تم ضغطه، واستخدمنا الثانية لعرض معلومات مفاتيح التبديل.

```
1  //  KeyDemoForm.cs
2  // Displaying information about the key the user pressed.
3  using System;
4  using System.Windows.Forms;
5  namespace KeyDemo
6  {
7  // Form to display key information when key is pressed
8  public partial class KeyDemo : Form
9  {
10 // default constructor
11 public KeyDemo()
12 {
13 InitializeComponent();
14 } // end constructor
15 // display the character pressed using KeyChar
16 private void KeyDemoForm_KeyPress( object sender, KeyPressEventArgs e )
17 {
18  charLabel.Text = "Key pressed: " + e.KeyChar;
19 } // end method KeyDemo_KeyPress
20 // display modifier keys, key code, key data and key value
21 private void KeyDemoForm_KeyDown( object sender, KeyEventArgs e )
22 {
23 keyInfoLabel.Text =
24 "Alt: " + ( e.Alt ? "Yes" : "No" ) + '\n' +
25 "Shift: " + ( e.Shift ? "Yes" : "No" ) + '\n' +
26 "Ctrl: " + ( e.Control ? "Yes" : "No" ) + '\n' +
27 "KeyCode: " + e.KeyCode + '\n' +
28 "KeyData: " + e.KeyData + '\n' +
29 "KeyValue: " + e.KeyValue;
30 } // end method KeyDemo_KeyDown
31 // clear Labels when key released
32 private void KeyDemoForm_KeyUp( object sender, KeyEventArgs e )
33 {
34 charLabel.Text = "";
35 keyInfoLabel.Text = "";
36 } // end method KeyDemo_KeyUp
37 } // end class KeyDemoForm
38 } // end namespace KeyDemo
```

- تكون كلتا اللصاقتان خاليتين في البداية. تعرض اللصاقة charLabel القيمة المحرفية character للمفتاح الذي تم ضغطه، بينما تعرض اللصاقة keyInfoLabel معلومات عنه. وبما أن الحدثين KeyPress و KeyDown يحملان معلومات مختلفة فإن النموذج KeyDemoForm يعالجهما كلاهما.
- يستخدم معالج الحدث KeyPress (في الأسطر من 16 إلى 19) الخاصية KeyChar للغرض KeyPressEventArgs، والتي تعيد المحرف الموافق للمفتاح والذي قمنا بعرضه في اللصاقة

charLabel (السطر 18). إذا ضُغط مفتاح لا يقابل محرف ASCII فلن يحصل الحدث KeyPress ولن يتم عرض أي نص في اللصاقة charLabel. تذكّر أن ASCII هو نظام ترميز موحد للأحرف والأرقام وعلامات الترقيم وغيرها من المحارف، إلا أنه لا يدعم مفاتيح الوظائف function keys مثل F1 أو مفاتيح التعديل (Alt,Ctrl,Shift).

- يعرض معالج الحدث KeyDown (في الأسطر من 21 إلى 30) معلومات من الوسيط KeyEventArgs، يختبر معالج الحدث المفاتيح Alt و Shift و Ctrl باستخدام الخصائص Alt و Shift و Control وكل منها هو قيمة منطقية تكون true إذا كان المفتاح الموافق مضغوطاً وإلا فهو false، ثم يقوم معالج الحدث بعرض الخصائص KeyCode و KeyData و KeyValue.
- تعيد الخاصية KeyCode (السطر 27) رمز المفتاح على شكل قيمة من نمط التعداد Keys ولا تحتوي معلومات مفاتيح التبديل، لذا فإن المحرف a (الصغير) والمحرف A (الكبير) يُمثلان بالمحرف A (في نمط التعداد Keys).
- تُعيد الخاصية KeyData (السطر 28) رمز المفتاح على شكل قيمة من نمط التعداد Keys وتحتوي معلومات مفاتيح التبديل، لذا فإن المحرف A (الكبير) يظهر المحرف A (في نمط التعداد Keys) والمحرف Shift.
- تعيد الخاصية KeyValue (السطر 29) رمز المفتاح على شكل عدد صحيح، ويسمى رمز ويندوز الافتراضي للمفتاح Windows virtual key code ويفيد في اختبار محارف لا تمثل في ASCII مثل F12.
- وأخيراً يقوم معالج الحدث KeyUp في الأسطر 34 إلى 35 بمسح كلتا اللصاقتين عند تحرير المفتاح.

الفصل الرابع: عناصر التحكم (3)

عنوان الموضوع:

عناصر التحكم (3)

الكلمات المفتاحية:

القوائم، MonthCalender، DateTimePicker، LinkLabel.

ملخص:

نستعرض في هذا الفصل أولاً إنشاء القوائم والتعامل معها ثم نعرض لعناصر التحكم للتاريخ والوقت، كما نعرض استخدام لصاقة الرابط.

أهداف تعليمية:

يتعرف الطالب في هذا الفصل على:

- القوائم Menus.
- التقويم الشهري MonthCalender.
- اختيار التاريخ DateTimePicker.
- لصاقة الربط LinkLabel.

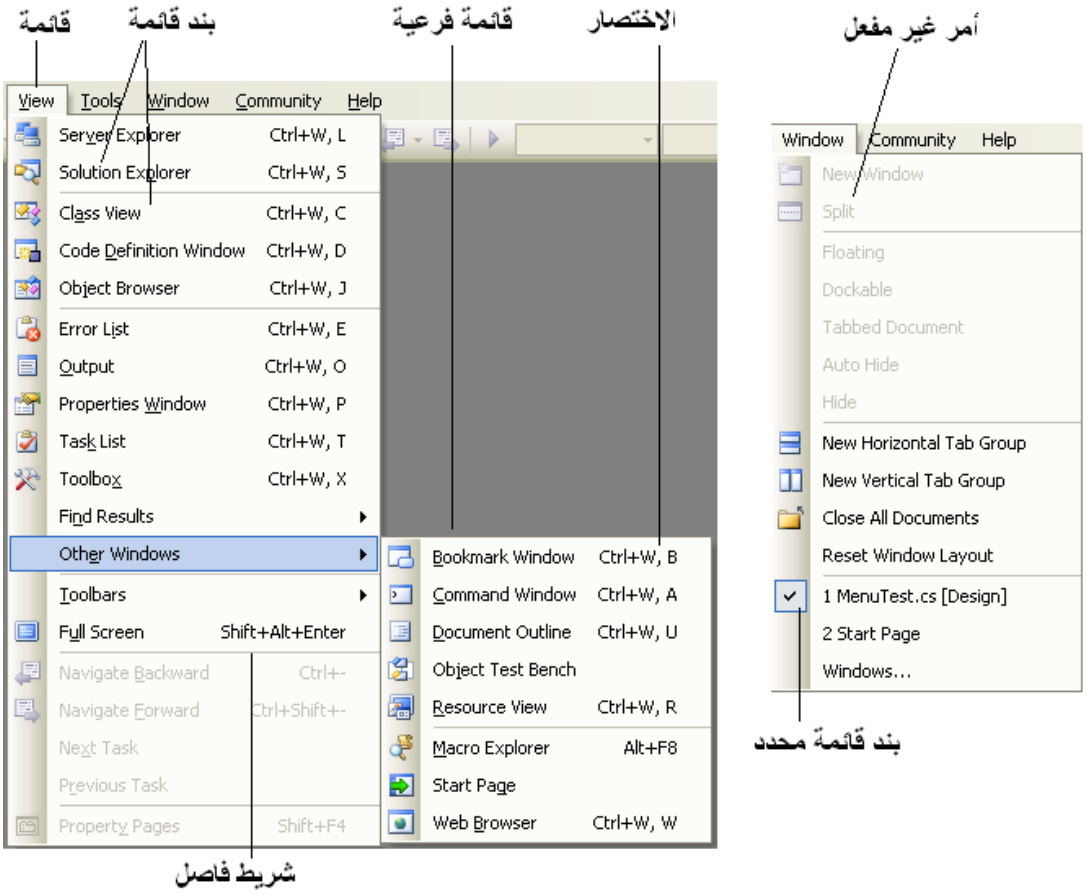
المخطط:

عناصر التحكم (3)

- 3 وحدات (Learning Objects)

القوائم Menus

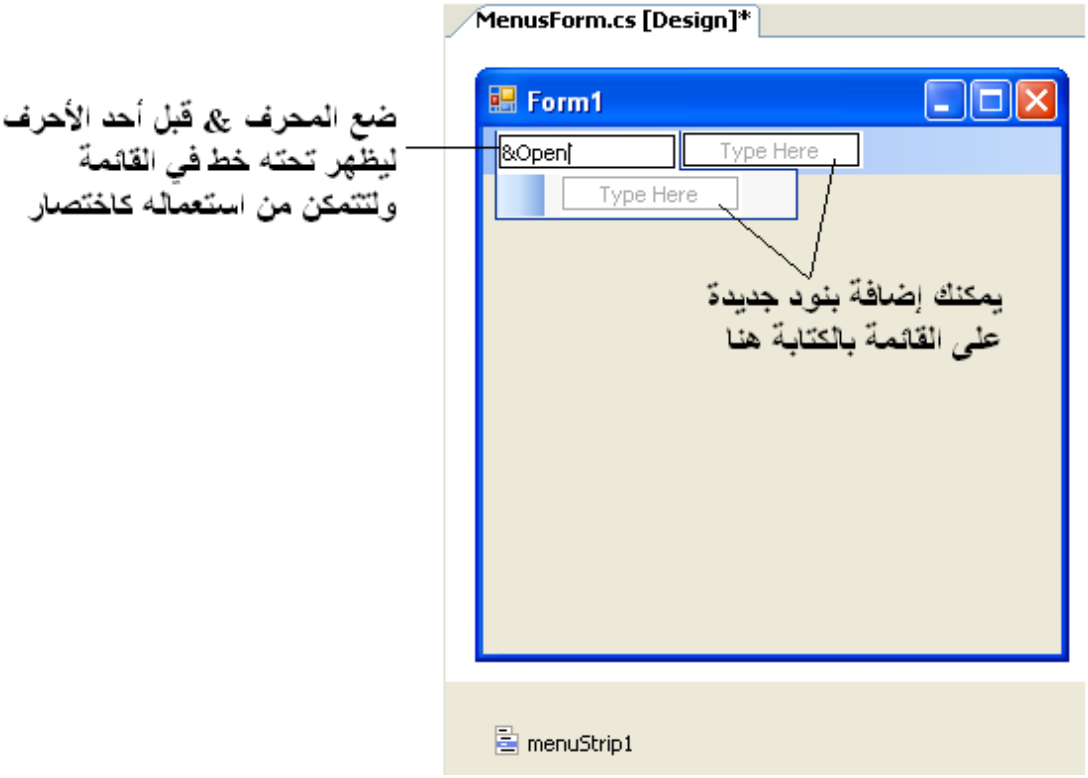
- توفر القوائم مجموعات من الأوامر المرتبطة في تطبيقات Windows.
- ومع أن هذه الأوامر تتبع البرنامج، إلا أن بعض الأوامر – مثل فتح Open وحفظ Save– مألوفة في العديد من التطبيقات.
- تُشكل القوائم جزءاً متكاملاً من واجهة المستخدم البيانية، لأنها ترتب الأوامر دون أن تزحم الواجهة بشكل مزعج.
- يوضح الشكل التالي قائمة ضمن محيط Visual Studio تحوي العديد من الأوامر (والتي تدعى بنود القائمة menu items)، بالإضافة إلى قوائم جزئية Submenus (قائمة ضمن قائمة)، لاحظ أن القوائم في المستوى الأعلى top–level تظهر في القسم العلوي الأيسر من الشكل، بينما تظهر القوائم الجزئية إلى يمينها.

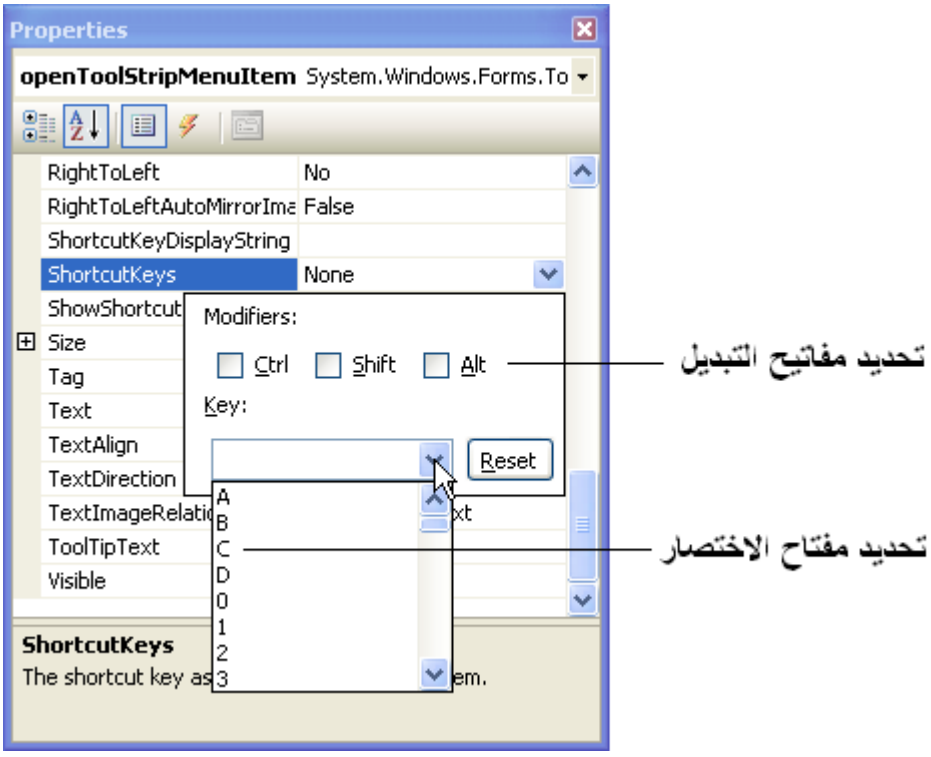


- تُدعى القائمة التي تحوي بنداً ما بالقائمة الأم parent menu لذلك البند.
- يُمكن لجميع البنود ضمن قائمة أن يكون لها اختصار باستعمال المفتاح Alt فيُمكن الوصول إليها بالضغط على المفتاح Alt والحرف المسطر تحته (مثلاً Alt+F للقائمة File).

- يُمكن أن يكون للقوائم التي ليست في المستوى الأعلى مفاتيح اختصار أيضاً (توليفات من المحارف Ctrl, Shift, Alt, F1, F2 والمحارف الحرفية).
- تعرض بعض بنود القائمة علامات اختيار Check Marks، مما يعني عادةً أنه يُمكن اختيار عدة بنود من القائمة في نفس الوقت.
- لإنشاء قائمة، نفتح صندوق الأدوات ToolBox ونسحب منه MenuStrip إلى النموذج، يؤدي ذلك إلى إنشاء شريط قوائم في أعلى النموذج (تحت شريط العنوان) و تظهر أيقونة MenuStrip في شريط العناصر أسفل النموذج. نقوم بالنقر على هذه الأيقونة لتحديد القائمة. يُمكن في وضع التصميم إنشاء وتحرير قوائم خاصة بالتطبيق. تملك القوائم –كغيرها من العناصر– أحداثاً وخصائص والتي يمكن الوصول إليها عن طريق نافذة الخصائص.

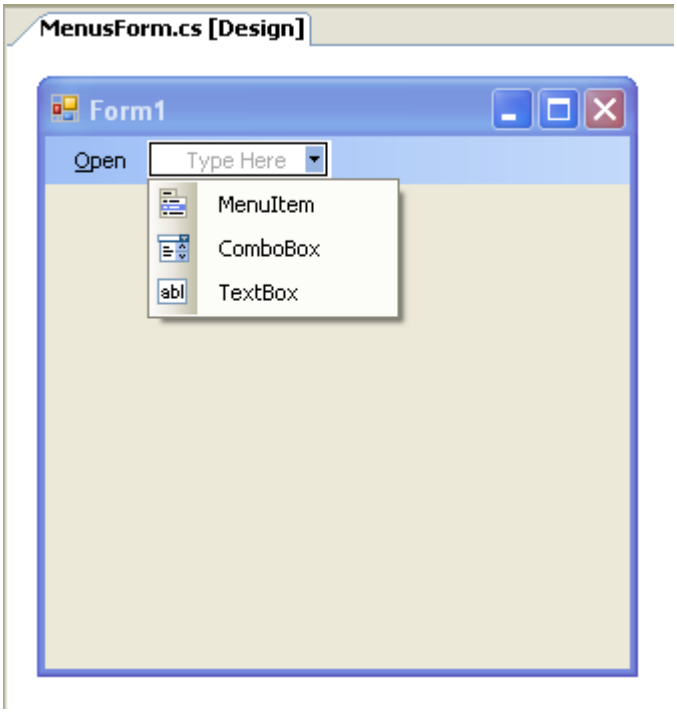
- ننقر على صندوق النص الذي يحوي النص Type Here لإضافة بند إلى القائمة (كما في الشكل) ونكتب اسم البند. يضيف هذا الفعل بنداً إلى القائمة من النمط ToolStripMenuItem، يتم إضافة البند بعد ضغط المحرف Enter، ثم يظهر المزيد من صناديق النص ذات النص Type Here تسمح بإضافة بنود تحت أو جانب البند الأصلي.





- لإنشاء اختصار وصول، نكتب الرمز & قبل المحرف الذي يجب أن يُسطّر. مثلاً، لإنشاء البند File مع تسطير الحرف F نكتب File&. ولإظهار الرمز & نفسه نكرره مرتين &&.

- لإضافة المزيد من مفاتيح الاختصار (مثل <Ctrl>+F9) نُحدّد قيمة الخاصية ShortcutKeys للبند المطلوب. عند النقر على السهم الذي يظهر إلى يمين هذه الخاصية في نافذة الخصائص تظهر النافذة المبينة في الشكل السابق. نستعمل صناديق الاختيار والقائمة المنسدلة في هذه النافذة لتحديد الاختصار المطلوب.
- يُمكن إخفاء مفاتيح الاختصار بجعل قيمة الخاصية ShowShortcutKeys تأخذ false. كما يُمكن تغيير طريقة عرض المفاتيح بتعديل الخاصية ShortcutKeyDisplayString.
- يُمكن حذف بند من قائمة عن طريق تحديده بالفأرة وضغط المفتاح Delete.
- يُمكن تجميع البنود منطقياً باستعمال أشرطة فاصلة Separator Bars، والتي يُمكن إضافتها بالنقر بالزر الأيمن على القائمة ثم اختيار Add Separator أو بكتابة (-) في مربع النص.
- بالإضافة إلى النص، يسمح Visual Studio بإضافة صناديق نص TextBoxes وقوائم منسدلة ComboBoxes كبنود ضمن القائمة. لاحظ أنه عند إضافة بند في وضع التصميم ظهور قائمة منسدلة قبل إدخال النص للبند الجديد. يسمح النقر على السهم باتجاه الأسفل باختيار نمط البند الذي نريد إضافته: بند قائمة MenuItem (من الصف ToolStripMenuItem) وهو الافتراضي، قائمة منسدلة (من الصف ToolStripComboBox)، مربع نص TextBox (من الصف ToolStripTextBox). لاحظ أنه إذا استعرضنا هذه البنود من أجل قائمة ليست من المستوى الأعلى سيظهر لنا خيار رابع هو الشريط الفاصل Separator Bar.



- تولد العناصر من النمط ToolStripMenuItem الحدث Click عند اختيارها. لإنشاء معالج حدث فارغ لهذا الحدث، ننقر نقراً مزدوجاً على البند في وضع التصميم.
- يعرض الجدول التالي أهم خصائص وأحداث القوائم:

الخاصية أو الحدث	الوصف
خصائص MenuStrip	
RightToLeft	تجعل النص يظهر من اليمين إلى اليسار، تفيد هذه الخاصية اللغات التي تُقرأ من اليمين إلى اليسار (كاللغة العربية).
خصائص ToolStripMenuItem	
Checked	تحدد فيما إذا كان البند مختاراً، القيمة الافتراضية هي false (غير مختار).
CheckOnClick	تحدد فيما إذا كان البند يجب أن يظهر مختاراً أو غير مختار بحسب النقر عليه.
ShortcutKeyDisplayString	النص الذي يظهر جانب العنصر عوضاً عن مفاتيح الاختصار.
ShortcutKeys	مفاتيح الاختصار.
ShowShortcutKeys	إظهار مفاتيح الاختصار أم لا.
Text	النص الذي يظهر على العنصر.
أحداث ToolStripMenuItem	
Click	يتولد عند النقر على العنصر أو عند استخدام مفاتيح الاختصار.

- يقوم الصف MenuTestForm في المثال التالي بإنشاء قائمة بسيطة ضمن نموذج. يحوي هذا النموذج في المستوى الأعلى القائمة File، والتي تحوي البندين About (يعرض رسالة) و Exit (الذي يغلق البرنامج). كما يحوي النموذج أيضاً القائمة Format و التي تحوي بنوداً لتعديل تنسيق نص ضمن لصاقة. تحوي القائمة Format قائمتين فرعيتين:Color و Font لتغيير لون و خط النص ضمن اللصاقة.
- لإنشاء هذه الواجهة، نقوم بسحب MenuStrip إلى النموذج ثم نستعمل وضع التصميم لإنشاء قائمة بالبنية الظاهرة في المثال.
- تحوي القائمة File (fileToolStripMenuItem) البندين About (aboutToolStripMenuItem) و Exit (exitToolStripMenuItem) ، كما تحوي القائمة

Format (formatToolStripMenuItem) قائمتين فرعيتين:

1. القائمة Color (colorToolStripMenuItem) والتي تحوي البنود:
   1.1. Black (blackToolStripMenuItem)
   1.2. Blue (blueToolStripMenuItem)
   1.3. Red (redToolStripMenuItem)
   1.4. Green (greenToolStripMenuItem)
2. القائمة Font التي تحوي البنود:
   1.5. Times New Roman (timesToolStripMenuItem)
   1.6. Courier (courierToolStripMenuItem)
   1.7. Comic San (comicToolStripMenuItem)
   1.8. شريط فاصل (dashToolStripMenuItem)
   1.9. Bold (boldToolStripMenuItem)
   1.10. Italic (italicToolStripMenuItem)

- يقوم البند About في القائمة File بعرض رسالة عند النقر عليه (الأسطر من 18 إلى 23). يُغلق البند Exit البرنامج باستدعاء الطريقة الساكنة Exit للصف Application (السطر 28). تتحكم الطرائق الساكنة static للصف Application بسير تنفيذ البرنامج، ويؤدي استدعاء الطريقة Exit إلى إنهاء التطبيق.
- جعلنا البنود في القائمة Color برمجياً (كما سنبيّن فيما يلي) في حالة استبعاد متبادل Mutually Execlusive، أي أن المستخدم يستطيع اختيار بند واحد فقط في نفس الوقت. للإشارة إلى اللون الحالي، نجعل للخاصية Checked للبند الموافق القيمة true، يؤدي ذلك إلى ظهور علامة الاختيار إلى يسار البند.
- يكون لكل بند في القائمة Color معالج للحدث Click (مثل blackToolStripMenuItem_Click للبند Black). وبما أننا نريد أن تكون الألوان في حالة استبعاد متبادل Mutually Execlusive، نقوم باستدعاء الطريقة ClearColor (الأسطر من 32 إلى 39) في كل معالج حدث قبل أن نجعل الخاصية Checked للبند الموافق القيمة true، تقوم هذه الطريقة بإسناد القيمة false إلى الخاصية Checked لجميع البنود في القائمة Color، مما يمنع اختيار أكثر من بند في نفس الوقت. كما أننا نعطي الخاصية Checked للبند Black القيمة true في زمن التصميم لأن لون النص يكون أسود عند بدء البرنامج.
- تحوي القائمة Font ثلاثة بنود لتحديد اسم الخط (Times New Roman, Courier, Comic San) وبندين لتحديد نمط الخط Style (Bold, Italic)، وقد أضفنا شريطاً فاصلاً بينهما للإشارة إلى أنها خيارات منفصلة.

- يُمكن اختيار خط واحد فقط، بينما يُمكن اختيار أكثر من نمط في نفس الوقت (يمكن للخط أن يكون عريضاً bold ومائلاً italic في نفس الوقت). نجعل هذه البنود قابلة للاختيار. كما نقوم بتطبيق الاستبعاد المتبادل على الخطوط فقط كما فعلنا في الألوان.
- تعمل معالجات الأحداث لبنود الخط بنفس الآلية التي تعمل بها مقابلاتها في بنود الألوان، وتستعمل الطريقة ClearFont (الأسطر من 86 إلى 92) لمسح الخاصية Checked لكل الخطوط، ونعطي الخاصية Checked للبند Times New Roman القيمة true في زمن التصميم لأنه هو خط النص عند بدء البرنامج.
- يستعمل كل من معالجي الحدث Click للبندين Bold و Italic (الأسطر من 132 إلى 154) العملية المنطقية XOR (^) لتكوين نمط الخط كما سبق وشرحنا في الفصل السابق.

```
1    // MenuTest.cs
2    // Using Menus to change font colors and styles.
3    using System;
4    using System.Drawing;
5    using System.Windows.Forms;
6
7    // our Form contains a Menu that changes the font color
8    // and style of the text displayed in Label
9    public partial class MenuTest : Form
10   {
11      // default constructor
12      public MenuTest()
13      {
14         InitializeComponent();
15      } // end constructor
16
17      // display MessageBox when About MenuItem is selected
18   private void aboutToolStripMenuItem_Click(object
sender,EventArgs e)
19 {
20 MessageBox.Show(
21      "This is an example\nof using menus.",
22      "About", MessageBoxButtons.OK, MessageBoxIcon.Information );
23      } // end method aboutMenuItem_Click
```

```
24
25      // exit program when Exit MenuItem is selected
26   private void exitToolStripMenuItem_Click(object sender,
EventArgs e)
27      {
28         Application.Exit();
29      } // end method exitMenuItem_Click
30
31      // reset checkmarks for Color MenuItems
32      private void ClearColor()
33      {
34         // clear all checkmarks
35         blackToolStripMenuItem.Checked = false;
36         blueToolStripMenuItem.Checked = false;
37         redToolStripMenuItem.Checked = false;
38         greenToolStripMenuItem.Checked = false;
39      } // end method ClearColor
40
41      // update Menu state and color display black
42  private void blackToolStripMenuItem_Click(object sender,
EventArgs e)
43      {
44         // reset checkmarks for Color MenuItems
45         ClearColor();
46
47         // set Color to Black
48         displayLabel.ForeColor = Color.Black;
49         blackToolStripMenuItem.Checked = true;
50      } // end method blackMenuItem_Click
51
52      // update Menu state and color display blue
53      private void blueToolStripMenuItem_Click(object sender,
EventArgs e)
54      {
55         // reset checkmarks for Color MenuItems
```

```
56          ClearColor();
57
58          // set Color to Blue
59          displayLabel.ForeColor = Color.Blue;
60          blueToolStripMenuItem.Checked = true;
61       } // end method blueMenuItem_Click
62
63       // update Menu state and color display red
64       private void redToolStripMenuItem_Click(object sender,
EventArgs e)
65       {
66          // reset checkmarks for Color MenuItems
67          ClearColor();
68
69          // set Color to Red
70          displayLabel.ForeColor = Color.Red;
71          redToolStripMenuItem.Checked = true;
72       } // end method redMenuItem_Click
73
74       // update Menu state and color display green
75       private void greenToolStripMenuItem_Click(object sender,
EventArgs e)
76       {
77          // reset checkmarks for Color MenuItems
78          ClearColor();
79
80          // set Color to Green
81          displayLabel.ForeColor = Color.Green;
82          greenToolStripMenuItem.Checked = true;
83       } // end method greenMenuItem_Click
84
85       // reset checkmarks for Font MenuItems
86       private void ClearFont()
87       {
88          // clear all checkmarks
```

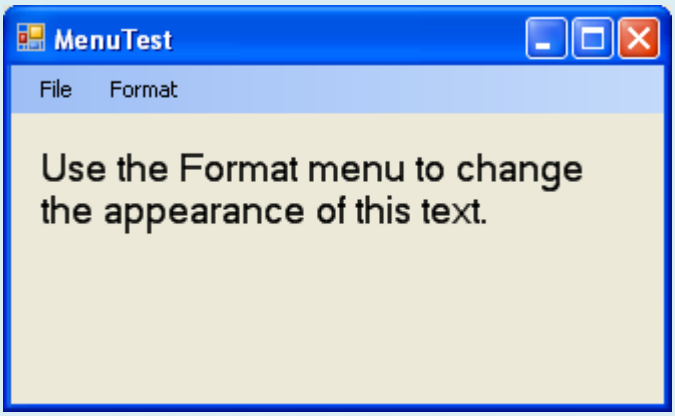
```
89          timesToolStripMenuItem.Checked = false;
90          courierToolStripMenuItem.Checked = false;
91          comicToolStripMenuItem.Checked = false;
92       } // end method ClearFont
93
94       // update Menu state and set Font to Times New Roman
95       private void timesToolStripMenuItem_Click(object sender,
EventArgs e)
96       {
97          // reset checkmarks for Font MenuItems
98          ClearFont();
99
100         // set Times New Roman font
101         timesToolStripMenuItem.Checked = true;
102         displayLabel.Font = new Font(
103            "Times New Roman", 14, displayLabel.Font.Style );
104      } // end method timesMenuItem_Click
105
106      // update Menu state and set Font to Courier
107      private void courierToolStripMenuItem_Click(
108          object sender, EventArgs e)
109      {
110         // reset checkmarks for Font MenuItems
111         ClearFont();
112
113         // set Courier font
114         courierToolStripMenuItem.Checked = true;
115         displayLabel.Font = new Font(
116            "Courier", 14, displayLabel.Font.Style );
117      } // end method courierMenuItem_Click
118
119      // update Menu state and set Font to Comic Sans MS
120      private void comicToolStripMenuItem_Click(object sender,
EventArgs e)
121      {
```

```
122         // reset checkmarks for Font MenuItems
123         ClearFont();
124
125         // set Comic Sans font
126         comicToolStripMenuItem.Checked = true;
127         displayLabel.Font = new Font(
128            "Comic Sans MS", 14, displayLabel.Font.Style );
129      } // end method comicMenuItem_Click
130
131      // toggle checkmark and toggle bold style
132      private void boldToolStripMenuItem_Click(object sender,
EventArgs e)
133      {
134         // toggle checkmark
135         boldToolStripMenuItem.Checked =
!boldToolStripMenuItem.Checked;
136
137         // use Xor to toggle bold, keep all other styles
138         displayLabel.Font = new Font(
139            displayLabel.Font.FontFamily, 14,
140            displayLabel.Font.Style ^ FontStyle.Bold );
141      } // end method boldMenuItem_Click
142
143      // toggle checkmark and toggle italic style
144      private void italicToolStripMenuItem_Click(
145          object sender, EventArgs e)
146      {
147         // toggle checkmark
148         italicToolStripMenuItem.Checked =
!italicToolStripMenuItem.Checked;
149
150         // use Xor to toggle italic, keep all other styles
151         displayLabel.Font = new Font(
152            displayLabel.Font.FontFamily, 14,
153            displayLabel.Font.Style ^ FontStyle.Italic );
```

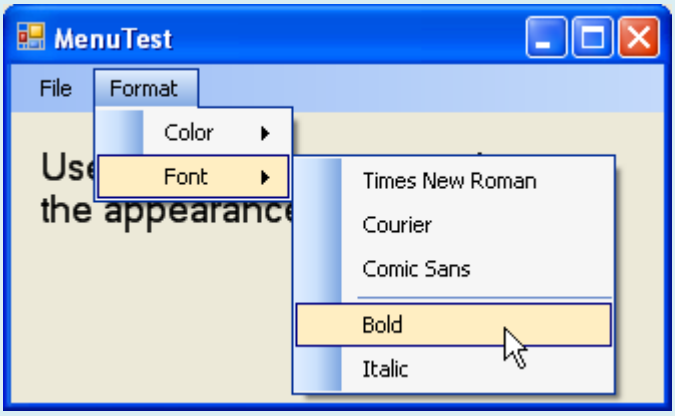
```
154     } // end method italicMenuItem_Click

155  } // end class MenuTest
```

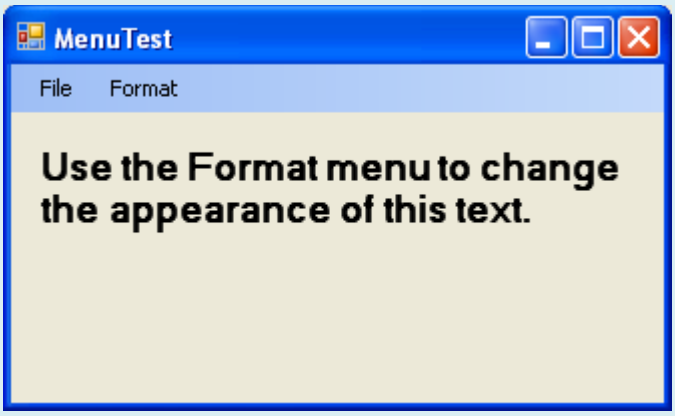
(a)



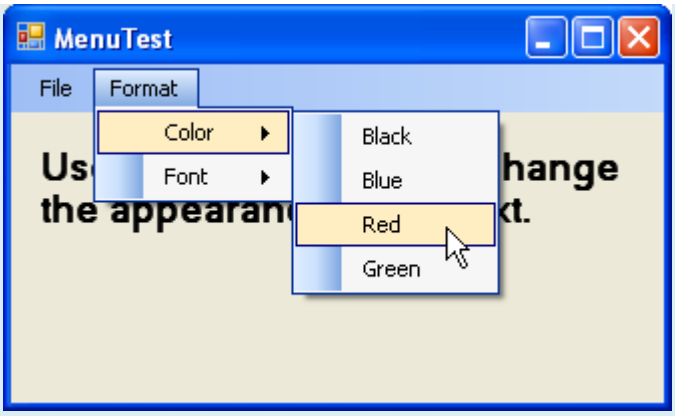
(b)



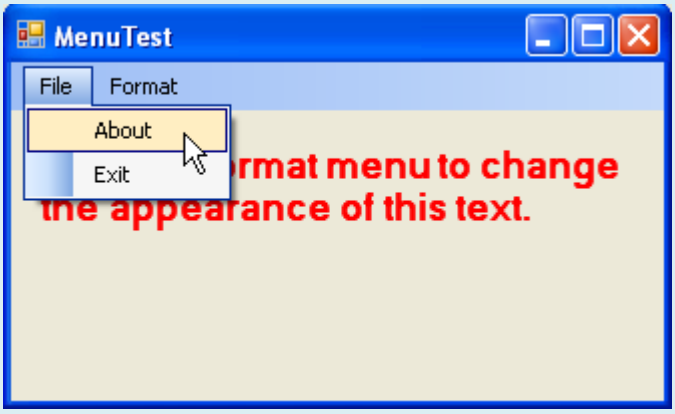
(c)



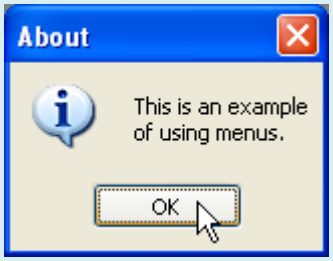
(d)


MenuTest
File
Format
Color
Font
Black
Blue
Red
Green

(e)


MenuTest
File
Format
About
Exit

(f)

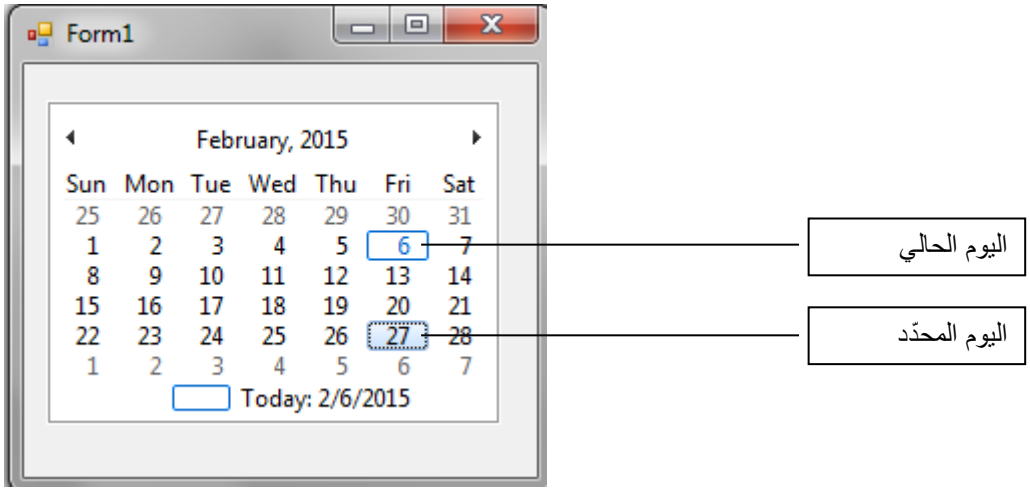

About
This is an example
of using menus.
OK

عناصر تحكم التاريخ والوقت

عنصر التحكم MonthCalender

- تحتاج العديد من التطبيقات إلى إجراء حسابات على الوقت والتاريخ.
- توفر منصة .NET عنصري تحكم يسمحان للبرنامج بالحصول على معلومات عن التاريخ والوقت: العنصر MonthCalender والعنصر DateTimePicker.
- يعرض العنصر MonthCalender تقويماً شهرياًعلى النموذج. يستطيع المستخدم اختيار تاريخ من التقويم، كما يستطيع استخدام الروابط المتوفرة ليتصفح أشهراً أخرى.
- يظهر التاريخ المحدد بلون مميز، ويُمكن تحديد أكثر من تاريخ عن طريق النقر على التقويم مع الضغط المستمر على الزر Shift.
- يكون الحدث الافتراضي لهذا العنصر هو DateChanged، والذي يتولد عندما يتم اختيار تاريخ جديد.
- تتوفر عدة خصائص للتحكم في مظهر التقويم والعدد الأعظمي للتواريخ التي يمكن تحديدها معاً والحد الأدنى والأعظم لهذه التواريخ.
- يُبين الجدول التالي أهم خصائص وأحداث العنصر MonthCalender:

الخاصية أو الحدث	الوصف
الخصائص	
FirstDayOfWeek	تحدد اليوم سيكون بداية كل أسبوع في التقويم.
MaxDate	آخر تاريخ يمكن تحديده.
MaxSelectionCount	الحد الأعظمي لعدد التواريخ التي يمكن تحديدها في نفس الوقت.
MinDate	أصغر تاريخ يمكن تحديده.
MonthlyBoldedDates	مصفوفة من التواريخ التي ستعرض باللون الغامق في التقويم.
SelectionEnd	آخر تاريخ من التواريخ التي حددها المستخدم.
SelectionRange	مجال التواريخ التي حددها المستخدم.
SelectionStart	أول تاريخ من التواريخ التي حددها المستخدم.
الأحداث	
DateChanged	يتولد عند تحديد تاريخ في التقويم.



عنصر التحكم DateTimePicker

- يشبه العنصر DateTimePicker العنصر MonthCalender، إلا أنه يُظهر التقويم عندما يقوم المستخدم بالنقر على الزر الذي يحوي سهماً موجهاً نحو الأسفل.
- يُمكن استعمال العنصر DateTimePicker للحصول على تاريخ أو توقيت من المستخدم. كما أنه يوفر إمكانيات أكبر لتخصيص مظهر التقويم.
- تُحدد الخاصية Format طريقة عرض خيار المستخدم عن طريق نمط التعداد DateTimePickerFormat والذي يُمكن أن يأخذ أحد القيم:
  - Long والذي يعني عرض التاريخ بصيغته الطويلة (Friday, July 1, 2005)
  - Short والذي يعني عرض التاريخ بصيغته القصيرة (7/1/2005)
  - Time والذي يعني عرض الوقت (11:48:02 PM)
  - Custom والذي يعني أن طريقة العرض سيتم تحديدها في الخاصية CustomFormat للعنصر.
- يكون الحدث الافتراضي لعنصر التحكم هذا هو ValueChanged، ويحدث عندما تتغير القيمة المحددة (التاريخ أو الوقت).
- يُبين الجدول التالي أهم خصائص العنصر DateTimePicker وأحداثه:

الخاصية أو الحدث	الوصف
الخصائص	
CalendarForeColor	تحديد لون خط نص التقويم.
CalendarMonthBackground	تحديد لون خلفية التقويم.
CustomFormat	تحديد تنسيق مخصص.
Format	تحديد تنسيق التاريخ والوقت.

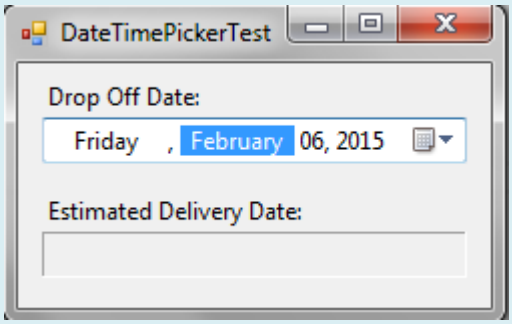
Maxdate	أكبر قيمة تاريخ.
MinDate	أصغر قيمة تاريخ.
ShowCheckBox	إظهار مربع تحقق أمام التاريخ المحدد.
ShowUpDown	إظهار الأسهم المساعدة في زيادة أو انقاص القيم.
Value	القيمة المحددة من قبل المستخدم.
الأحداث	
ValueChanged	يتولد الحدث عند تحديد قيمة.

- يقوم المستخدم في المثال التالي بتحديد تاريخ طلبية مثلاً، فيظهر له تاريخ التسليم المتوقع. يكون تاريخ التسليم بعد يومين ما لم تحوي الفترة يوم أحد فيصبح عندها تاريخ التسليم بعد ثلاثة أيام.

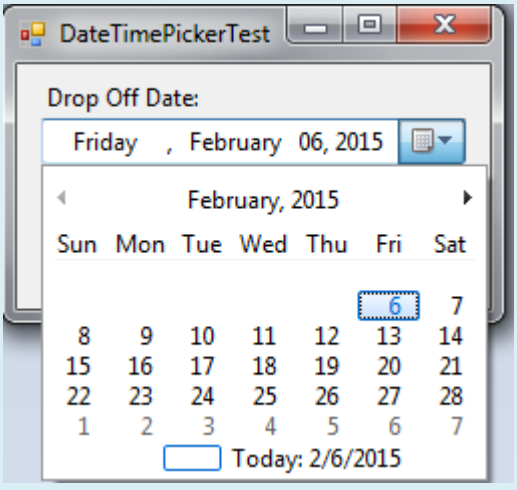
```
1 // Using a DateTimePicker
2 //to select a drop off time.
3 using System;
4 using System.Windows.Forms;
5
6 public partial class DateTimePickerForm : Form
7 {
8   // default constructor
9   public DateTimePickerForm()
10   {
11      InitializeComponent();
12   } // end constructor
13
14 private void dateTimePickerDropOff_ValueChanged( object sender,
EventArgs e )
15    {
16      DateTime dropOffDate = dateTimePickerDropOff.Value;
17
18      // add extra time when items are dropped off around Sunday
19      if ( dropOffDate.DayOfWeek == DayOfWeek.Friday ||
20         dropOffDate.DayOfWeek == DayOfWeek.Saturday ||
21         dropOffDate.DayOfWeek == DayOfWeek.Sunday )
22
```

```
23          //estimate three days for delivery
24     outputLabel.Text = dropOffDate.AddDays( 3 ).ToLongDateString();
25       else
26          // otherwise estimate only two days for delivery
27     outputLabel.Text = dropOffDate.AddDays( 2 ).ToLongDateString();
28    } // end method dateTimePickerDropOff_ValueChanged
29
30 private void DateTimePickerForm_Load( object sender, EventArgs e )
31    {
32       // user cannot select days before today
33       dateTimePickerDropOff.MinDate = DateTime.Today;
34
35       // user can only select days of this year
36       dateTimePickerDropOff.MaxDate = DateTime.Today.AddYears( 1 );
37    } // end method DateTimePickerForm_Load
38 } // end class DateTimePickerForm
```

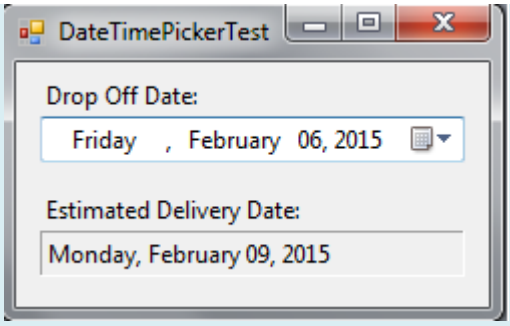
(a)



(b)



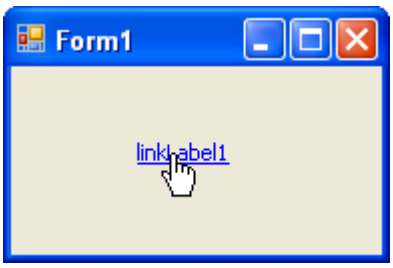
(c)


DateTimePickerTest
Drop Off Date:
Friday , February 06, 2015
Estimated Delivery Date:
Monday, February 09, 2015

لصاقة الرابط

عنصر التحكم LinkLabel

- يعرض عنصر التحكم LinkLabel روابط لمصادر أخرى، كالملفات وصفحات الوب.
- يظهر عنصر التحكم LinkLabel كنص تحته خط (لونه أزرق بشكل افتراضي)، وعندما نضع مؤشر الفأرة فوقه يتغير شكله ليصبح على شكل يد Hand، وهذا مشابه لتصرف الروابط في صفحة وب.



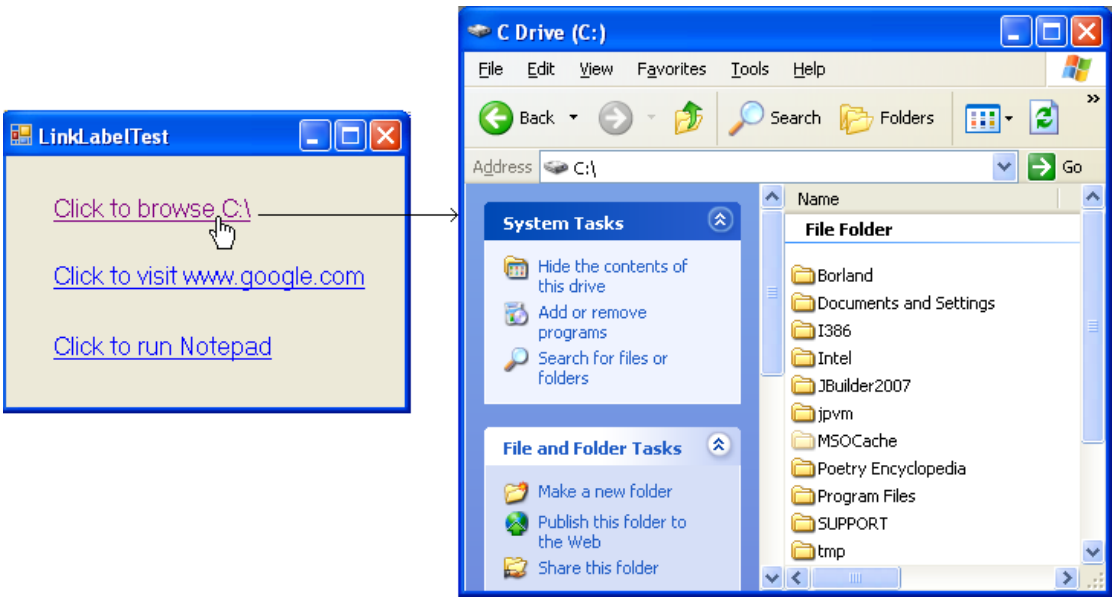
- يمكن للرابط أن يغير لونه ليوحي فيما إذ كان جديداً، أو تمت زيارته سابقاً، أو فعال.
- عندما يُنقر على LinkLabel يقوم بتوليد الحدث LinkClicked.
- يُشتق الصف LinkLabel من الصف Label لذا فهو يرث كل وظائفه.
- يُبين الجدول التالي أهم خصائص وأحداث العنصر LinkLabel:

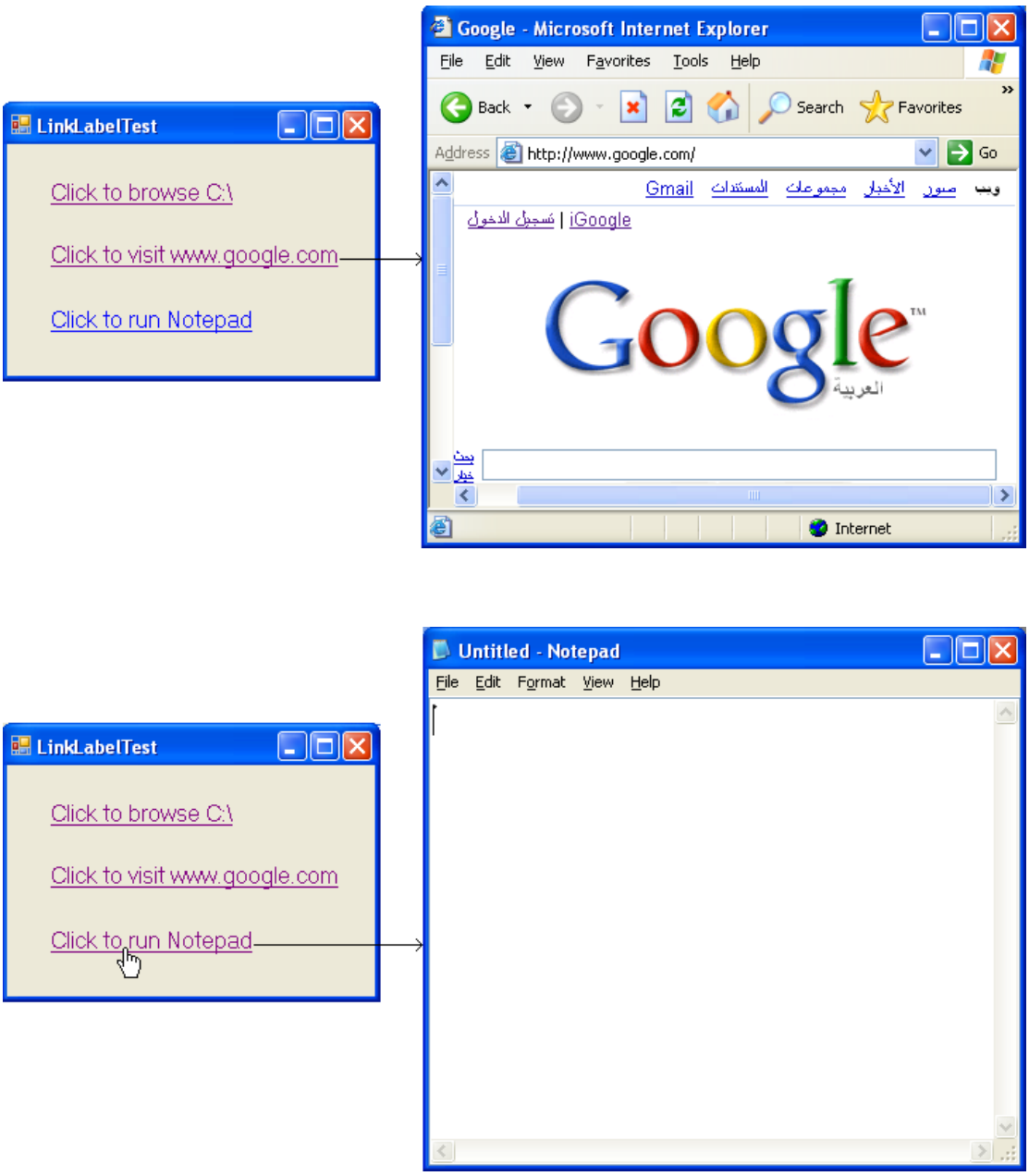
الخاصية أو الحدث	الوصف
الخصائص	
ActiveLinkColor	لون الرابط عند النقر (أحمر افتراضياً).
LinkArea	يُحدّد جزء النص ليكون جزء من الرابط.
LinkBehavior	يُحدّد مظهر الرابط عند توضع الفأرة فوقه.
LinkColor	لون الرابط (أزرق افتراضياً).
LinkVisited	يظهر الرابط كما لو أنه تمت زيارته إذا كانت قيمة الخاصية True.
Text	النص.
VisitedLinkColor	أول تاريخ من التواريخ التي حددها المستخدم.
الأحداث	
LinkClicked	يتولد عند نقر الرابط.

- يُبين المثال التالي ستخدام هذ العنصر مثلاً لاستعراض مجلد أو صفحة وب أو فتح تطبيق:

```
1  // LinkLabelTestForm.cs
2  // Using LinkLabels to create hyperlinks.
3  using System;
4  using System.Windows.Forms;
5
6  // Form using LinkLabels to browse the C:\ drive,
7  // load a webpage and run Notepad
8  public partial class LinkLabelTestForm : Form
9  {
10    // default constructor
11    public LinkLabelTestForm()
12    {
13       InitializeComponent();
14    } // end constructor
15
16    // browse C:\ drive
17    private void driveLinkLabel_LinkClicked( object sender,
18       LinkLabelLinkClickedEventArgs e )
19    {
20       // change LinkColor after it has been clicked
21       driveLinkLabel.LinkVisited = true;
22
23       System.Diagnostics.Process.Start( @"C:\" );
24    } // end method driveLinkLabel_LinkClicked
25
26    // load www.google.com in web browser
27    private void googleLinkLabel_LinkClicked( object sender,
28       LinkLabelLinkClickedEventArgs e )
29    {
30        // change LinkColor after it has been clicked
31       googleLinkLabel.LinkVisited = true;
32
33       System.Diagnostics.Process.Start(
```

```
34          "IExplore", "http://www.google.com" );
35    } // end method googleLinkLabel_LinkClicked
36
37    // run application Notepad
38    private void notepadLinkLabel_LinkClicked( object sender,
39       LinkLabelLinkClickedEventArgs e )
40    {
41        // change LinkColor after it has been clicked
42       notepadLinkLabel.LinkVisited = true;
43
44       // program called as if in run
45       // menu and full path not needed
46       System.Diagnostics.Process.Start( "notepad" );
47    } // end method driveLinkLabel_LinkClicked
48 } // end class LinkLabelTestForm
```





- تستدعي معالجات الأحداث لهذه العناصر الطريقة Start للصف Proccess (في فضاء الأسماء System.Diagnostics) والتي تسمح بتنفيذ برامج أخرى من التطبيق.
- تأخذ الطريقة Start وسيطاً واحداً من النمط string (سلسلة محرفية) ويمثل الملف المراد فتحه. أو وسيطين من النمط string، الأول هو التطبيق المراد فتحه، والثاني هو وسائط التطبيق command-line arguments. يُمكن لهذه الوسائط أن تكون بنفس الشكل المستخدم في الاستدعاء باستعمال الأمر Run في Windows. تتميز التطبيقات المعروفة في Windows بأنه يُمكن استدعاؤها بدون المسار الكامل full path، وغالباً ما يُمكن حذف اللاحقة exe.، ويمكن لفتح ملف معروف النمط في Windows أن نستعمل اسم الملف فقط.

- يقوم معالج الحدث LinkClicked الخاص بـ driveLinkLabel بفتح السواقة C: للتصفح (الأسطر من 17 إلى 24)، يُسند السطر 21 القيمة true للخاصية LinkVisited، مما يؤدي إلى تغيير لون الرابط من الأزرق إلى البنفسجي (يُمكن تحديد اللون باستخدام الخاصية VisetedLinkColor). يُمرر هذا الحدث الوسيط `@"C:\"` للطريقة Start (السطر 23)، والتي تقوم بفتح نافذة Windows Explorer. يدل الرمز `@` قبل `"C:\"` على أن جميع المحارف ضمن الشريط المحرفي يجب أن تُعتبر حرفياً. لذا، فلا يُعتبر المحرف `"\"` بداية سلسلة هروب escape sequence. تجعل هذه الطريقة السلاسل المحرفية التي تُمثّل مسارات ملفات أكثر وضوحاً لأنه لا داعي لاستعمال `"\\"` من أجل كل `"\"` في المسار.
- يقوم معالج الحدث LinkClicked الخاص بـ googleLinkLabel (الأسطر من 27 إلى 35) بفتح صفحة الإنترنت www.google.com ضمن Internet Explorer، ويتم إنجاز ذلك بتمرير عنوان الصفحة على شكل سلسلة محرفية (الأسطر 33،34). يُسند السطر 31 القيمة true للخاصية LinkVisited.
- يقوم معالج الحدث LinkClicked الخاص بـ notepadLinkLabel (الأسطر من 38 إلى 47) بفتح برنامج المفكرة Notepad. يُسند السطر 42 القيمة true للخاصية LinkVisited.

الفصل الخامس: عناصر التحكم (4)

عنوان الموضوع:

عناصر التحكم (4)

الكلمات المفتاحية:

List Box ، Checked List Box، Combo Box.

ملخص:

نستعرض في هذا الفصل مختلف أنواع القوائم.

أهداف تعليمية:

يتعرف الطالب في هذا الفصل على:

- صندوق القائمة.
- صندوق قائمة الخيارات.
- القائمة المنسدلة

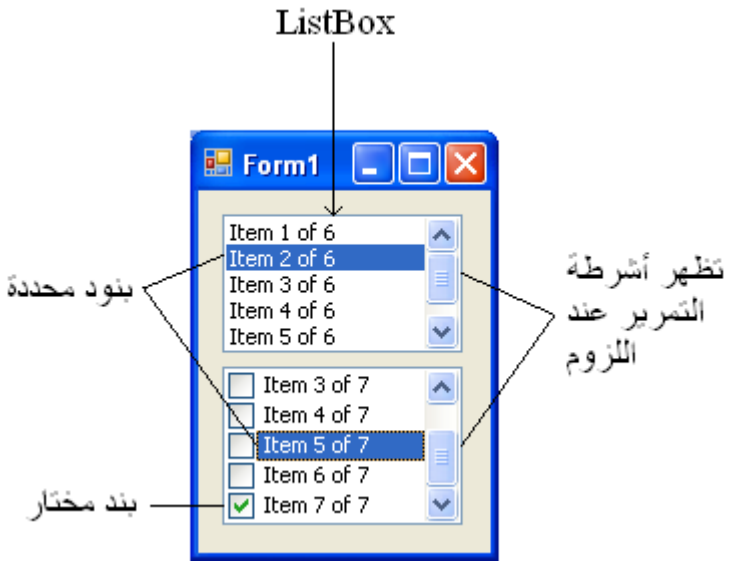
المخطط:

عناصر التحكم (4)

- 3 وحدات (Learning Objects)

عنصر التحكم ListBox

- يسمح عنصر التحكم صندوق القائمة ListBox للمستخدم أن يستعرض ويختار من قائمة من البنود. يُمكن إضافة البنود إلى القائمة (أو حذفها) في وضع التصميم، كما يُمكن إضافتها برمجياً في وضع التنفيذ. يُمكن عملياً أن نضع للمستخدم مجموعة من مربعات النص TextBoxes والأزرار ليقم بإضافة بنود إلى قائمة العنصر ListBox.
- يُعتبر عنصر التحكم صندوق قائمة الاختيار CheckedListBox امتداداً لعنصر صندوق القائمة حيث يضيف مربع اختيار بجانب كل بند.
- يُمكن للمستخدم أن يُحدد أكثر من بند بتغيير قيمة الخاصية SelectionMode.
- يُظهر الشكل التالي عنصري التحكم ListBox و CheckedListBox. تظهر أشرطة التمرير في كلا العنصرين عندما لا تكفي المساحة المعروضة لإظهار كافة البنود.



- يُظهر الجدول التالي أهم خصائص وأحداث وطرق عنصر التحكم ListBox.

الخاصية أو الحدث أو الطريقة	الوصف
الخصائص	
Items	تجميعة البنود في ListBox
MultiColumn	تُحدد فيما إذا كان من الممكن تقسيم قائمة البنود إلى أعمدة، يؤدي ذلك إلى اختفاء شريط التمرير العمودي.
SelectedIndex	تُعيد فهرس البند المحدد، وفي حال عدم تحديد أي بند تعيد القيمة -1، أما إذا حدد المستخدم أكثر من بند، فتُعيد هذه الخاصية فهرس واحد منها، في هذه الحالة نستعمل الخاصية SelectedIndices.
SelectedIndices	تُعيد تجميعة تحوي فهارس جميع البنود المحددة.
SelectedItem	تُعيد مؤشراً على البند المحدد، وتعيد البند ذو الفهرس الأدنى في حال تحديد أكثر من بند.
SelectedItems	تُعيد تجميعة تحوي جميع البنود المحددة.
SelectionMode	تُحدد عدد البنود التي يمكن تحديدها والآلية التي يمكن من خلالها تحديد أكثر من بند. تأخذ القيم None(لايُسمح بالتحديد)، One(تحديد بند واحد فقط)، MultiSimple (يسمح بتحديد أكثر من بند) و MultiExtended (يسمح بتحديد أكثر من بند باستعمال تركيبة من الأسهم أو نقرات الفأرة والمفاتيح Ctrl أو Shift).
Sorted	تُحدد فيما إذا كانت البنود مرتبة أبجدياً، القيمة الافتراضية هي false.
الطرائق	
ClearSelected	تلغي تحديد جميع البنود.
GetSelected	تأخذ فهرساً كوسيط وتعيد true إذا كان البند محدداً.
الأحداث	
SelectedIndexChanged	يتولد عندما يتغير فهرس البند المحدد، وهو الحدث الافتراضي عند النقر المزدوج على العنصر في وضع التصميم.

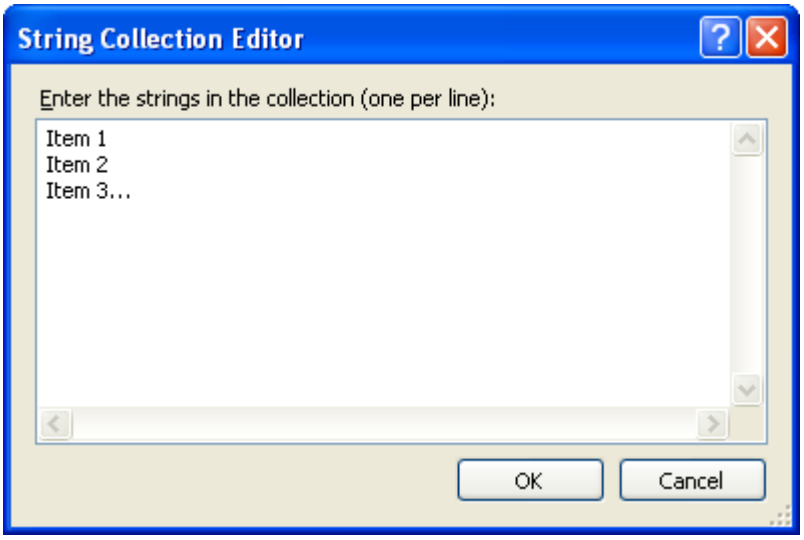
- تُحدد الخاصية SelectionMode عدد البنود التي يمكن تحديدها، وتأخذ إحدى القيم None, One, MultiSimple, MultiExtended (من نمط التعداد SelectionMode)، يوضح الجدول السابق الفروق بين هذه القيم. يتولد الحدث SelectedIndexChanged عندما يُحدد المستخدم بنداً جديداً.
- يكون لكلا العنصرين ListBox و CheckedListBox الخواص Items و SelectedItem و SelectedIndex.
- تُعيد الخاصية Items جميع البنود ضمن تجميعة.
- تُعتبر التجميعات Collections طريقة مألوفة لإدارة قوائم الأغراض في بيئة NET.. يستعمل العديد من عناصر الواجهة (مثل ListBox) التجميعات لتعريف قوائم من الأغراض الداخلية (مثل البنود ضمن ListBox). تكون التجميعة التي تعيدها الخاصية Items غرض من الصف ObjectCollection.
- تُعيد الخاصية SelectedItem البند المحدد في ListBox، وإذا حدد المستخدم أكثر من بند فيُمكن الحصول عليها جميعاً من الخاصية SelectedItems.
- تُعيد الخاصية SelectedIndex فهرس البند المحدد، وفي حال عدم تحديد أي بند تعيد القيمة -1. أما إذا حدد المستخدم أكثر من بند، فتُعيد هذه الخاصية فهرس واحد منها. في هذه الحالة يُمكن استخدام الخاصية SelectedIndices للحصول على جميع فهارس العناصر المحددة.
- تأخذ الطريقة GetSelected فهرساً كوسيط وتعيد true إذا كان البند الموافق محدداً.
- لإضافة بنود للعنصر ListBox أو العنصر CheckedListBox يجب إضافة أغراض للتجميعة Items الخاصة به، ويمكن إنجاز ذلك باستدعاء الطريقة Add لإضافة بند إلى التجميعة Items الخاصة بالعنصر ListBox أو العنصر CheckedListBox. يُمكن مثلاً أن نكتب:

```
myListBox.Items.Add(myListItem)
```

وذلك لنضيف السلسلة المحرفية myListItem إلى myListBox من الصف ListBox.

ولإضافة مجموعة من الأغراض، يُمكن استدعاء الطريقة Add عدة مرات أو استدعاء الطريقة AddRange والتي تُضيف مصفوفة من الأغراض.

- يقوم الصف ListBox و CheckedListBox باستدعاء الطريقة ToString للعنصر المضاف لتحديد النص الموافق له، مما يسمح بإضافة أغراض مختلفة ثم الحصول عليها من الخاصتين SelectedItem و SelectedItems.
- يُمكن إضافة بنود إلى ListBox أو CheckedListBox عن طريق الخاصية Items في نافذة الخصائص Properties Windows في وضع التصميم. يؤدي ضغط زر الانتقال بجانب الخاصية Items إلى فتح محرر تجميعة السلاسل المحرفية String Collection Editor، والتي تحوي مربع نص لإضافة البنود، حيث يظهر كل بند في سطر مستقل، ثم يقوم Visual Studio بتوليد الكود اللازم لإضافة هذه السلاسل المحرفية إلى التجميعة Items ضمن الطريقة InitializeComponent.

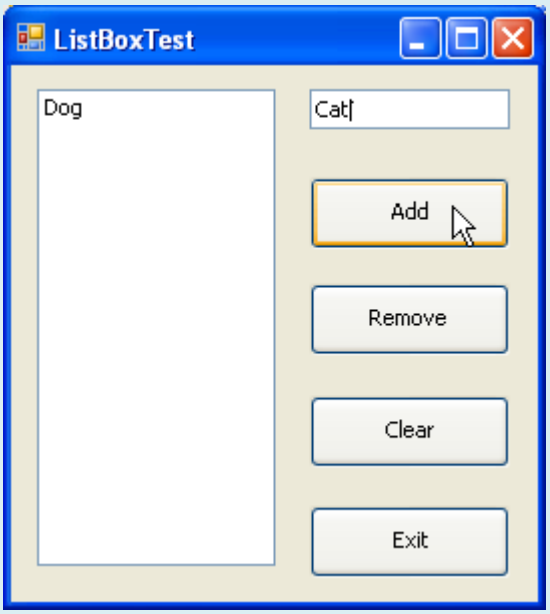


- يستعمل المثال التالي الصف ListBoxTestForm لإضافة وحذف ومسح البنود من ListBox المسماة displayListBox، يستعمل الصف ListBoxTestForm مربع النص inputTextBox ليتيح للمستخدم أن يكتب نص بند جديد. عندما ينقر المستخدم الزر Add يظهر بند جديد في displayListBox. كما يتم إزالة البند المحدد عند نقر الزر Remove. وعند النقر على الزر Clear يتم حذف جميع البنود من القائمة. ويُمكن للمستخدم إنهاء التطبيق بالنقر على الزر Exit.
- يستدعي معالج الحدث addButton_Click (الأسطر من 18 إلى 22) الطريقة Add للتجميعة Items، تأخذ هذه الطريقة سلسلة محرفية تُمثل البند المراد إضافته، والذي يوجد في inputTextBox.Text، ثم يتم مسح النص (السطر 21).
- يستدعي معالج الحدث addRemove_Click (الأسطر من 25 إلى 30) الطريقة RemoveAt لتحذف بنداً من القائمة، حيث يستعمل الخاصية SelectedIndex ليحدد فهرس العنصر المحدد، وإذا لم تكن قيمتها -1 (أي أنه يوجد بند محدد) فيقوم السطر 29 بحذف البند المحدد.

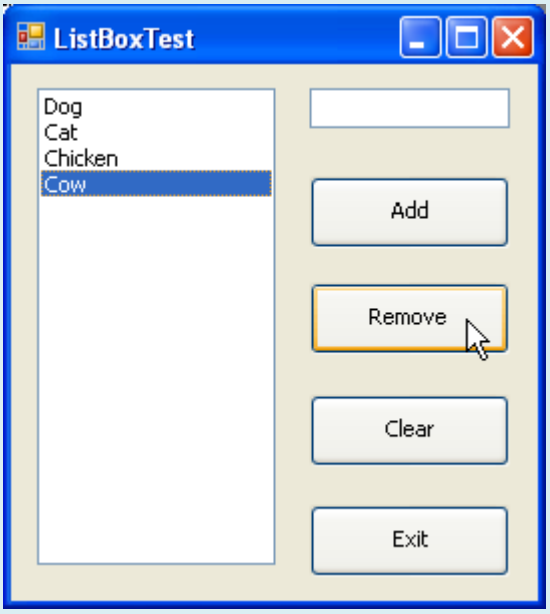
```
1  // ListBoxTestForm.cs
2  // Program to add, remove and clear ListBox items
3  using System;
4  using System.Windows.Forms;
5
6  // Form uses a TextBox and Buttons to add,
7  // remove, and clear ListBox items
8  public partial class ListBoxTestForm : Form
9  {
```

```
10    // default constructor
11    public ListBoxTestForm()
12    {
13       InitializeComponent();
14    } // end constructor
15
16    // add new item to ListBox (text from input TextBox)
17    // and clear input TextBox
18    private void addButton_Click( object sender, EventArgs e )
19    {
20       displayListBox.Items.Add( inputTextBox.Text );
21       inputTextBox.Clear();
22    } // end method addButton_Click
23
24    // remove item if one is selected
25    private void removeButton_Click( object sender, EventArgs e )
26    {
27       // check if item is selected, remove if selected
28       if ( displayListBox.SelectedIndex != -1 )
29      displayListBox.Items.RemoveAt(displayListBox.SelectedIndex);
30    } // end method removeButton_Click
31
32    // clear all items in ListBox
33    private void clearButton_Click( object sender, EventArgs e )
34    {
35       displayListBox.Items.Clear();
36    } // end method clearButton_Click
37
38    // exit application
39    private void exitButton_Click( object sender, EventArgs e )
40    {
41       Application.Exit();
42    } // end method exitButton_Click
43 } // end class ListBoxTestForm
```

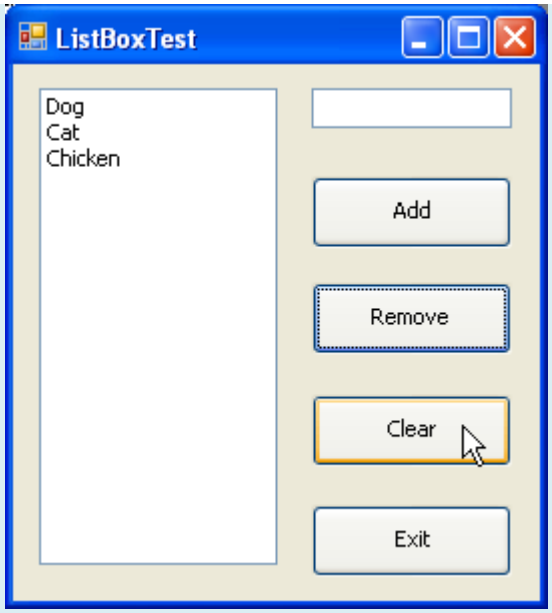
(a)


ListBoxTest
Dog
Cat
Add
Remove
Clear
Exit

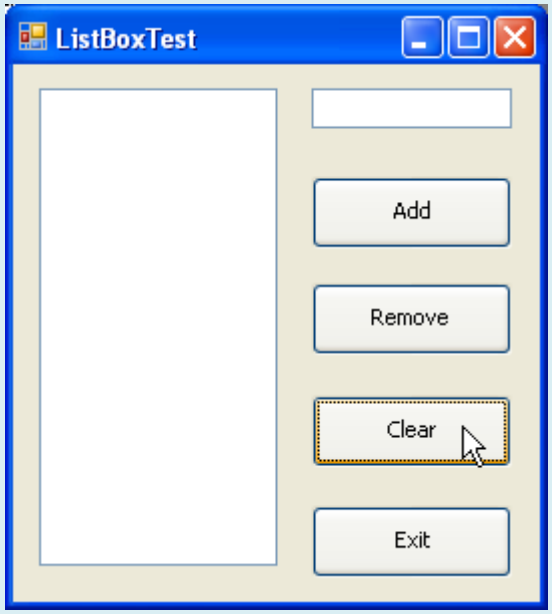
(b)


ListBoxTest
Dog
Cat
Chicken
Cow
Add
Remove
Clear
Exit

(c)



(d)



- يستدعي معالج الحدث addClear_Click (الأسطر من 33 إلى 36) الطريقة Clear للتجميعة Items (السطر 35)، والتي تقوم بحذف كل المدخلات من القائمة displayListBox.
- أخيراً يقوم معالج الحدث exitButton_Click (الأسطر من 39 إلى 42) بإنهاء التطبيق باستدعاء الطريقة Application.Exit (السطر 41).

عنصر التحكم CheckedListBox

- يُشتق الصف CheckedListBox من الصف ListBox، ويتميز بأنه يحوي مربع اختيار CheckBox بجانب كل بند. وكما في الصف ListBox، يُمكن إضافة بنود باستدعاء الطريقة Add أو AddRange برمجياً. أو باستعمال String Collection Editor في وضع التصميم. يسمح العنصر CheckedListBox بتحديد أكثر من بند. و تكون القيم الممكنة للخاصية SelectionMode هي None و One فقط. يسمح الخيار One بالاختيار المتعدد لأن مربعات الاختيار تضمن تعدد الخيارات بدون شروط. أما الخيار None فيعني عدم إمكانية اختيار أي بند من القائمة.
- يعرض الجدول التالي أهم خصائص وأحداث وطرق الصف CheckedListBox :

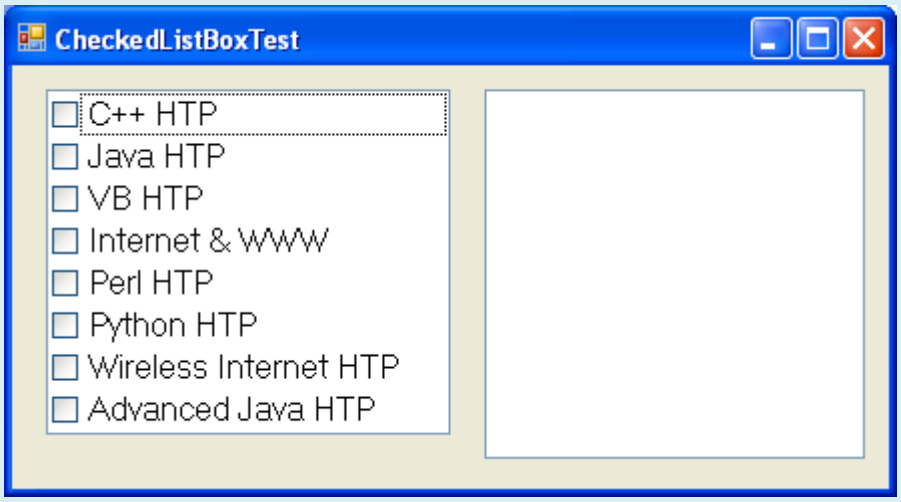
الخاصية أو الحدث أو الطريقة	الوصف
الخصائص (يرث الصف CheckedListBox كل خصائص وأحداث وطرق الصف ListBox)	
CheckedItems	تحوي تجميعة من البنود المختارة، وتختلف عن البند المحدد الذي يبدو مظللاً (ليس بالضرورة مختاراً)، مع ملاحظة أنه لا يمكن وجود أكثر من بند محدد في نفس الوقت.
CheckedIndices	تعيد فهارس كل البنود المختارة.
CheckOnClick	عندما تكون true، عندما ينقر المستخدم عنصر يُصبح هذا العنصر محدداً ويتم اختياره أو إزالة الاختيار عنه. القيمة الافتراضية لهذه الخاصية false مما يعني أن المستخدم عليه أولاً تحديد العنصر ومن ثم النقر عليه ثانيةً لاختياره أو لإزالة الاختيار عنه.
SelectionMode	تُحدد عدد البنود التي يمكن اختيارها. تأخذ أحد القيمتين One (يسمح باختيار أكثر من بند) أو None (لا يسمح باختيار أي بند).
الطرائق	
GetItemChecked	تأخذ فهرساً كوسيط وتعيد true إذا كان البند مختاراً.
الأحداث	
ItemCheck	يتولد عندما يتم اختيار بند أو إلغاء اختياره.
خصائص الصف ItemCheckEventArgs	
CurrentValue	تحدد حالة البند، القيم الممكنة هي Checked, Unchecked و Indeterminate.
Index	يعيد الفهرس (بدءاً من 0) للبند المتغير.
NewValue	تحدد الحالة الجديدة للبند.

- يحدث الحدث ItemCheck كلما قام المستخدم باختيار أو إلغاء اختيار بند ضمن القائمة CheckedListBox. يكون لهذا الحدث وسيطين هما CurrentValue و NewValue يُعيدان قيماً من النمط CheckState للحالة الحالية والجديدة على الترتيب. يُمكن عن طريق هاتين القيمتين تحديد فيما إذا كان البند مختاراً أم لا.
- ومع أن العنصر CheckedListBox يحافظ على الخاصتين SelectedItems و SelectedIndices (اللتان يرثهما من ListBox)، إلا أنه يملك أيضاً الخاصيتين CheckedItems و CheckedIndices واللتان تعيدان معلومات عن البنود المختارة وفهارسها.
- نستخدم في المثال التالي قائمة اختيارات CheckedListBox وقائمة ListBox لنُظهر خيارات المستخدم من مجموعة من الكتب. حيث يسمح العنصر CheckedListBox باختيار متعدد لعناوين الكتب. أضفنا في وضع التصميم وباستعمال String Collection Editor مجموعة من العناوين. يقوم العنصر ListBox بعرض خيارات المستخدم التي يقوم بها من خلال قائمة الاختيار.

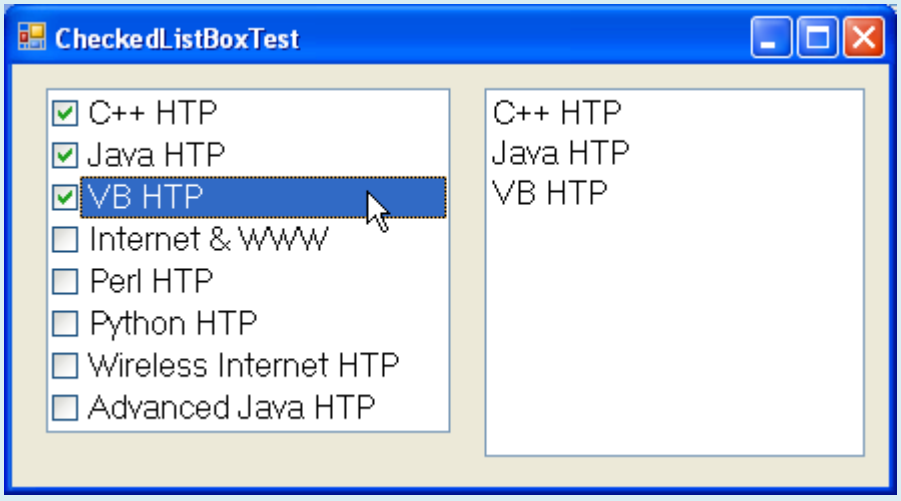
```
1  // CheckedListBoxTestForm.cs
2  // Using the checked ListBox to add items to a display ListBox
3  using System;
4  using System.Windows.Forms;
5
6  //Form uses a checked ListBox to add items to a display ListBox
7  public partial class CheckedListBoxTestForm : Form
8  {
9     // default constructor
10    public CheckedListBoxTestForm()
11    {
12       InitializeComponent();
13    } // end constructor
14
15    // item about to change
16    // add or remove from display ListBox
17    private void inputCheckedListBox_ItemCheck(
18       object sender, ItemCheckEventArgs e )
19    {
20       // obtain reference of selected item
21      string item = inputCheckedListBox.SelectedItem.ToString();
```

```
22
23       // if item checked add to ListBox
24       // otherwise remove from ListBox
25       if ( e.NewValue == CheckState.Checked )
26          displayListBox.Items.Add( item );
27       else
28          displayListBox.Items.Remove( item );
29    } // end method inputCheckedListBox_ItemCheck
30 } // end class CheckedListBoxTestForm
```

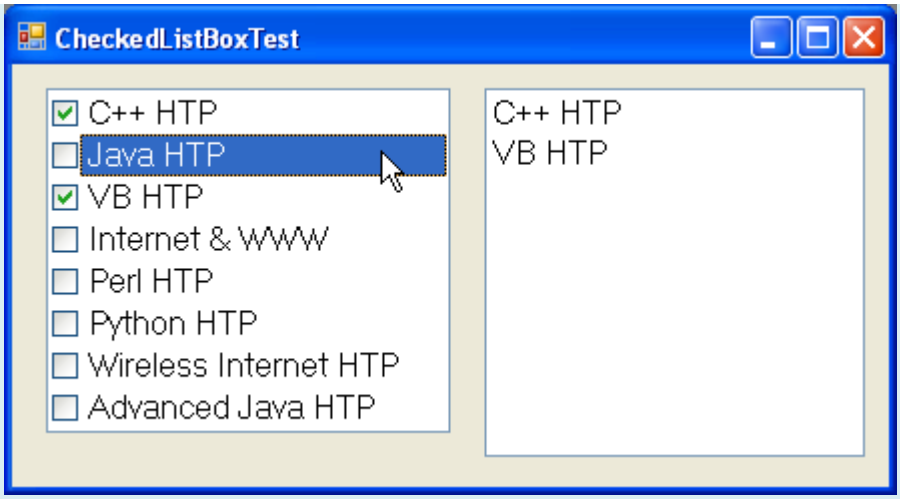
(a)



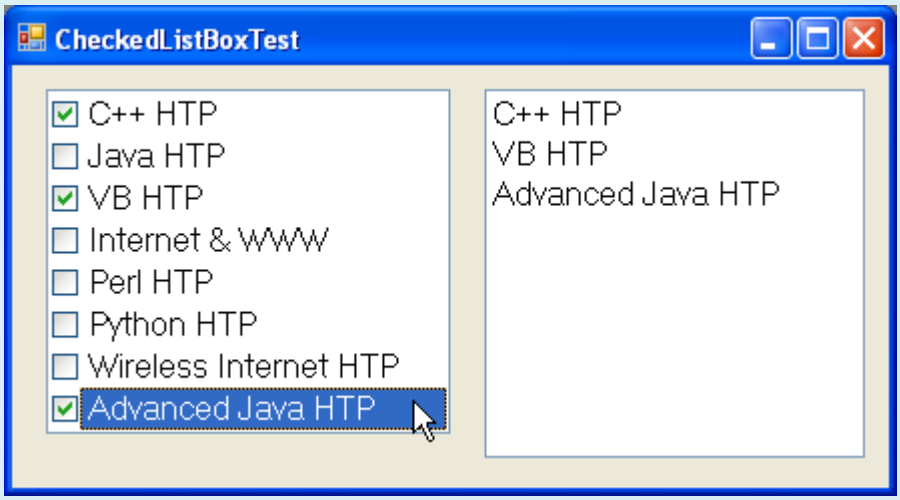
(b)



(c)



(d)



- عندما يقوم المستخدم باختيار أو إلغاء اختيار بند من inputCheckedListBox يتولد الحدث ItemCheck ويتم تنفيذ المعالج (من السطر 17 إلى 29)

inputCheckedListBox_ItemCheck، تُحدد العبارة الشرطية if...else (الأسطر 25 إلى 28) فيما إذا كان المستخدم قد اختار أو ألغى اختيار بنداً من inputCheckedListBox، يستعمل السطر 25 الخاصية NewValue ليحدد فيما إذا كان يتم اختيار بند (CheckState.Checked)، إذا قام المستخدم باختيار بند ما فسيقوم السطر 26 بإضافة هذا البند إلى displayListBox، أما إذا قام بإلغاء اختيار بند ما فسيقوم السطر 28 بحذف البند الموافق من displayListBox.

عنصر التحكم ComboBox

- يحوي عنصر التحكم ComboBox (القائمة المنسدلة) على قائمة من القيم التي يُمكن اختيار أحدها، ويظهر ComboBox عادةً على شكل مربع نص إلى يمينه سهم.
- يُمكن للمستخدم بشكل افتراضي كتابة نص في مربع النص أو النقر على زر السهم لعرض قائمة من البنود مسبقة التعريف. وإذا اختار المستخدم أحدها يظهر نصه في مربع النص. كما يظهر شريط تمرير إذا احتوت القائمة عدداً من البنود أكبر مما يُمكن عرضه. يُمكن تحديد العدد الأعظمي للبنود التي يمكن عرضها بتحديد قيمة الخاصية MaxDropDownItems. يُظهر الشكل التالي مثالاً عن ComboBox في ثلاث حالات مختلفة.

- يُمكن –كما في عنصر القائمة ListBox– إضافة أغراض إلى التجميعة Items برمجياً باستدعاء الطريقتين Add و AddRange، أو باستعمال String Collection Editor في وضع التصميم.
- يُظهر الجدول التالي أهم خصائص وأحداث الصف ComboBox:

الخاصية أو الحدث أو الطريقة	الوصف
الخصائص	
DropDownStyle	تُحدد نمط ComboBox، تعني القيمة Simple أن النص في مربع النص قابل للتعديل وأن القائمة ظاهرة بشكل دائم، وتعني القيمة DropDown (الافتراضية) أن النص قابل للتعديل إلا أنه يجب النقر على الزر لإظهار القائمة، وتعني القيمة DropDownList أن النص غير قابل للتعديل ويجب النقر على الزر لإظهار القائمة.
Items	تجميعة البنود في ComboBox.
MaxDropDownItems	تُحدد العدد الأعظمي (بين 1 و100) للبنود التي يُمكن عرضها

	في القائمة المنسدلة، وإذا تجاوز عدد البنود في القائمة هذه القيمة يظهر شريط تمرير.
SelectedIndex	تُعيد فهرس البند المحدد، وإذا لم يكن هناك بند محدد فتعيد 1-.
SelectedItem	تُعيد البند المحدد.
Sorted	تحدد فيما إذا كانت البنود مرتبة أبجدياً، القيمة الافتراضية هي false.
الأحداث	
SelectedIndexChanged	يتولد عندما يتغير فهرس البند المحدد، وهو الحدث الافتراضي عند النقر المزدوج على العنصر في وضع التصميم.

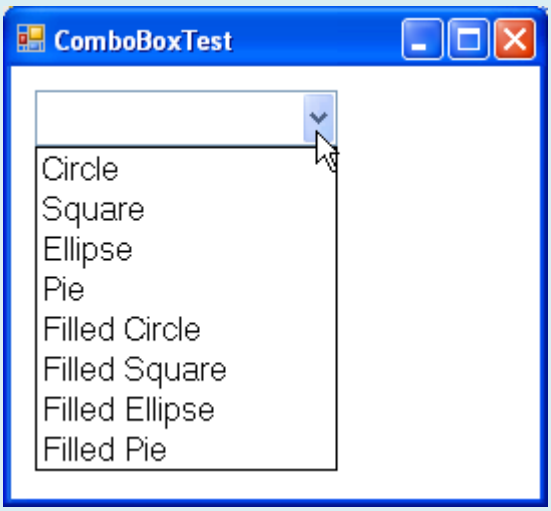
- تُحدد الخاصية DropDownStyle نوع القائمة المنسدلة ComboBox، وتأخذ قيمة من نمط التعداد ComboBoxStyle، والذي يحوي القيم Simple, DropDown, DropDownList.
- يؤدي الخيار Simple إلى عدم عرض زر القائمة المنسدلة، ويظهر بدلاً عنه شريط تمرير بجانب العنصر ليسمح للمستخدم أن يختار من القائمة، كما يمكن للمستخدم أن يكتب نصاً. يعرض الخيار DropDown (وهو الافتراضي) القائمة المنسدلة عند الضغط على الزر (أو ضغط مفتاح السهم نحو الأسفل)، كما يُمكن للمستخدم أن يكتب نصاً في مربع النص. يعرض الخيار الأخير DropDownList قائمة منسدلة إلا أنه لا يسمح للمستخدم أن يكتب في مربع النص.
- يملك العنصر ComboBox الخصائص Items, SelectedItem, SelectedIndex وهي مشابهة لمقابلاتها في ListBox. يُمكن على الأكثر اختيار بند واحد في ComboBox، وعند عدم اختيار أي بند تكون قيمة SelectedIndex مساوية 1-، كما يتم توليد الحدث SelectedIndexChanged عند تغيير البند المحدد.
- يسمح المثال التالي للمستخدم باختيار شكل لرسمه: دائرة، قطع ناقص، مربع، فطيرة، في نسخها المملوءة وغير المملوءة، وذلك باستعمال ComboBox. لا يُمكن للمستخدم في مثالنا الكتابة في مربع النص لأن العنصر ComboBox غير قابل للتعديل.

```
1  // ComboBoxTestForm.cs
2  // Using ComboBox to select a shape to draw.
3  using System;
4  using System.Drawing;
5  using System.Windows.Forms;
6
7  // Form uses a ComboBox to select different shapes to draw
```

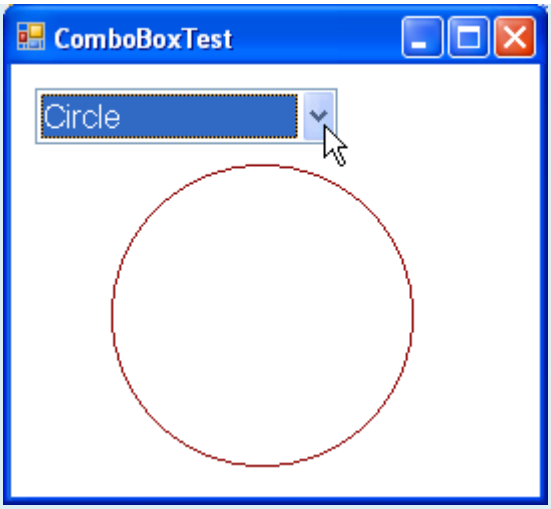
```
8  public partial class ComboBoxTestForm : Form
9  {
10    // default constructor
11    public ComboBoxTestForm()
12    {
13       InitializeComponent();
14    } // end constructor
15
16    // get index of selected shape, draw shape
17    private void imageComboBox_SelectedIndexChanged(
18       object sender, EventArgs e )
19    {
20       // create graphics object, Pen and SolidBrush
21       Graphics myGraphics = base.CreateGraphics();
22
23       // create Pen using color DarkRed
24       Pen myPen = new Pen( Color.DarkRed );
25
26       // create SolidBrush using color DarkRed
27       SolidBrush mySolidBrush = new SolidBrush( Color.DarkRed );
28
29       // clear drawing area setting it to color white
30       myGraphics.Clear( Color.White );
31
32       // find index, draw proper shape
33       switch ( imageComboBox.SelectedIndex )
34       {
35          case 0: // case Circle is selected
36             myGraphics.DrawEllipse( myPen, 50, 50, 150, 150 );
37             break;
38          case 1: // case Rectangle is selected
39             myGraphics.DrawRectangle( myPen, 50, 50, 150, 150 );
40             break;
41          case 2: // case Ellipse is selected
```

```
42              myGraphics.DrawEllipse( myPen, 50, 85, 150, 115 );
43              break;
44           case 3: // case Pie is selected
45              myGraphics.DrawPie( myPen, 50, 50, 150, 150, 0, 45 );
46              break;
47           case 4: // case Filled Circle is selected
48            myGraphics.FillEllipse(mySolidBrush, 50, 50, 150, 150);
49            break;
50           case 5: // case Filled Rectangle is selected
51            myGraphics.FillRectangle(mySolidBrush, 50, 50, 150,150);
52            break;
53           case 6: // case Filled Ellipse is selected
54            myGraphics.FillEllipse(mySolidBrush, 50, 85, 150, 115);
55            break;
56           case 7: // case Filled Pie is selected
57            myGraphics.FillPie(mySolidBrush,50,50, 150, 150, 0, 45);
58            break;
59        } // end switch
60
61        myGraphics.Dispose(); // release the Graphics object
62     } // end method imageComboBox_SelectedIndexChanged
63 } // end class ComboBoxTestForm
```

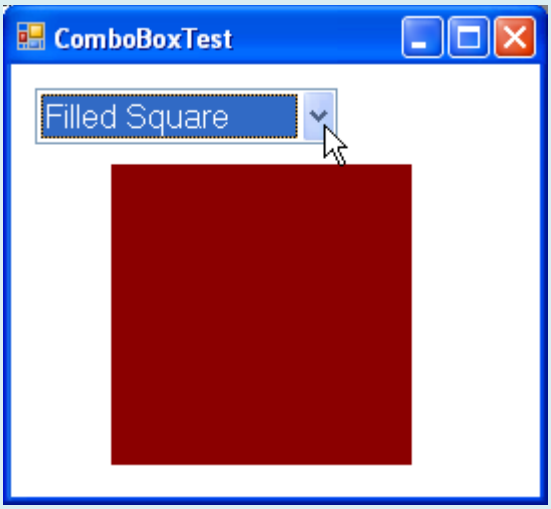
(a)



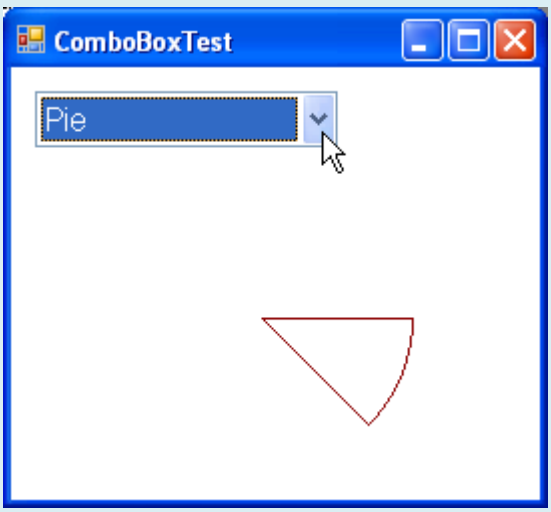
(b)


ComboBoxTest
Circle

(c)


ComboBoxTest
Filled Square

(d)


ComboBoxTest
Pie

- بعد إنشاء قائمة منسدلة ComboBox باسم imageComboBox، نجعلها غير قابلة للتعديل بإسناد القيمة DropDownList للخاصية DropDownStyle في نافذة الخصائص، ثم نضيف البنود باستعمال String Collection Editor في وضع التصميم.
- كلما اختار المستخدم بنداً من بنود imageComboBox يُرفع الحدث SelectedIndexChanged وينفذ معالج الحدث (الأسطر من 17 إلى 60) imageComboBox_SelectedIndexChanged. تقوم الأسطر من 21 إلى 27 بإنشاء غرض من الصف Graphics وقلم Pen وفرشاة SolidBrush، والتي ستُستعمل للرسم على النموذج Form. يسمح الغرض من الصف Graphics (السطر 21) باستعمال القلم والفرشاة لرسم عدة أشكال باستدعاء العديد من التوابع. يُستعمل القلم (السطر 24) في استدعاء الطرق DrawEllipse, DrawRectangle, DrawPie في الأسطر (36, 39, 42, 45) لرسم حدود هذه الأشكال، وتُستعمل الفرشاة (السطر 27) في استدعاء الطرق FillEllipse, FillRectangle, FillPie في الأسطر (48, 51, 54, 57) لملء هذه الأشكال، يلون السطر 30 النموذج كله باللون الأبيض باستعمال الطريقة Clear.
- يرسم التطبيق الشكل بالاعتماد على فهرس البند المحدد في القائمة، حيث تستعمل عبارة switch (الأسطر من 33 إلى 59) قيمة imageComboBox.SelectedIndex لتُحدد البند الذي اختاره المستخدم.
- تأخذ الطريقة DrawEllipse (السطر 36) قلماً وإحداثيي مركز القطع الناقص (X,Y) وعرضه وارتفاعه، حيث أن مركز الإحداثيات هو الزاوية العليا اليسرى من النموذج، يزداد الإحداثي X باتجاه اليمين والإحداثي Y باتجاه الأسفل. تكون الدائرة حالة خاصة من القطع الناقص (يتساوى فيها العرض والارتفاع). يرسم السطر 36 دائرة، ويرسم السطر 42 قطعاً ناقصاً بعرض وارتفاع مختلفين.
- تأخذ الطريقة DrawRectangle (السطر 39) قلماً وإحداثيي الزاوية العليا–اليسرى للمستطيل (X,Y) وعرضه وارتفاعه.
- ترسم الطريقة DrawPie (السطر 45) فطيرة على أنها جزء من قطع ناقص، ويُحدد القطع الناقص بمستطيل، حيث تأخذ الطريقة DrawPie قلماً وإحداثيي الزاوية العليا–اليسرى للمستطيل (X,Y) وعرضه وارتفاعه، وزاوية البدء (مقدرة بالدرجات) وزاوية المسح (مقدرة بالدرجات)، حيث تتزايد الزوايا باتجاه عقارب الساعة.
- تشبه الطرائق FillEllipse (الأسطر من 48 إلى 54)، FillRectangle (السطر 51) وFillPie (السطر 57) مقابلاتها التي لا تملأ الشكل، إلا أنها تأخذ فرشاة بدل قلم، يوضح الشكل بعض هذه الرسوم.

الفصل السادس: عناصر التحكم (5)

عنوان الموضوع:

عناصر التحكم (5)

الكلمات المفتاحية:

TreeView ، ListView، TabControl.

ملخص:

نستعرض في هذا الفصل عناصر التحكم المتقدمة: عرض الشجرة، عرض القائمة، التبويب.

أهداف تعليمية:

يتعرف الطالب في هذا الفصل على:

- عرض الشجرة TreeView.
- عرض القائمة ListView.
- التبويب TabControl.

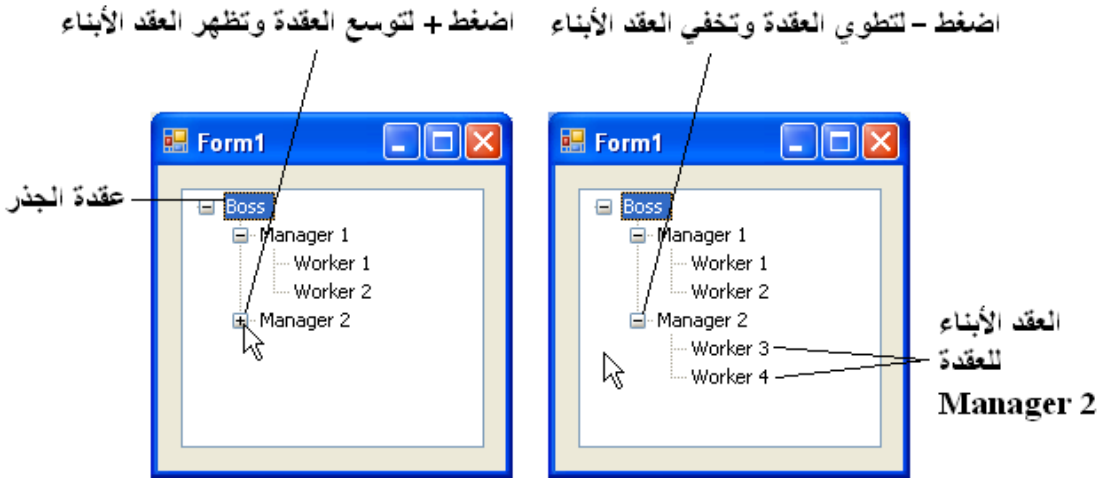
المخطط:

عناصر التحكم (5)

- 3 وحدات (Learning Objects)

عنصر التحكم TreeView

- يُستخدم العنصر TreeView لعرض مجموعة من العقد ذات تنظيم هرمي على شكل شجرة. تكون العقد عبارة عن أغراض يُمكن أن تحوي قيماً وتشير إلى عقد أخرى عادةً. تحوي العقدة الأب Parent Node على مجموعة من العقد الأبناء Child Nodes، والتي يُمكن أن تكون كل منها بدورها أباً لعقد أخرى، وهكذا... . تُدعى العقد التي لها نفس العقدة الأب بالعقد الإخوة Sibling Nodes.
- تكون الشجرة عبارة عن تجميعة collection من العقد تُرتب عادةً بتسلسل هرمي. تُدعى العقدة الأب الأولى في الشجرة بالعقدة الجذر Root Node. فعلى سبيل المثال، يُمكن تمثيل نظام الملفات الخاص بحاسوب ما على شكل شجرة. تُشكّل المجلدات في المستويات العليا (مثل C:) الجذر، كما يُشكّل كل مجلد فرعي من C: عقدة ابن، وكل منها يُمكن أن يكون له أبنائه.
- يفيد العنصر TreeView إذاً لعرض المعلومات بتسلسل هرمي، مثل بنية الملفات مثلاً. يوضح الشكل التالي مثالاً للعنصر TreeView.



- يُمكن توسيع expand و طي collapse عقدة ما بالنقر على المربع الذي يحوي رمز + أو – إلى يسار العقدة، لا تملك العقد التي لا أبناء لها مثل هذا المربع.
- تكون العقد ضمن العنصر TreeView عبارة عن منتسخات من الصف TreeNode. يكون لكل عقدة TreeNode التجميعة Nodes الخاصة بها (من النمط TreeNodeCollection)، وهي تحوي قائمة بالعقد الأبناء. تُعيد الخاصية Parent مؤشراً للعقدة الأب (أو null في حالة العقدة الجذر).
- يوضح الجدول التالي أهم خصائص وأحداث وطرائق الصف TreeView:

الخاصية أو الحدث	الوصف
الخصائص	
CheckBoxes	تُحدّد فيما إذا كان هناك صناديق اختيار بجانب العقد، القيمة الافتراضية هي false (لا تظهر صناديق الاختيار).
ImageList	تُحدّد غرضاً من النمط ImageList يحوي أيقونات العقد، ويحوي الغرض ImageList على أغراض من الصف Image.
Nodes	تُعيد تجميعة من TreeNode الموجودة ضمن العنصر، وتحوي الطريقة Add (لإضافة TreeNode) و Clear (لحذف جميع العقد) و Remove (لحذف عقدة معينة)، إن حذف عقدة يؤدي إلى حذف جميع أبنائها.
SelectedNode	العقدة المحددة.
الحدث (وسيط الحدث من النمط TreeViewEventArgs)	
AfterSelect	يتولد عند تغيير العقدة المحددة، وهو الحدث الافتراضي عند النقر المزدوج على العنصر في زمن التصميم.

- يوضح الجدول التالي أهم خصائص وأحداث وطرائق الصف TreeNode:

الخاصية أو الحدث	الوصف
الخصائص	
Checked	تُحدّد فيما إذا كانت العقدة مختارة (يجب أن تكون الخاصية CheckBoxes تحمل القيمة true في الشجرة الحاوية للعقدة).
FirstNode	تحدد أول عقدة في الخاصية Nodes (أول عقدة فرعية في الشجرة).
FullPath	مسار العقدة بدءاً من جذر الشجرة.
ImageIndex	فهرس الصورة عندما تكون العقدة غير محددة.
LastNode	تحدد آخر عقدة في الخاصية Nodes (آخر عقدة فرعية في الشجرة).
NextNode	العقدة التالية المجاورة.
Nodes	تجميعة من العقد المحتواة ضمن العقدة الحالية (العقد الأبناء) وتحوي الطريقة Add (لإضافة TreeNode) و Clear (لحذف جميع العقد) و Remove (لحذف عقدة معينة)، إن حذف عقدة

	يؤدي إلى حذف جميع أبنائها.
PrevNode	العقدة السابقة المجاورة.
SelectedImageIndex	فهرس الصورة عندما تكون العقدة محددة.
Text	نص العقدة.
الطرائق	
Collapse	تقوم بطي العقدة.
Expand	تقوم بتوسيع العقدة.
ExpandAll	تقوم بتوسيع جميع أبناء العقدة.
GetNodeCount	تعيد عدد العقد الأبناء.

- لإضافة عقد إلى TreeView في وضع التصميم، ننقر الزر الذي بجانب الخاصية Nodes في نافذة الخصائص، سيؤدي هذا إلى فتح المحرر TreeNode Editor والذي يعرض شجرة فارغة، يوجد أزرار لإنشاء عقدة جذر ولإضافة و حذف عقد، كما يظهر في الجانب الأيمن خصائص العقدة الحالية.
- لإضافة العقد برمجياً يجب إنشاء عقدة جذر أولاً: نُنشئ غرضاً جديداً من الصف TreeNode ونمرر له النص الذي سيعرضه، ثم نستدعي الطريقة Add لإضافة هذه العقدة الجديدة للتجميعة Nodes للعنصر TreeView.
- مثال:

```
myTreeView.Nodes.Add( new TreeNode( rootLabel ) );
```

وحيث أن myTreeView هي الشجرة TreeView التي نضيف إليها الجذر، وrootLabel هو النص المراد عرضه. ولإضافة عقد إلى الجذر نقوم بإضافة العقد إلى الخاصية Nodes للعقدة الجذر، يُمكننا اختيار العقدة الجذر المطلوبة كما يلي:

```
myTreeView.Nodes[ myIndex ]
```

وحيث يُمثل myIndex فهرس العقدة الجذر، وتتم إضافة العقد الأبناء بنفس الآلية السابقة:

```
myTreeView.Nodes[myIndex].Nodes.Add(new TreeNode(rootLabel));
```

- نستعمل في المثال التالي عنصر الشجرة TreeView لنعرض محتويات مجلد يختاره المستخدم عن طريق مربع نص وزر. حيث يكتب المستخدم مسار المجلد في مربع النص ثم ينقر الزر ليُحدّد الجذر في TreeView، ويصبح كل مجلد فرعي عقدة ابن، مما يشابه Windows Explorer. كما يمكن توسيع expand وطي collapse المجلد باستعمال المربع الذي يحمل الرمز + أو – إلى يسار المجلد.
- عندما ينقر المستخدم الزر enterButton، يتم مسح جميع العقد من الشجرة displayTreeView (السطر 57)، ثم يُستعمل المسار المدخل لإنشاء العقدة الجذر. يضيف السطر 75 المجلد إلى displayTreeView على أنه عقدة جذر، ويستدعي السطران 78 و 79 الطريقة

PopulateTreeView (الأسطر من 19 إلى 61)، والتي تأخذ مسار مجلد وعقدة أب، ثم تقوم الطريقة PopulateTreeView بإضافة عقد أبناء بحسب المجلدات الفرعية للمجلد الممرر كوسيط.

- تحصل الطريقة PopulateTreeView (الأسطر من 19 إلى 61) على قائمة من المجلدات الفرعية باستعمال الطريقة GetDirecories للصف Directory (في فضاء الأسماء System.IO) في السطرين 23 و24.
- تأخذ هذه الطريقة سلسلة محرفية (المجلد الحالي) وتعيد مصفوفة من السلاسل المحرفية (المجلدات الفرعية)، إذا كان الدخول إلى مجلد غير مسموح لأسباب الصلاحيات ، يتم إطلاق الاستثناء UnauthorizedAccessException. تقوم الأسطر من 57 إلى 60 بالتقاط هذا الاستثناء وتضيف عقدة تحوي العبارة "Access Denied" عوضاً عن عرض المجلدات الفرعية.
- إذا وجدت مجلدات فرعية يمكن الدخول إليها، تستعمل الأسطر من 40 إلى 42 الطريقة Substring لتزيد من قابلية القراءة باختصار المسارات الكاملة إلى اسم المجلد فقط ثم تُستعمل كل سلسلة محرفية في المصفوفة directoryArray لإنشاء عقدة ابن (السطر 45)، ثم نستعمل الطريقة Add لإضافة العقدة الابن (السطر 48)، ثم يتم استدعاء الطريقة PopulateTreeView عودياً على كل مجلد فرعي (السطر 51) مما يؤدي في المحصلة إلى بناء العنصر TreeView باستعمال بنية المجلد. لاحظ أن هذه الخوارزمية العودية قد تُسبب بطئاً عندما يُحّمل البرنامج مجلدات كبيرة .
- بعد إضافة أسماء المجلدات إلى الخاصية Nodes يمكن توسيع وطي هذه العقد.

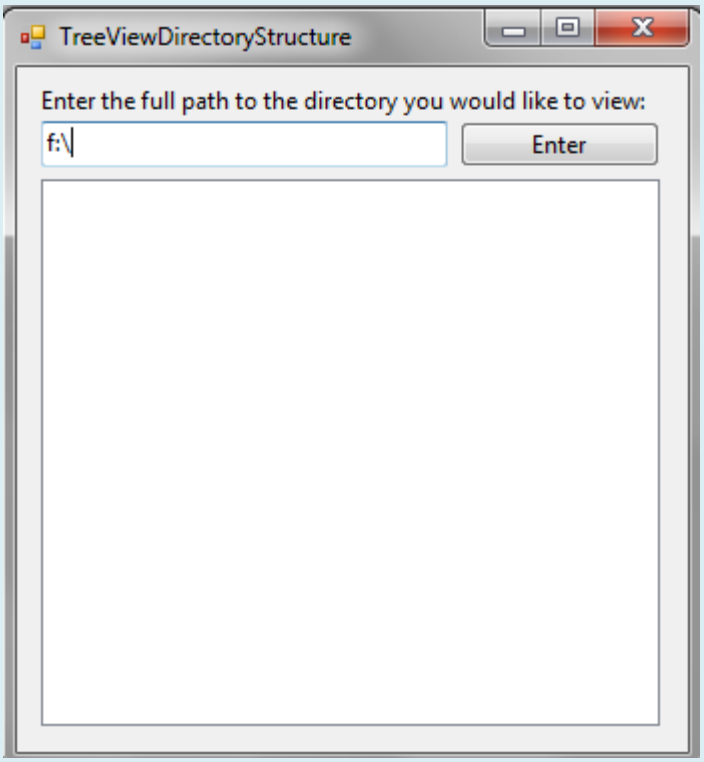
```
1  // TreeViewDirectoryStructureForm.cs
2  // Using TreeView to display directory structure.
3  using System;
4  using System.Windows.Forms;
5  using System.IO;
6
7  // Form uses TreeView to display directory structure
8  public partial class TreeViewDirectoryStructureForm : Form
9  {
10    string substringDirectory; // store last part of full path name
11
12    // default constructor
13    public TreeViewDirectoryStructureForm()
14    {
15       InitializeComponent();
16    } // end constructor
```

```
17
18    // populate current node with subdirectories
19    public void PopulateTreeView(
20       string directoryValue, TreeNode parentNode )
21    {
22       // array stores all subdirectories in the directory
23       string[] directoryArray =
24          Directory.GetDirectories( directoryValue );
25
26       // populate current node with subdirectories
27       try
28       {
29          // check to see if any subdirectories are present
30          if ( directoryArray.Length != 0 )
31          {
32             // for every subdirectory, create new TreeNode,
33             // add as a child of current node and recursively
34             // populate child nodes with subdirectories
35             foreach ( string directory in directoryArray )
36             {
37          // obtain last part of path name from the full path name
38          // using
39          // GetFileNameWithoutExtension
40             substringDirectory =
41               Path.GetFileNameWithoutExtension( directory );
42
43
44             // create TreeNode for current directory
45             TreeNode myNode = new TreeNode( substringDirectory );
46
47             // add current directory node to parent node
48             parentNode.Nodes.Add( myNode );
49
50             // recursively populate every subdirectory
```

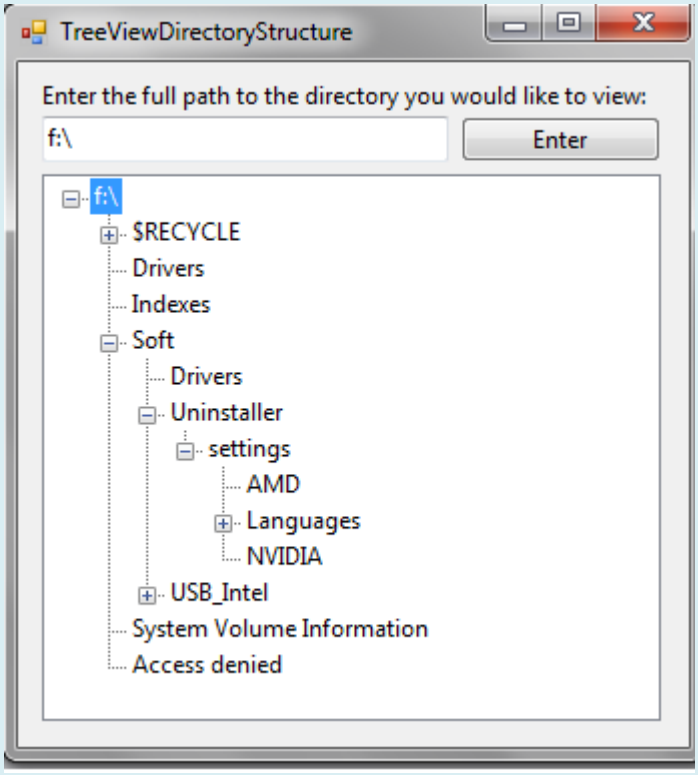
```
51              PopulateTreeView( directory, myNode );
52            } // end foreach
53          } // end if
54       } // end try
55
56       // catch exception
57       catch ( UnauthorizedAccessException )
58       {
59          parentNode.Nodes.Add( "Access denied" );
60       } // end catch
61    } // end method PopulateTreeView
62
63    // handles enterButton click event
64    private void enterButton_Click( object sender, EventArgs e )
65    {
66       // clear all nodes
67       directoryTreeView.Nodes.Clear();
68
69       // check if the directory entered by user exists
70       // if it does then fill in the TreeView,
71       // if not display error MessageBox
72       if ( Directory.Exists( inputTextBox.Text ) )
73       {
74          // add full path name to directoryTreeView
75          directoryTreeView.Nodes.Add( inputTextBox.Text );
76
77          // insert subfolders
78          PopulateTreeView(
79             inputTextBox.Text, directoryTreeView.Nodes[ 0 ] );
80       } // end if
81       // display error MessageBox if directory not found
82       else
83          MessageBox.Show(inputTextBox.Text+" could not be found.",
84             "Directory Not Found", MessageBoxButtons.OK,
```

```
85              MessageBoxIcon.Error );
86     } // end method enterButton_Click
87 } // end class TreeViewDirectoryStructureForm
```

(a)



(b)



عنصر التحكم: عرض القائمة ListView

الأهداف التعليمية:

- عرض القائمة ListView

ISSN: 2617-989X

عنصر التحكم ListView

- يعرض العنصر ListView قائمة يُمكن أن يختار منها المستخدم بنداً أو أكثر. كما يُمكن أن يعرض أيقونات icons بجانب البنود (تُحدّدها الخاصية ImageList). تُحدّد الخاصية MultiSelect (قيمة منطقية Boolean) فيما إذ كان من الممكن تحديد أكثر من بند. يُمكن تضمين مربعات اختيار بإعطاء الخاصية CheckBoxes (قيمة منطقية Boolean) القيمة true. تُحدد الخاصية View كيفية توضع البنود، وتُحدد الخاصية Activation طريقة اختيار المستخدم لبند من القائمة.
- يوضح الجدول التالي هذه الخصائص مع الحدث ItemActivate:

الخاصية أو الحدث	الوصف
الخصائص	
Activation	تُحدّد طريقة تفعيل المستخدم لبند، تأخذ قيمة من نمط التعداد ItemActivation، القيم الممكنة هي: OneClick للتفعيل بنقرة واحدة، TwoClick للتفعيل بالنقر المزدوج ويتغير لون البند عند تحديده، Standard للتفعيل بالنقر المزدوج ولا يتغير لون البند.
CheckBoxes	تُحدّد فيما إذا كان هناك مربعات اختيار بجانب العقد، القيمة الافتراضية هي false (لا تظهر مربعات الاختيار).
LargeImageList	تُحدّد قائمة الصور ImageList التي تحوي الأيقونات الكبيرة.
Items	تُعيد تجميعة من ListViewItem الموجودة ضمن العنصر.
MultiSelect	تُحدّد فيما إذا كان الاختيار المتعدد مسموحاً. القيمة الافتراضية هي true.
SelectedItems	تُعيد تجميعة العناصر المحددة.
SmallImageList	تُحدّد قائمة الصور ImageList التي تحوي الأيقونات الصغيرة.
View	تُحدّد مظهر البنود ضمن العنصر، القيم الممكنة هي: LargeIcon (عرض أيقونات كبيرة، يُمكن للبنود أن تشغل أكثر من عمود)، SmallIcon (عرض أيقونات صغيرة، يُمكن للبنود أن تشغل أكثر من عمود)، List (عرض أيقونات صغيرة تظهر البنود في عمود واحد)، Details (مثل List ، إلا أنه يمكن عرض معلومات في أعمدة أخرى لكل بند)،

	Tile (عرض أيقونات كبيرة ، مع عرض معلومات إلى يمين الأيقونة).
الأحداث	
ItemActivate	يتولد عند تفعيل بند، لا يحوي معلمات عن البند المفعل.

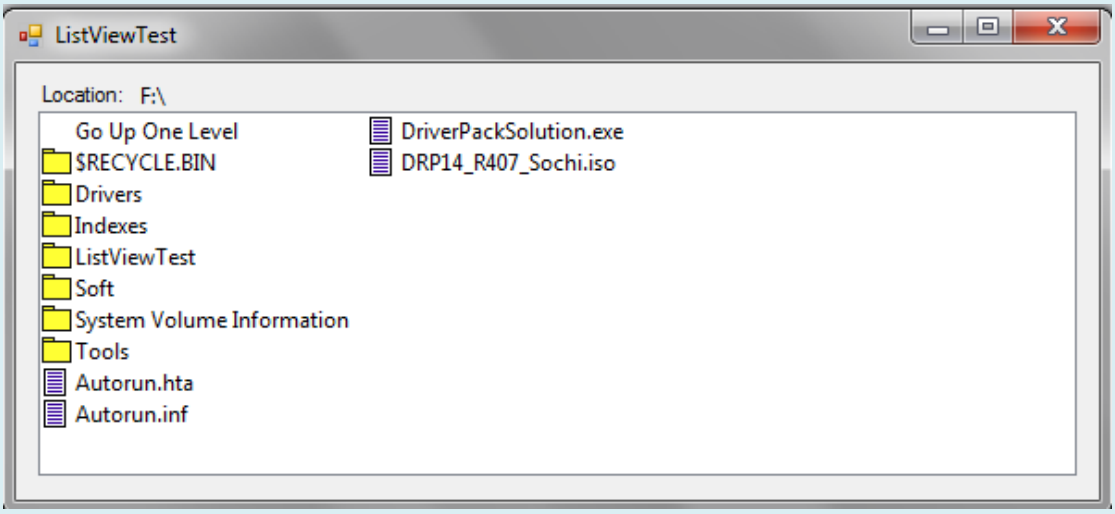
- يسمح العنصر ListView بتحديد الصور الخاصة بالبنود، ولعرض هذه الصور نحتاج إلى استخدام العنصر ImageList (قائمة صور). يُمكن إنشاء هذا العنصر بسحبه من صندوق الأدوات ToolBox وإفلاته فوق النموذج Form.
- نُضيف الصور إلى مصادر Resources المشروع.
- نقوم بتحميل الصور في ImageList في إجرائية Load للنموذج.
- تكون بنود العنصر ListView من النمط ListViewItem، و تُحدّد أيقونة كل منها بتحديد الخاصية ImageIndex.
- يعرض المثال التالي مجلدات وملفات ضمن ListView لاستعمال أيقونات صغيرة لكل منها، تظهر رسالة إذا كان الدخول إلى ملف أو مجلد غير مسموح لأسباب الصلاحيات، يمسح البرنامج محتويات المجلد عند استعراضه بدل من فهرسة سواقة مرة واحدة.

```
1  // ListViewTestForm.cs
2  // Displaying directories and their contents in ListView.
3  using System;
4  using System.Windows.Forms;
5  using System.IO;
6  namespace ListViewTest
7  {
8  // Form contains a ListView which displays
9  // folders and files in a directory
10 public partial class ListViewTestForm : Form
11 {
12 // store current directory
13 string currentDirectory = Directory.GetCurrentDirectory();
14 // constructor
15 public ListViewTestForm()
16 {
17  InitializeComponent();
18 } // end constructor
19 // browse directory user clicked or go up one level
20 private void browserListView_Click( object sender, EventArgs e )
21 {
22 // ensure an item is selected
23 if ( browserListView.SelectedItems.Count != 0 )
24 {
25 // if first item selected, go up one level
26 if ( browserListView.Items[0].Selected )
27 {
28 // create DirectoryInfo object for directory
29 DirectoryInfo directoryObject =
30     new DirectoryInfo( currentDirectory );
31 // if directory has parent, load it
```

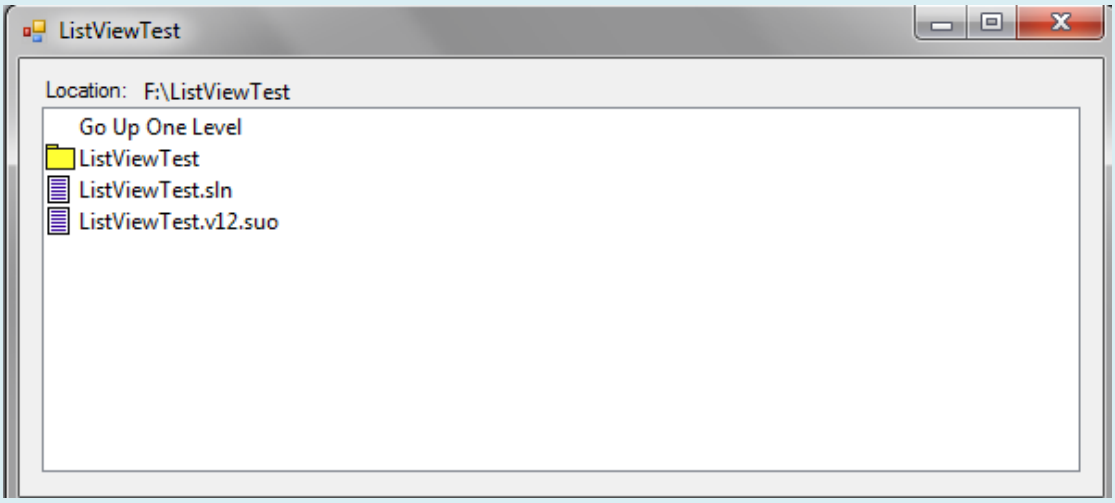
```
32 if ( directoryObject.Parent != null )
33 {
34  LoadFilesInDirectory(
35      directoryObject.Parent.FullName );
36 } // end if
37 } // end if
38 // selected directory or file
39 else
40 {
41 // directory or file chosen
42  string chosen = browserListView.SelectedItems[0].Text;
43 // if item selected is directory, load selected directory
44 if ( Directory.Exists(
45       Path.Combine( currentDirectory, chosen ) ) )
46 {
47  LoadFilesInDirectory(
48      Path.Combine( currentDirectory, chosen ) );
49 } // end if
50 } // end else
51 // update displayLabel
52  displayLabel.Text = currentDirectory;
53 } // end if
54 } // end method browserListView_Click
55 // display files/subdirectories of current directory
56 public void LoadFilesInDirectory( string currentDirectoryValue )
57 {
58 // load directory information and display
59 try
60 {
61 // clear ListView and set first item
62  browserListView.Items.Clear();
63  browserListView.Items.Add( "Go Up One Level" );
64  // update current directory
65  currentDirectory = currentDirectoryValue;
66  DirectoryInfo newCurrentDirectory =
67        new DirectoryInfo( currentDirectory );
68 // put files and directories into arrays
69 DirectoryInfo[] directoryArray =
70       newCurrentDirectory.GetDirectories();
71 FileInfo[] fileArray = newCurrentDirectory.GetFiles();
72 // add directory names to ListView
73 foreach ( DirectoryInfo dir in directoryArray )
74 {
75  // add directory to ListView
76  ListViewItem newDirectoryItem =
77    browserListView.Items.Add( dir.Name );
78  newDirectoryItem.ImageIndex = 0;  // set directory image
79 } // end foreach
80 // add file names to ListView
81 foreach ( FileInfo file in fileArray )
82 {
83  // add file to ListView
84  ListViewItem newFileItem =
85       browserListView.Items.Add( file.Name );
86  newFileItem.ImageIndex = 1;  // set file image
87 } // end foreach
88 } // end try
89 // access denied
90 catch ( UnauthorizedAccessException )
91 {
92  MessageBox.Show( "Warning: Some files may not be " +
93   "visible due to permission settings",
```

```
94    "Attention", 0, MessageBoxIcon.Warning );
95  } // end catch
96  } // end method LoadFilesInDirectory
97  // handle load event when Form displayed for first time
98  private void ListViewTestForm_Load( object sender, EventArgs e )
99  {
100  // add images to ImageList
101  fileFolderImageList.Images.Add( Properties.Resources.folder );
102  fileFolderImageList.Images.Add( Properties.Resources.file );
103  // load current directory into browserListView
104  LoadFilesInDirectory( currentDirectory );
105  displayLabel.Text = currentDirectory;
106 } // end method ListViewTestForm_Load
107 } // end class ListViewTestForm
108 } // end namespace ListViewTest
```

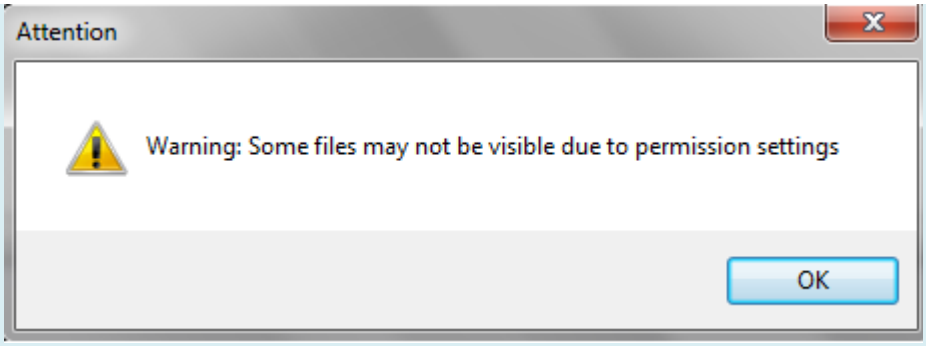
(a)



(b)



(c)



- تقوم الطريقة LoadFilesInDirectory (الأسطر من 56 إلى 96) ببناء الغرض browserListView باستعمال المجلد الممرر كوسيط (currentDirectoryValue)، حيث تقوم بمسح البنود في browserListView وتضيف البند "Go Up One Level"، وعندما ينقر المستخدم هذا البند يحاول البرنامج أن ينتقل لمستوى واحد باتجاه الأعلى (سنوضح الكيفية باختصار)، ثم تقوم الطريقة بإنشاء غرض من الصف DirectoryInfo و تهيئته بالسلسلة المحرفية currentDirectory (السطرين 66 و 67)، إذا لم يمنح إذن بتصفح المجلد يتم إطلاق استثناء (ويتم التقاطه في السطر 90).
- تقوم الطريقة LoadFilesInDirectory بتحميل المجلدات في المجلد الحالي فقط.
- يُمكّن الصف DirectoryInfo (في فضاء الأسماء System.IO) من تصفح و معالجة بنية المجلد بسهولة. تُعيد الطريقة GetDirectories (السطر 70) مصفوفة من DirectoryInfo تحوي المجلدات الفرعية للمجلد الحالي، وبشكل مشابه تُعيد الطريقة GetFiles (السطر 71) مصفوفة من FileInfo تحوي الملفات الموجودة في المجلد الحالي. تحوي الخاصية Name (لكلا الصفين DirectoryInfo و FileInfo) اسم الملف أو المجلد فقط مثل temp بدلاً من C:\myfolder\temp. للحصول على المسار الكامل نستعمل الخاصية FullName.
- تدور الأسطر من 73 إلى 80 ومن 81 إلى 87 على المجلدات الفرعية والملفات في المجلد الحالي وتضيفهم إلى browserListView. يقوم السطران 78 و 86 بتحديد الخاصية ImageIndex للبنود الجديدة، إذا كان البند مجلداً نعطيها القيمة 0 أما إذا كان ملفاً نعطيها القيمة 1.
- يستجيب حدث النقر browserListView_Click (الأسطر من 20 إلى 54) لنقر المستخدم على العنصر، يفحص السطر 23 فيما إذا كان هناك بند محدد، في حال وجود بند محدد يُحدد السطر 26 فيما إذا كان المستخدم قد اختار البند الأول، إن البند الأول هو دائماً "Go Up One Level"، لذا يحاول البرنامج الانتقال إلى المستوى الأعلى، يُنشئ السطران 29 و 30 غرضاً من DirectoryInfo من أجل المجلد الحالي، يفحص السطر 32 الخاصية Parent ليتأكد أن المستخدم ليس في جذر شجرة المجلدات، تحدد الخاصية Parent المجلد الأب على شكل DirectoryInfo. إذا لم يكن هناك مجلد أب تُعيد الخاصية Parent القيمة null، إذا وجد مجلد أب يقوم السطر 34 بتمرير المسار الكامل له للطريقة LoadFilesInDirectory.

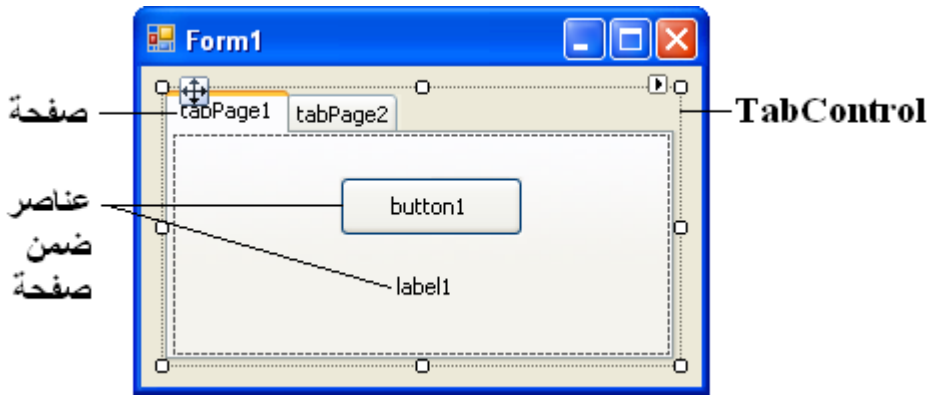
- إذا لم يقم المستخدم بتحديد البند الأول تسمح الأسطر من 40 إلى 50 للمستخدم بمتابعة التصفح ضمن بنية المجلدات. يُنشئ السطر 42 السلسلة المحرفية chosen والتي تحمل نص البند المحدد (أول بند في التجميعية SelectedItems)، يُحدّد السطر 44 فيما إذا كان المستخدم قد اختار مجلداً صالحاً (ليس ملفاً)، يجمع البرنامج المتحولين currentDirectory و chosen (المجلد الجديد)، و يمرر هذه القيمة للطريقة Exists في الصف Directory، تُعيد هذه الطريقة القيمة true إذا كانت السلسلة المحرفية تُمثل مجلداً، وفي هذه الحالة، يُمرر البرنامج السلسلة المحرفية للطريقة LoadFilesInDirectory. وفي النهاية تُحدّث displayLabel بقيمة المجلد الجديد (السطر 52).
- يعمل هذا البرنامج بسرعة لأنه يُفهرس فقط الملفات في المجلد الحالي.

عنصر التحكم TabControl

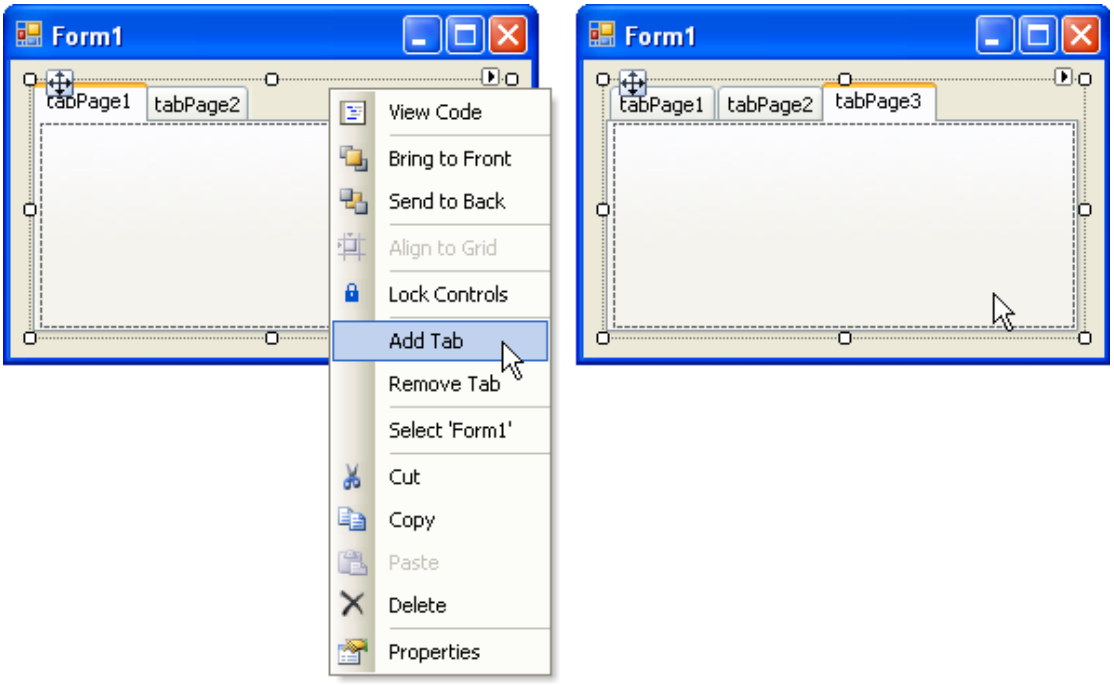
- يُمكّن هذا العنصر من عرض معلومات أكثر في نفس المساحة على نموذج.
- يحوي العنصر TabControl أغراضاً من الصف TabPage (صفحة)، والتي تُشبه اللوحات Panel وصناديق المجموعات GroupBox في إمكانيتها على احتواء عناصر أخرى. يجب أن نضيف أولاً العناصر إلى الصفحة، ثم نضيف الصفحة إلى TabControl، ويمكن عرض صفحة واحدة فقط في لحظة ما.
- لإضافة أغراض لصفحة ثم إلى TabControl نكتب برمجياً:

```
myTabPage.Controls.Add(myControl)
myTabControl.Controls.Add(myTabPage)
```

- يضيف هذا المثال عنصراً (myControl) إلى صفحة (myTabPage) ثم يقوم بإضافتها إلى myTabControl.
- يُمكننا بدلاً من ذلك استدعاء الطريقة AddRange بإضافة مصفوفة من الصفحات أو العناصر إلى كل من TabControl و TabPage على الترتيب.
- يوضح الشكل التالي مثالاً عن العنصر TabControl.



- يمكن إضافة TabControl في وضع التصميم بسحبه من صندوق الأدوات وإفلاته فوق النموذج، لإضافة صفحات في وضع التصميم ننقر على العنصر بالزر الأيمن ونختار Add Tab كما يُبين الشكل التالي.



- يُمكننا بدلاً من ذلك النقر على الخاصية TabPages في نافذة الخصائص وإضافة الصفحات في مربع الحوار الذي يظهر. يُمكن تغيير نص صفحة ما عن طريق الخاصية Text. لاحظ أن النقر على عناوين الصفحات يُحدد TabControl الحاوي لها. لتحديد صفحة ما، ننقر على العنصر الظاهر تحت العنوان، يُمكن إضافة عناصر للصفحة بالسحب والإفلات. لتغيير الصفحة المعروضة، ننقر على عنوان الصفحة المطلوبة (في وضع التصميم والتنفيذ).
- يوضح الجدول التالي أهم خصائص وأحداث العنصر TabControl.

الخاصية أو الحدث	الوصف
الخصائص	
ImageList	تُحدد قائمة الصور التي تظهر في عناوين الصفحات.
ItemSize	تُحدد حجم عنوان الصفحة.
MultiLine	تُحدد فيما إذا كان من الممكن عرض العناوين في أكثر من سطر.
SelectedIndex	فهرس الصفحة المحددة.
SelectedTab	الصفحة المحددة.
TabCount	تُعيد عدد الصفحات.
TabPages	تجميعة كل الصفحات.
الأحداث	

SelectedIndexChanged	يتولد عندما يتغير SelectedIndex (عند تحديد صفحة جديدة).

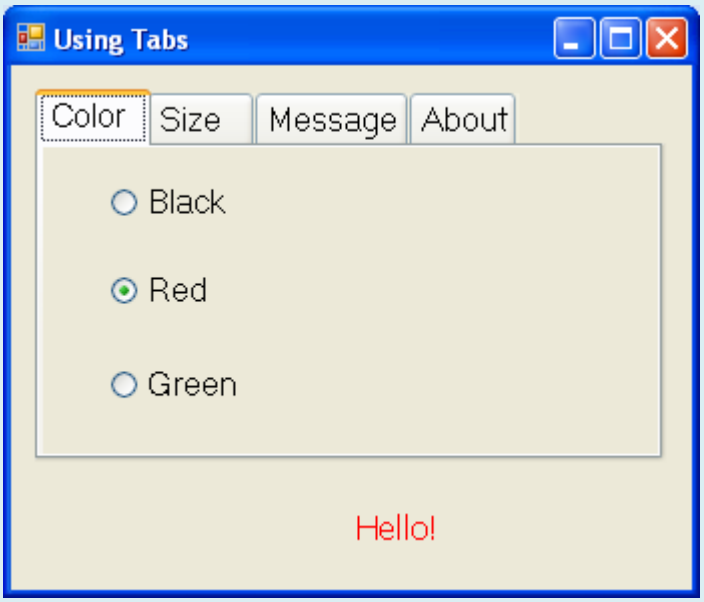
- تولد كل صفحة الحدث Click عند النقر على عنوانها، يُمكن توليد معالجات أحداث لهذا الحدث بالنقر المزدوج على جسم الصفحة.
- يستعمل المثال التالي العنصر TabControl لعرض مجموعة من الخيارات المتعلقة بنص ضمن لصاقة Label (اللون Color، الحجم Size، نص الرسالة Message). تُظهر آخر صفحة About نص يشرح كيفية استعمال الصفحات.

```
1  // UsingTabsForm.cs
2  // Using TabControl to display various font settings.
3  using System;
4  using System.Drawing;
5  using System.Windows.Forms;
6
7  //Form uses Tabs and RadioButtons to display various font settings
8  public partial class UsingTabsForm : Form
9  {
10    // default constructor
11    public UsingTabsForm()
12    {
13       InitializeComponent();
14    } // end constructor
15
16    // event handler for Black RadioButton
17    private void blackRadioButton_CheckedChanged(
18       object sender, EventArgs e )
19    {
20  displayLabel.ForeColor = Color.Black;//change font color to black
21   } // end method blackRadioButton_CheckedChanged
22
23    // event handler for Red RadioButton
24    private void redRadioButton_CheckedChanged(
```

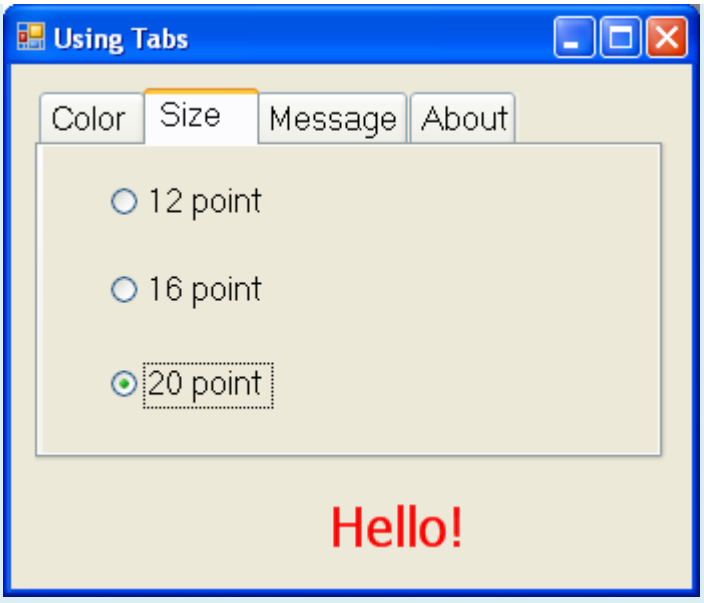
```
25          object sender, EventArgs e )
26       {
27     displayLabel.ForeColor = Color.Red; // change font color to red
28       } // end method redRadioButton_CheckedChanged
29
30       // event handler for Green RadioButton
31       private void greenRadioButton_CheckedChanged(
32          object sender, EventArgs e )
33       {
34    displayLabel.ForeColor = Color.Green;//change font color to green
35       } // end method greenRadioButton_CheckedChanged
36
37       // event handler for 12 point RadioButton
38       private void size12RadioButton_CheckedChanged(
39          object sender, EventArgs e )
40       {
41          // change font size to 12
42          displayLabel.Font = new Font( displayLabel.Font.Name, 12 );
43       } // end method size12RadioButton_CheckedChanged
44
45       // event handler for 16 point RadioButton
46       private void size16RadioButton_CheckedChanged(
47          object sender, EventArgs e )
48       {
49          // change font size to 16
50          displayLabel.Font = new Font( displayLabel.Font.Name, 16 );
51       } // end method size16RadioButton_CheckedChanged
52
53       // event handler for 20 point RadioButton
54       private void size20RadioButton_CheckedChanged(
55          object sender, EventArgs e )
56       {
57          // change font size to 20
58          displayLabel.Font = new Font( displayLabel.Font.Name, 20 );
```

```
59     } // end method size20RadioButton_CheckedChanged
60
61     // event handler for  Hello! RadioButton
62     private void helloRadioButton_CheckedChanged(
63        object sender, EventArgs e )
64     {
65        displayLabel.Text = "Hello!"; // change text to Hello!
66     } // end method helloRadioButton_CheckedChanged
67
68     // event handler for Goodbye! RadioButton
69     private void goodbyeRadioButton_CheckedChanged(
70        object sender, EventArgs e )
71     {
72        displayLabel.Text = "Goodbye!"; // change text to Goodbye!
73     } // end method goodbyeRadioButton_CheckedChanged
74 } // end class UsingTabsForm
```

(a)



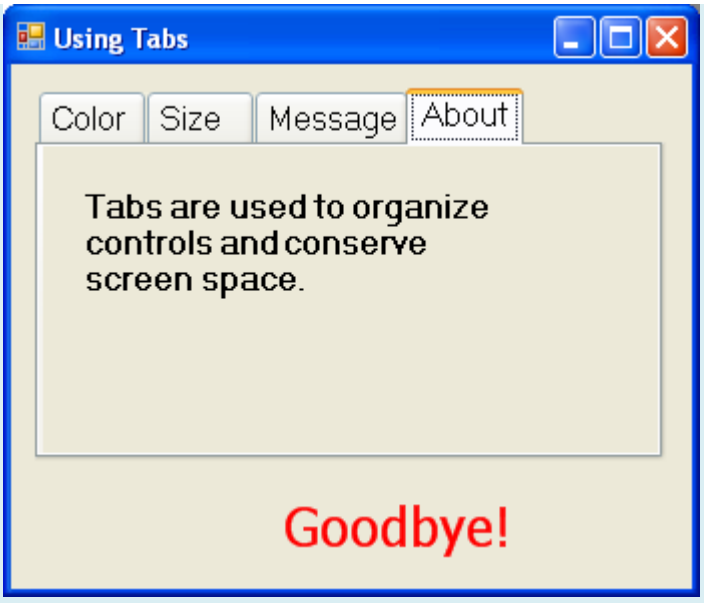
(b)


Using Tabs
Color
Size
Message
About
12 point
16 point
20 point
Hello!

(c)


Using Tabs
Color
Size
Message
About
Hello!
Goodbye!
Goodbye!

(d)



- قمنا بإنشاء العناصر textOptionsTabControl, colorTabPage, sizeTabPage, messageTabPage و aboutTabPage في وضع التصميم باستعمال المصمم Designer.
- تحوي الصفحة colorTabPage ثلاثة أزرار خيار RadioBottons للألوان أسود، أحمر وأخضر . يحدث معالج الحدث CheckChanged لكل زر خيار لون ويقوم بتلوين النص في displayLabel (الأسطر 20 و 27 و 34).
- تملك الصفحة sizeTabPage ثلاثة أزرار خيار لقياسات الخط 12 و 16 و 20 والتي تغير حجم الخط في displayLabel (الأسطر 42 و50 و58).
- تحوي الصفحة messageTabPage زري خيار لنصي الرسالة Hello! و Goodbye! واللذان يحددان النص ضمن displayLabel (السطرين 65 و 72).
- تحوي الصفحة aboutTabPage لصاقة تشرح الهدف من استعمال العنصر TabControl.

الفصل السابع: مواضيع متقدمة

عنوان الموضوع:

مواضيع متقدمة.

الكلمات المفتاحية:

الوثائق متعددة المستندات، الوراثة المرئية، إنشاء عناصر مخصصة.

ملخص:

نعرض في هذا الفصل بعض المواضيع المتقدمة. نبدأ أولاً بتقانة إنشاء الواجهات ذات الوثائق المتعددة، ثم نُبين استخدامات الوراثة المرئية، وأخيراً نتعلم بناء عناصر جديدة مخصصة.

أهداف تعليمية:

يتعرف الطالب في هذا الفصل على:

- كيفية بناء الواجهات ذات الوثائق المتعددة.
- استخدام الوراثة المرئية.
- بناء عناصر مخصصة.

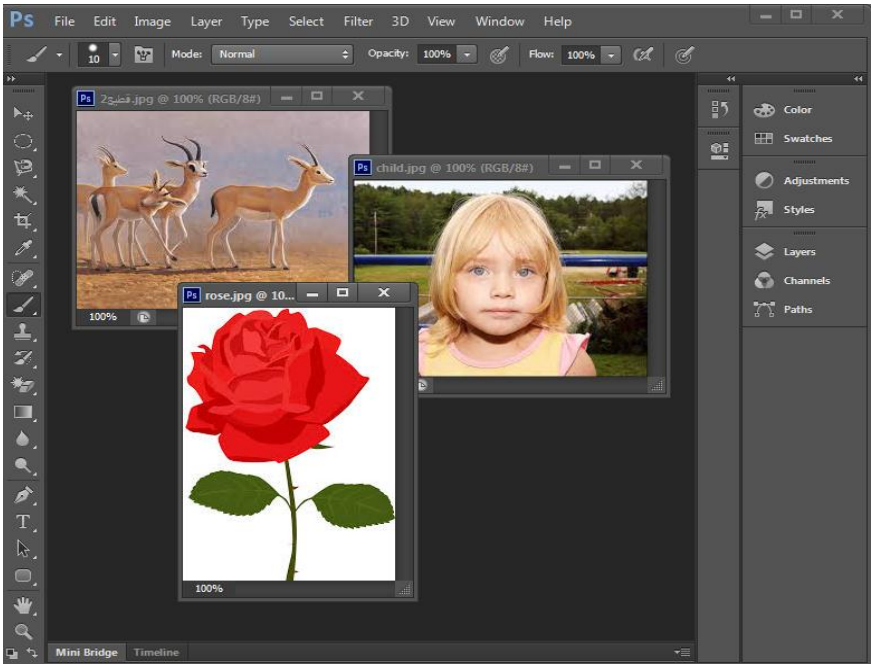
المخطط:

مواضيع متقدمة

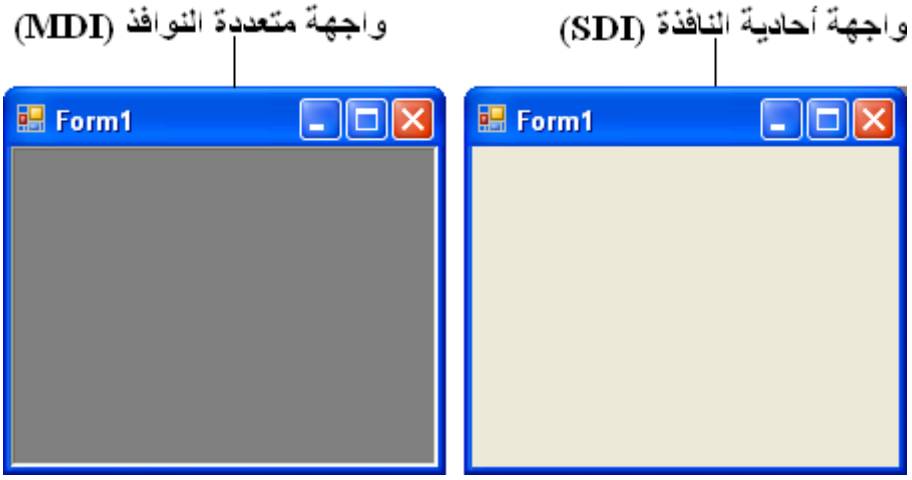
- 3 وحدات (Learning Objects)

نوافذ الواجهة متعددة المستندات MDI

- قمنا فيما سبق ببناء تطبيقات ذات واجهات أحادية المستندات Single Document Interface (SDI)، تستطيع البرامج من هذا النوع (مثل Notepad و Paint) أن تُظهر نافذة واحدة فقط في نفس الوقت. عادةً ما يكون لهذه التطبيقات قدرات محدودة، فالتطبيقين المذكورين آنفاً لهما قدرات تحرير محدودة للنص والصورة، يجب على المستخدم تشغيل نسخة جديدة من البرنامج لتحرير عدة مستندات.
- تكون معظم التطبيقات المتقدمة برامج ذات واجهات متعددة المستندات Multiple Document Interface (MDI)، والتي تسمح للمستخدم تحرير عدة مستندات في آن معاً (مثال: تطبيقات Microsoft Office). كما أن التطبيقات ذات الواجهات متعددة المستندات تميل لأن تكون أكثر تعقيداً، فالتطبيقين PaintShop Pro و Photoshop على سبيل المثال يملكان عدداً أكبر من إمكانيات تحرير الصور والنصوص من Paint.
- تُدعى النافذة الرئيسية لتطبيق MDI بالنافذة الأم Parent Window، كما توصف كل نافذة أخرى ضمن التطبيق بأنها نافذة ابن Child Window. وعلى الرغم من أن تطبيق MDI يُمكن أن يحوي العديد من النوافذ الأبناء، إلا أن كلاً منها لها نافذة أم واحدة. كما أنه يُمكن لنافذة ابن وحيدة على الأكثر أن تكون فعالة في نفس الوقت.
- لا يُمكن أن تكون النافذة الابن أماً، كما لايُمكن تحريكها خارج نافذتها الأم، وإلا لكانت تتصرف كأي نافذة أخرى (مع الأخذ بعين الاعتبار الإغلاق والتصغير وتغيير الحجم ...). كما يُمكن أن تختلف وظيفة نافذة ابن عن نافذة ابن أخرى تشترك معها في النافذة الأم، فيمكن على سبيل المثال لنافذة ابن أن تسمح للمستخدم بتحرير الصور، ويمكن لأخرى أن تسمح له بتحرير نص، ويمكن لثالثة أن تعرض مخططاً لنشاط الشبكة، ويمكن لجميع هذه النوافذ أن تنتمي لنفس النافذة الأم MDI Parent.
- يوضح الشكل التالي مثالاً عن تطبيق MDI:



- لإنشاء نموذج MDI (MDI Form)، نقوم بإنشاء نموذج مع وضع الخاصية IsMdiContainer مساوية true. يتغير عندها مظهر النموذج كما في الشكل التالي:



- نقوم بعد ذلك بإنشاء صف للنموذج الابن ليُضاف إلى النموذج. للقيام بذلك ننقر على المشروع في Solution Explorer بالزر الأيمن ونختار (Project → Add → Windows Form) ونسمي الملف، ونحرر النموذج الجديد كما نريد.
- لإضافة النموذج الابن للنافذة الأم، يجب إنشاء غرض جديد منه، وإسناد النموذج الأم إلى الخاصية MdiParent، واستدعاء الطريقة Show للنموذج الابن.
- وبشكل عام، فلإضافة نموذج ابن إلى نموذج أم نكتب:

```
ChildFormClass childForm = new ChildFormClass();
childForm. MdiParent = parentForm;
childForm.Show();
```

- يقوم النموذج الأم بإنشاء الابن في معظم الحالات، لذا يكون المؤشر parentForm هو this.
- عادةً، يتوضع الكود اللازم لإنشاء نموذج ابن ضمن معالج حدث، والذي يقوم بإنشاء نافذة جديدة كرد فعل على حدث قام به المستخدم، تُشكل خيارات القوائم (مثل File → New → Window) تقنية مألوفة لإنشاء نوافذ جديدة.
- تُعيد الخاصية MdiChildren للصف Form مصفوفة من المؤشرات على النماذج الأبناء، وهذا مفيد إذا كانت النافذة الأم تريد فحص حالة كل أبنائها (كالتحقق من حفظ جميع النوافذ الأبناء قبل إغلاق النافذة الأم). تُعيد الخاصية ActiveMdiChild مؤشراً على النافذة الابن الفعالة، ولا تُعيد شيئاً في حال عدم وجود أي نافذة ابن فعالة.
- يوضح الجدول التالي بعض المزايا الأخرى لنوافذ MDI:

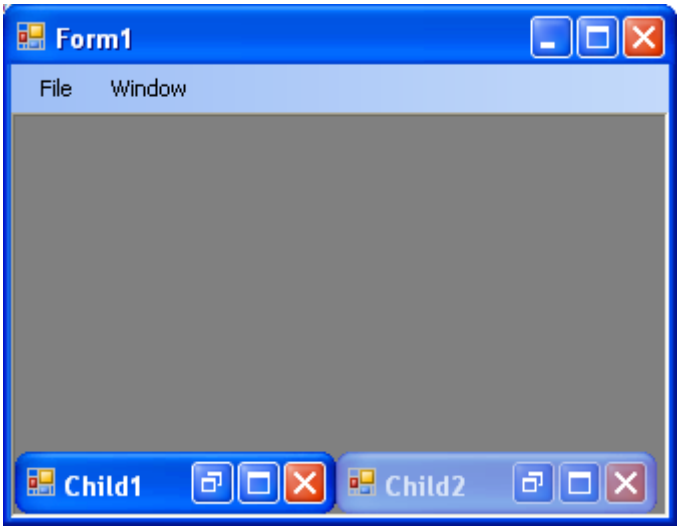
الخاصية أو الحدث	الوصف
خصائص النافذة الابن	
IsMdiChild	تُحدد فيما إذا كان النموذج هو نموذج MDI ابن (للقراءة فقط read-only).
MdiParent	تُحدد النموذج الأب.
خصائص النافذة الأم	
ActiveMdiChild	تُعيد النموذج الابن الفعال (null في حال عدم وجود ابن فعال).
IsMdiContainer	تُحدد فيما إذا كان يُمكن للنموذج أن يكون نموذجاً أباً في تطبيق Mdi، القيمة الافتراضية هي false.
MdiChildren	تُعيد نوافذ الأبناء على شكل مصفوفة من النماذج.
الطرائق	
LayoutMdi	تُحدد طريقة عرض النوافذ الأبناء ضمن النافذة الأم، تأخذ هذه الطريقة وسيطاً من نمط التعداد MdiLayout الذي يأخذ أحد القيم ArrangeIcons ، Cascade ، TileHorizental ، TileVertical.
الأحداث	
MdiChildActivate	يتولد عندما يتم تفعيل أو إغلاق نافذة ابن.

- يُمكن إغلاق وتصغير وتكبير النوافذ الأبناء بشكل مستقل عن النوافذ الأبناء الأخرى أو النافذة الأم.
- عندما يتم تصغير أو إغلاق النافذة الأم، يتم تصغير أو إغلاق النوافذ الأبناء أيضاً.
- توفر C# خاصية تُساعد على متابعة النوافذ الأبناء المفتوحة في حاوية MDI، حيث تُحدد الخاصية MdiWindowListItem للصف MenuStrip القائمة -في حال وجودها- والتي تُظهر قائمة بالنوافذ الأبناء المفتوحة، وعندما تُفتح نافذة ابن جديدة، يُضاف بند إلى القائمة.
- إذا تم فتح تسع نوافذ أبناء أو أكثر يظهر الخيار ...More Windows ضمن القائمة، والذي يسمح للمستخدم بتحديد النافذة من قائمة في مربع حوار.
- تسمح حاويات MDI بتنظيم توضع النوافذ الأبناء، يمكن ترتيب النوافذ الأبناء في تطبيق MDI باستدعاء الطريقة LayoutMdi للنموذج الأب، تأخذ الطريقة LayoutMdi وسيطاً من نمط التعداد MdiLayout، والذي يأخذ أحد القيم: ArrangeIcons, Cascade, TileHorizental و TileVertical. تملأ النوافذ الأبناء المرصوفة Tiles النافذة الأم بشكل كامل ولا تتداخل، ويمكن ترتيبها أفقياً (القيمة TileHorizental) أو عمودياً (القيمة TileVertical)، تتداخل النوافذ المتسلسلة Cascaded (القيمة Cascade) بحيث يكون لكل منها نفس الحجم وتظهر جزءاً من شريط العنوان،

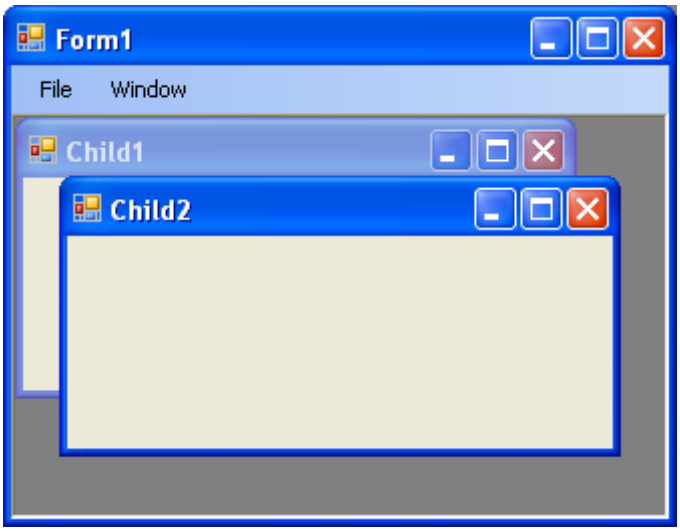
ترتب القيمة ArrangeIcons أيقونات النوافذ الأبناء المصغرة (عند الإمكان)، إذا كانت النوافذ المصغرة متناثرة ضمن النافذة الأم تقوم القيمة ArrangeIcons بترتيبها بشكل أنيق في الزاوية اليسرى– السفلى للنافذة الأم.

- يوضح الشكل التالي القيم المختلفة لنمط التعداد MdiLayout.

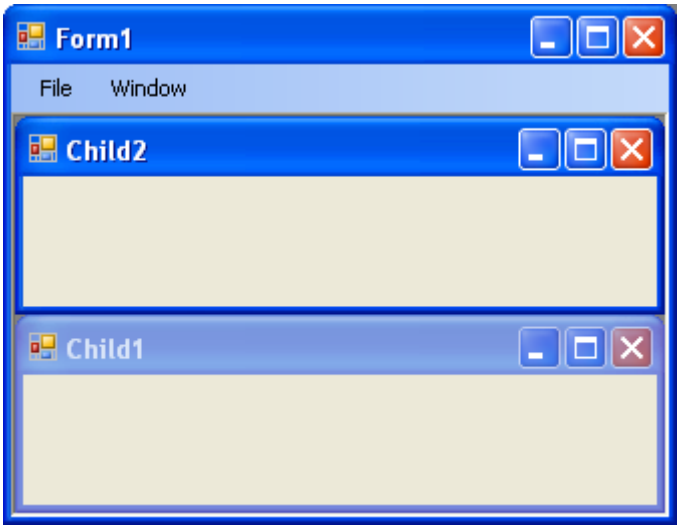
(a) ArrangeIcons



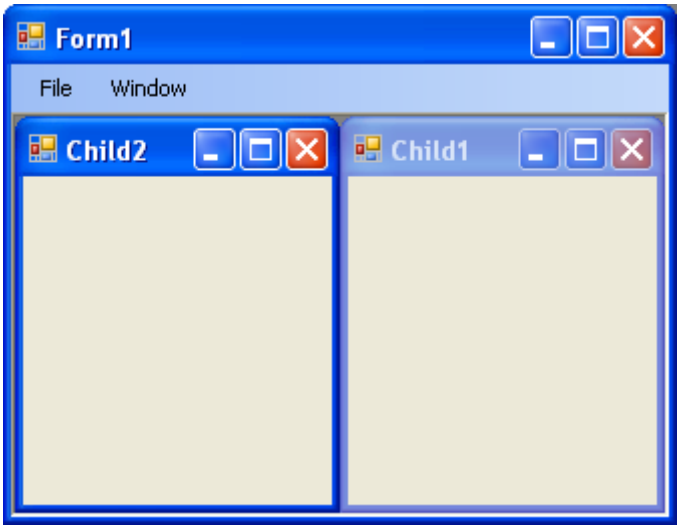
(b) Cascade



(c) TileHorizental



(d) TileVertical



- يوضح المثال التالي طريقة استعمال نوافذ MDI. يستعمل الصف UsingMDIForm ثلاث منتسخات من النماذج الأبناء من الصف ChildForm، يحتوي كل منها صندوق صورة، يحوي النموذج الأب قائمة تسمح للمستخدم بإنشاء وترتيب النماذج الأبناء.
- يحوي النموذج الأب –الذي يتم إنشاؤه أولاً– قائمتين في المستوى الأعلى، تحوي أولاها (File) القائمة الفرعية New والبند Exit، تتألف القائمة الفرعية New من بند لكل نافذة ابن، وتوفر القائمة الثانية Window خيارات لتوضع نوافذ MDI الأبناء، بالإضافة إلى قائمة بالنوافذ الأبناء المفتوحة.

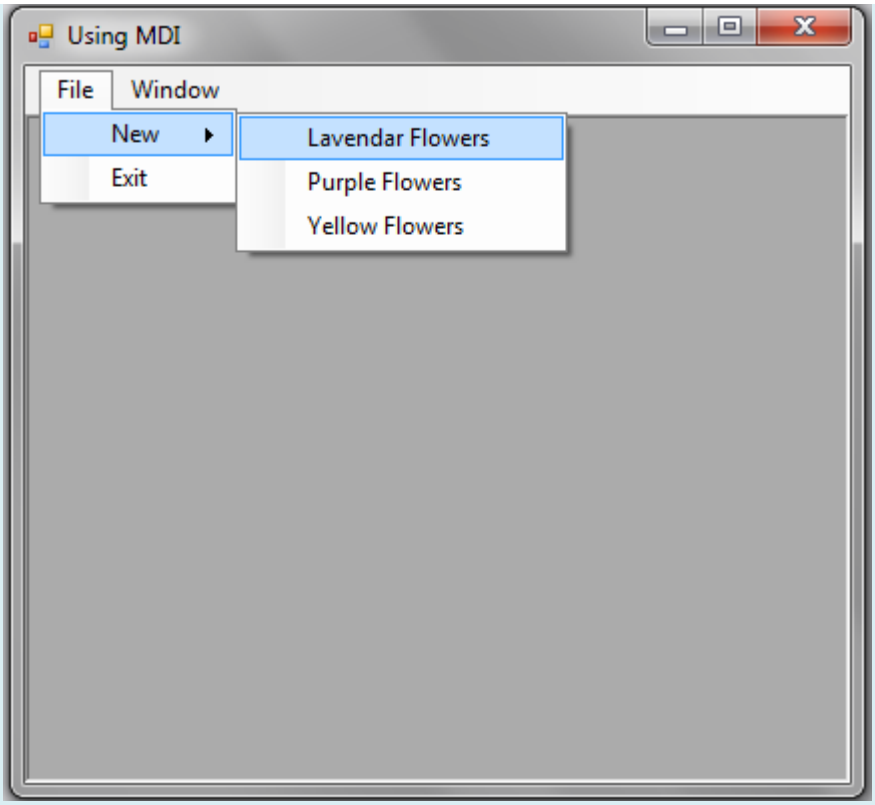
- نجعل قيمة الخاصية IsMdiContainer مساوية true في نافذة الخصائص لنجعل من النموذج نافذة أم، نجعل قيمة الخاصية MdiWindowListItem للقائمة (menuStrip1) مساوية windowToolStripMenuItem، يجعل هذا القائمة Window حاوية على قائمة النوافذ الأبناء.
- يملك البند Cascade معالج الحدث cascadeToolStripMenuItem_Click (الأسطر من 55 إلى 59) والذي يُرتب النوافذ الأبناء بشكل متشلشل، يستدعي معالج الحدث هذا الطريقة LayoutMdi ويمرر لها القيمة Cascade (السطر 58).
- يملك البند TileHorizental معالج الحدث tileHorizentalToolStripMenuItem_Click (الأسطر من 62 إلى 66) والذي يرتب النوافذ الأبناء بشكل أفقي، يستدعي معالج الحدث هذا الطريقة LayoutMdi ويمرر لها القيمة TileHorizental (السطر 65).
- أخيراً، يملك البند TileVertical معالج الحدث tileVerticalToolStripMenuItem_Click (الأسطر من69 إلى 73) والذي يُرتب النوافذ الأبناء بشكل أفقي، يستدعي معالج الحدث هذا الطريقة LayoutMdi ويمرر لها القيمة TileVertical (السطر 72).
- لم يكتمل بناء التطبيق حتى الآن، يجب أن نُحدد صف النافذة الابن ولفعل ذلك ننقر على المشروع في Solution Explorer بالزر الأيمن ونختار (Project → Add → Windows Form) ونسمي الملف ChildForm، وثم نضيف مربع صورة له.

```
1  // UsingMDIForm.cs
2  // Demonstrating use of MDI parent and child windows.
3  using System;
4  using System.Windows.Forms;
5
6  // Form demonstrates the use of MDI parent and child windows
7  public partial class UsingMDIForm : Form
8  {
9     // default constructor
10    public UsingMDIForm()
11    {
12       InitializeComponent();
13    } // end constructor
14
15    //create Child 1 window when child1ToolStripMenuItem is clicked
16    private void child1ToolStripMenuItem_Click(
17       object sender, EventArgs e )
```

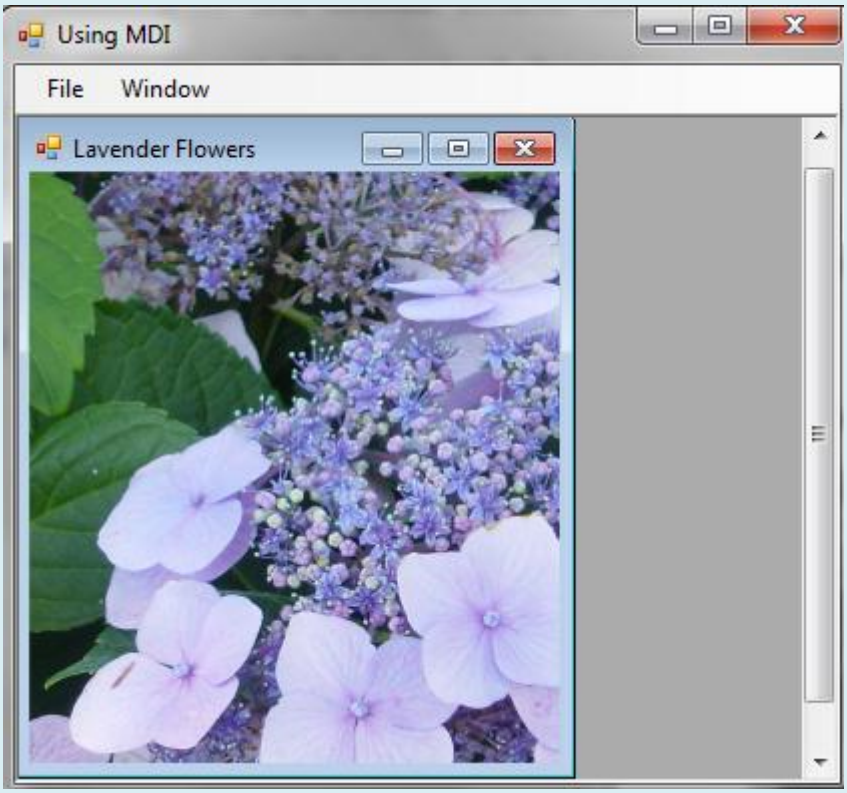
```
18    {
19       // create new child
20       ChildForm formChild = new ChildForm(
21          "Lavender Flowers", "lavenderflowers" ); );
22       formChild.MdiParent = this; // set parent
23       formChild.Show(); // display child
24    } // end method child1ToolStripMenuItem_Click
25
26    //create Child 2 window when child2ToolStripMenuItem is clicked
27    private void child2ToolStripMenuItem_Click(
28       object sender, EventArgs e )
29    {
30       // create new child
31       ChildForm formChild = new ChildForm(
32          "Purple Flowers", "purpleflowers" );
33       formChild.MdiParent = this; // set parent
34       formChild.Show(); // display child
35    } // end method child2ToolStripMenuItem_Click
36
37    //create Child 3 window when child3ToolStripMenuItem is clicked
38    private void child3ToolStripMenuItem_Click(
39       object sender, EventArgs e )
40    {
41       // create new child
42       ChildForm formChild = new ChildForm(
43          "Yellow Flowers", "yellowflowers" );
44       formChild.MdiParent = this; // set parent
45       formChild.Show(); // display child
46    } // end method child3MenuItem_Click
47
48    // exit application
49    private void exitToolStripMenuItem_Click(object sender,
EventArgs e )
50    {
```

```
51          Application.Exit();
52       } // end method exitToolStripMenuItem_Click
53
54       // set Cascade layout
55       private void cascadeToolStripMenuItem_Click(
56          object sender, EventArgs e )
57       {
58          this.LayoutMdi( MdiLayout.Cascade );
59       } // end method cascadeToolStripMenuItem_Click
60
61       // set TileHorizontal layout
62       private void tileHorizontalToolStripMenuItem_Click(
63          object sender, EventArgs e )
64       {
65          this.LayoutMdi( MdiLayout.TileHorizontal );
66       } // end method tileHorizontalToolStripMenuItem
67
68       // set TileVertical layout
69       private void tileVerticalToolStripMenuItem_Click(
70          object sender, EventArgs e )
71       {
72          this.LayoutMdi( MdiLayout.TileVertical );
73       } // end method tileVerticalToolStripMenuItem_Click
74    } // end class UsingMDIForm
```

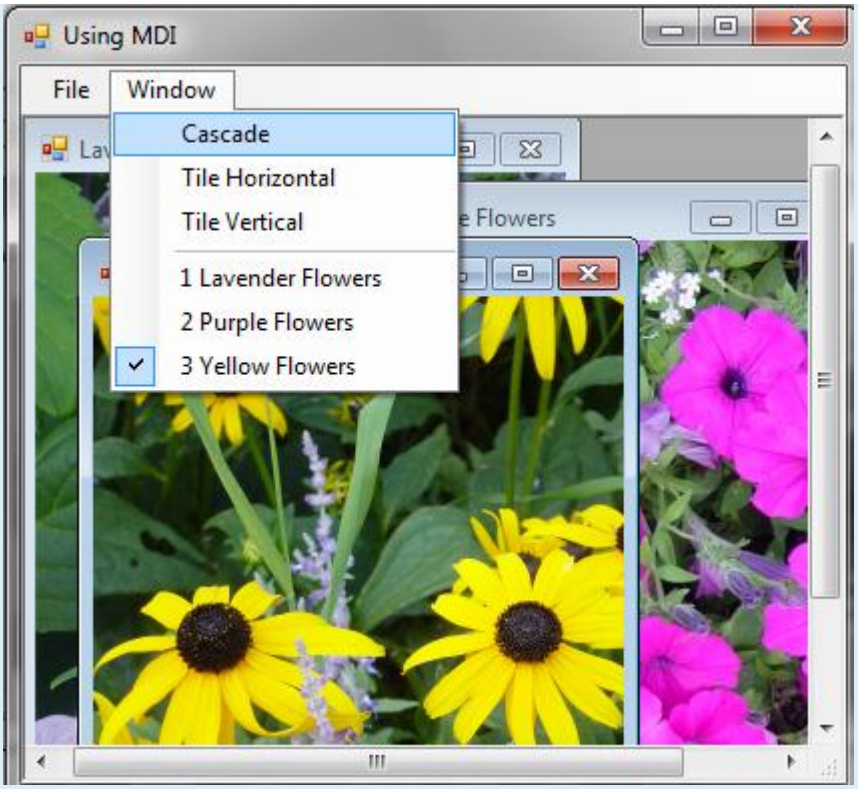
(a)


Using MDI
File
Window
New
Exit
Lavendar Flowers
Purple Flowers
Yellow Flowers

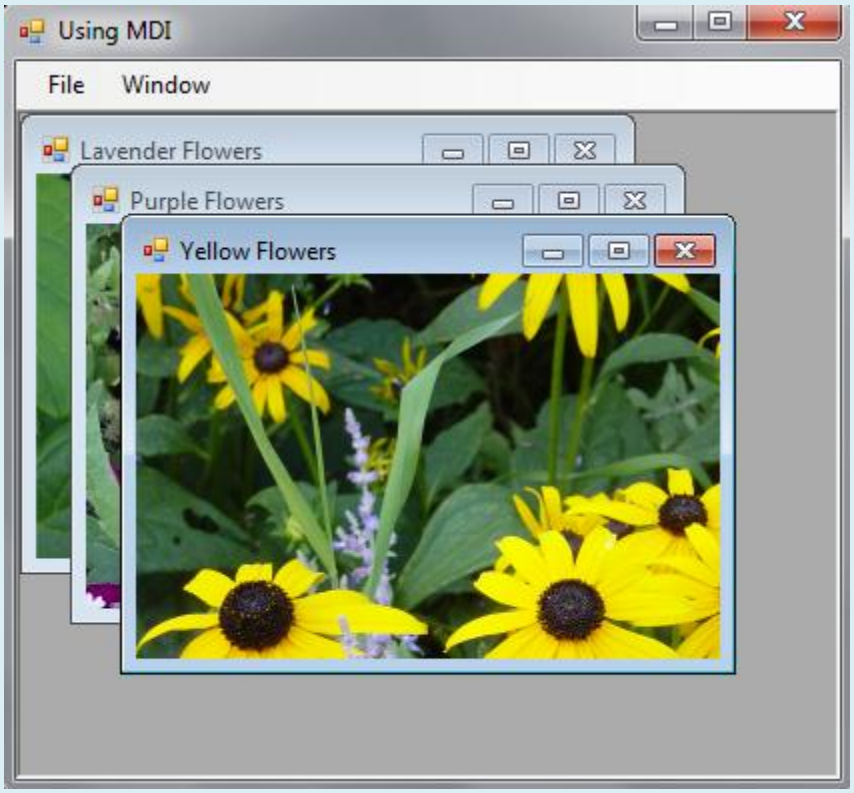
(b)


Using MDI
File
Window
Lavender Flowers

(c)



(d)



- بعد تعريف صف النافذة الابن، يُمكن للنافذة الأم إنشاء نوافذ أبناء. تنشئ معالجات الأحداث في الأسطر من 16 إلى 46 نموذج ابن جديد حسب بند القائمة التي تم نقرها. تقوم الأسطر 20 و 21 ،

30 و 32 ، 42 و 43 بإنشاء منتسخات جديدة من ChildForm، و تقوم الأسطر 22 و 33 و 44 بتحديد الخاصية MdiParent و تستدعي الأسطر 23 و 34 و 45 الطريقة Show لعرض النافذة الابن.

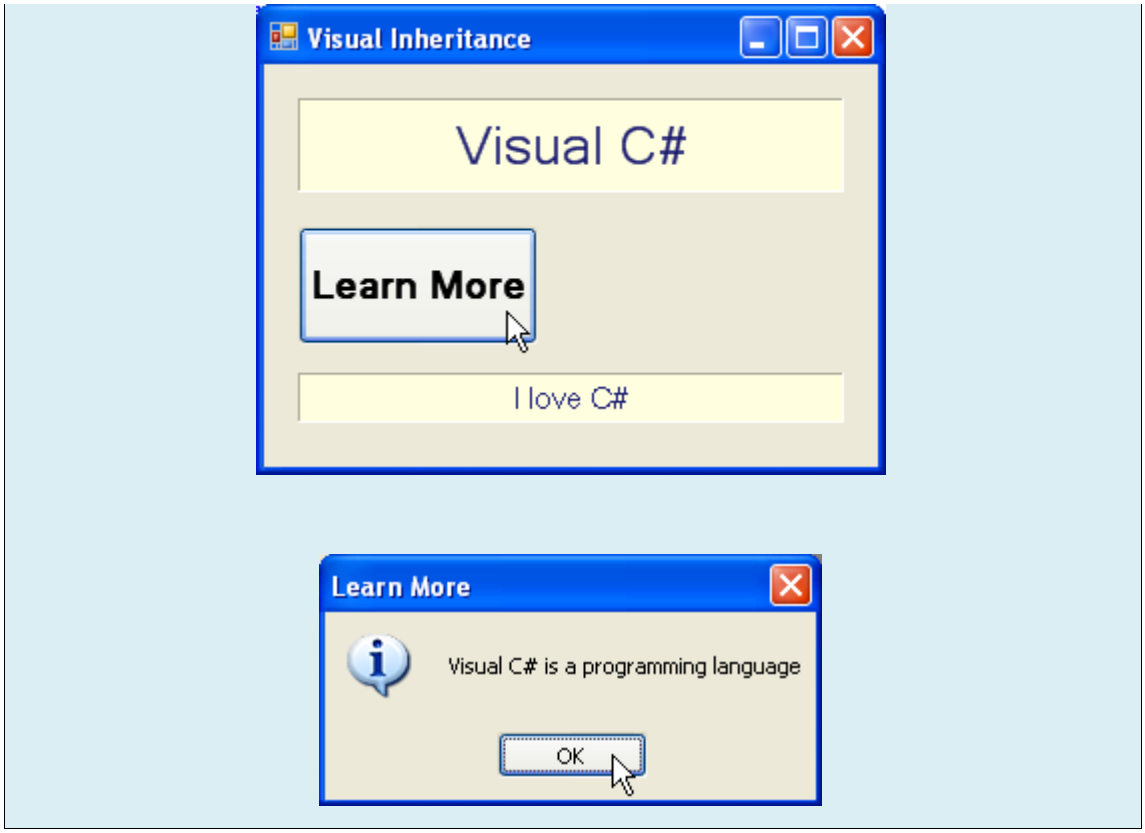
- في الباني constructor للنموذج الابن، يُحدد السطر 15 نص شريط العنوان، يقوم السطران 18 و 19 بتحديد الخاصية Image.

```
1  // Child.cs
2  // Child window of MDI parent.
3  using System;
4  using System.Drawing;
5  using System.Windows.Forms;
6  using System.IO;
7
8  public partial class ChildForm : Form
9  {
10    public ChildForm( string title, string resourceName)
11    {
12       // Required for Windows Form Designer support
13       InitializeComponent();
14
15       Text = title; // set title text
16
17       // set image to display in pictureBox
18       picDisplay.Image =
19      (Image)(Properties.Resources.ResourceManager.GetObject(resourceName);
20    } // end constructor
21 } // end class ChildForm
```

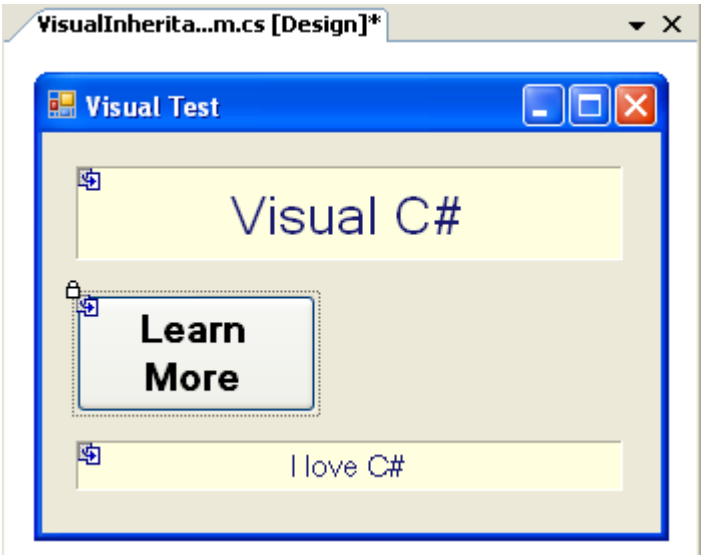
الوراثة المرئية Visual Inheritance

- تُمكّن الوراثة المرئية من تحقيق توافقية في المظهر ضمن تطبيق. فعلى سبيل المثال، يُمكن تعريف نموذج أب يحوي شعار المنتج ولون خلفية معين وشريط قوائم مسبق التعريف ومجموعة من العناصر الأخرى، ثم يُمكن استعمال النموذج الأب ضمن التطبيق لتوحيد المظهر.
- في المثال التالي، يحوي النموذج VisualInheritanceForm لصاقتين تعرضان النص "Visual C#" و "I love C#"، و زراً يعرض النص Learn more، عندما يضغط المستخدم الزر يتم استدعاء الطريقة learnMoreButton_Click (الأسطر من 16 إلى 22)، و التي تقوم بعرض رسالة MessageBox تُظهر بعض المعلومات.

```
1  // VisualInheritanceForm.cs
2  // Base Form for use with visual inheritance.
3  using System;
4  using System.Windows.Forms;
5
6  // base Form used to demonstrate visual inheritance
7  public partial class VisualInheritanceForm : Form
8  {
9     // default constructor
10    public VisualInheritanceForm()
11    {
12       InitializeComponent();
13    } // end constructor
14
15    // display MessageBox when Button is clicked
16    private void learnMoreButton_Click( object sender,EventArgs e )
17    {
18       MessageBox.Show(
19          "Visual C# is a programming language",
20          "Learn More", MessageBoxButtons.OK,
21          MessageBoxIcon.Information );
22    } // end method learnMoreButton_Click
23 } // end class VisualInheritanceForm
```



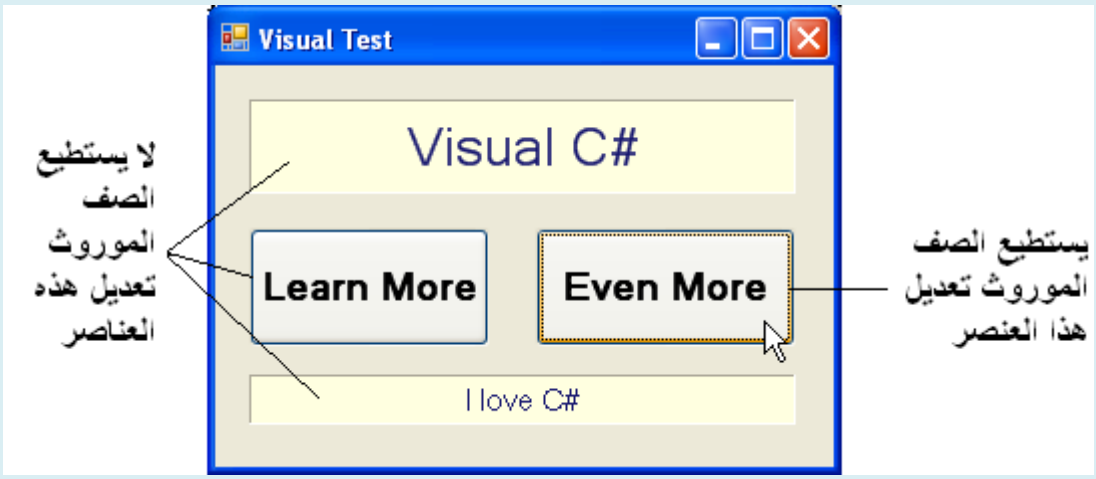
- لنسمح لنماذج أخرى بالوراثة المرئية من VisualInheritanceForm يجب حزمها ضمن مكتبة صفوف class library (.dll). ننقر بالزر الأيمن على اسم المشروع في Solution Explorer ونختار Properties، ثم نختار Application ونغير الخيار في القائمة المنسدلة Output type من Windows Application إلى Class Library، يؤدي بنا هذا إلى توليد (.dll).
- لنرث VisualInheritanceForm مرئياً، نقوم أولاً بإنشاء تطبيق جديد، ونضيف إليه مرجعاً reference إلى المكتبة التي أنشأناها (الموجودة في المجلد bin\Release للتطبيق السابق)، ثم نفتح الملف الذي يُعرّف واجهة التطبيق الجديد ونعدل السطر الأول ليرث من الصف VisualInheritanceForm. لاحظ أننا بحاجة لتحديد اسم الصف، يجب أن تظهر الآن عناصر النموذج الأب في وضع التصميم كما في الشكل التالي، ما زال بإمكاننا إضافة عناصر إلى النموذج.

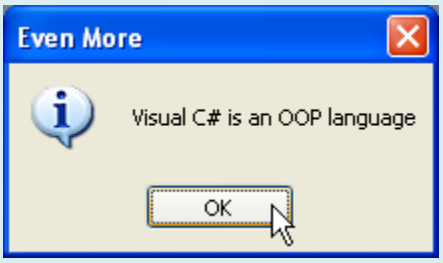


- تحوي الواجهة العناصر الموروثة من الصف VisualInheritanceForm بالإضافة إلى زر إضافي يحمل النص Even More. عندما ينقر المستخدم هذا الزر يقوم معالج الحدث الاسطر من 17 إلى 22 بعرض رسالة تحوي معلومات مختلفة عن الرسالة السابقة. يوضح المثال أن العناصر و توضعها و عملها والتي توجد في الصف VisualInheritanceForm تورّث جميعها للصف VisualInheritanceTestForm.
- إذا نقر المستخدم الزر Learn More سيقوم معالج الحدث بعرض رسالة، و بما أن الصف VisualInheritanceForm يستعمل محدد الوصول private لتعريف عناصره فلا يمكن للصف VisualInheritanceTestForm تعديل هذه العناصر.

```
1  // VisualInheritanceTestForm.cs
2  // Derived Form using visual inheritance.
3  using System;
4  using System.Windows.Forms;
5
6  // derived form using visual inheritance
7  public partial class VisualInheritanceTestForm :
8     VisualInheritanceForm   // code for inheritance
9  {
10    // default constructor
11    public VisualInheritanceTestForm()
12    {
13       InitializeComponent();
```

```
14     } // end constructor
15
16     // display MessageBox when Button is clicked
17     private void evenMoreButton_Click(object sender, EventArgs e)
18     {
19        MessageBox.Show( "Visual C# is an OOP language",
20           "Even More", MessageBoxButtons.OK,
21           MessageBoxIcon.Information );
22     } // end method evenMoreButton_Click
23 } // end class VisualInheritanceTestForm
```





العناصر المخصصة User-Defined Controls

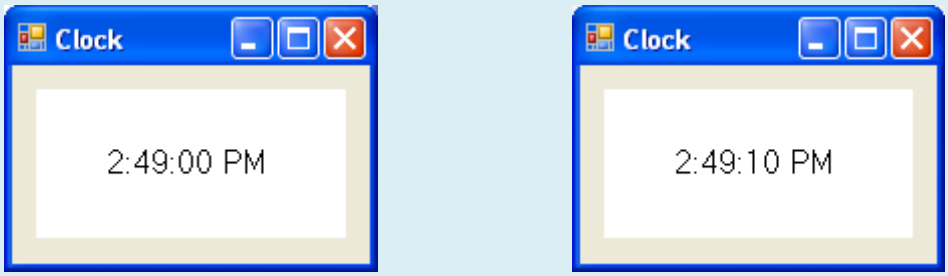
- تسمح منصة NET. بإنشاء عناصر تحكم إضافية مخصصة. تظهر هذه العناصر في صندوق الأدوات ToolBox ويمكن إضافتها إلى النماذج Forms واللوحات Panels ومربعات المجموعات GroupBoxes كما نضيف الأزرار واللصاقات وعناصر التحكم المعرفة مسبقاً.
- إن أسهل طريقة لتعريف عنصر تحكم جديد مخصص هي أن نشتقه من عنصر تحكم موجود سابقاً كاللصاقة مثلاً. يفيد هذا في حال كنا نحتاج إلى إضافة وظائف إلى عنصر موجود بدلاً من القيام بإعادة بناء العنصر كله من أجل إضافة بعض الوظائف. فيُمكن على سبيل المثال، أن نُنشئ نمطاً جديداً من اللصاقات التي تتصرف مثل اللصاقات العادية إلا أنها تختلف عنها في المظهر، ويُمكن تحقيق ذلك بالوراثة من الصف Label وتجاوز إعادة تعريف الطريقة OnPaint مثلاً.
- تمتلك كل عناصر التحكم الطريقة OnPaint والتي يستدعيها النظام عند لزوم إعادة رسم العنصر (في حال تغيير حجمه مثلاً)، ويمرر لهذه الطريقة غرض من النمط PaintEventArgs والذي يحوي معلومات خاصة بالرسم، فالخاصية Graphics تمثل الغرض الرسومي المستعمل للرسم، والخاصية ClipRectangle تعرف المنطقة مستطيلة الشكل التي تحدد حدود العنصر، عندما يقوم النظام برفع الحدث Paint يقوم الصف الأب لصفنا باستدعاء الطريقة OnPaint باستعمال تعدد الأشكال Polymorphism، وبما أنه لن يتم استدعاء الطريقة OnPaint الخاصة بالأب فيجب علينا القيام بذلك صراحة ضمن الحدث OnPaint الخاص بنا قبل كتابة كود رسم مخصص. يكون الهدف –في معظم الحالات– التأكد من تنفيذ كود الرسم الأصلي إضافة إلى الكود المخصص الجديد، أما إذا لم يكن من المطلوب السماح للأب بالرسم فلن نقوم باستدعاء للطريقة OnPaint الخاصة بالأب.
- نستعمل الصف UserControl لإضافة عنصر تحكم جديد يتركب من عدة عناصر تحكم موجودة، حيث تدعى العناصر المضافة بعناصر التأسيس constituent controls، يمكن للمبرمج على سبيل المثال بناء UserControl مكون من مربع نص ولصاقة وزر، كل منها مرتبط بوظيفة ما (يمكن أن يعمل الزر على نسخ قيمة النص ضمن مربع النص إلى اللصاقة)، يلعب العنصر UserControl دور الحاوية للعناصر المضافة إليه، إن العنصر UserControl يحوي عناصر التأسيس لذا فهو لا يحدد طريقة رسمها، كما أنه لا يمكن تجاوز الطريقة OnPaint له، لذا يجب استعمال الحدث Paint لكل عنصر للتحكم في مظهره، يمرر لمعالج هذا الحدث غرض من الصف PaintEventArgs والذي يمكن استعماله للرسم على عناصر التأسيس.
- كما يمكن للمبرمج أن يستعمل تقنية أخرى لبناء عنصر جديد كلياً بأن يرث من الصف Control، لا يحدد هذا الصف سلوكاً معيناً، وتترك لك هذه المهمة، وعوضاً عن ذلك يقوم الصف Control بمعالجة ما يتعلق بجميع عناصر التحكم، كالأحداث وتغيير الحجم، كما يجب أن تحوي الطريقة OnPaint استدعاءً للطريقة OnPaint الخاصة بالصف الأب، والتي تقوم باستدعاء معالج الحدث Paint، يجب أيضاً وضع كود يقوم بعملية رسم العنصر في الطريقة OnPaint الجديدة، تزيد هذه التقنية من المرونة ولكنها بحاجة إلى تخطيط أكبر، يوضح الجدول التالي الفرق بين الطرق المذكورة.

التقنية المتبعة لبناء عنصر مخصص	الوصف
الوراثة من عنصر تحكم معرف مسبقاً	يمكنك فعل ذلك لإضافة وظائف إلى عنصر تحكم موجود،عندما تعيد تعريف الطريقة OnPaint قم باستدعاء النسخة الخاصة بالصف الأب، لاحظ أنه يمكنك فقط الإضافة على المظهر الأصلي لعنصر التحكم و لا يمكنك إعادة تصميمه.
إنشاء UserControl	يمكنك إنشاء UserControl مكون من العديد من عناصر التحكم الموجودة لاحظ أنه لا يمكنك إعادة تعريف الطريقة OnPaint لهذا العنصر، بل يجب عليك أن تكتب كود الرسم في معالج الحدث Paint عوضاً عن ذلك، لاحظ أنه يمكنك فقط الإضافة على المظهر الأصلي لعنصر التحكم و لا يمكنك إعادة تصميمه.
الوراثة من الصف Control	لتعريف عنصر تحكم جديد بالكامل، قم بإعادة تعريف الطريقة OnPaint ثم استدعِ الطريقة OnPaint الخاصة بالصف الأب ثم اكتب كوداً لرسم عنصر التحكم، يمكنك باستعمال هذهالطريقة تحديد مظهر العنصر بشكل كامل.

الخاصية أو الحدث	الوصف
الخصائص	
Graphics	الغرض الرسومي للعنصر، يستعمل للرسم فوق العنصر.
ClipRectangle	تحدد المستطيل الذي يعرف حدود العنصر.

- نقوم في المثال التالي ببناء العنصر "ساعة" "Clock"، وهو UserControl يتألف من لصاقة Label ومؤقت Timer، حيث يتم تحديث نص اللصاقة ليمثل الوقت الحالي كلما رفع المؤقت حدثاً.

```
1  // ClockUserControl.cs
2  // User-defined control with a timer and a Label.
3  using System;
4  using System.Windows.Forms;
5
6  // UserControl that displays the time on a Label
7  public partial class ClockUserControl : UserControl
8  {
9     // default constructor
10    public ClockUserControl()
11    {
12       InitializeComponent();
13    } // end constructor
14
15    // update Label at every tick
16    private void clockTimer_Tick(object sender, EventArgs e)
17    {
18       // get current time (Now), convert to string
19       displayLabel.Text = DateTime.Now.ToLongTimeString();
20    } // end method clockTimer_Tick
21 } // end class ClockUserControl
```



- إن المؤقت Timer (في فضاء الأسماء System.Windows.Forms) هو عنصر غير مرئي يكمن في النموذج، ويولد الحدث Tick كل فترة محددة بالخاصية Interval مقدرة بالميللي ثانية milliseconds (جزء من ألف من الثانية).

- يحوي هذا التطبيق UserControl يدعى ClockUserControl ونموذجاً يقوم بعرضه، نبدأ بإنشاء Windows Application، ثم نختار

Project → Add User Control…

مما يؤدي إلى عرض مربع حوار يمكننا من تحديد نمط العنصر المنشود، ثم نسمي الملف (والصف) ClockUserControl، فيظهر هذا العنصر الحالي على شكل مستطيل رمادي.

- يمكن التعامل مع هذا العنصر كما لو كان نموذجاً، أي أننا نستطيع إضافة عناصر باستعمال صندوق الأدوات وتحديد الخصائص باستعمال نافذة الخصائص. إلا أننا بدلاً من بناء تطبيق نقوم ببناء عنصر جديد يتألف من عناصر أخرى، نضيف لصاقة (displayLabel) ومؤقتاً (clockTimer) إلى العنصر، ونضبط الخاصية Interval للمؤقت لتكون 1000 ميللي ثانية ونعطي قيمة للنص ضمن اللصاقة في معالج الحدث Tick للمؤقت (الأسطر من 16 إلى 20)، يجب أن يكون المؤقت فعالاً ليولد أحداثاً، وذلك عن طريق إعطاء الخاصية Enabled القيمة true في نافذة الخصائص.
- يحوي DateTime (في فضاء الأسماء System) الخاصية Now التي تمثل الوقت الحالي، تستعمل الطريقة ToLongTimeString لتحويل Now إلى سلسلة محرفية تحوي الساعة والدقيقة والثانية (مع أحد الرمزين AM وPM)، وقد استعملناها لتحديث النص في displayLabel في السطر 19.
- يظهر هذا العنصر في صندوق الأدوات بمجرد إنشائه، وقد يلزم الانتقال إلى النموذج قبل أن يظهر في صندوق الأدوات، لاستعمال هذا العنصر نقوم بسحبه وإفلاته على النموذج ونشغل التطبيق، لقد أعطينا العنصر ClockUserControl خلفية بيضاء ليظهر واضحاً على النموذج، وبما أنه لا يوجد معالجات أحداث فسنعرض كود العنصر ClockUserControl فقط.
- لبناء عنصر مخصص UserControl، يُمكن استخدامه في تطبيقات أخرى نتبع الخطوات التالية:

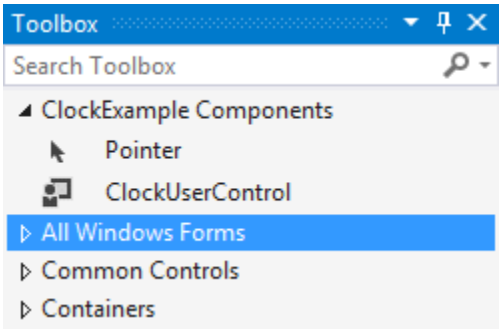
1. ننشئ مشروع مكتبة صفوف Class Library جديد.
2. نحذف الملف Class1.cs الذي يظهر.
3. ننقر الزر الأيمن على المشروع في Solution Explorer ونختار

Add → User Control…

ثم نسمي الملف وننقر Add.

4. نقوم بإضافة العناصر المكونة للعنصر.
5. نقوم ببناء Build المشروع، سيقوم Visual Studio بإنشاء ملف له اللاحقة dll. في مجلد الخرج.
6. ننشئ تطبيق جديد.
7. في التطبيق الجديد ننقر بالزر الأيمن على صندوق الأدوات ونختار Choose Items...، سيظهر مربع الحوار Choose ToolBox Items، ننقر Browse ونختار ملف dll. الناتج عن التطبيق الذي أنشئناه في الخطوات من 1 إلى 5، سيظهر بند في مربع الحوار Choose ToolBox Items نقوم

باختياره إذا لم يكن مختاراً، ننقر OK لإضافة العنصر إلى صندوق الأدوات. يمكن الآن إضافة هذا العنصر إلى النموذج مثل أي عنصر آخر.



الفصل الثامن: الرسوميات

عنوان الموضوع:

الرسوميات.

الكلمات المفتاحية:

صفوف الرسم، نظام الإحداثيات، التحكم بالألوان، خطوط النص، الخطوط، المستطيلات، الأشكال البيضوية، الأقواس، المضلعات.

ملخص:

نستعرض في هذا الفصل الأدوات المتوفرة في .NET لرسم الأشكال ثنائية الأبعاد، إضافةً إلى عمليات التحكم في الألوان والخطوط.

أهداف تعليمية:

يتعرف الطالب في هذا الفصل على:

- الصفوف التي توفرها اللغة للتعامل مع الرسوميات المتجهة.
- التعامل مع الألوان.
- التعامل مع خطوط النص.
- رسم الخطوط.
- رسم المستطيلات.
- رسم الأشكال البيضوية.
- رسم الأقواس.
- رسم المضلعات.

المخطط:

الرسوميات

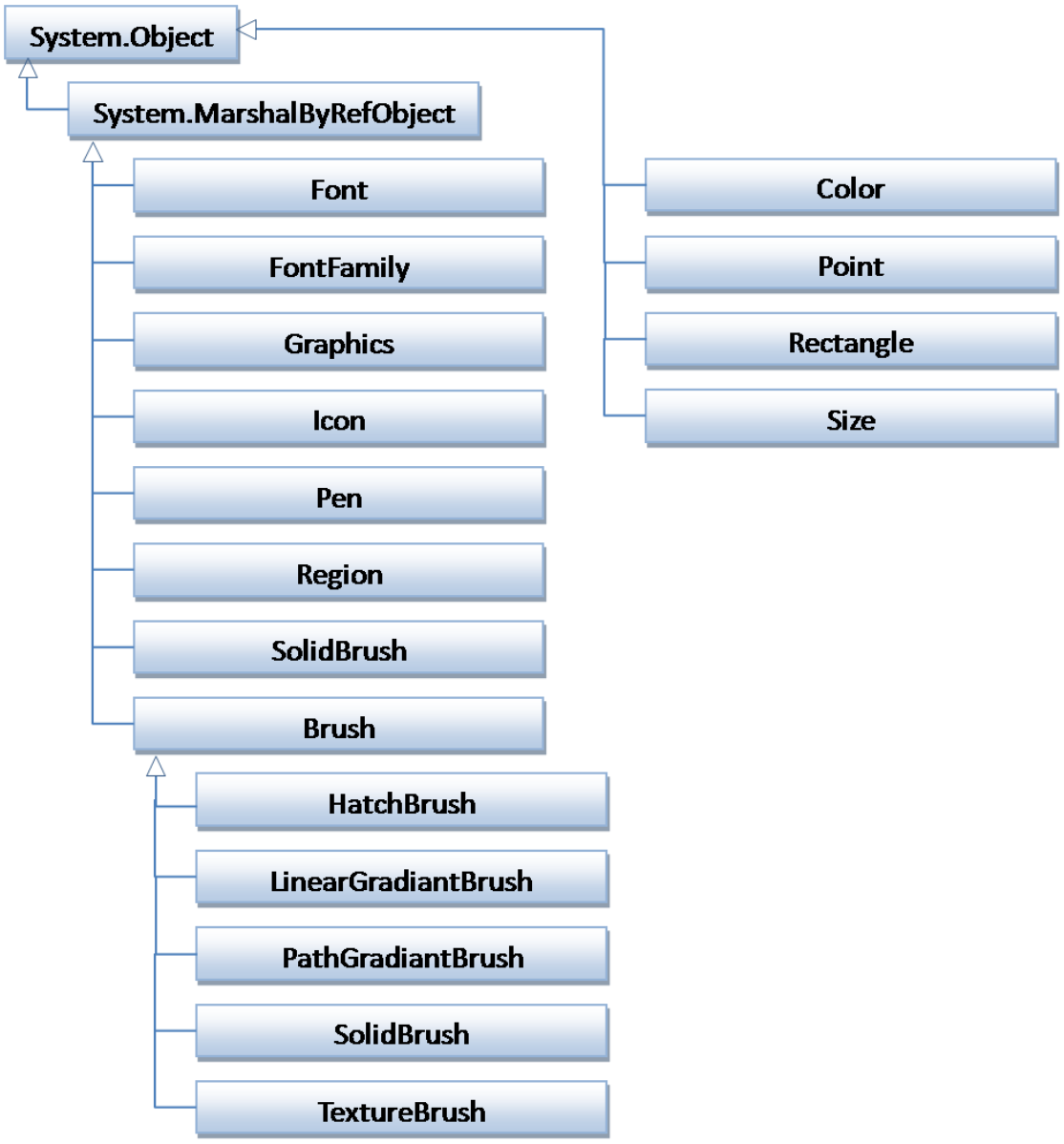
- 3 وحدات (Learning Objects)

مقدمة

- تدعم NET. الرسوميات، مما يساعد المبرمجين على تحسين الواجهات البيانية لتطبيقاتهم.
- تحوي مكتبة صفوف المنصة Framework Class Library (FCL) العديد من إمكانيات الرسم المتطورة كجزء من فضاء الأسماء System.Drawing وعدة فضاءات أخرى تشكل بمجموعها واجهة تصميم الرسوميات Graphics Design Interface (GDI) الخاص بمنصة NET.، وهو عبارة عن واجهة برمجة تطبيقات Application Programming Interface (API) تؤمن مجموعة من الصفوف لإنشاء رسوميات متجهة Vector Graphics (وهي طريقة لتوصيف الرسوميات بحيث يمكن معالجتها بسهولة وبتقنيات عالية الأداء)، ولمعالجة الخطوط والصور.

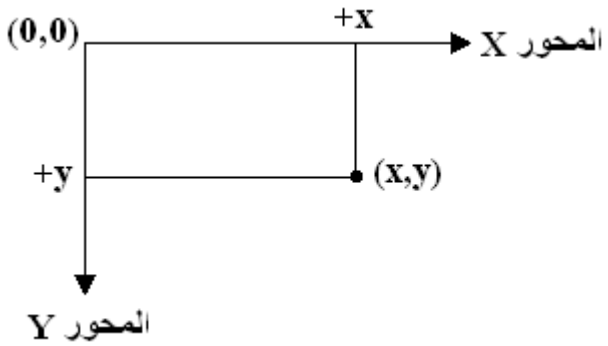
صفوف الرسم ونظام الإحداثيات

- يعرض المخطط التالي جزءاً من فضاء الأسماء System.Drawing والذي يحوي العديد من الصفوف والبنى التي يغطيها هذا الفصل. يحوي فضاءا الأسماء System.Drawing و System.Drawing.Drawing2D عناصر GDI الأكثر استعمالاً.



- يحوي الصف Graphics العديد من الطرائق لرسم السلاسل المحرفية Strings والخطوط Lines والمستطيلات Rectangles والعديد من الأشكال فوق عنصر ما. تحتاج طرائق الرسم في الصف Graphics عادةً إلى قلم Pen أو فرشاة Brush. لرسم شكل ما، يرسم القلم محيط الشكل، في حين ترسم الفرشاة شكلاً مصمتاً.
- تحوي البنية Color عدداً كبيراً من الخصائص الساكنة Static Properties تُمثّل ألوان العديد من العناصر الرسومية، بالإضافة إلى طريقة تسمح بتشكيل ألوان جديدة.
- كما يحوي الصف Font الخصائص التي تُعرّف خطوط الكتابة، ويحوي الصف FontFamily طرائق للحصول على معلومات عن خط الكتابة.
- قبل أن نبدأ بالرسم في C#، يجب أن نستعرض نظام الإحداثياتCoordinates System الخاص بـ GDI، وهو عبارة عن نظام يُعرّف كل نقطة على الشاشة.

- بشكل افتراضي، يكون للزاوية العليا اليسرى الإحداثيات (0،0). تحوي ثنائية الإحداثيات قيمتين: الإحداثي X–Coordinate X (الإحداثي الأفقي) والإحداثي Y–Coordinate Y (الإحداثي العمودي).
- يقيس الإحداثي X المسافة الأفقية –باتجاه اليمين– عن الزاوية العليا اليسرى، ويقيس الإحداثي Y المسافة العمودية –باتجاه الأسفل– عن الزاوية العليا اليسرى. يُعرّف المحور X كل الإحداثيات الأفقية، في حين يُعرّف المحور Y كل الإحداثيات العمودية. نستخدم في نظام الإحداثيات الواحدة "بيكسل" (pixel = picture element) وهي أصغر واحدة لقياس دقة Resolution شاشة العرض.
- يؤمن فضاء الأسماء System.Drawing العديد من البنى Structures التي تُعرّف القياسات والمواقع في نظام الإحداثيات.
- تُمثّل البنية Point إحداثيي نقطة في مستو. وتُحدد البنية Rectangle عرض وارتفاع منطقة مستطيلة الشكل. كما تُمثّل البنية Size عرض وارتفاع شكل ما.



سياقات وأغراض الرسم

- يُمثل سياق الرسم في C# سطحاً يسمح بالرسم على الشاشة. يُدير غرض من النمط Graphics سياق الرسم عن طريق التحكم بكيفية الرسم، حيث يحوي طرائق للرسم ومعالجة خط النص واللون والعديد من الأمور المتعلقة بالرسوميات.
- يرث كل صف يُشتق من System.Windows.Forms.Form الطريقة الظاهرية (Virtual) OnPaint والتي تتم فيها معظم العمليات الرسومية. تتضمن وسائط هذه الطريقة غرضاً من النمط PaintEventArgs والذي يُمكن أن نحصل منه على غرض Graphics خاص بالنموذج. يجب الحصول على الغرض Graphics عند كل استدعاء للطريقة لأن خصائص سياق الرسم التي يُمثلها هذا الغرض قد تتغير. تستدع الطريقة OnPaint الحدث Paint الخاص بالعنصر.
- يُمكننا عند الرسم على النموذج إعادة تعريف الطريقة OnPaint لجلب غرض من النمط Graphics من الوسيط PaintEventArgs أو لإنشاء غرض جديد يرتبط بالسطح المناسب، سنتحدث عن تقنيات الرسم هذه في C# فيما بعد ضمن هذا الفصل.
- لإعادة تعريف الطريقة الموروثة OnPaint نستعمل الترويسة التالية:

protected override void OnPaint (PaintEventArgs e)

ثم نستخرج غرض Graphics الوارد من الوسيط PaintEventArgs كما يلي:

Graphics graphicsObject = e.Graphics;

يُمكن الآن استعمال المتحول graphicsObject لرسم أشكال على النموذج.

- يؤدي استدعاء الطريقة OnPaint إلى رفع الحدث Paint، يُمكن للمبرمجين بدلاً من إعادة تعريف الطريقة OnPaint أن يضيفوا معالجاً للحدث Paint، يولد Visual Studio معالج الحدث Paint على الشكل التالي:

protected void MyEventHandler_Paint(
object sender, PaintEventArgs e)

- يندر أن يقوم المبرمجون باستدعاء الطريقة OnPaint بشكل مباشر، لأن عملية الرسم توصف بأنها "مقادة بالأحداث" Event-Driven، حيث يستدعي حدث ما -مثل تغطية و كشف و تغيير حجم النافذة- الطريقة OnPaint الخاصة بالنموذج. وبشكل مشابه، يتم استدعاء الطريقة OnPaint عند عرض أي عنصر كمربع نص TextBox أو لصاقة Label.
- يُمكن استدعاء الطريقة OnPaint بشكل إجباري وذلك باستدعاء الطريقة Invalidate للعنصر. تُحدّث هذه الطريقة العنصر و تُعيد رسم عناصره الرسومية. يحوي الصف Control العديد من نسخ الطريقة Invalidate التي تسمح للمبرمجين بتحديث أجزاء العنصر.
- لا تملك عناصر التحكم -كاللصاقات و الأزرار- سياقات رسم خاصة فيها، إلا أنه يُمكن إنشاء هذه السياقات. فللرسم على عنصر تحكم، نقوم أولاً بإنشاء غرض Graphics باستدعاء الطريقة CreateGraphics الخاصة بالعنصر كما يلي:

Graphics graphicsObject = controlName.CreateGraphics();

- ثم يُمكن استعمال الطرائق المتوفرة في الصف Graphics للرسم على العنصر.

التحكم باللون

- تُحسّن الألوان من مظهر الواجهات وتساعد على توضيح المعنى. تُعرّف البنية Color طرائق وثوابت Constants تُستعمل لمعالجة الألوان.
- يُمكن إنشاء أي لون عن طريق توليفة من المكونات: ألفا Alpha، أحمر Red، أخضر Green، أزرق Blue (تُدعى قيم ARGB). تأخذ كل من هذه المكونات الأربعة ARGB قيمة من النمط byte تُمثل قيماً صحيحة بين 0 و 255. تُحدد القيمة alpha شفافية اللون، حيث تُمثل القيمة 0 لوناً شفافاً Transparent و تُمثل القيمة 255 لوناً غير شفاف Opaque، أما القيم بين 0 و 255 فتؤدي إلى تأثير مزج موزون بين الألوان RGB مع اللون الخلفي لتعطي تأثيراً شبه شفاف SemiTransparent. يُعرّف الرقم الأول من RGB درجة الأحمر ضمن اللون، ويُعرّف الرقم الثاني درجة الأخضر، ويُعرّف الثالث درجة الأخضر. كلما كبرت هذه الأرقام كلما زادت درجة اللون الموافق.
- تسمح C# للمبرمجين بالاختيار بين ما يقارب 17 مليون لون، وإذا كان لا يُمكن لحاسوب ما أن يظهر كل هذه الألوان فإنه يستعمل اللون الأقرب. يوضح الجدول التالي بعض ثوابت الألوان مسبقة التعريف (وكلها عامة public وساكنة static).

اللون	قيمة RGB	اللون	قيمة RGB
Orange	255,200,0	White	255,255,255
Pink	255,175,175	Gray	128,128,128
Cyan	0,255,255	DarkGray	64,64,64
Magenta	255,0,255	Red	255,0,0
Yellow	255,255,0	Green	0,255,0
Black	0,0,0	Blue	0,0,255

- يوضح الجدول التالي أهم خصائص وطرائق البنية Color، للحصول على قائمة كاملة بثوابت الألوان مسبقة التعريف و خصائص وطرائق البنية Color يُمكن الاطلاع على توثيق البنية Color على الرابط:

msdn2.microsoft.com/en-us/library/system.drawing.color

الخاصية أو الطريقة	الوصف
الطرائق	
FromArgb	طريقة ساكنة static تقوم بإنشاء لون بالاعتماد على قيم المكونات أحمر red، أخضر green، أزرق blue على شكل أعداد صحيحة بين 0 و255. كما تسمح نسخة أخرى من هذه الطريقة بتحديد ألفا alpha، أحمر red، أخضر green، أزرق blue.
FromName	طريقة ساكنة static تقوم بإنشاء لون بالاعتماد على اسمه الممرر على شكل سلسلة محرفية.
الخصائص	
A	قيمة من النمط byte (بين 0 و 255) تمثل المكون ألفا alpha.
R	قيمة من النمط byte (بين 0 و 255) تمثل المكون أحمر red.
G	قيمة من النمط byte (بين 0 و 255) تمثل المكون أخضر green.
B	قيمة من النمط byte (بين 0 و 255) تمثل المكون أزرق blue.

- يوجد نسختين للطريقة FromArgb، تأخذ إحداها ثلاثة أعداد صحيحة، وتأخذ الأخرى أربعة أعداد صحيحة (يجب أن تكون جميعها بين 0 و 255). حيث تأخذ كلاهما ثلاثة وسائط تحوي قيم الألوان الأحمر والأخضر والأزرق، كما تسمح النسخة الثانية بتحديد المكون ألفا، حيث تعطي النسخة الأولى القيمة 255 للمكون ألفا (غير شفاف)، وتُعيد كلا الطريقتين غرضاً من النمط Color. تُعيد الخصائص A,R,G,B قيم المكونات alpha, red, green, blue على الترتيب.
- يقوم المبرمجون برسم الخطوط والسلاسل المحرفية باستعمال الفراشي Brushes والأقلام Pens. يُستعمل القلم Pen –والذي يعمل بطريقة مشابهة للقلم الحقيقي– لرسم الخطوط، وتحتاج معظم طرائق الرسم إلى غرض من النمط Pen، حيث تسمح النسخ المختلفة لتابع البناء الخاص بالنمط Pen بتحديد لون وعرض الخطوط المراد رسمها، كما يحوي فضاء الأسماء System.Drawing الصف Pens الذي يحوي العديد من الأقلام مسبقة التعريف.
- تُعرّف كل الصفوف التي تُشتق من الصف المجرد Brush abstract class أغراضاً تُستعمل في تلوين الجزء الداخلي من الشكل. فعلى سبيل المثال، يأخذ التابع البناء للصف SolidBrush غرضاً من

النمط Color لتحديد اللون المستخدم في الملء. تقوم الفراشي في معظم الطرائق Fill بملء المساحة بلون أو عينة Pattern أو صورة، يلخص الجدول التالي بعض أنواع الفراشي واستعمالاتها.

الصف	الوصف
HatchBrush	تقوم بملء المساحة باستعمال عينة Pattern، يتم تعريف العينة كأحد قيم نمط التعداد HatchStyle وباللون الأمامي (الذي ترسم به العينة) واللون الخلفي.
LinearGradientBrush	تقوم بملء المساحة بمزيج متدرج بين لونين، وتعرف هذه الفرشاة على قطعة مستقيمة، حيث يتم تعريفها بتحديد اللونين وزاوية التدرج وإما عرض مستطيل أو نقطتين.
SolidBrush	تقوم بملء المساحة بلون واحد.
TextureBrush	تقوم بملء المساحة بتكرار صورة.

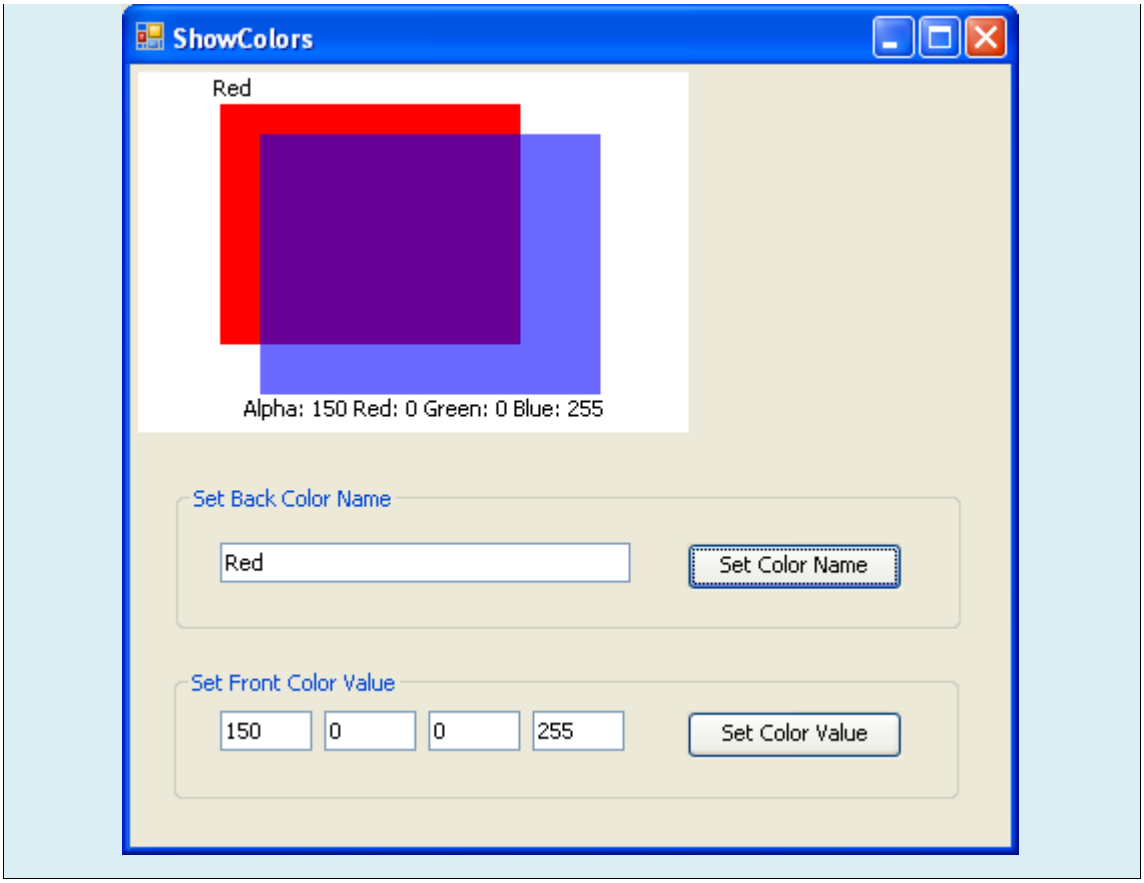
معالجة الألوان

- يوضح المثال التالي كيفية استعمال الطرائق والخصائص الموضحة في الجدول السابق، حيث يظهر مستطيلين متداخلين تسمح للمستخدم بتجريب قيم الألوان وأسماءها وقيم ألفا (للشفافية).

```
1  // ShowColors.cs
2  // Color value and alpha demonstration.
3  using System;
4  using System.Drawing;
5  using System.Windows.Forms;
6
7  public partial class ShowColors : Form
8  {
9     // color for back rectangle
10    private Color backColor = Color.Wheat;
11
12    // color for front rectangle
13    private Color frontColor = Color.FromArgb( 100, 0, 0, 255 );
14
15    // default constructor
```

```
16    public ShowColors()
17    {
18       InitializeComponent();
19    } // end constructor
20
21    // override Form OnPaint method
22    protected override void OnPaint( PaintEventArgs e )
23    {
24       Graphics graphicsObject = e.Graphics; // get graphics
25
26       // create text brush
27       SolidBrush textBrush = new SolidBrush( Color.Black );
28
29       // create solid brush
30       SolidBrush brush = new SolidBrush( Color.White );
31
32       // draw white background
33       graphicsObject.FillRectangle( brush, 4, 4, 275, 180 );
34
35       // display name of backColor
36       graphicsObject.DrawString( backColor.Name, this.Font,
37          textBrush, 40, 5 );
38
39       // set brush color and display back rectangle
40       brush.Color = backColor;
41       graphicsObject.FillRectangle( brush, 45, 20, 150, 120 );
42
43       // display Argb values of front color
44       graphicsObject.DrawString( "Alpha: " + frontColor.A +
45          " Red: " + frontColor.R + " Green: " + frontColor.G +
46          " Blue: " + frontColor.B, this.Font, textBrush, 55, 165);
47
48       // set brush color and display front rectangle
49       brush.Color = frontColor;
```

```
50          graphicsObject.FillRectangle( brush, 65, 35, 170, 130 );
51       } // end method OnPaint
52
53       // handle colorNameButton click event
54       private void colorNameButton_Click( object sender,EventArgs e )
55       {
56          // set backColor to color specified in text box
57          backColor = Color.FromName( colorNameTextBox.Text );
58
59          Invalidate(); // refresh Form
60       } // end method colorNameButton_Click
61
62       // handle colorValueButton click event
63       private void colorValueButton_Click( object sender,EventArgs e)
64       {
65          // obtain new front color from text boxes
66          frontColor = Color.FromArgb(
67             Convert.ToInt32( alphaTextBox.Text ),
68             Convert.ToInt32( redTextBox.Text ),
69             Convert.ToInt32( greenTextBox.Text ),
70             Convert.ToInt32( blueTextBox.Text ) );
71
72          Invalidate(); // refresh Form
73       } // end method colorValueButton_Click
74    } // end class ShowColors
```



- يظهر النموذج عند بداية تشغيل البرنامج، ويؤدي ذلك إلى استدعاء الطريقة OnPaint لرسم محتويات النموذج. يستخرج السطر 24 مؤشر على الغرض Graphics من الوسيط PaintEventArgs ويُسنده إلى graphicsObject، ثم ينشئ السطران 27 و30 فرشاتان من النمط SolidBrush إحداهما بيضاء والأخرى سوداء لرسم أشكال مصمتة Solid على النموذج، وبما أن الصف SolidBrush مشتق من الصف المجرد Brush فيُمكن تمريره لأي طريقة تأخذ وسيطاً من النمط Brush.
- يستعمل السطر 33 الطريقة FillRectangle لترسم مستطيلاً مصمتاً بلون أبيض باستعمال الفرشاة التي أنشأناها في السطر 30، تأخذ الطريقة FillRectangle فرشاة وإحداثيي الزاوية العليا اليسرى للمستطيل المراد رسمه وعرضه وارتفاعه، ويقوم السطران 36 و37 بعرض الخاصية Name للون backColor باستعمال الطريقة DrawString. هناك العديد من النسخ لهذه الطريقة، حيث يوضح المثال نسخة تأخذ السلسلة المحرفية المراد عرضها وخط النص وإحداثيي أول حرف في النص.
- يقوم السطر 40 بإسناد القيمة backColor للون الفرشاة brush (الخاصية Color) ثم يقوم السطر 41 برسم مستطيل باستعمال الفرشاة، تقوم الأسطر من 44 إلى 46 باستخراج القيم ARGB من frontColor وترسم سلسلة محرفية تحوي هذه القيم، يقوم السطر 49 بإسناد القيمة frontColor للون الفرشاة brush ثم يقوم السطر 50 برسم مستطيل يتداخل مع المستطيل السابق باستعمال الفرشاة.
- يستعمل معالج الحدث colorNameButton_Click (الأسطر من 54 إلى 60) الطريقة الساكنة FormName الخاصة بالبنية Color لإنشاء غرض جديد من النمط Color بناء على colorName

الذي قام المستخدم بإدخاله في مربع النص، ثم يسند هذه القيمة إلى المتحول backColor (السطر 57)، بعدها يستدعي السطر 59 الطريقة Invalidate الخاصة بالنموذج مشيراً إلى ضرورة إعادة رسم النموذج، مما يؤدي إلى استدعاء الطريقة OnPaint ليتم تحديث النموذج.

- يستعمل معالج الحدث colorValueButton_Click (الأسطر من 63 إلى 73) الطريقة الساكنة FormArgb الخاصة بالبنية Color لإنشاء غرض جديد من النمط Color بناء على قيم ARGB التي قام المستخدم بإدخالها في مربعات النص، ثم يسند هذه القيمة إلى المتحول frontColor، بعدها يستدعي السطر 72 الطريقة Invalidate الخاصة بالنموذج مشيراً إلى ضرورة إعادة رسم النموذج، مما يؤدي إلى استدعاء الطريقة OnPaint ليتم تحديث النموذج.
- إذا أسند المستخدم للقيمة ألفا رقماً بين 0 و 255 (للون frontColor) فسيظهر تأثير المزج بين اللونين، في الخرج الموضح للمثال يمتزج اللون الأحمر للمستطيل الخلفي مع الأزرق للمسطيل الأمامي ليظهر اللون البنفسجي في منطقة التداخل.
- لاحظ أنه ليس بالإمكان تغيير خصائص غرض من النمط Color، بل نحتاج لإنشاء غرض جديد من النمط Object لنستعمل لوناً جديداً.

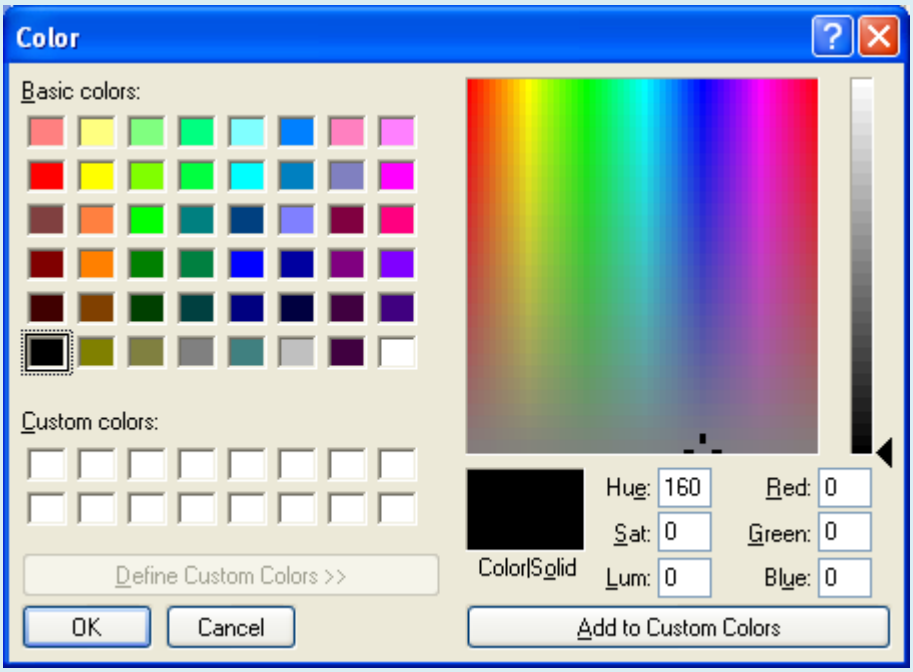
استعمال مربع حوار الألوان ColorDialog لاختيار لون من لوحة الألوان Color Palette

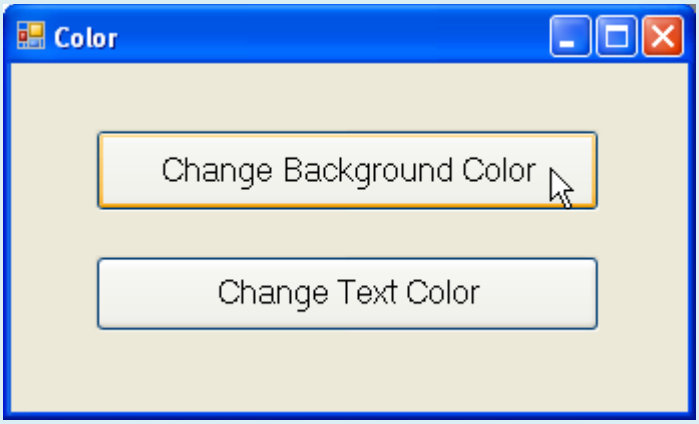
- يسمح عنصر الواجهة المُعرّف ColorDialog للمستخدم باختيار لون مُعرّف في لوحة الألوان Color Palette أو تعريف لون مخصص. عندما ينقر المستخدم الزر OK يحصل البرنامج على اللون الذي اختاره المستخدم عن طريق الخاصية Color لمربع الحوار ColorDialog.

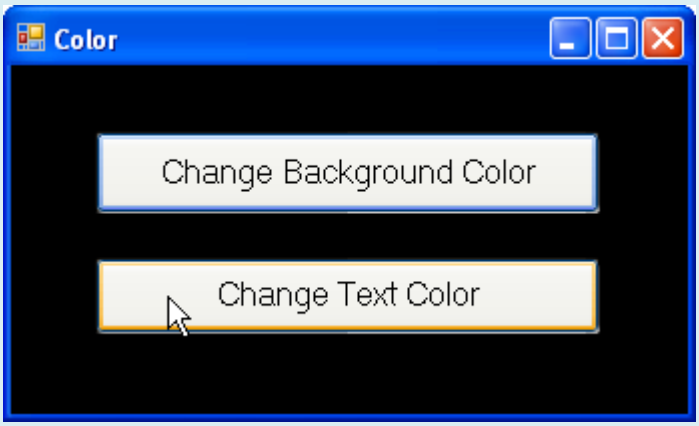
```
1  // ShowColorsComplex.cs
2  // ColorDialog used to change background and text color.
3  using System;
4  using System.Drawing;
5  using System.Windows.Forms;
6
7  // allows users to change colors using a ColorDialog
8  public partial class ShowColorsComplex : Form
9  {
10    // create ColorDialog object
11    private static ColorDialog colorChooser = new ColorDialog();
12
```

```
13    // default constructor
14    public ShowColorsComplex()
15    {
16       InitializeComponent();
17    } // end constructor
18
19    // change text color
20    private void textColorButton_Click( object sender, EventArgs e)
21    {
22       // get chosen color
23       DialogResult result = colorChooser.ShowDialog();
24
25       if ( result == DialogResult.Cancel )
26          return;
27
28       // assign forecolor to result of dialog
29       backgroundColorButton.ForeColor = colorChooser.Color;
30       textColorButton.ForeColor = colorChooser.Color;
31    } // end method textColorButton_Click
32
33    // change background color
34    private void backgroundColorButton_Click( object sender,
EventArgs e )
35    {
36       // show ColorDialog and get result
37       colorChooser.FullOpen = true;
38       DialogResult result = colorChooser.ShowDialog();
39
40       if ( result == DialogResult.Cancel )
41          return;
42
43       // set background color
44       this.BackColor = colorChooser.Color;
45    } // end method backgroundColorButton_Click
```

```
46 } // end class ShowColorsComplex
```







- تحوي واجهة هذا التطبيق زرين، يسمح الزر backgroundColorButton للمستخدم بتغيير اللون الخلفي للنموذج، ويسمح الزر textColorButton بتغيير لون النص ضمن الأزرار. يقوم السطر 11 بإنشاء ColorDialog اسمه colorChooser، والذي يُستعمل في معالجي حدث الزرين.
- تُعرّف الأسطر من 20 إلى 31 معالج حدث الزر textColorButton والذي يُظهر مربع الحوار colorChooser باستدعاء الطريقة ShowDialog (السطر 23)، تحوي الخاصية Color لمربع الحوار على اللون الذي اختاره المستخدم، حيث يقوم السطران 29 و30 بإسناده إلى لون النص في كلا الزرين.
- تُعرّف الأسطر من 34 إلى 45 معالج حدث الزر backgroundColorButton الذي يقوم بتعديل اللون الخلفي للنموذج عن طريق إسناد الخاصية Color لمربع الحوار إلى الخاصية BackColor للنموذج (السطر 44). يُسند السطر 37 القيمة true إلى الخاصية FullOpen لمربع الحوار مما يؤدي إلى عرض كل الألوان الممكنة في مربع الحوار كما يظهر في الشكل، عندما تكون قيمة هذه الخاصية false تظهر عينات الألوان فقط.
- لا ينحصر اختيار المستخدم في 48 لوناً التي تظهر في العينات، إذ يُمكن للمستخدم أن ينشئ لوناً مخصصاً بالنقر على أي مكان في المستطيل الكبير ضمن مربع الحوار والذي يحوي طيفاً واسعاً من الألوان المتدرجة، ثم يُمكنه معايرة القيم اللونية المختلفة لتحسين اللون، وعند الانتهاء ينقر الزر Add to Custom Colors الذي يضيف اللون المخصص إلى مربع ضمن قسم الألوان المخصصة، ثم يضغط OK ليختار ذلك اللون.

التحكم بخط النص

- لا يُمكن تعديل خصائص غرض من النمط Font، لذا يجب إنشاء غرض جديد من النمط Font عند الحاجة إلى خط جديد. هناك العديد من النسخ للتابع البناء في النمط Font. يوضح الجدول التالي بعض خصائص الصف Font.

الخاصية	الوصف
Bold	تعيد true إذا كان الخط عريضاً.
FontFamily	تعيد عائلة الخط FontFamily، بنية لتجميع خصائص الخطوط المتماثلة.
Height	تعيد ارتفاع الخط.
Italic	تعيد true إذا كان الخط مائلاً.
Name	تعيد اسم الخط على شكل سلسلة محرفية.
Size	تعيد قيمة من النمط float تعبر عن حجم الخط مقاساً بواحدات التصميم (واحدات لقياس حجم الخط).
SizeInPoints	تعيد قيمة من النمط float تعبر عن حجم الخط مقاساً بالنقط.
Strikeout	تعيد true إذا كان الخط يتوسطه خط.
Underline	تعيد true إذا كان الخط مسطراً تحته.

- لاحظ أن الخاصية Size لنمط Font تعيد حجم الخط مقدراً بوحدات التصميم Design Units، أما SizeInPoints فتُعيد حجم الخط مقاساً بالنقط Points وهو الأكثر شيوعاً. تسمح وحدات التصميم بتحديد حجم الخط بواحدة من العديد من واحدات القياس، مثل البوصات inches أو الميلليمترات millimeters. تأخذ بعض نسخ البناء في الصف Font وسيطاً من النمط GraphicsUnit، وهو نمط تعداد يسمح بتحديد واحدة قياس حجم الخط، ويأخذ هذا النمط القيم Point (1/72 بوصة)، Display (1/75 بوصة)، Document (1/300 بوصة)، Millimeter، Inch، Pixel.
- إذا تم تمرير هذا الوسيط فستحوي القيمة Size حجم الخط مقدراً بواحدة التصميم الممررة، وستحوي الخاصية SizePoints حجم الخط مقدراً بالنقاط Points. فعلى سبيل المثال، إذا أنشأنا خطاً حجمه 1 وقمنا بتحديد واحدة القياس GraphicsUnit.Inch، فستحوي الخاصية Size القيمة 1 والخاصية SizeInPoints القيمة 72، لأنه يوجد 72 نقطة في البوصة الواحدة. عادةً، تكون واحدة القياس الافتراضية هي GraphicsUnit.Point ولذا تكون الخاصيتان Size و SizeInPoints متساويتين.
- يحوي الصف Font العديد من توابع البناء، يحتاج معظمها إلى اسم الخط وهو سلسلة محرفية تُمثل خطاً يدعمه النظام. من بعض هذه الخطوط هي Microsoft SansSerif و Serif. كما يحتاج معظم توابع البناء إلى حجم الخط Size ونمطه FontStyle كوسطاء. تُحدد قيمة FontStyle باستعمال ثابت من نمط التعداد FontStyle (عريض Bold، مائل Italic، عادي Regular، يتوسطه خط

Strikeout، ومسطر تحته Underline أو توليفة من هذه القيم)، يُمكن ضم هذه القيم باستعمال المعامل | (أو OR)، كما في:

FontStyle.Italic | FontStyle.Bold

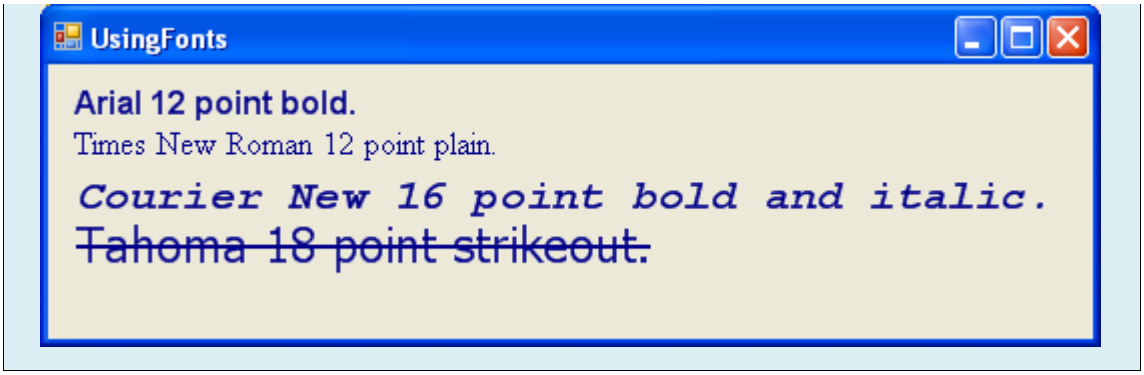
مما يجعل الخط مائلاً وعريضاً في نفس الوقت. تُحدد الطريقة DrawString في النمط Graphics الخط المستعمل في رسم السلسلة المحرفية بحسب الوسيط Font الممرر لها.

رسم سلاسل محرفية باستعمال أكثر من خط نص

- يعرض المثال التالي نصوصاً باستعمال العديد من الخطوط والقياسات، حيث يستعمل التابع البناء للصف Font لتهيئة الخطوط (الأسطر 24 و 28 و 32 و 36). يُمرر كل استدعاء لتابع بناء الخط اسم الخط (مثل Arial, Time New Roman, Courier New, Tahoma) على شكل سلسلة محرفية وحجم الخط (من النمط float) وغرض من النمط FontStyle.
- تقوم الطريقة DrawString في الصف Graphics بتحديد الخط وترسم النص في الموقع المحدد، لاحظ أن السطر 20 يُنشئ فرشاة SolidBrush ذات لون أزرق الغامق DarkBlue ، جميع النصوص التي ترسم باستعمال هذه الفرشاة تظهر باللون الأزرق الغامق.

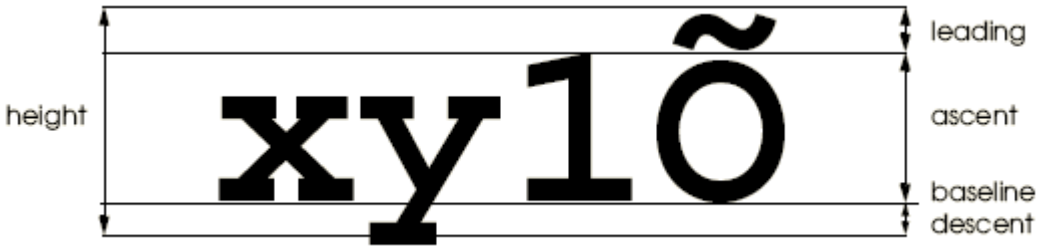
```
1  // UsingFonts.cs
2  // Fonts and FontStyles.
3  using System;
4  using System.Drawing;
5  using System.Windows.Forms;
6
7  // demonstrate font constructors and properties
8  public partial class UsingFonts : Form
9  {
10    // default constructor
11    public UsingFonts()
12    {
13       InitializeComponent();
14    } // end constructor
15
16    // demonstrate various font and style settings
17    protected override void OnPaint( PaintEventArgs paintEvent )
18    {
```

```
19        Graphics graphicsObject = paintEvent.Graphics;
20        SolidBrush brush = new SolidBrush( Color.DarkBlue );
21
22        // arial, 12 pt bold
23        FontStyle style = FontStyle.Bold;
24        Font arial = new Font( "Arial" , 12, style );
25
26        // times new roman, 12 pt regular
27        style = FontStyle.Regular;
28        Font timesNewRoman = new Font("Times New Roman", 12, style);
29
30        // courier new, 16 pt bold and italic
31        style = FontStyle.Bold | FontStyle.Italic;
32        Font courierNew = new Font( "Courier New", 16, style );
33
34        // tahoma, 18 pt strikeout
35        style = FontStyle.Strikeout;
36        Font tahoma = new Font( "Tahoma", 18, style );
37
38        graphicsObject.DrawString( arial.Name +
39           " 12 point bold.", arial, brush, 10, 10 );
40
41        graphicsObject.DrawString( timesNewRoman.Name +
42           " 12 point plain.", timesNewRoman, brush, 10, 30 );
43
44        graphicsObject.DrawString( courierNew.Name +
45           " 16 point bold and italic.", courierNew,
46           brush, 10, 54 );
47
48        graphicsObject.DrawString( tahoma.Name +
49           " 18 point strikeout.", tahoma, brush, 10, 75 );
50     } // end method OnPaint
51 } // end class UsingFonts
```



معايير خط النص Font Metrics

- يُمكن تحديد معلومات دقيقة لمعايير الخط (خصائصه) مثل الارتفاع height، والهامش السفلي descent (المقدار الذي ينزل إليه الحرف تحت الخط القاعدي baseline)، الهامش العلوي ascent (المقدار الذي يرتفع إليه الحرف فوق الخط القاعدي baseline)، والتباعد leading (المسافة بين الهامش العلوي للسطر و الهامش السفلي للسطر السابق). يوضح الشكل التالي معايير الخط.



- يُعرّف الصف FontFamily خصائص مشتركة لمجموعة خطوط مرتبطة، يوفر هذا الصف عدة طرائق تُستعمل لتحديد معايير الخط المشتركة بين أعضاء هذه العائلة. يوضح الجدول التالي هذه الطرائق:

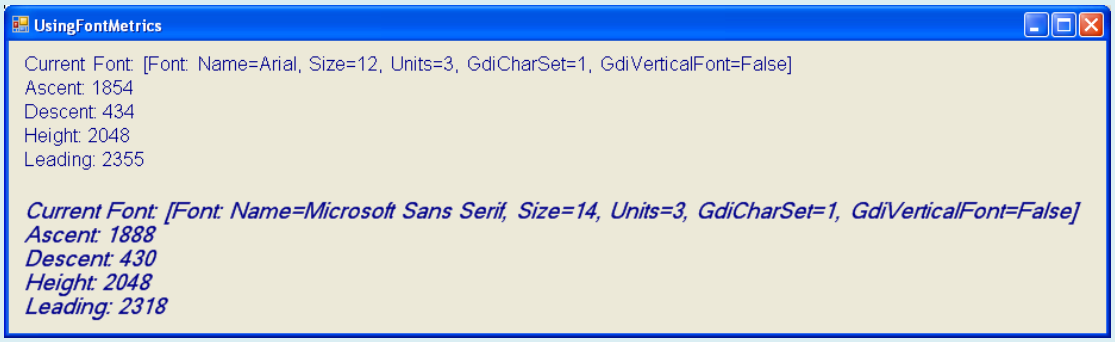
الطريقة	الوصف
GetCellAscent	تُعيد عدداً صحيحاً يمثل الهامش العلوي ascent مقدراً بواحدات التصميم.
GetCellDescent	تُعيد عدداً صحيحاً يمثل الهامش السفلي descent مقدراً بواحدات التصميم.
GetEmHeight	تُعيد عدداً صحيحاً يمثل ارتفاع الخط height مقدراً بواحدات التصميم.
GetLineSpacing	تُعيد عدداً صحيحاً يمثل التباعد leading مقدراً بواحدات التصميم (وهو مقدار المسافة بين سطرين متتاليين).

- يوضح المثال التالي معايير خطين مختلفين. يُنشئ السطر 23 غرضاً من النمط Font يسمى arial بقياس 12 نقطة والخط Arial، يستعمل السطر 24 الخاصية FontFamily للغرض arial للحصول على الغرض FontFamily الموافق، ويعرض السطرين27 و 28 نصاً يمثل هذا الخط، ثم تستعمل الأسطر من 30 إلى 44 طرائق الصف FontFamily للحصول على الهامش العلوي ascent والهامش السفلي descent والارتفاع height والتباعد leading، ثم ترسم سلاسل محرفية تعرض هذه المعلومات، وتكرر الأسطر من 47 إلى 68 هذه العملية من أجل الخط sanSerif وهو خط مشتق من العائلة MS Sans Serif.

```
1  // UsingFontMetrics.cs
2  // Displaying font metric information
3  using System;
4  using System.Drawing;
5  using System.Windows.Forms;
6
7  // display font information
8  public partial class UsingFontMetrics : Form
9  {
10   // default constructor
11    public UsingFontMetrics()
12    {
13       InitializeComponent();
14    } // end constructor
15
16    // displays font information
17    protected override void OnPaint( PaintEventArgs paintEvent )
18    {
19       Graphics graphicsObject = paintEvent.Graphics;
20       SolidBrush brush = new SolidBrush( Color.DarkBlue );
21
22       // Arial font metrics
23       Font arial = new Font( "Arial", 12 );
24       FontFamily family = arial.FontFamily;
25
26       // display Arial font metrics
27       graphicsObject.DrawString( "Current Font: " +
```

```
28          arial, arial, brush, 10, 10 );
29
30       graphicsObject.DrawString( "Ascent: " +
31          family.GetCellAscent( FontStyle.Regular ), arial,
32          brush, 10, 30 );
33
34       graphicsObject.DrawString( "Descent: " +
35          family.GetCellDescent( FontStyle.Regular ), arial,
36          brush, 10, 50 );
37
38       graphicsObject.DrawString( "Height: " +
39          family.GetEmHeight( FontStyle.Regular ), arial,
40          brush, 10, 70 );
41
42       graphicsObject.DrawString( "Leading: " +
43          family.GetLineSpacing( FontStyle.Regular ), arial,
44          brush, 10, 90 );
45
46       // display Sans Serif font metrics
47       Font sanSerif = new Font( "Microsoft Sans Serif",
48          14, FontStyle.Italic );
49       family = sanSerif.FontFamily;
50
51       graphicsObject.DrawString( "Current Font: " +
52          sanSerif, sanSerif, brush, 10, 130 );
53
54       graphicsObject.DrawString( "Ascent: " +
55          family.GetCellAscent( FontStyle.Regular ), sanSerif,
56          brush, 10, 150 );
57
58       graphicsObject.DrawString( "Descent: " +
59          family.GetCellDescent( FontStyle.Regular ), sanSerif,
60          brush, 10, 170 );
61
```

```
62          graphicsObject.DrawString( "Height: " +
63             family.GetEmHeight( FontStyle.Regular ), sanSerif,
64             brush, 10, 190 );
65
66          graphicsObject.DrawString( "Leading: " +
67             family.GetLineSpacing( FontStyle.Regular ), sanSerif,
68             brush, 10, 210 );
69       } // end method OnPaint
70 } // end class UsingFontMetrics
```



رسم الخطوط والمستطيلات والأشكال البيضوية

- تحتاج الطرائق التي ترسم أشكالاً مجوفة Hollow إلى قلم Pen وأربعة أعداد صحيحة تُمرر كوسائط، في حين تحتاج الطرائق التي ترسم أشكالاً مصمتة Solid إلى فرشاة Brush وأربعة أعداد صحيحة تُمرر كوسائط، يُمثل أول عددان إحداثيي الزاوية العليا اليسرى من الشكل (أو المنطقة الحاوية له)، ويُمثل الآخران عرض وارتفاع الشكل (أو المنطقة الحاوية له).
- يوضح الجدول التالي عدة طرائق في الصف Graphics مع وسائطها، لأغلب هذه الطرائق نسخ أخرى يُمكن مراجعتها على الرابط:

msdn2.microsoft.com/en-us/library/system.drawing.graphics

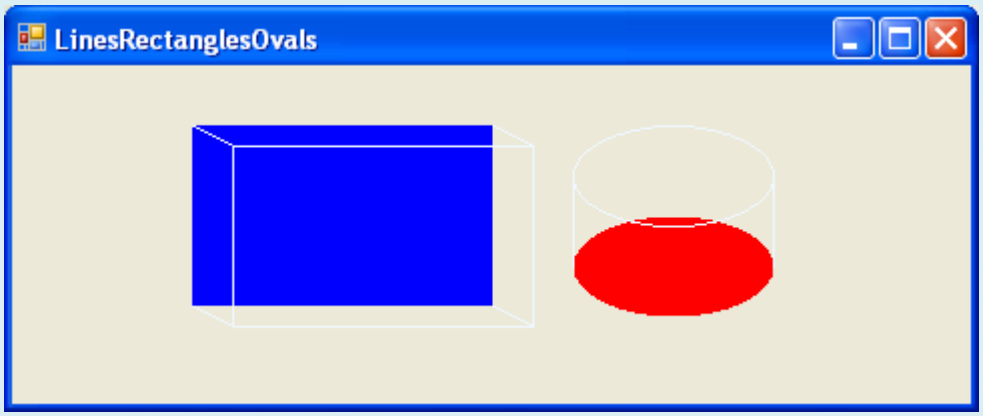
طرائق الرسم في الصف Graphics
DrawLine(Pen p, int x1, int y1, int x2, int y2) لون الخط ونمطه وعرضه.Pen)، حيث يُحدد (x2,y2 و (x1,y1ترسم خطاً مستقيماً بين النقطتين (
DrawRectangle(Pen p, int x, int y, int width, int height) لون Pen)، يُحدد x,yترسم مستطيلاً بالعرض والارتفاع المحددين، حيث أن الزاوية العليا اليسرى هي (المستطيل ونمط وعرض محيطه.
FillRectangle(Brush b, int x, int y, int width, int height))، تُحدد x,yترسم مستطيلاً مصمتاً بالعرض والارتفاع المحددين، حيث أن الزاوية العليا اليسرى هي (نمط ملء المستطيل.Brush
DrawEllipse(Pen p, int x, int y, int width, int height) ترسم قطعاً ناقصاً يحدده مستطيل بالعرض والارتفاع المحددين، حيث أن الزاوية العليا اليسرى للمستطيل لون القطع ونمط وعرض محيطه.Pen)، يُحدد x,yهي (
FillEllipse(Brush b, int x, int y, int width, int height) ترسم قطعاً ناقصاً مصمتاً يحدده مستطيل بالعرض والارتفاع المحددين، حيث أن الزاوية العليا اليسرى نمط ملء القطع.Brush)، تحدد x,yللمستطيل هي (

- يوضح المثال التالي كيفية رسم خطوط ومستطيلات وقطوع ناقصة (مجوفة ومصمتة).

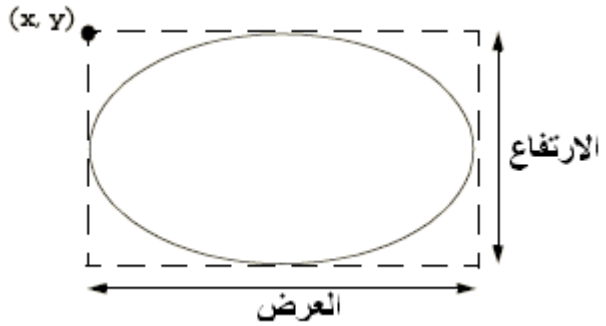
```
1  // LinesRectanglesOvals.cs
2  // Demonstrating lines, rectangels and ovals.
3  using System;
4  using System.Drawing;
5  using System.Windows.Forms;
6
```

```
7  // draw shapes on Form
8  public partial class LinesRectanglesOvals : Form
9  {
10    // default constructor
11    public LinesRectanglesOvals()
12    {
13       InitializeComponent();
14    } // end constructor
15
16    // override Form OnPaint method
17    protected override void OnPaint( PaintEventArgs paintEvent )
18    {
19       // get graphics object
20       Graphics g = paintEvent.Graphics;
21       SolidBrush brush = new SolidBrush( Color.Blue );
22       Pen pen = new Pen( Color.AliceBlue );
23
24       // create filled rectangle
25       g.FillRectangle( brush, 90, 30, 150, 90 );
26
27       // draw lines to connect rectangles
28       g.DrawLine( pen, 90, 30, 110, 40 );
29       g.DrawLine( pen, 90, 120, 110, 130 );
30       g.DrawLine( pen, 240, 30, 260, 40 );
31       g.DrawLine( pen, 240, 120, 260, 130 );
32
33       // draw top rectangle
34       g.DrawRectangle( pen, 110, 40, 150, 90 );
35
36       // set brush to red
37       brush.Color = Color.Red;
38
39       // draw base Ellipse
40       g.FillEllipse( brush, 280, 75, 100, 50 );
```

```
41
42          // draw connecting lines
43          g.DrawLine( pen, 380, 55, 380, 100 );
44          g.DrawLine( pen, 280, 55, 280, 100 );
45
46          // draw Ellipse outline
47          g.DrawEllipse( pen, 280, 30, 100, 50 );
48       } // end method OnPaint
49 } // end class LinesRectanglesOvals
```

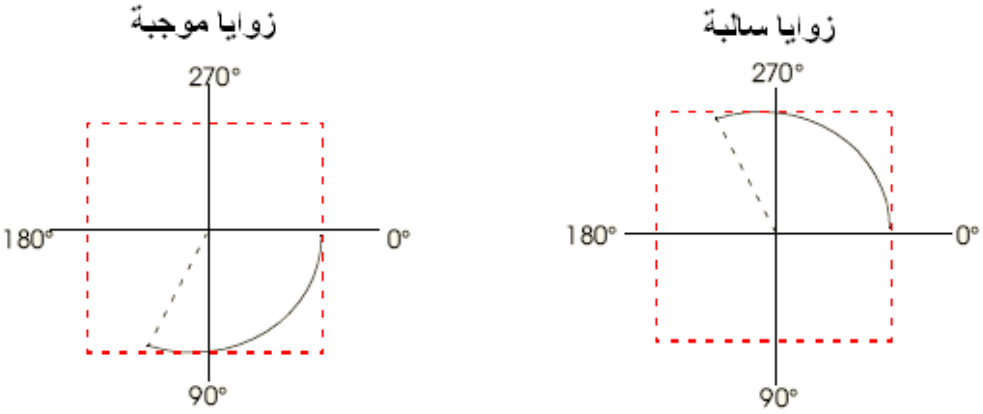


- ترسم الطريقتان FillRectangle و DrawRectangle (السطران 25 و34) مستطيلاً على الشاشة، وتأخذ كلا الطريقتين كوسيط أول غرض رسومي تستعمله في رسم المستطيل، حيث تأخذ الطريقة FillRectangle فرشاة (غرضاً من النمط Brush) وهي في حالتنا غرض من الصف SolidBrush المشتق من Brush، بينما تأخذ الطريقة DrawRectangle قلماً (غرضاً من النمط Pen)، ويمثل الوسيطان الثاني والثالث إحداثيي الزاوية العليا اليسرى للمستطيل الحاوي bounding rectangle والذي يمثل المنطقة التي سيرسم فيها المستطيل، ويحدد الوسيطان الرابع والخامس عرض وارتفاع المستطيل.
- تأخذ الطريقة DrawLine (الأسطر 28 إلى 31) قلماً Pen وزوجين من الأعداد الصحيحة يحددان بداية الخط ونهايته، ثم تقوم برسم خط بين النقطتين باستعمال القلم.
- ترسم الطريقتان FillEllipse و DrawEllipse (السطران 40 و47) قطعاً ناقصاً، يُحدد الوسيط الأول الغرض الرسومي المستخدم في الرسم، ويحدد الوسيطان الثاني والثالث إحداثيي الزاوية العليا اليسرى للمستطيل الذي يُمثل المنطقة التي سيُرسم فيها القطع، ويُحدد الوسيطان الرابع والخامس عرض وارتفاع هذا المستطيل، يُبين الشكل التالي قطعاً ناقصاً والمستطيل الحاوي له، يمس القطع منتصفات أضلاع المستطيل من الداخل، لا يتم رسم المستطيل الحاوي أثناء رسم القطع.



رسم الأقواس

- تقاس الأقواس والتي هي أجزاء من قطع ناقص بالدرجات، تبدأ عند زاوية محددة تسمى زاوية البدء Starting Angle وتستمر عدداً معيناً من الدرجات تسمى زاوية القوس Arc Angle. تمسح الأقواس زاويتها ابتداء من زاوية البدء، إذا كان المسح باتجاه عقارب الساعة clockwise تكون زاوية المسح موجبة، وتكون سالبة في الاتجاه المعاكس (عكس عقارب الساعة counterclockwise). ويوضح الشكل التالي كلا الحالتين، لاحظ أن القوس في الجهة اليمنى من الشكل يمسح باتجاه الأعلى ابتداء من الزاوية 0 وحتى الزاوية −110 تقريباً، كما أن القوس في الجهة اليمنى من الشكل يمسح باتجاه الأسفل ابتداء من الزاوية 0 وحتى الزاوية 110 تقريباً.



- لاحظ المربعات المنقطة حول الأقواس، يُرسم كل قوس على أنه جزء من شكل بيضوي (لا تظهر باقي أجزائه)، وعند رسم شكل بيضوي يجب تحديد المستطيل الحاوي له، تمثل المربعات في الشكل هذه المستطيلات، يوضح الجدول التالي طرائق الصف Graphics المستعملة لرسم الأقواس وهي DrawArc و DrawPie و FillPie.

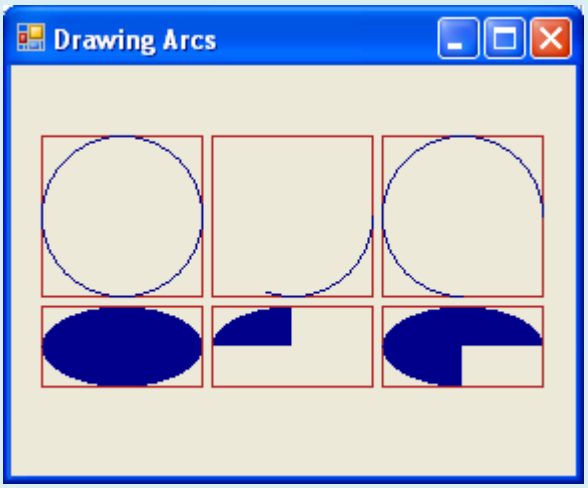
طرائق الرسم في الصف Graphics
DrawArc (Pen p, int x, int y, int width, int height, int startAngle, int sweepAngle) ترسم قوساً بدءاً من الزاوية startAngle (بالدرجات) وتمسح sweepAngle درجة، حيث يعرَّف القطع الناقص بالمستطيل الحاوي له والمعرف بزاويته العليا اليسرى (x,y) وعرضه وارتفاعه، يحدد Pen لون القوس ونمط وعرض محيطه.
DrawPie (Pen p, int x, int y, int width, int height, int startAngle, int sweepAngle) ترسم جزءاً من فطيرة بدءاً من الزاوية startAngle (بالدرجات) وتمسح sweepAngle درجة، حيث يعرَّف القطع الناقص بالمستطيل الحاوي له والمعرف بزاويته العليا اليسرى (x,y) وعرضه وارتفاعه، يحدد Pen لون القوس ونمط وعرض محيطه.
FillPie (Brush b, int x, int y, int width, int height, int startAngle, int sweepAngle) يعمل بطريقة مشابهة للطريقة DrawPie إلا أنه يرسم قوساً مصمتاً (أي قطاعاً زاوياً)، تحدد Brush نمط الملء.

- يعرض المثال الحالي ست صور (ثلاثة أقواس مجوفة وثلاثة مصمتة)، ولتوضيح المستطيلات الحاوية لهذه الأقواس رسمنا مستطيلات حمر بنفس الموقع والأبعاد.

```
1  // DrawingArcs.cs
2  // Drawing various arcs on a Form.
3  using System;
4  using System.Drawing;
5  using System.Windows.Forms;
6
7  // draws various arcs
8  public partial class DrawArcs : Form
9  {
10    // default constructor
11    public DrawArcs()
12    {
13       InitializeComponent();
```

```
14    } // end constructor
15
16    // draw arcs
17    private void DrawArcs_Paint( object sender, PaintEventArgs e )
18    {
19       // get graphics object
20       Graphics graphicsObject = e.Graphics;
21       Rectangle rectangle1 = new Rectangle( 15, 35, 80, 80 );
22       SolidBrush brush1 = new SolidBrush( Color.Firebrick );
23       Pen pen1 = new Pen( brush1, 1 );
24       SolidBrush brush2 = new SolidBrush( Color.DarkBlue );
25       Pen pen2 = new Pen( brush2, 1 );
26
27       // start at 0 and sweep 360 degrees
28       graphicsObject.DrawRectangle( pen1, rectangle1 );
29       graphicsObject.DrawArc( pen2, rectangle1, 0, 360 );
30
31       // start at 0 and sweep 110 degrees
32       rectangle1.Location = new Point( 100, 35 );
33       graphicsObject.DrawRectangle( pen1, rectangle1 );
34       graphicsObject.DrawArc( pen2, rectangle1, 0, 110 );
35
36       // start at 0 and sweep -270 degrees
37       rectangle1.Location = new Point( 185, 35 );
38       graphicsObject.DrawRectangle( pen1, rectangle1 );
39       graphicsObject.DrawArc( pen2, rectangle1, 0, -270 );
40
41       // start at 0 and sweep 360 degrees
42       rectangle1.Location = new Point( 15, 120 );
43       rectangle1.Size = new Size( 80, 40 );
44       graphicsObject.DrawRectangle( pen1, rectangle1 );
45       graphicsObject.FillPie( brush2, rectangle1, 0, 360 );
46
47       // start at 270 and sweep -90 degrees
```

```
48          rectangle1.Location = new Point( 100, 120 );
49          graphicsObject.DrawRectangle( pen1, rectangle1 );
50          graphicsObject.FillPie( brush2, rectangle1, 270, -90 );
51
52          // start at 0 and sweep -270 degrees
53          rectangle1.Location = new Point( 185, 120 );
54          graphicsObject.DrawRectangle( pen1, rectangle1 );
55          graphicsObject.FillPie( brush2, rectangle1, 0, -270 );
56       } // end method DrawArcs_Paint
57 } // end class DrawArcs
```



- تقوم الأسطر من 20 إلى 25 بإنشاء الأعراض الرسموية اللازمة من مستطيلات وأقلام وفراشي، ثم يرسم السطران 28 و29 مستطيلاً بداخله قوس يمسح 360 درجة مشكلاً دائرة، ويغير السطر 32 موقع المستطيل بإسناد غرض جديد من النمط Point إلى الخاصية Location، يأخذ التابع البناء للنمط Point وسيطين يمثلان إحداثيي النقطة الجديدة، وتحدد الخاصية Location موقع الزاوية العليا اليسرى للمستطيل. بعد رسم المستطيل، نقوم برسم قوس يبدأ عند الزاوية 0 ويمسح 110 درجات، وبما أن الزاوية موجبة فسيكون اتجاه المسح نحو الأسفل. وتقوم الأسطر من 37 إلى 39 بنفس العملية إلا أنها تمسح −270 درجة. يُسند السطر 43 غرضا جديداً من النمط Size إلى الخاصية Size للمستطيل مما يؤدي إلى تغيير حجمه.
- يعمل باقي البرنامج بنفس الطريقة السابقة غير أنه يستعمل فرشاة لرسم قوس مصمت باستعمال الطريقة FillPie، ويظر ذلك في الجزء السفلي من الخرج.

رسم المضلعات ومتعددات الخطوط

- تكون المضلعات عبارة عن أشكال لها عدة أضلاع، وهناك العديد من الطرائق في الصف Graphics لرسم المضلعات: DrawLines لرسم سلسلة خطوط مترابطة، DrawPolygone لرسم مضلع مغلق، FillPolygone لرسم مضلع مصمت. يحوي الجدول التالي توضيحاً لهذه الطرائق، كما يسمح المثال التالي للمستخدم أن يرسم مضلعات وخطوطاً مترابطة باستعمال هذه الطرائق.

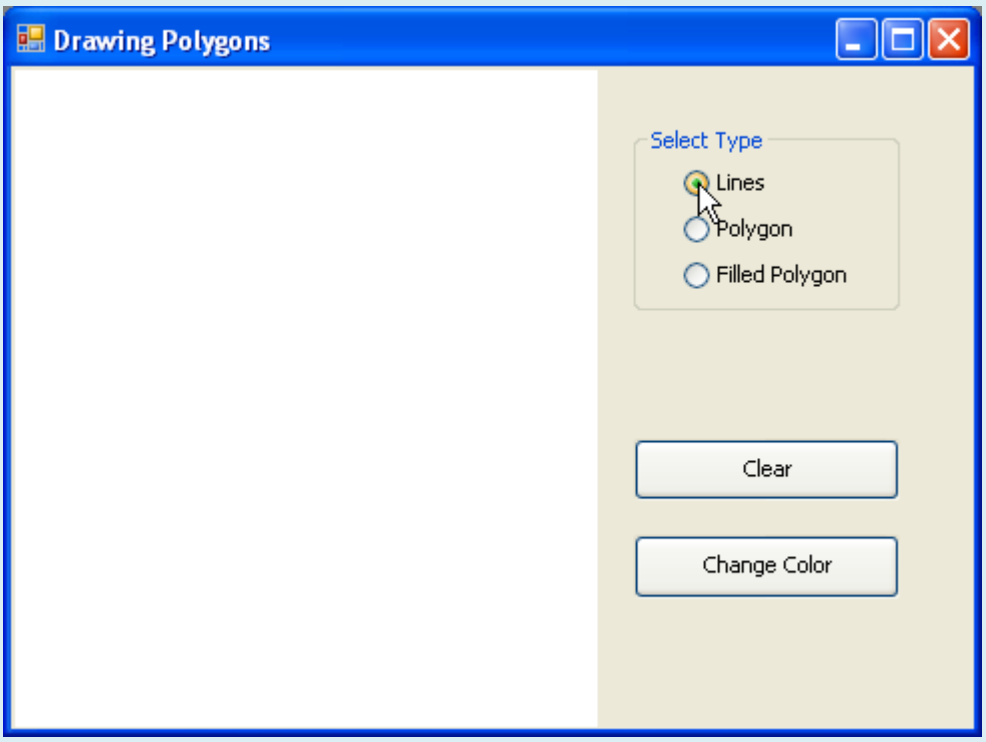
الطريقة	الوصف
DrawLines	تقوم برسم سلسلة من المستقيمات المترابطة، حيث تُحدد إحداثيات النقاط بمصفوفة عناصرها من النمط Point، إذا كانت النقطة الأخيرة لا تساوي النقطة الأولى فلن ينتج شكل مغلق.
DrawPolygon	تقوم برسم مضلع، حيث تُحدد إحداثيات النقاط بمصفوفة عناصرها من النمط Point، إذا كانت النقطة الأخيرة لا تساوي النقطة الأولى يتم وصلهما لإغلاق المضلع.
FillPolygon	تقوم برسم مضلع مصمت، حيث تُحدد إحداثيات النقاط بمصفوفة عناصرها من النمط Point، إذا كانت النقطة الأخيرة لا تساوي النقطة الأولى يتم وصلهما لإغلاق المضلع.

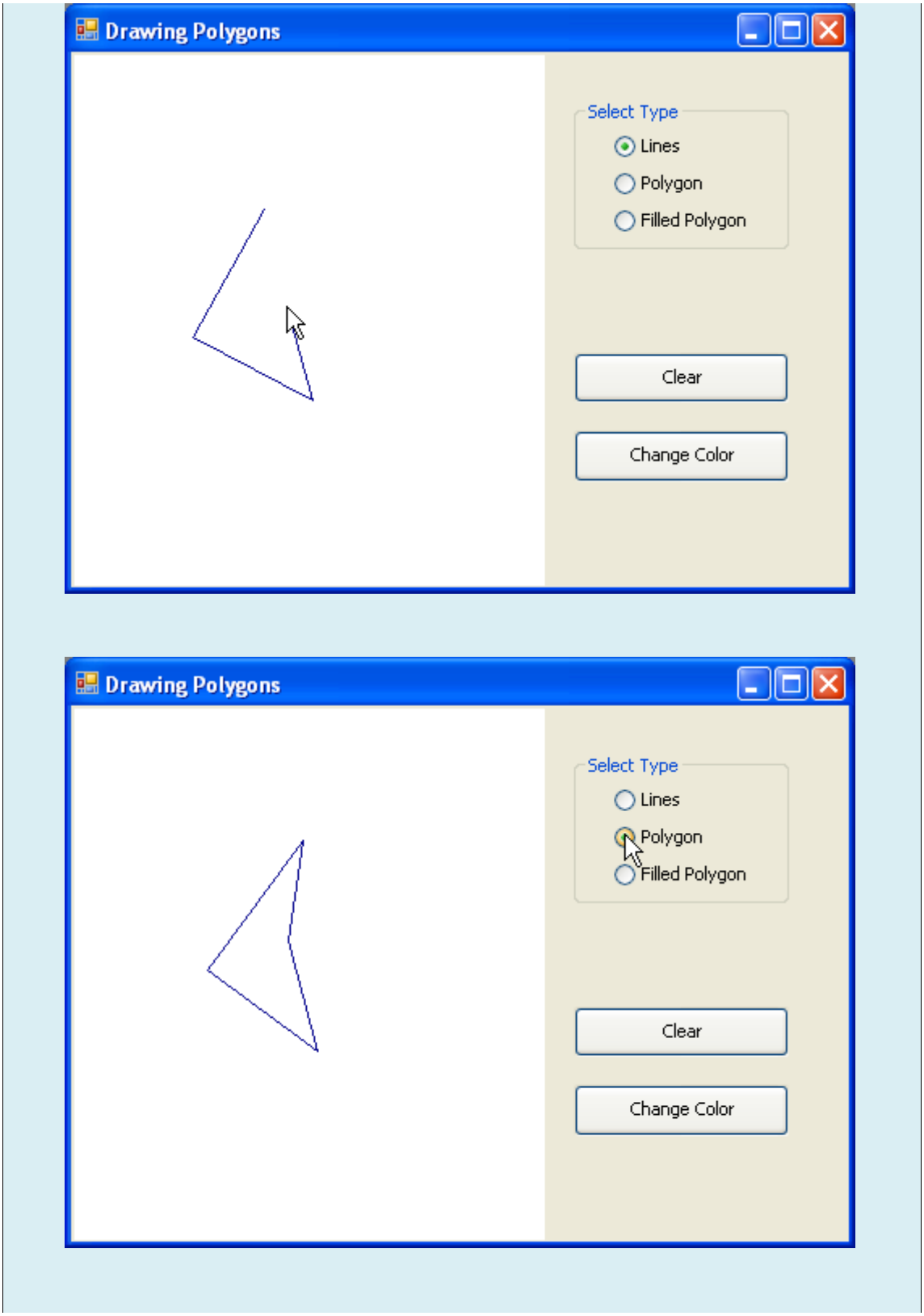
```
1  // DrawPolygons.cs
2  // Demonstrating polygons.
3  using System;
4  using System.Collections;
5  using System.Drawing;
6  using System.Windows.Forms;
7
8  // demonstrating polygons
9  public partial class PolygonForm : Form
10 {
11    // default constructor
12    public PolygonForm()
13    {
14       InitializeComponent();
```

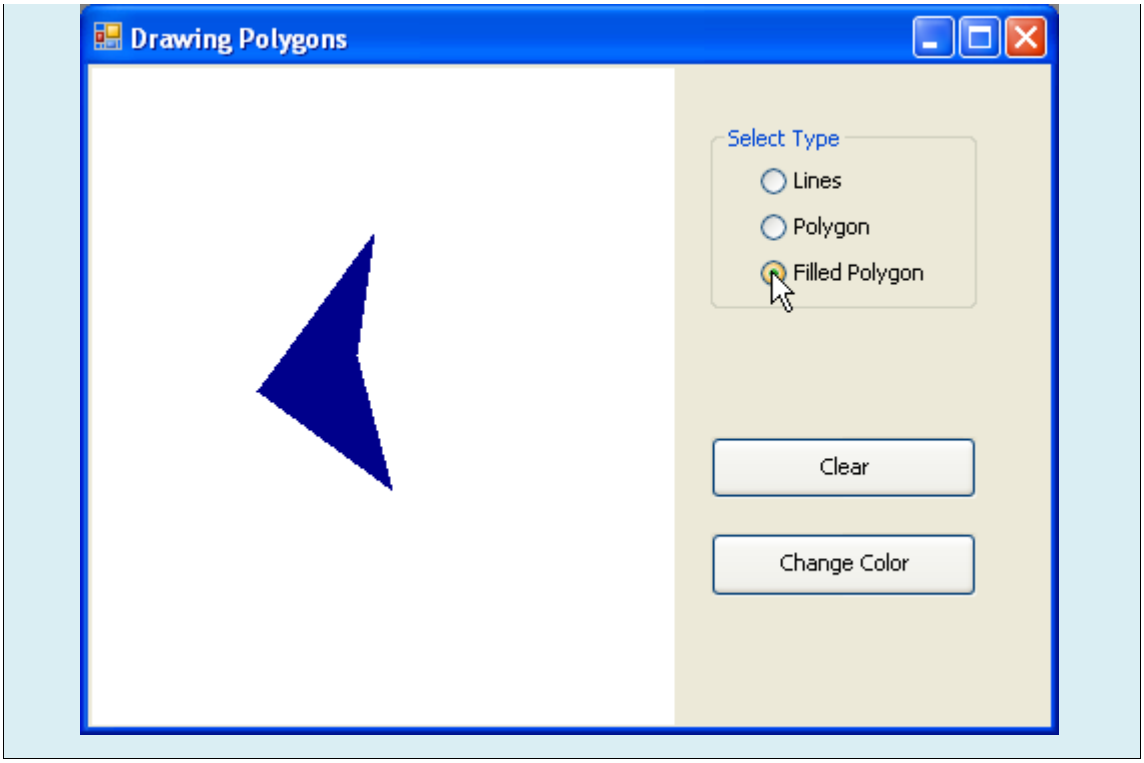
```
15    } // end constructor
16
17    // contains list of polygon vertices
18    private ArrayList points = new ArrayList();
19
20    // initialize default pen and brush
21    Pen pen = new Pen( Color.DarkBlue );
22    SolidBrush brush = new SolidBrush( Color.DarkBlue );
23
24    // draw panel mouse down event handler
25   private void drawPanel_MouseDown(object sender,MouseEventArgs e)
26    {
27       // add mouse position to vertex list
28       points.Add( new Point( e.X, e.Y ) );
29       drawPanel.Invalidate(); // refresh panel
30    } // end method drawPanel_MouseDown
31
32    // draw panel Paint event handler
33    private void drawPanel_Paint( object sender, PaintEventArgs e )
34    {
35       // get graphics object for panel
36       Graphics graphicsObject = e.Graphics;
37
38       // if arraylist has 2 or more points, display shape
39       if ( points.Count > 1 )
40       {
41          // get array for use in drawing functions
42          Point[] pointArray =
43             ( Point[] ) points.ToArray( points[ 0 ].GetType() );
44
45          if ( lineOption.Checked )
46             graphicsObject.DrawLines( pen, pointArray );
47          else if ( polygonOption.Checked )
48             graphicsObject.DrawPolygon( pen, pointArray );
```

```
49          else if ( filledPolygonOption.Checked )
50             graphicsObject.FillPolygon( brush, pointArray );
51       } // end if
52    } // end method drawPanel_Paint
53
54    // handle clearButton click event
55    private void clearButton_Click( object sender, EventArgs e )
56    {
57       points.Clear(); // remove points
58       drawPanel.Invalidate(); // refresh panel
59    } // end method clearButton_Click
60
61    // handle polygon RadioButton CheckedChanged event
62    private void polygonOption_CheckedChanged(
63       object sender, System.EventArgs e )
64    {
65       drawPanel.Invalidate(); // refresh panel
66    } // end method polygonOption_CheckedChanged
67
68    // handle line ReadioButton CheckedChanged event
69    private void lineOption_CheckedChanged(
70       object sender, System.EventArgs e )
71    {
72       drawPanel.Invalidate(); // refresh panel
73    } // end method lineOption_CheckedChanged
74
75    // handle filled polygon RadioButton CheckedChanged event
76    private void filledPolygonOption_CheckedChanged(
77       object sender, System.EventArgs e )
78    {
79       drawPanel.Invalidate(); // refresh panel
80    } // end method filledPolygonOption_CheckedChanged
81
82    // handle colorButton Click event
```

```
83    private void colorButton_Click( object sender, EventArgs e )
84    {
85       // create new color dialog
86       ColorDialog dialogColor = new ColorDialog();
87
88       // show dialog and obtain result
89       DialogResult result = dialogColor.ShowDialog();
90
91       // return if user cancels
92       if ( result == DialogResult.Cancel )
93          return;
94
95       pen.Color = dialogColor.Color; // set pen to color
96       brush.Color = dialogColor.Color; // set brush
97       drawPanel.Invalidate(); // refresh panel;
98    } // end method colorButton_Click
99 } // end class PolygonForm
```




Drawing Polygons
Select Type
Lines
Polygon
Filled Polygon
Clear
Change Color
Drawing Polygons
Select Type
Lines
Polygon
Filled Polygon
Clear
Change Color



- يُعرّف السطر 18 ArrayList باسم points كحاوية للنقاط التي حددها المستخدم. تُشبه ArrayList المصفوفة array إلا أنه من الممكن زيادة حجمها لإضافة عناصر جديدة. يُعرّف السطران 21 و 22 قلماً Pen وفرشاة Brush لرسم الأشكال، ويخزن معالج الحدث MouseDown (الأسطر من 25 إلى 30) الخاص باللوحة drawPanel مواقع نقرات الفأرة على شكل نقاط باستعمال الطريقة Add الخاصة بالصف ArrayList (السطر 28)، ثم يقوم السطر 29 باستدعاء الطريقة Invalidate للوحة الرسم للتأكد من تحديث اللوحة بالنقاط الجديدة. يقوم معالج الحدث Paint للوحة drawPanel (الأسطر من 33 إلى 52) بالحصول على غرض Graphics للوحة (السطر 36) ثم يرسم مضلعاً حسب الطريقة التي حددها المستخدم عن طريق أزرار الخيار في الواجهة (الأسطر من 45 إلى 50) إذا كانت ArrayList تحوي نقطتين على الأقل (السطر 39)، يقوم السطران 42 و43 باستخراج مصفوفة من النقاط من ArrayList باستعمال الطريقة ToArray التي تأخذ وسيطاً يعرف نمط المصفوفة المُعادة، استخدمنا الطريقة GetType للعنصر الأول للحصول على النمط المطلوب.
- يقوم معالج الحدث clearButton (الأسطر من 55 إلى 59) بمسح القائمة points باستدعاء الطريقة Clear وتحديث لوحة الرسم، كما يقوم معالج الحدث CheckedChanged لأزرار الخيار (الأسطر من 62 إلى 80) بتحديث لوحة الرسم لتعكس خيار المستخدم الذي يحدد نوع الشكل، كما يسمح معالج حدث الزر colorButton للمستخدم بتغيير لون الرسم باستعمال ColorDialog كما وضحنا سابقاً.

الفصل التاسع: أساسيات LINQ

عنوان الموضوع:

أساسيات LINQ.

الكلمات المفتاحية:

var، where، select، orderby.

ملخص:

نستعرض في هذا الفصل أساسيات تقانة الاستعلام LINQ.

أهداف تعليمية:

يتعرف الطالب في هذا الفصل على:

- الاستعلام في مصفوفة من الأنماط الأساسية.
- الاستعلام في مصفوفة من الأغراض.

المخطط:

أساسيات LINQ

- 2 وحدة (Learning Objects)

مقدمة

يكتب المبرمجون استعلاماتهم على قواعد البيانات العلائقية بلغة الاستعلام المعروفة SQL.

تُقدّم لغة **LINQ (Language Integrated Query)** إمكانيات جديدة تسمح لك بكتابة تعابير للاستعلام (شبيهة بلغة SQL) تُعيد مجموعة من النتائج من مصادر بيانات مختلفة الأنواع وليس حصراً قواعد بيانات علائقية.

الاستعلام في مصفوفة من الأعداد

نقوم في المثال التالي أولاً بالاستعلام عن القيم التي أكبر من 4 مثلاً.

```
from value in values
    where value > 4
    select value;
```

بالطبع كان يُمكن أن نقوم بحلقة لإجراء هذا الفحص. إلا أن LINQ تُعزز فصل مسائل الاستعلام عن مسائل العرض.

لترتيب قيم المصفوفة تصاعدياً:

```
   from value in values
   orderby value
   select value;
```

يُمكن الاستعلام والترتيب كما يلي:

```
from value in values
   where value > 4
   orderby value descending
   select value;
```

يكون الكود:

```
1  //  LINQWithSimpleTypeArray.cs
2  // LINQ to Objects using an int array.
3  using System;
4  using System.Linq;
5  class LINQWithSimpleTypeArray
6  {
7  public static void Main( string[] args )
8  {
9  // create an integer array
10 int[] values = { 2, 9, 5, 0, 3, 7, 1, 4, 8, 5 };
11 // display original values
12 Console.Write( "Original array:" );
13 foreach ( var element in values )
14 Console.Write( " {0}", element );
15 // LINQ query that obtains values greater than 4 from the array
16 var filtered =
17     from value in values
18     where value > 4
```

```
19      select value;
20  // display filtered results
21  Console.Write( "\nArray values greater than 4:" );
22  foreach ( var element in filtered )
23  Console.Write( " {0}", element );
24  // use orderby clause to original values in ascending order
25  var sorted =
26     from value in values
27     orderby value
28     select value;
29  // display sorted results
30  Console.Write( "\nOriginal array, sorted:" );
31  foreach ( var element in sorted )
32  Console.Write( " {0}", element );
33  // sort the filtered results into descending order
34  var sortFilteredResults =
35     from value in filtered
36     orderby value descending
37     select value;
38  // display the sorted results
39  Console.Write(
40  "\nValues greater than 4, descending order (separately):" );
41  foreach ( var element in sortFilteredResults )
42  Console.Write( " {0}", element );
43  // filter original array and sort results in descending order
44  var sortAndFilter =
45     from value in values
46     where value > 4
47     orderby value descending
48     select value;
49  // display the filtered and sorted results
50  Console.Write(
51  "\nValues greater than 4, descending order (one query):" );
52  foreach ( var element in sortAndFilter )
53      Console.Write( " {0}", element );
54  Console.WriteLine();
55  } // end Main
56  } // end class LINQWithSimpleTypeArray
```

يُظهر التنفيذ النتائج التالية:

```
Original array: 2 9 5 0 3 7 1 4 8 5
Array values greater than 4: 9 5 7 8 5
Original array, sorted: 0 1 2 3 4 5 5 7 8 9
Values greater than 4, descending order (separately): 9 8 7 5 5
Values greater than 4, descending order (one query): 9 8 7 5 5
Press any key to continue. . .
```

الكلمة المفتاحية var

يُمكن استخدام الكلمة المفتاحية var لتصريح عن متغير وترك مهمة تحديد نوع المتغير للمترجم استناداً إلى عملية التهيئة الأولية للمتغير. فمثلاً إذا كتبنا:

```
var x = 7;
```

سيقوم المترجم باعتبار المتغير x من النمط int.

بينما لو كتبنا:

```
var y = −123.45;
```

فسيقوم المترجم باعتبار المتغير y من النمط double.

العبارة where

عندما يكون الشرط المحدد في الفقرة محققاً. سيتم تضمين العنصر الموافق في النتائج.

العبارة select

تُحدد القيم التي ستظهر في النتائج.

العبارة orderby

تقوم بترتيب النتائج.

الاستعلام في مصفوفة من الأغراض باستخدام LINQ

نقوم في المثال التالي بعرض إمكانيات الاستعلام من مصفوفة من الأغراض:

ليكن لدينا الصف البسيط التالي Employee:

```
1  // Employee.cs
2  // Employee class with FirstName, LastName and MonthlySalary properties.
3  public class Employee
4  {
5  private decimal monthlySalaryValue; // monthly salary of employee
6  // auto-implemented property FirstName
7  public string FirstName { get; set; }
8  // auto-implemented property LastName
9  public string LastName { get; set; }
10 // constructor initializes first name, last name and monthly salary
11 public Employee( string first, string last, decimal salary )
12 {
13 FirstName = first;
14 LastName = last;
15 MonthlySalary = salary;
16 } // end constructor
17 // property that gets and sets the employee's monthly salary
18 public decimal MonthlySalary
19 {
20 get
21 {
22 return monthlySalaryValue;
23 } // end get
24 set
25 {
26 if ( value >= 0M ) // if salary is non-negative
27 {
28 monthlySalaryValue = value;
29 } // end if
30 } // end set
31 } // end property MonthlySalary
32 // return a String containing the employee's information
33 public override string ToString()
34 {
35 return string.Format( "{0,-10} {1,-10} {2,10:C}",
36 FirstName, LastName, MonthlySalary );
37 } // end method ToString
38 } // end class Employee
```

نقوم في المثال التالي باستخدام الصف Employee:

```
1  // LINQWithArrayOfObjects.cs
2  // LINQ to Objects using an array of Employee objects.
3  using System;
4  using System.Linq;
5  public class LINQWithArrayOfObjects
6  {
7  public static void Main( string[] args )
8  {
9  // initialize array of employees
10 Employee[] employees = {
11   new Employee( "Jason", "Red", 5000M ),
12   new Employee( "Ashley", "Green", 7600M ),
13   new Employee( "Matthew", "Indigo", 3587.5M ),
14   new Employee( "James", "Indigo", 4700.77M ),
```

```
15   new Employee( "Luke", "Indigo", 6200M ),
16   new Employee( "Jason", "Blue", 3200M ),
17   new Employee( "Wendy", "Brown", 4236.4M ) }; // end init list
18  // display all employees
19  Console.WriteLine( "Original array:" );
20  foreach ( var element in employees )
21  Console.WriteLine( element );
22  // filter a range of salaries using && in a LINQ query
23  var between4K6K =
24   from e in employees
25   where e.MonthlySalary >= 4000M && e.MonthlySalary <= 6000M
26   select e;
27  // display employees making between 4000 and 6000 per month
28  Console.WriteLine( string.Format(
29   "\nEmployees earning in the range {0:C}-{1:C} per month:",
30                 4000, 6000 ) );
31  foreach ( var element in between4K6K )
32  Console.WriteLine( element );
33  // order the employees by last name, then first name with LINQ
34  var nameSorted =
35   from e in employees
36   orderby e.LastName, e.FirstName
37   select e;
38  // header
39  Console.WriteLine( "\nFirst employee when sorted by name:" );
40  // attempt to display the first result of the above LINQ query
41  if ( nameSorted.Any() )
42   Console.WriteLine( nameSorted.First() );
43  else
44   Console.WriteLine( "not found" );
45  // use LINQ to select employee last names
46  var lastNames =
47   from e in employees
48   select e.LastName;
49  // use method Distinct to select unique last names
50  Console.WriteLine( "\nUnique employee last names:" );
51  foreach ( var element in lastNames.Distinct() )
52   Console.WriteLine( element );
53  // use LINQ to select first and last names
54  var names =
55   from e in employees
56   select new { e.FirstName, Last = e.LastName };
57  // display full names
58  Console.WriteLine( "\nNames only:" );
59  foreach ( var element in names )
60   Console.WriteLine( element );
61  Console.WriteLine();
62  } // end Main
63  } // end class LINQWithArrayOfObjects
```

- نقوم في الأسطر (10 إلى 17) بتعريف مصفوفة تحوي ستة موظفين.
- نقوم في الأسطر (24 إلى 26) بالاستعلام عن الموظفين الذين معاشهم محصور بين 4000 و 6000.
- نقوم في الأسطر (35 إلى 37) بترتيب الموظفين وفق الاسم الأخير ومن ثم وفق الاسم الأول.
- نقوم في الأسطر (47 إلى 48) بالاستعلام عن الاسم الأخير لكل موظف .
- نقوم في الأسطر (55 إلى 56) بإنشاء نمط جديد .

سيتم إظهار ما يلي:

Original array:

Jason Red $5,000.00

Ashley Green $7,600.00

Matthew Indigo $3,587.50

James Indigo $4,700.77

Luke Indigo $6,200.00

Jason Blue $3,200.00

Wendy Brown $4,236.40

Employees earning in the range $4,000.00-$6,000.00 per month:

Jason Red $5,000.00

James Indigo $4,700.77

Wendy Brown $4,236.40

First employee when sorted by name:

Jason Blue $3,200.00

Unique employee last names:

Red

Green

Indigo

Blue

Brown

Names only:

{ FirstName = Jason, Last = Red }

{ FirstName = Ashley, Last = Green }

{ FirstName = Matthew, Last = Indigo }

{ FirstName = James, Last = Indigo }

{ FirstName = Luke, Last = Indigo }

{ FirstName = Jason, Last = Blue }

```
{ FirstName = Wendy, Last = Brown }

Press any key to continue . . .
```

إنشاء نمط جديد في العبارة select

نقوم في الاستعلام الأخير (الأسطر) بإنشاء نمط مجهول **anonymous type** بدون اسم والذي يقوم المترجم بتوليده إعتمادا على الخصائص المضمنة في الأقواس { }. يتألف النمط في مثالنا من الاسم الأول والاسم الأخير. لاحظ أنه يُمكن إعطاء اسم جديد للخاصية (وضعنا الاسم last للاسم الأخير) وإلا فستأخذ الخاصية نفس الاسم.

الفصل العاشر: التعامل مع قواعد البيانات (1)

عنوان الموضوع:

التعامل مع قواعد البيانات (1).

الكلمات المفتاحية:

صف نموذج كائن البيانات ADO.NET Entity Data Model Class Library، الربط بين عناصر التحكم وصف نموذج كائن البيانات.

ملخص:

نستعرض في هذا الفصل أساسيات التعامل مع قواعد البيانات. فنعرض أولاً لإنشاء مكتبة صف نموذج كائن البيانات لاستخدامها في تطبيقات مختلفة. ثم نُبين استخدام المكتبة في تطبيق ويندوز والربط بين عناصر التحكم ونموذج كائن البيانات.

أهداف تعليمية:

يتعرف الطالب في هذا الفصل على:

- إنشاء صف نموذج كائن البيانات.
- إنشاء تطبيق ويندوز يتعامل مع صف نموذج كائن البيانات.
- الربط بين عناصر التحكم ونموذج كائن البيانات.

المخطط:

التعامل مع قواعد البيانات (1)

- 2 وحدة (Learning Objects)

إنشاء مكتبة صف نموذج كائن البيانات ADO.NET Entity Data Model Class Library

نعرض فيما يلي آلية إنشاء مكتبة مخصصة للتعامل مع قاعدة بيانات.

سيكون بإمكانك بعدها التعامل مع هذه المكتبة في مشاريع مختلفة.

الخطوة الأولى: إنشاء صف المكتبة *Class Library*:

- افتح محيط العمل Visual Studio 2013 ومن ثم:

File ⟶ New ⟶ Project

- قم بتسمية المكتبة BooksExamples.
- امسح الصف المنشئ تلقائياً Class1.cs.

الخطوة الثانية: إنشاء قاعدة البيانات:

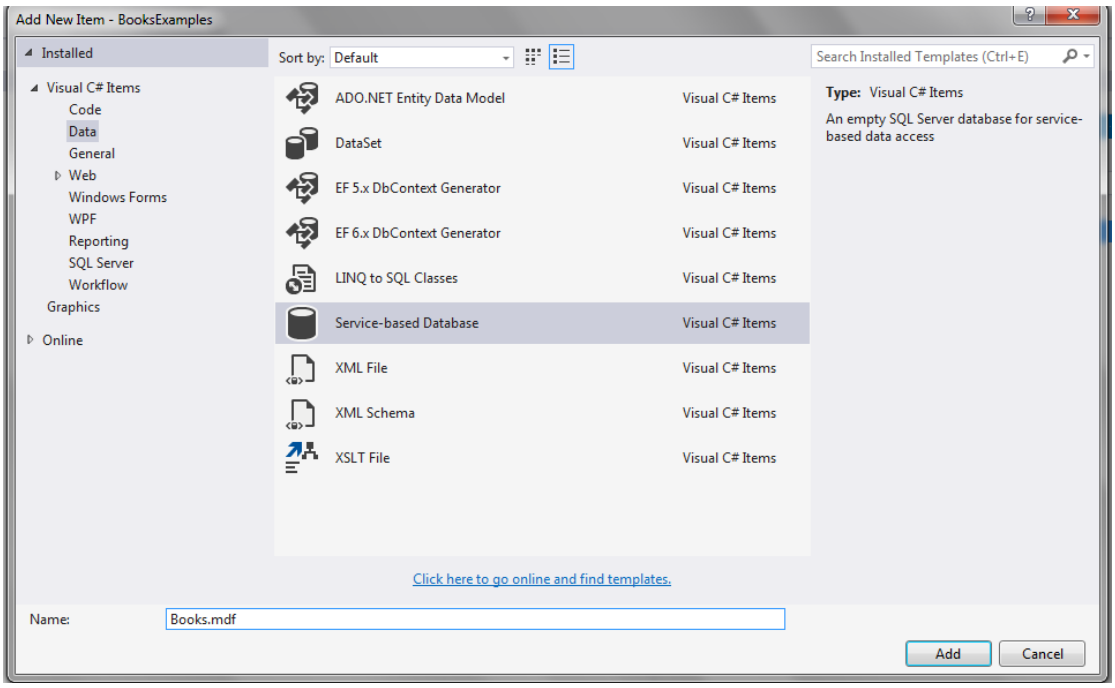
- انقر بالزر الأيمن على أيقونة المكتبة BooksExamples في مستعرض التطبيق ثم اختر:

Add ⟶ New Item

- ثم قم باختيار:

Data / Service-based Database

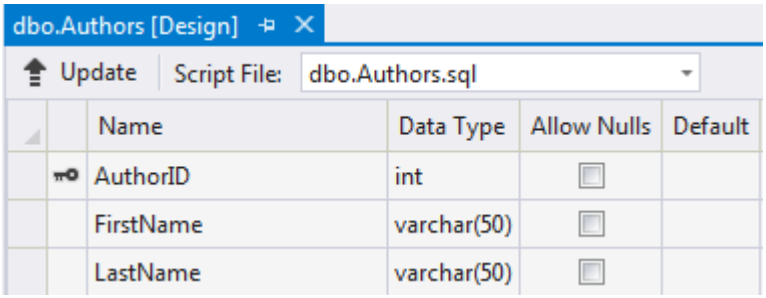
- قم بتسمية قاعدة البيانات Books.mdf.



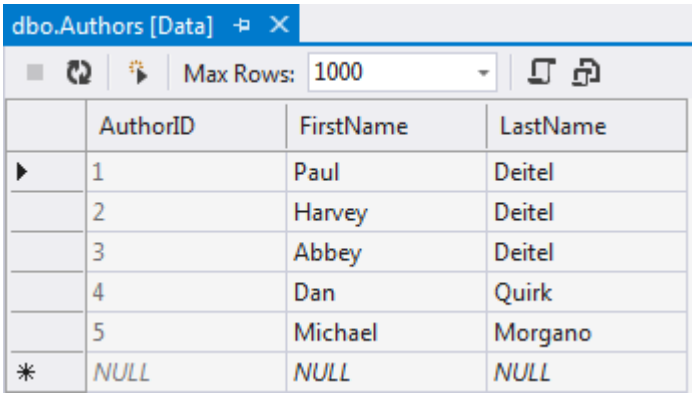
الخطوة الثالثة: إنشاء جداول قاعدة البيانات:

- قم بإنشاء جداول البيانات التالية:

- يحوي الجدول الأول Authors بيانات المؤلفين:



- تكون البيانات مثلاً:



- يحوي الجدول الثاني Titles بيانات الكتب:

dbo.Titles [Design]

Update | Script File: dbo.Titles.sql

Name	Data Type	Allow Nulls	Default
ISBN	varchar(20)	☐	
Title	varchar(100)	☐	
EditionNumber	int	☐	
Copyright	varchar(4)	☐	
		☐	

- تكون البيانات مثلاً:

dbo.Titles [Data]

Max Rows: 1000

ISBN	Title	EditionNumber	Copyright
0132121360	Android for Programmers: An App-Dr...	1	2012
0132151006	Internet & World Wide Web How to P...	5	2012
0132575663	Java How to Program	9	2012
013299044X	C How to Program	7	2013
0132990601	Simply Visual Basic 2010	4	2013
0133378713	C++ How to Program	9	2014
0133379337	Visual C# 2012 How to Program	5	2014
0133406954	Visual Basic 2012 How to Program	6	2014
0136151574	Visual C++ 2008 How to Program	2	2008
NULL	*NULL*	*NULL*	*NULL*

- الجدول الثالث AuthorISBN ناتج عن كسر العلاقة كتب/مؤلفين للربط بين الكتب والمؤلفين:

dbo.AuthorISBN [Design]

Update | Script File: dbo.AuthorISBN.sql

Name	Data Type	Allow Nulls	Default
AuthorID	int	☐	
ISBN	varchar(20)	☐	
		☐	

- تكون البيانات مثلاً:

dbo.AuthorISBN [Data]

Max Rows: 1000

AuthorID	ISBN
1	0132151006
1	0132575663
1	013299044X
1	0132990601
1	0133378713
1	0133379337
1	0133406954
1	0136151574
2	0132121360
2	0132151006
2	0132575663
2	013299044X
2	0132990601
2	0133378713
2	0133379337
2	0133406954
2	0136151574
3	0132121360
3	0132151006
3	0132990601
3	0133406954
4	0136151574
5	0132121360

الخطوة الرابعة: إضافة نموذج كائن البيانات إلى مكتبة الصف:

أولاً: إضافة نموذج كائن البيانات *ADO.NET Entity Data Model*

للتفاعل مع قاعدة البيانات، نضيف نموذج كائن البيانات إلى الصف:

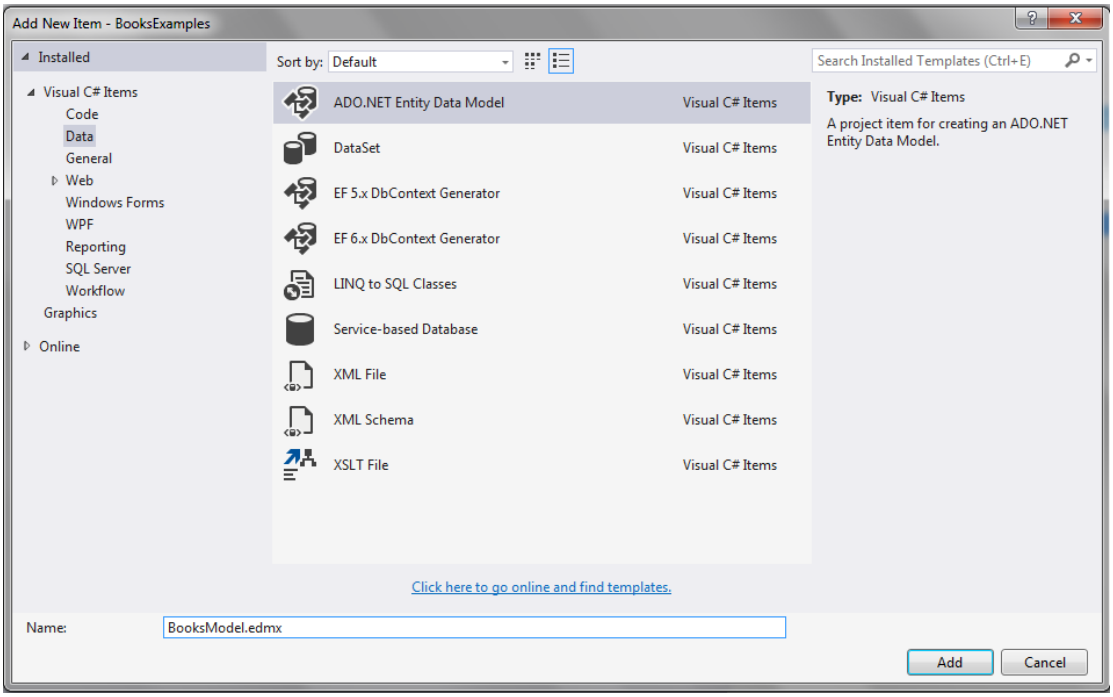
- انقر بالزر الأيمن على أيقونة المكتبة BooksExamples في مستعرض الحل solution explorer ثم اختر:

Add → New Item

- ثم اختر:

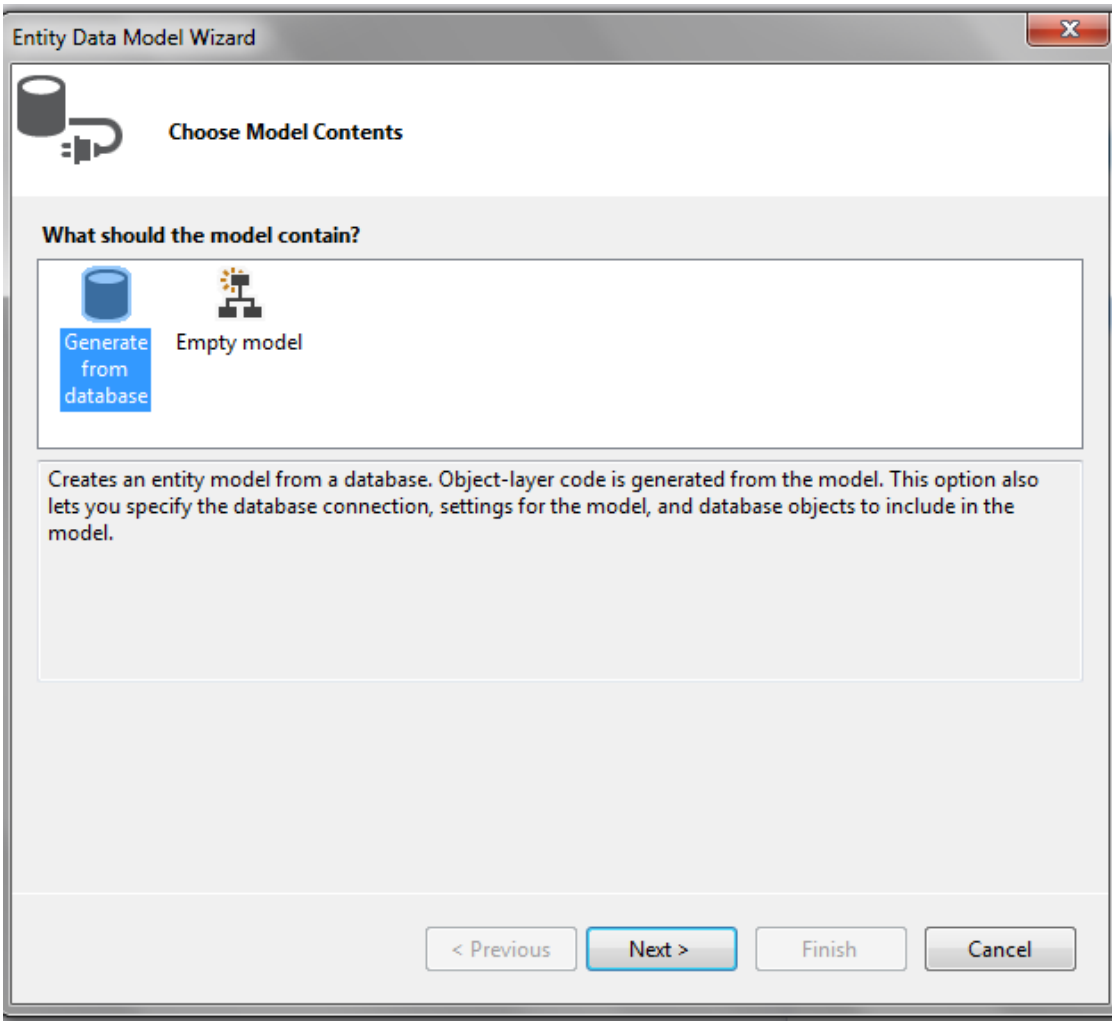
Data / ADO.NET Entity Data Model

وقم بتسميته BooksModel.edmx.



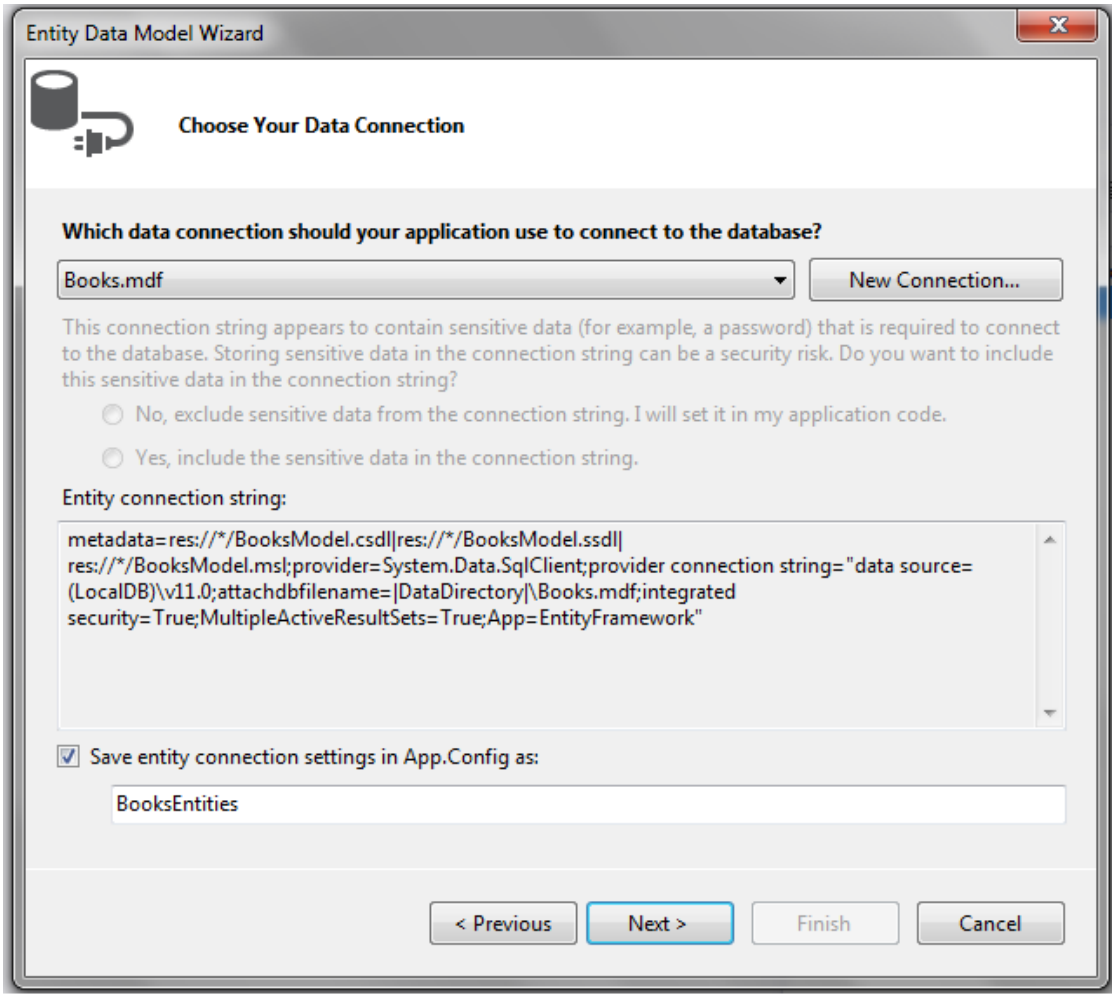
ثانياً: اختيار محتويات النموذج *Model Contents*

- قم باختيار التوليد من قاعدة بيانات Generate from database من نافذة المعالج اختيار محتوى النموذج Choose Model Contents:



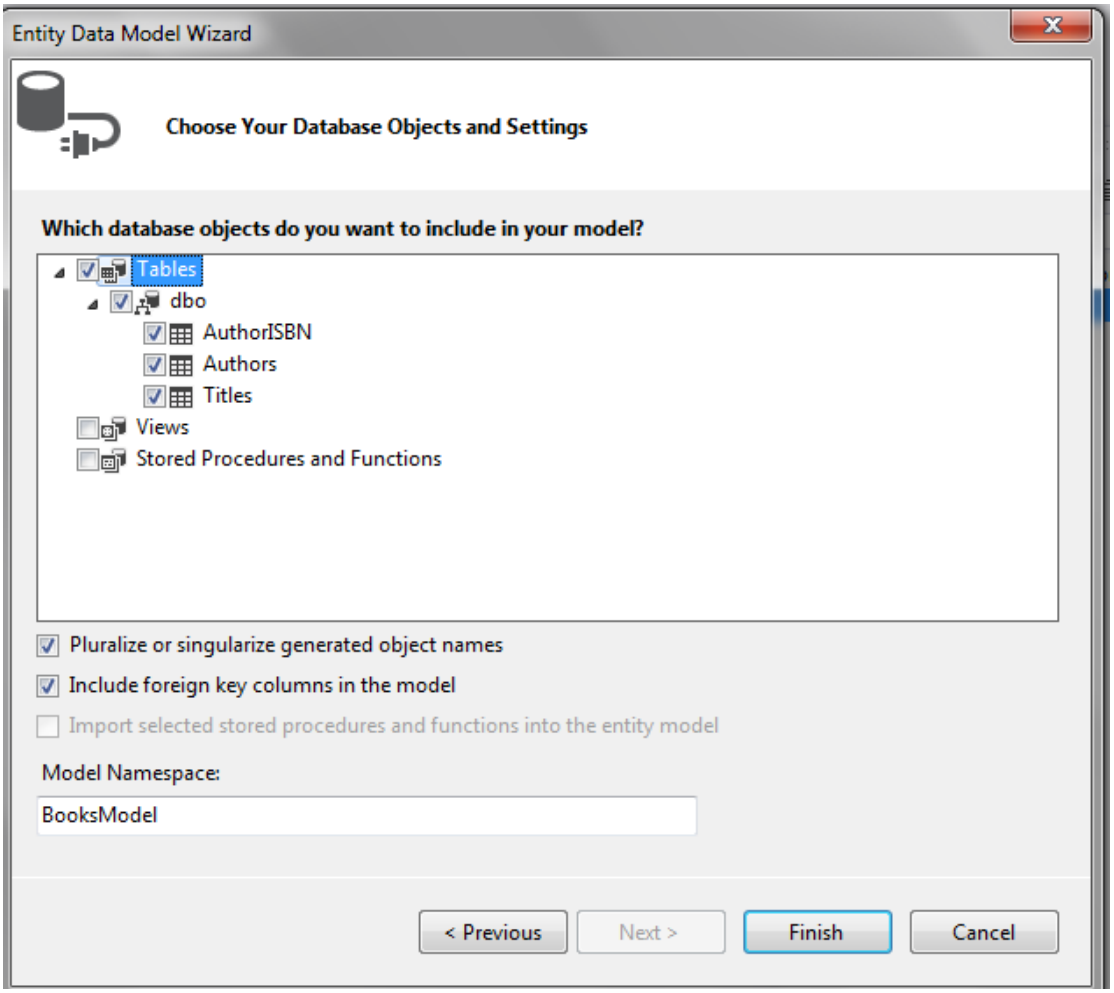
ثالثاً: اختيار الاتصال مع البيانات *Data Connection:*

- قم باختيار الاتصال مع قاعدة البيانات Books.mdf.
- في حال تريد الاتصال مع قاعدة بيانات أخرى، انقر الزر New Connection ثم حدّد الاتصال المطلوب.
- سيتم تخزين الاتصال في ملف إعدادات التطبيق App.Config.



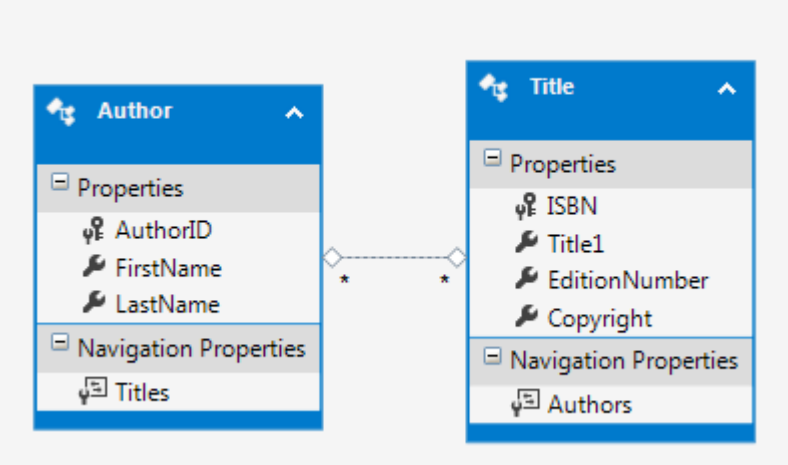
رابعاً: اختيار كائنات قاعدة البيانات المطلوب تضمينها في النموذج:

- قم باختيار الجداول الثلاثة:



خامساً: معاينة مخطط نموذج كائن البيانات *Model Diagram Entity Data* في تصميم النموذج

- يتم إظهار مخطط الجداول والعلاقات بينها:



سادساً: بناء صف المكتبة:

- قم ببناء المكتبة:

Build → Build Solution

ستكون قادراً على استخدام هذه المكتبة في مشاريع مختلفة.

بناء تطبيق ويندوز وربطه مع نموذج كائن البيانات

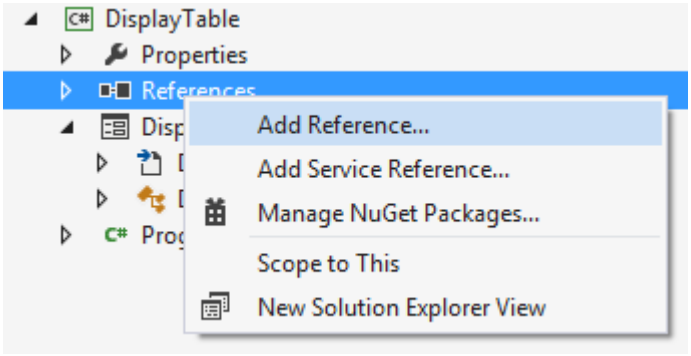
قم باتباع الخطوات التالية لبناء تطبيق ويندوز مرتبط مع نموذج كائن البيانات:

الخطوة الأولى: إنشاء المشروع:

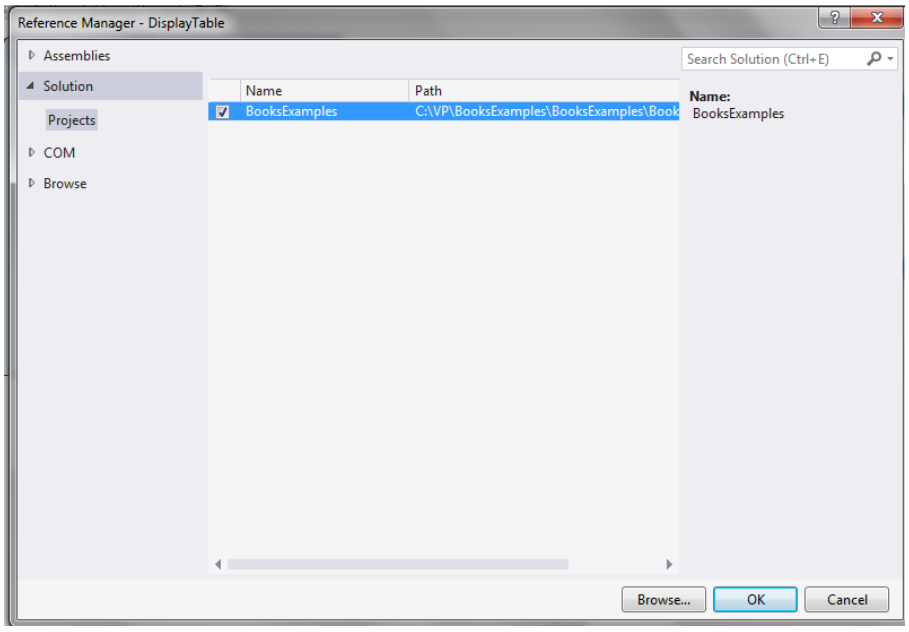
- قم بإنشاء مشروع جديد من النوع Windows Form Application.
- قم بتسمية المشروع DisplayTable.
- عدّل اسم النموذج الافتراضي من Form1 إلى DisplayAuthorsTable.

الخطوة الثانية: إضافة مرجع إلى صف المكتبة *BooksExamples*:

- انقر بالزر الأيمن على عقدة المراجع References في مستعرض الحل واختر إضافة مرجع Add Reference.

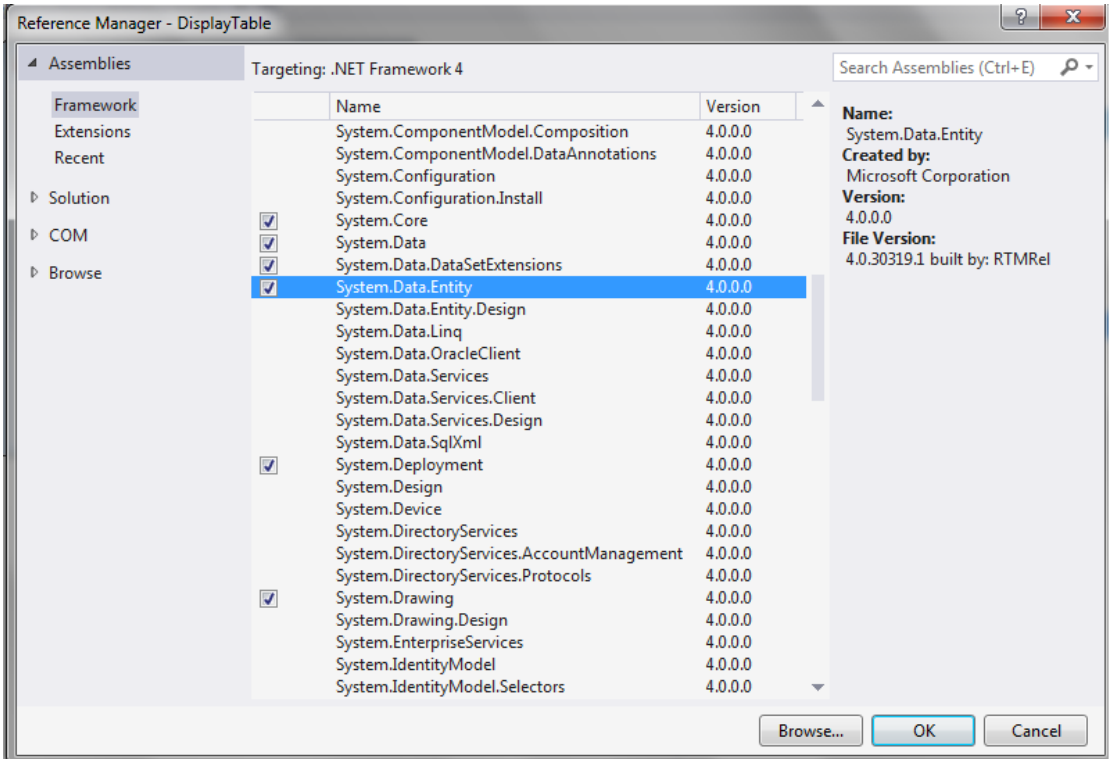


- اختر في نافذة إدارة المراجع Reference Manager التبويب solution.
- حدّد مشروع المكتبة السابقة BooksExamples.



الخطوة الثالثة: إضافة مرجع إلى المكتبة *System.Data.Entity* :

- افتح في نافذة إدارة المراجع Reference Manager التبويب Assemblies.
- قم بتحديد System.Data.Entity.



- سوف تلاحظ ظهور System.Data.Entity تحت عقدة المراجع References في المشروع.

الخطوة الرابعة: إضافة مرجع إلى المكتبة *EntityFramework*:

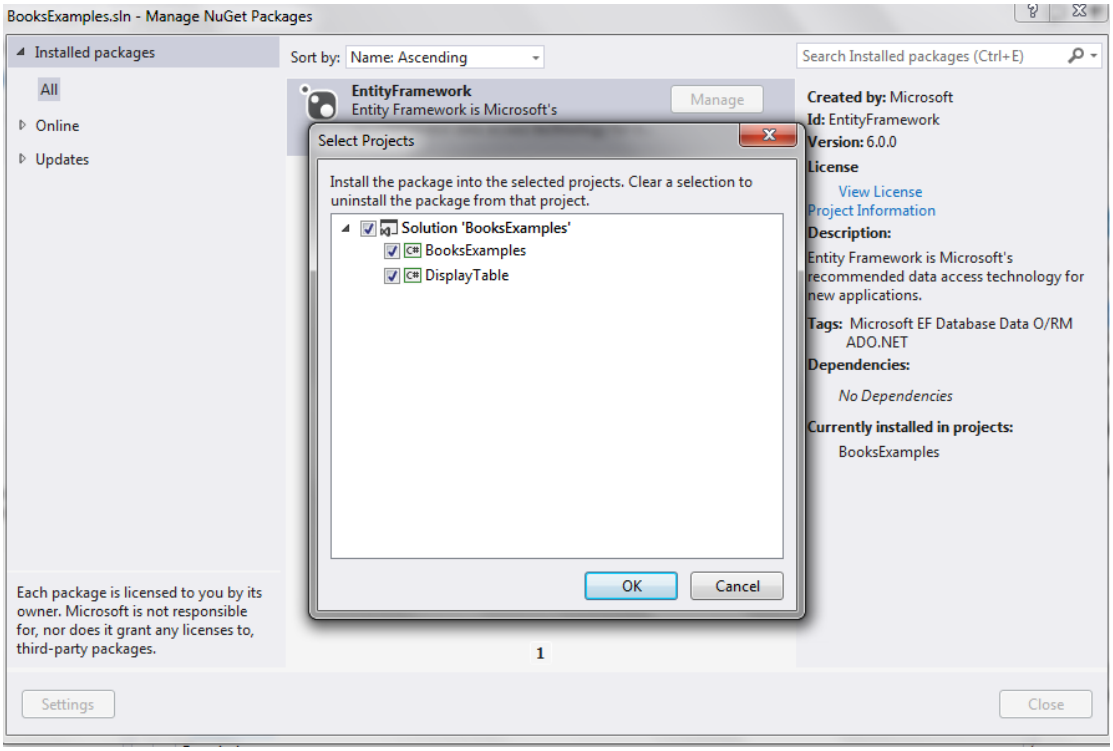
تم إضافة المكتبة EntityFramework تلقائياً إلى الصف BooksExamples عندما قمنا بإنشاء نموذج كائن البيانات. إلا أن هذه المكتبة مطلوبة في كل مشروع يستخدم نموذج كائن البيانات.

لإضافة مرجع إلى المكتبة EntityFramework:

- انقر بالزر الأيمن على عقدة التطبيق الأولى solution واختر:

Manage NuGet Packages for Solution…

- انقر الزر إدارة Manage وحدّد المشروع DisplayTable:



- لاحظ ظهور EntityFramework تحت عقدة المراجع References في المشروع.

الخطوة الخامسة: إضافة سلسلة الاتصال إلى التطبيق ويندوز:

- يحتاج كل مشروع يستخدم نموذج كائن البيانات إلى سلسلة الاتصال التي تُخبره كيف يتصل مع قاعدة البيانات.
- تمّ توليد سلسلة الاتصال تلقائياً عند بناء المكتبة BooksExample.
- قم بفتح ملف الإعدادات App.Config في المكتبة BooksExample ونسخ سلسلة الاتصال ومن ثم لصقها في الملف App.Config في المشروع DisplayTable.

```
<connectionStrings>
  <add name="BooksEntities" connectionString="…… " />
</connectionStrings>
```

الربط بين عناصر التحكم ونموذج كائن البيانات

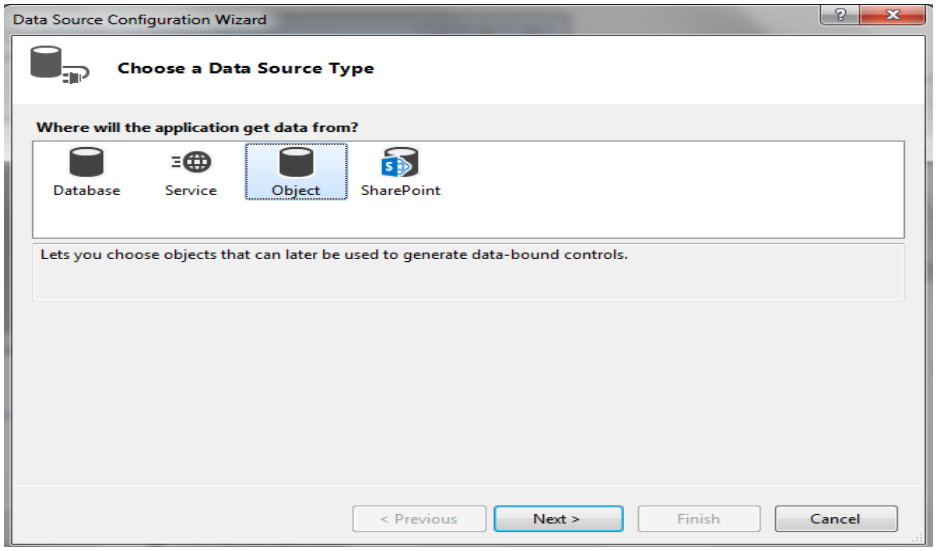
الخطوة الأولى :إضافة مصدر بيانات إلى جدول المؤلفين:

لاستخدام صفوف نموذج كائن البيانات للربط مع البيانات، يجب أولاً إضافتها كمصدر بيانات:

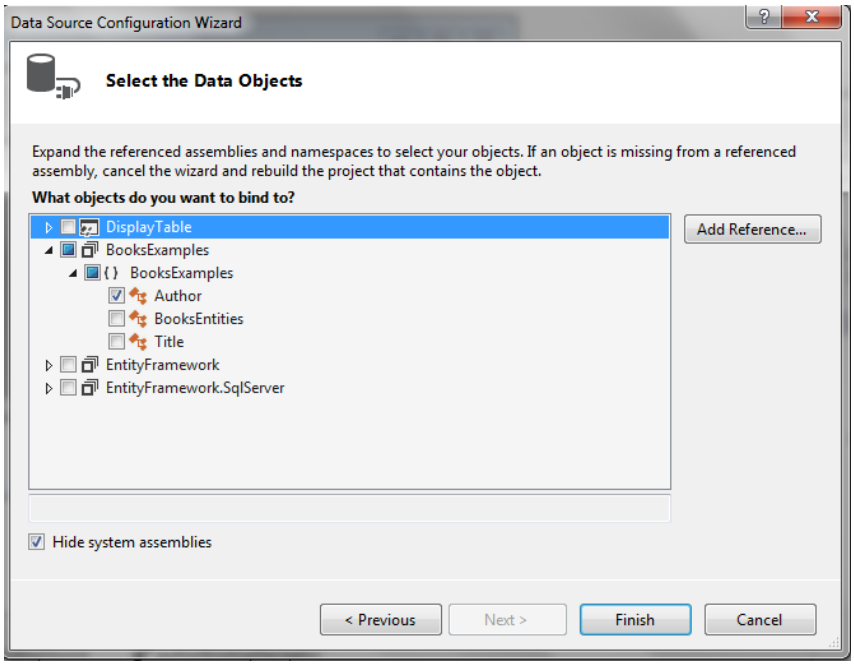
- افتح مصادر البيانات:

VIEW → Other Windows → Data Sources

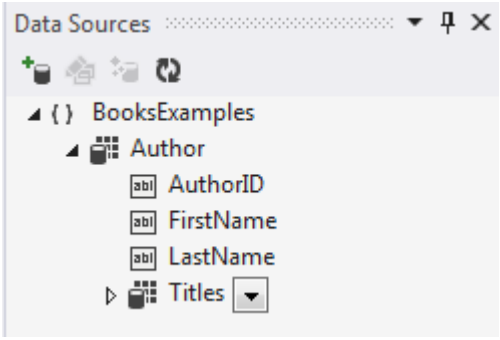
- ثم انقر الرابط إضافة مصدر بيانات جديد Add New Date Source لفتح معالج إعداد مصدر البيانات Data Source Configuration Wizard.
- بما أن صفوف نموذج كائن البيانات تُستخدم لإنشاء أغراض *objects* تُمثّل الجداول في قاعدة البيانات. قم باختيار الخيار *objects*.



- ثم اختر Author:



- لاحظ ظهور الصف Author في نافذة مصادر البيانات:



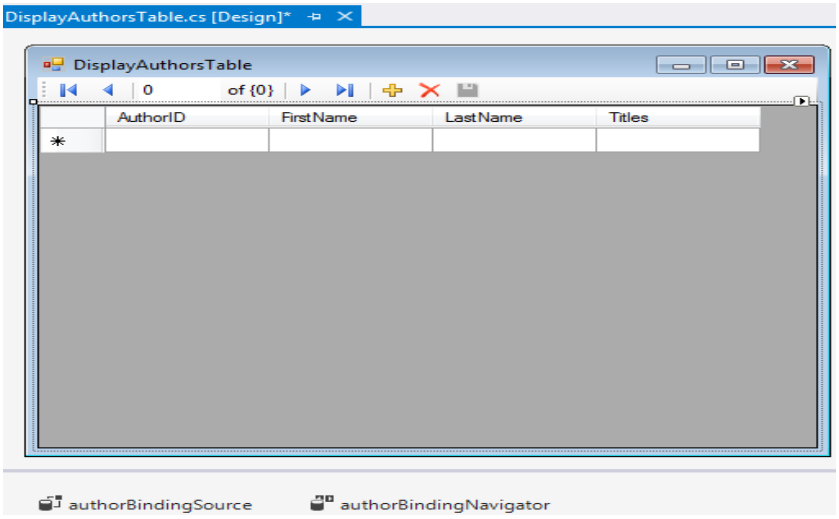
الخطوة الثانية: إضافة عناصر التحكم:

- انقر على عقدة الصف Author في نافذة مصادر البيانات ولاحظ أنها ستتحول إلى قائمة منسدلة. تأكد من اختيار عرض شبكة البيانات DataGridView.
- قم بسحب عقدة الصف Author من نافذة مصادر البيانات إلى النموذج (المفتوح بطريقة عرض التصميم).
- سيقوم محيط العمل بوضع شبكة بيانات DataGridView في النموذج وبحيث يقابل كل عمود فيها خاصية من الصف.
- كما يتم وضع شريط الربط:

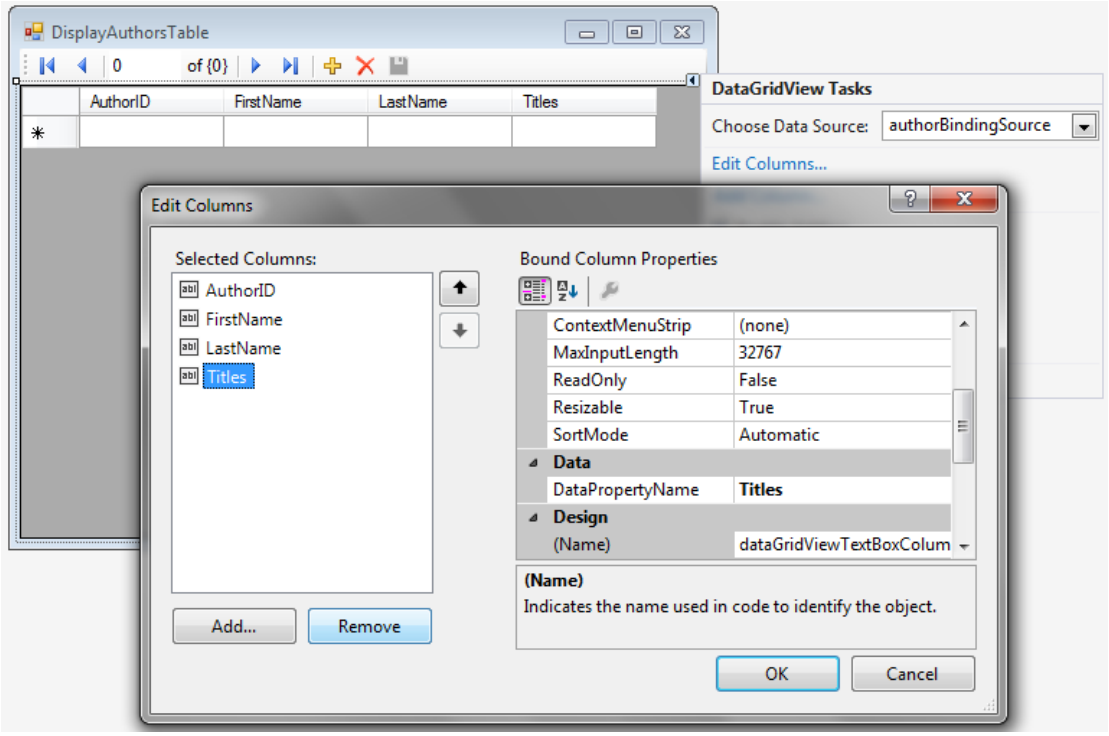
BindingNavigator (authorBindingNavigator)

الذي يحوي مجموعة من الأزرار التي تسمح بالتنقل بين التسجيلات، الإضافة، الحذف، حفظ التعديلات على قاعدة البيانات.

- يتم أيضاً توليد مصدر الربط Binding Source (authorBindingSource) والذي يتولى عمليات نقل البيانات بين مصدر البيانات وعناصر التحكم المرتبطة على النموذج.



- يُمكنك جعل عنصر شبكة البيانات يملأ كامل النموذج بضبط الخاصية Dock إلى Fill.
- بما أنك لن تُظهر العمود Titles في الشبكة، قم بإزالته كما يلي:
  - افتح قائمة الزر الأيمن للشبكة واختر تحرير الأعمدة Edit Columns.
  - ثم حدّد العمود Titles وانقر زر الحذف Remove.



الخطوة الثالثة: ربط مصدر البيانات مع مصدر الربط *authorBindingSource*:

يجب كتابة التعليمات البرمجية اللازمة:

```
1  // DisplayAuthorsTable.cs
2  // Displaying data from a database table in a DataGridView.
3  using System;
4  using System.Data.Entity;
5  using System.Data.Entity.Validation;
6  using System.Linq;
7  using System.Windows.Forms;
8  namespace DisplayTable
9  {
10 public partial class DisplayAuthorsTable : Form
11 {
12 // constructor
13 public DisplayAuthorsTable()
14 {
15  InitializeComponent();
16 } // end constructor
17 // Entity Framework DBContext
18 private BooksExamples.BooksEntities dbcontext =
19    new BooksExamples.BooksEntities();
20 // load data from database into DataGridView
```

```
21 private void DisplayAuthorsTable_Load( object sender, EventArgs e )
22 {
23 // load Authors table ordered by LastName then FirstName
24 dbcontext.Authors
25    .OrderBy( author => author.LastName )
26    .ThenBy( author => author.FirstName )
27    .Load();
28 // specify DataSource for authorBindingSource
29 authorBindingSource.DataSource = dbcontext.Authors.Local;
30 } // end method DisplayAuthorsTable_Load
31 // click event handler for the Save Button in the
32 // BindingNavigator saves the changes made to the data
33 private void authorBindingNavigatorSaveItem_Click(
34                   object sender, EventArgs e )
35 {
36    Validate(); // validate the input fields
37    authorBindingSource.EndEdit(); // complete current edit, if any
38 // try to save changes
39 try
40 {
41    dbcontext.SaveChanges(); // write changes to database file
42 } // end try
43 catch( DbEntityValidationException )
44 {
45    MessageBox.Show( "FirstName and LastName must contain values",
46                     "Entity Validation Exception" );
47 } // end catch
48 } // end method authorBindingNavigatorSaveItem_Click
49 } // end class DisplayAuthorsTable
50 } // end namespace DisplayTable
```

أولاً: إنشاء الكائن *DbContext*:

- يتعامل الغرض dbcontext من الصف BooksEntities مع قاعدة البيانات (السطرين 18 و 19). تم توليد الصف BooksEntities آلياً عند إنشاء نموذج كائن البيانات للتعامل مع قاعدة البيانات Books.

ثانياً: كتابة حدث التحميل DisplayAuthorsTable_Load:

- نسمح للبيانات بالانتقال بين الغرض dbcontext وبين قاعدة البيانات باستخدام طرق LINQ to Entities التي تسمح باستحصال البيانات من الخاصية Authors والتي تقابل جدول المؤلفين Authors في قاعدة البيانات.
- يُحدّد التعبير (السطر 24):

dbcontext.Authors

بأننا سنقوم بالحصول على البيانات من الجدول Authors.

- يُحدّد استدعاء الطريقة (السطر 25) OrderBy أن تسجيلات الجدول يجب أن تُسترجع بالترتيب التصاعدي للاسم الأخير للمؤلف.

.OrderBy(author => author.LastName)

- يكون معامل الطريقة الموسعة OrderBy (extension method) عبارة عن تعبير لامبدا lambda expression يقوم بتعريف طريقة مجهولة anonymous method.
- يبدأ التعبير لامبدا بقائمة المعاملات (في حالتنا author) والذي هو غرض من نموذج كائن البيانات Author.
- يقوم تعبير لامبدا باستنتاج نمط المعامل من dbcontext.Authors والذي يحوي الغرض Author.
- يلي قائمة المعاملات المعامل لامبدا => (يُقرأ يذهب إلى goes to) وتعبير يُمثّل جسم الطريقة.
- تكون القيمة المولدة من التعبير (الاسم الأخير لمؤلف) هي القيمة المعادة ضمنياً للتعبير.
- لانقوم بتحديد نوع بيانات القيمة المعادة حيث يتم استنتاجه تلقائياً من القيمة المعادة.
- في حال وجود أكثر من مؤلف لهم نفس الاسم الثاني، فنريد ترتيبهم تصاعدياً حسب الاسم الأول. نستخدم استدعاء الطريقة الموسعة (السطر 26) ThenBy التالي:

.ThenBy(author => author.FirstName)

والتي تسمح بترتيب النتائج وفق عمود إضافي آخر.

- نستدعي أخيراً الطريقة Load (السطر 27). تقوم هذه الطريقة بتنفيذ الاستعلام LINQ to Entities وتحميل النتائج في الذاكرة.
- يتم متابعة هذه النتائج من قبل الغرض dbcontext وبحيث أن أي تعديل على البيانات في الذاكرة يُمكن أن يُحفظ في قاعدة البيانات.
- تكافئ الأسطر (24 إلى 27) التعليمات التالية:

```
(from author in dbcontext.Authors
            orderby author.LastName, author.FirstName
            select author).Load();
```

- نقوم بإسناد الخاصية DataSource لـ authorBindingSource إلى الخاصية Local لـ dbcontext.
- تُمثّل الخاصية Local (من النوع ObservableCollection<T>) نتائج الاستعلام المحمّلة في الذاكرة.
- عندما نربط خاصية DataSource لـ BindingSource مع ObservableCollection<T> (من فضاء الأسماء System.Collections.ObjectModel) سيتم إعلام عنصر التحكم المرتبط مع BindingSource بأي تعديل على البيانات مما يسمح لعنصر التحكم بإظهار هذا التعديل. كما أن التعديلات التي يُمكن أن يقوم بها المستخدم من خلال عنصر التحكم سيتم متابعتها وبحيث يقوم dbcontext بحفظ هذه البيانات في قاعدة البيانات.

ثالثاً: إجرائية الحفظ:

- نريد حفظ البيانات في حال قام المستخدم بتعديلها في الشبكة.
- يكون زر الحفظ في BindingNavigator غير مفعل افتراضياً.
- لتفعيل زر الحفظ، انقر عليه بالزر الأيمن وحدّد الخيار Enabled.
- انقر نقراً مزدوجاً على زر الحفظ لكتابة معالج الحدث له.
- يتم حفظ التعديلات في ثلاث خطوات (الأسطر 33 إلى 48):

الخطوة الأولى:

يتم التحقق من جميع عناصر التحكم على النموذج باستدعاء الطريقة Validate. تقوم هذه الطريقة باستدعاء أي معالج حدث للحدث Validating مكتوب على أي عنصر تحكم (يُستخدم هذا الحدث للتحقق من صحة القيم المدخلة في عنصر التحكم).

الخطوة الثانية:

استدعاء الطريقة EndEdit لـ authorBindingSource والتي تقوم بفرض حفظ أي تعديل معلق إلى النموذج BooksEntities في الذاكرة.

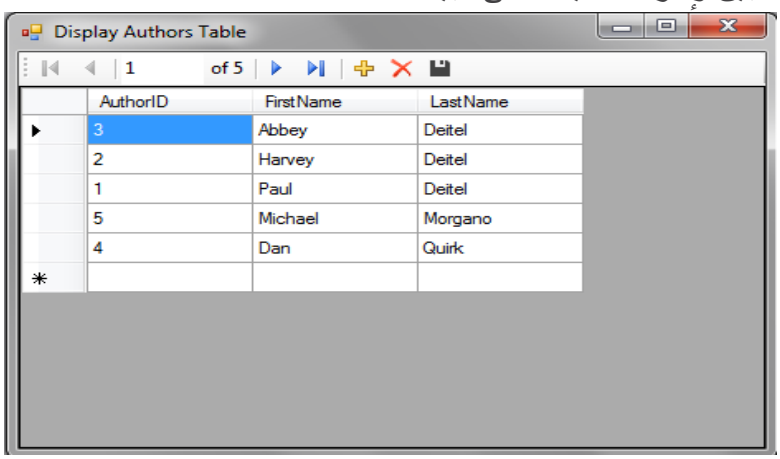
الخطوة الثالثة:

استدعاء الطريقة Save_Changes على الكائن dbcontext لحفظ أي تعديلات إلى قاعدة البيانات.

لاحظ أننا وضعنا هذا الاستدعاء ضمن تعليمة try لالتقاط أي خطأ قد يحصل عند الحفظ. فمثلاً قمنا أثناء تصميمنا لقاعدة البيانات بفرض حقول الاسم الأول والأخير واجبة الإدخال. لو حاول المستخدم حفظ تسجيلة جديدة دون إدخال قيم لهذه الحقول فستظهر له رسالة الخطأ الموافقة.

التنفيذ:

يُمكن الآن تشغيل التطبيق وإجراء العمليات على البيانات:



الفصل الحادي عشر: التعامل مع قواعد البيانات (2)

عنوان الموضوع:

التعامل مع قواعد البيانات (2).

الكلمات المفتاحية:

الربط الديناميكي، استرجاع البيانات من عدة جداول.

ملخص:

نتابع في هذا الفصل تقانات التعامل مع قواعد البيانات فنعرض أولاً الربط الديناميكي مع نتائج الاستعلامات ثم نعرض لاسترجاع البيانات من عدة جداول.

أهداف تعليمية:

يتعرف الطالب في هذا الفصل على:

- الربط الديناميكي مع نتائج الاستعلامات.
- استرجاع البيانات من عدة جداول.

المخطط:

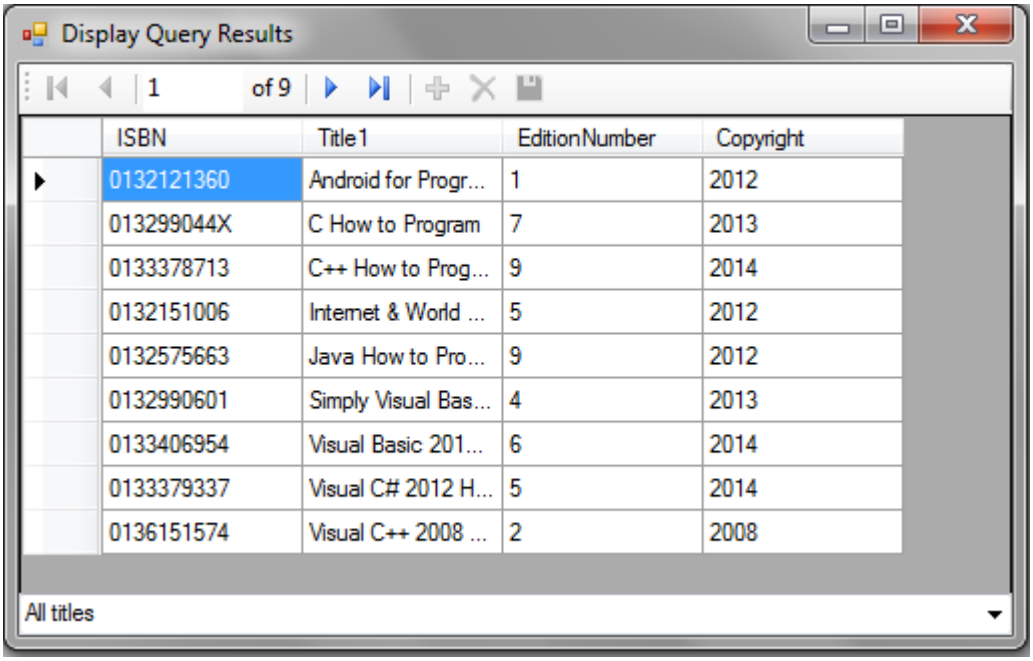
التعامل مع قواعد البيانات (2)

- 2 وحدة (Learning Objects)

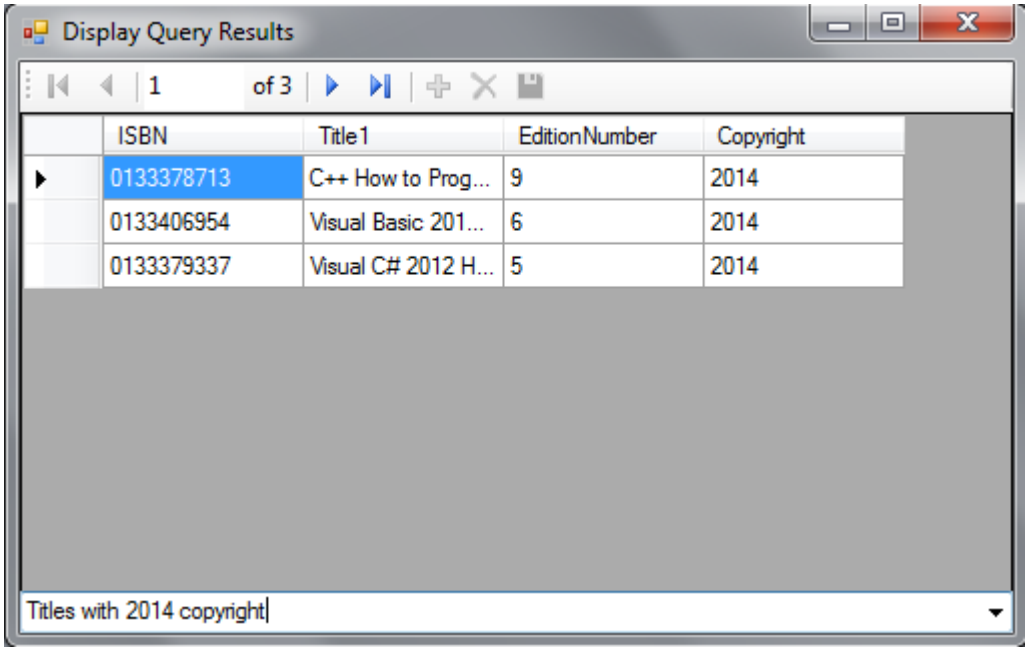
الربط الديناميكي مع نتائج الاستعلام

نقوم في التطبيق التالي بالسماح للمستخدم باختيار الاستعلام المطلوب من قائمة منسدلة أسفل النافذة لإظهار النتائج الموافقة.

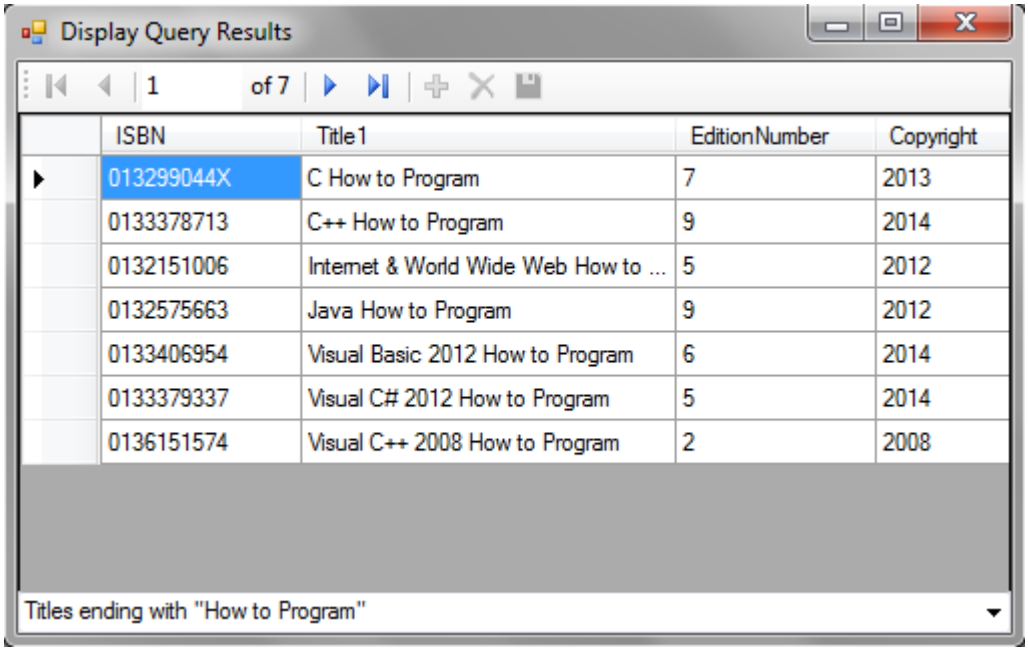
- يقوم الاستعلام الأول بإظهار جميع الكتب مرتبة تصاعدياً حسب عنوان الكتاب:



- يقوم الاستعلام الثاني بإظهار الكتب المحققة للشرط Titles with 2014 Copyrights:



- يقوم الاستعلام الثالث بإظهار الكتب التي عناوينها تنتهي بـ How to Program:



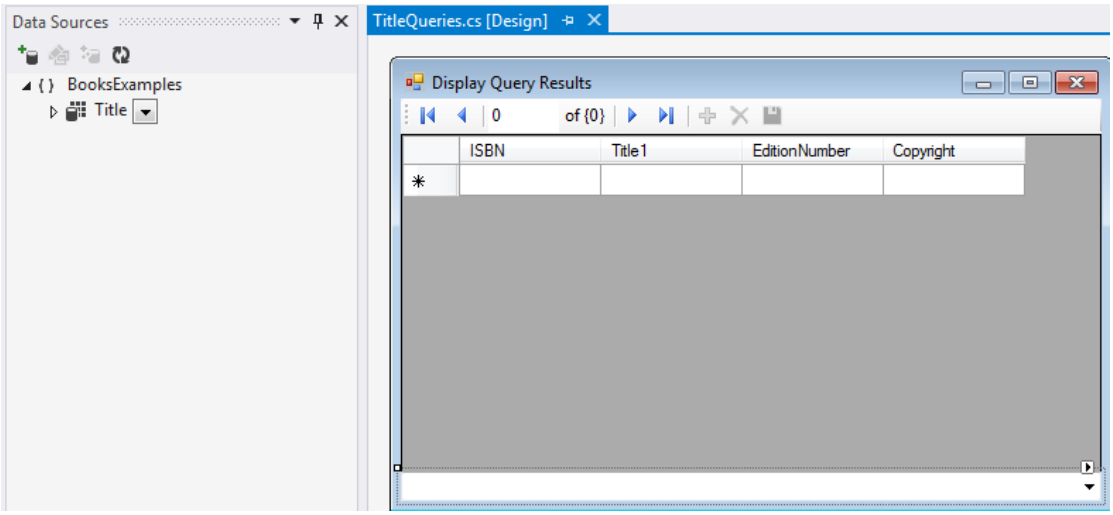
تُبين الخطوات التالية مراحل إنشاء المشروع:

الخطوة الأولى: إنشاء مشروع جديد:

- أنشئ مشروع جديد وسمّه DisplayQueryResult في نفس الحل solution السابق.
- قم بتعديل اسم ونص النموذج Form1 إلى DisplayQueryResult.

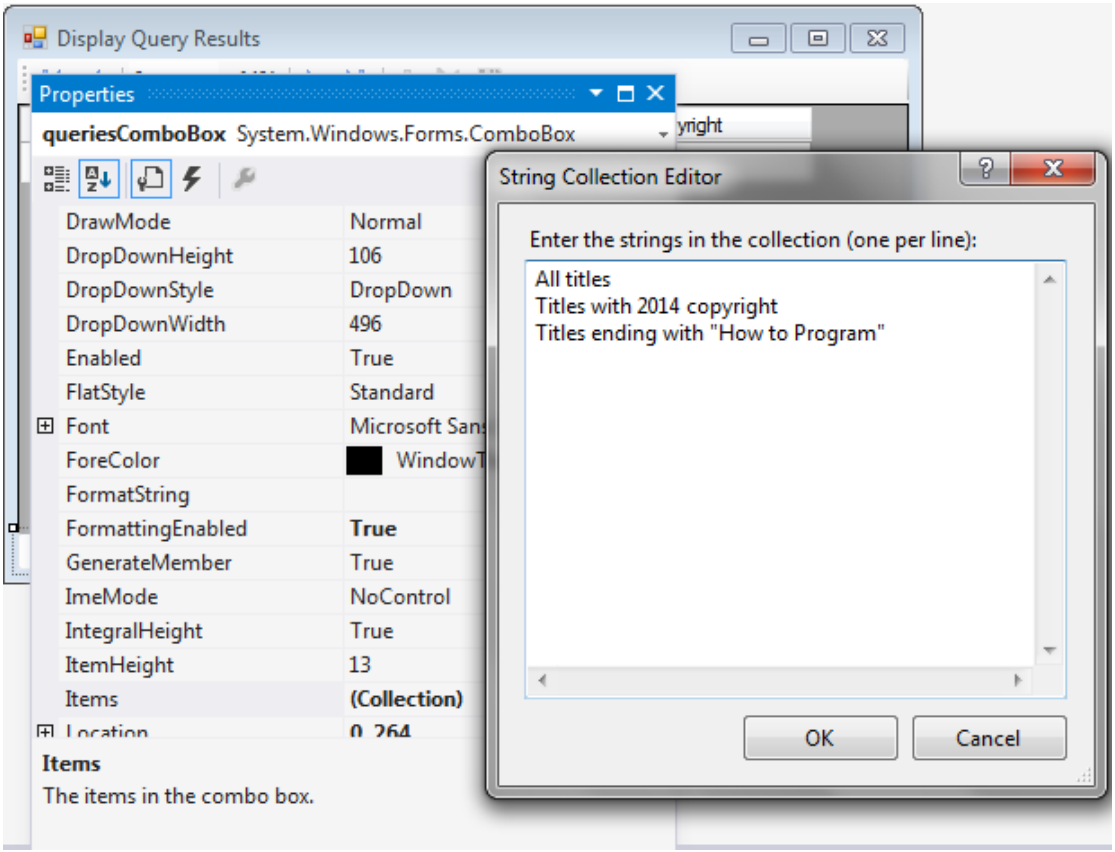
الخطوة الثانية: إضافة شبكة عرض بيانات *DataGridView* لإظهار جدول الكتب Titles:

- قم بإظهار نافذة مصادر البيانات Data Sources ثم اسحب أيقونة الصف Title إلى النموذج. قم بحذف العمود Authors من الشبكة.



الخطوة الثالثة: إضافة قائمة منسدلة إلى النموذج:

- قم بإضافة قائمة منسدلة إلى النموذج واضبط الخصائص التالية:
  - Name: queriesComboBox
  - Dock: Bottom
- قم بإدخال العناصر التالية في الخاصية Items للقائمة.



الخطوة الرابعة: كتابة الكود اللازم في كل من حدث تحميل النموذج وحدث اختيار عنصر من القائمة:

```
1  // TitleQueries.cs
2  // Displaying the result of a user-selected query in a DataGridView.
3  using System;
4  using System.Data.Entity;
5  using System.Linq;
6  using
7  namespace DisplayQueryResult
8  {
9  public partial class TitleQueries : Form
10 {
11 public TitleQueries()
12 {
13  InitializeComponent();
14 } // end constructor
15 // Entity Framework DBContext
16 private BooksExamples.BooksEntities dbcontext =
17 new BooksExamples.BooksEntities();
18 // load data from database into DataGridView
```

```
19 private void TitleQueries_Load( object sender, EventArgs e )
20 {
21  dbcontext.Titles.Load(); // load Titles table into memory
22 // set the ComboBox to show the default query that
23 // selects all books from the Titles table
24 queriesComboBox.SelectedIndex = 0;
25 } // end method TitleQueries_Load
26 // loads data into titleBindingSource based on user-selected query
27 private void queriesComboBox_SelectedIndexChanged(
28                                         object sender, EventArgs e )
29 {
30 // set the data displayed according to what is selected
31 switch ( queriesComboBox.SelectedIndex )
32 {
33 case 0: // all titles
34 // use LINQ to order the books by title
35  titleBindingSource.DataSource =
36  dbcontext.Titles.Local.OrderBy( book => book.Title1 );
37 break;
38 case 1: // titles with 2014 copyright
39 // use LINQ to get titles with 2014
40 // copyright and sort them by title
41  titleBindingSource.DataSource =
42  dbcontext.Titles.Local
43    .Where( book => book.Copyright == "2014" )
44    .OrderBy( book => book.Title1 );
45 break;
46 case 2: // titles ending with "How to Program"
47 // use LINQ to get titles ending with
48 // "How to Program" and sort them by title
49 titleBindingSource.DataSource =
50 dbcontext.Titles.Local
51  .Where( book =>
52           book.Title1.EndsWith( "How to Program" ) )
53 .OrderBy( book => book.Title1 );
54 break;
55 } // end switch
56  titleBindingSource.MoveFirst(); // move to first entry
57 } // end method queriesComboBox_SelectedIndexChanged
58 } // end class TitleQueries
59 } // end namespace DisplayQueryResult
```

- نقوم في حدث تحميل النموذج بتحميل جميع بيانات جدول الكتب في الذاكرة عن طريق استدعاء الطريقة Load() على الغرض dbcontext من الصف BooksEntities.
- نقوم في حدث اختيار عنصر من القائمة بما يلي:

العنصر الأول: اختيار جميع الكتب مرتبة حسب عناوينها:

يتم ذلك باستخدام الطريقة الموسعة OrderBy على dbcontext.Titles.Local.

العنصر الثاني: اختيار الكتب ذات حقوق النشر في 2014:

يتم استخدام الطريقة الموسعة Where مع المعامل التالي (تعبير لامبدا):

book => book.Copyright == "2014"

يأخذ هذا التعبير معامل غرض Title (يُسميه Book) ويستخدمه لفحص الخاصية copyright (سلسلة نصية في قاعدة البيانات) لتساوي 2014.

يجب أن يُعيد تعبير لامبدا مع الطريقة Where قيمة منطقية. سيتم إرجاع الأغراض من النوع Title التي تكون قيمة التعبير من أجلها True.

نستخدم أيضاً الطريقة الموسعة OrderBy لإعادة الكتب مرتبة وفق عناوين الكتب.

العنصر الثالث: اختيار الكتب التي ينتهي عنوانها بـ How to Program:

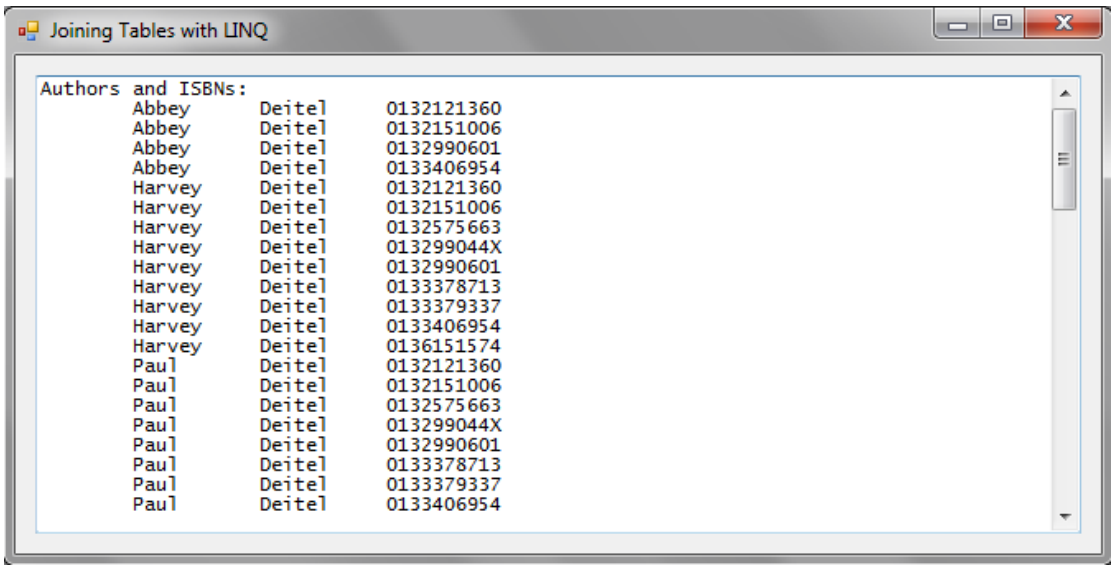
نستخدم الطريقة الموسعة Where مع تعبير لامبدا التالي:

```
book => book.Title1.EndsWith( "How to Program" )
```

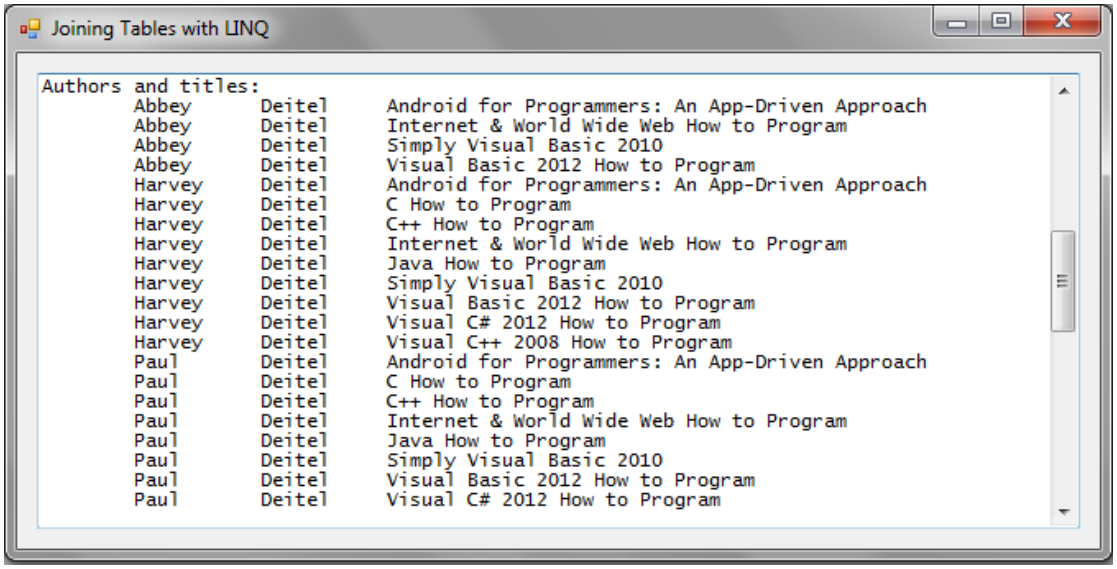
استرجاع البيانات من عدة جداول باستخدام LINQ

نقوم في المثال التالي بالاستعلام عن البيانات من أكثر جدول.

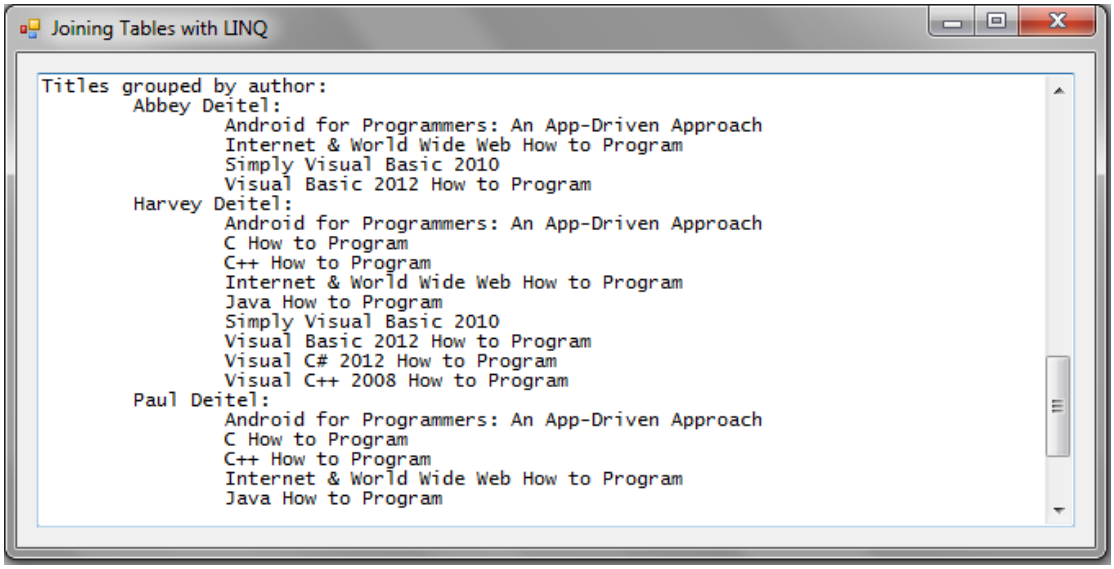
يتم أولاً إظهار قائمة المؤلفين وأرقام الكتب المؤلفين لها. يتم ترتيب المؤلفين وفق الاسم الأخير ومن ثم وفق الاسم الأول:



ثم يتم إظهار قائمة المؤلفين وعناوين الكتب المؤلفين لها. يتم ترتيب المؤلفين وفق الاسم الأخير ومن ثم وفق الاسم الأول:



ثم يتم إظهار الكتب مجمّعة وفق المؤلف:



يكون الكود الموافق:

```
1  // JoiningTableData.cs
2  // Using LINQ to perform a join and aggregate data across tables.
3  using System;
4  using System.Linq;
5  using System.Windows.Forms;
6  namespace JoinQueries
7  {
8  public partial class JoiningTableData : Form
9  {
10 public JoiningTableData()
11 {
12 InitializeComponent();
13 } // end constructor
14 private void JoiningTableData_Load( object sender, EventArgs e )
15 {
16 // Entity Framework DBContext
17 BooksExamples.BooksEntities dbcontext =
18                     new BooksExamples.BooksEntities();
19 // get authors and ISBNs of each book they co-authored
20 var authorsAndISBNs =
21       from author in dbcontext.Authors
22       from book in author.Titles
23       orderby author.LastName, author.FirstName
24       select new { author.FirstName, author.LastName, book.ISBN };
25 outputTextBox.AppendText( "Authors and ISBNs:" );
26 // display authors and ISBNs in tabular format
27 foreach ( var element in authorsAndISBNs )
28 {
29  outputTextBox.AppendText(
30  String.Format( "\r\n\t{0,-10} {1,-10} {2,-10}",
31  element.FirstName, element.LastName, element.ISBN ) );
32 } // end foreach
33 // get authors and titles of each book they co-authored
34 var authorsAndTitles =
35       from book in dbcontext.Titles
36       from author in book.Authors
37       orderby author.LastName, author.FirstName, book.Title1
```

```
38        select new { author.FirstName, author.LastName, book.Title1 };
39  outputTextBox.AppendText( "\r\n\r\nAuthors and titles:" );
40  // display authors and titles in tabular format
41  foreach ( var element in authorsAndTitles )
42  {
43  outputTextBox.AppendText(
44  String.Format( "\r\n\t{0,-10} {1,-10} {2}",
45  element.FirstName, element.LastName, element.Title1 ) );
46  } // end foreach
47  // get authors and titles of each book
48  // they co-authored; group by author
49  var titlesByAuthor =
50          from author in dbcontext.Authors
51          orderby author.LastName, author.FirstName
52          select new { Name = author.FirstName + " " + author.LastName,
53             Titles =
54                       from book in author.Titles
55                       orderby book.Title1
56                       select book.Title1 };
57  outputTextBox.AppendText( "\r\n\r\nTitles grouped by author:" );
58  // display titles written by each author, grouped by author
59  foreach ( var author in titlesByAuthor )
60  {
61  // display author's name
62     outputTextBox.AppendText( "\r\n\t" + author.Name + ":" );
63  // display titles written by that author
64  foreach ( var title in author.Titles )
65  {
66     outputTextBox.AppendText( "\r\n\t\t" + title );
67  } // end inner foreach
68  } // end outer foreach
69  } // end method JoiningTableData_Load
70  } // end class JoiningTableData
71  } // end namespace JoinQueries
```

لاحظ في الكود السابق ما يلي:

الربط بين أسماء المؤلفين وأرقام الكتب المؤلفينها:

يقوم الاستعلام الأول (الأسطر 21 إلى 24) بربط بيانات جدول المؤلفين وجدول الكتب:

```
from author in dbcontext.Authors

from book in author.Titles

orderby author.LastName, author.FirstName

select new { author.FirstName, author.LastName, book.ISBN };
```

تقوم عبارة from الأولى بالحصول على كل مؤلف author من جدول المؤلفين Authors. بينما تقوم عبارة from الثانية باستخدام الخاصية المولدة Titles من الصف Author للحصول على أرقام الكتب ISBN للمؤلف الحالي.

يستخدم نموذج كائن البيانات معلومات المفتاح الأجنبي الموجودة في الجدول AuthorISBN للحصول على أرقام الكتب الموافقة ISBN. تكوّن النتائج المرتبطة لكلا العبارتين from تجميعية collection من المؤلفين وكتبهم، كما تقوم هاتين العبارتين بإدخال متغيرين جديدين (author و book) إلى مجال هذا الاستعلام

يُمكن للعبارات الأخرى التعامل معهم. تقوم العبارة orderby بترتيب النتائج وفق LastName ومن ثم FirstName. يقوم السطر 24 بإنشاء نمط جديد مجهول يحوي الاسم الأول FirstName والاسم الأخير LastName لمؤلف author من جدول المؤلفين Authors مع رقم ISBN لكتاب book من جدول الكتب Titles مؤلف من قبل هذا المؤلف.

الأنماط المجهولة *Anonymous Types*

تسمح الأنماط المجهولة بتعريف صفوف بسيطة تُستخدم لتخزين البيانات وبدون كتابة تعريف الصفوف. يبدأ التصريح عن نمط مجهول بالكلمة المفتاحية new متبوعة بأعضاء الصف ضمن الأقواس { }. سيقوم المترجم بتوليد تعريف لصف يحوي الخصائص المحدّدة بالأعضاء. تكون هذه الخصائص عامة public وللقراءة فقط *read-only*. لايُمكنك تغيير قيمة خاصية بعد إنشاء الغرض. يقوم المترجم بإعطاء اسم للصف إلا أنك لاتعرف هذا الاسم ولذلك عندما تحتاج لتعريف متغير من هذا الصف تستخدم الكلمة المفتاحية var (السطر 41).
يقوم المترجم أيضاً بتعريف الطريقة ToString على الصف والتي تُعيد سلسلة نصية تتألف من الأقواس { } تحوي ضمنها قائمة من الأزواج *PropertyName = value* مفصولة بفواصل.

الربط بين أسماء المؤلفين وعناوين الكتب المؤلفينها:

يقوم الاستعلام الثاني (الأسطر 35 إلى 38) بربط بيانات جدول المؤلفين وجدول الكتب:

```
from book in dbcontext.Titles
from author in book.Authors
orderby author.LastName, author.FirstName, book.Title1
select new { author.FirstName, author.LastName, book.Title1 };
```

بشكل مشابهه للاستعلام الأول.

تجميع الكتب وفق المؤلف:

يقوم الاستعلام الثالث (الأسطر 50 إلى 56) بتجميع الكتب وفق المؤلف:

```
from author in dbcontext.Authors
orderby author.LastName, author.FirstName
select new { Name = author.FirstName + " " + author.LastName,
   Titles =
            from book in author.Titles
            orderby book.Title1
            select book.Title1 };
```

يتكون كل عنصر من نتائج هذا الاستعلام من اسم المؤلف وقائمة الكتب المؤلفها. يقوم الاستعلام بذلك باستخدام استعلام مضمن *nested query* في عبارة select. يقوم الاستعلام الخارجي outer بالدوران على المؤلفين

في قاعدة البيانات. بينما يقوم الاستعلام الداخلي inner بأخذ مؤلف معين واسترجاع كتبه. تقوم عبارة select بإنشاء نمط جديد مجهول مع الخاصيتين:

- الخاصية Name والتي تتكون من الاسم الأول و الاسم الأخير مفصولين بفراغ.
- الخاصية Titles والتي تحوي نتائج الاستعلام الداخلي وتحوي عنوان الكتب المؤلفة من قبل المؤلف الحالي.

لاحظ أننا قمنا بإعطاء أسماء للخصائص.

الفصل الثاني عشر: التعامل مع قواعد البيانات (3)

عنوان الموضوع:

التعامل مع قواعد البيانات (3).

الكلمات المفتاحية:

إنشاء نموذج رئيسي/فرعي.

ملخص:

نتابع في هذا الفصل مواضيع التعامل مع قواعد البيانات فنعرض لإنشاء نموذج رئيسي/فرعي ثم ننهي الموضوع بمثال تطبيقي.

أهداف تعليمية:

يتعرف الطالب في هذا الفصل على:

- إنشاء نموذج رئيسي/فرعي.
- مثال تطبيقي.

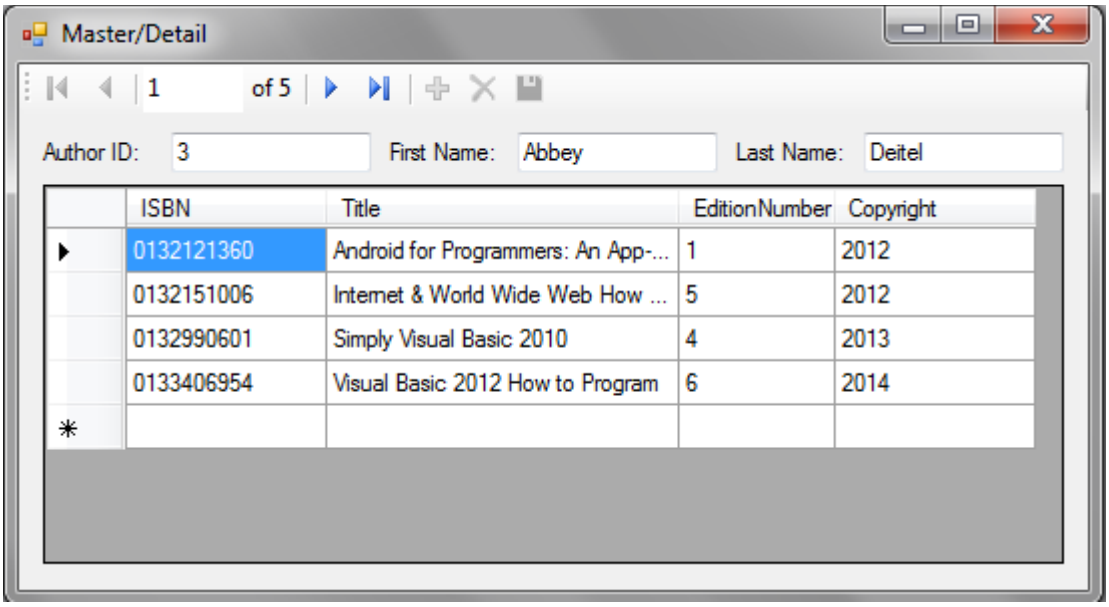
المخطط:

التعامل مع قواعد البيانات (3)

- 2 وحدة (Learning Objects)

إنشاء نموذج رئيسي/فرعي Master/Detail

يهدف هذا التطبيق إلى إنشاء نموذج رئيسي/فرعي يعرض كتب كل مؤلف كما يلي:



اتبع الخطوات التالية:

الخطوة الأولى: إنشاء مشروع جديد:

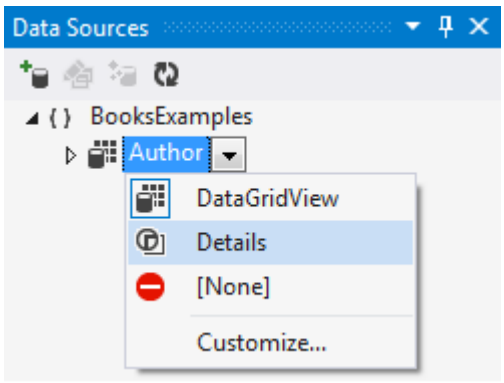
قم بإنشاء مشروع جديد من النمط Windows Forms Application وسمّه Master/Detail. سم النموذج Details.cs وقم بوضع خاصية النص له Master/Detail.

الخطوة الثانية: إضافة مصدر بيانات للجدول Author:

قم بإضافة مصدر بيانات إلى الجدول Author.

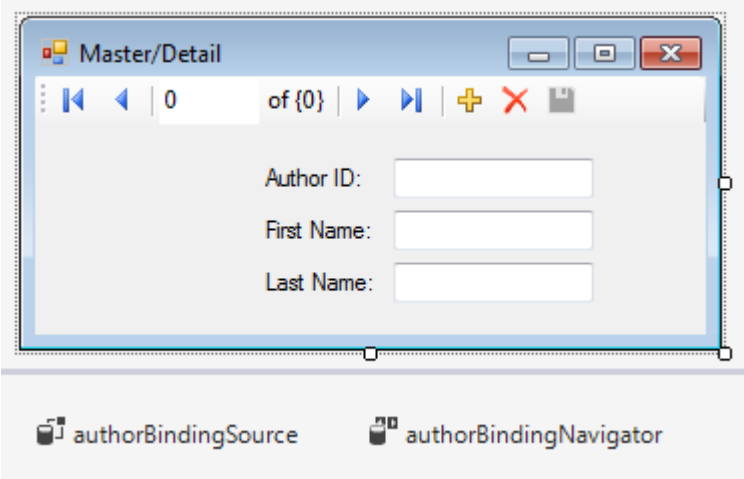
الخطوة الثالثة: إنشاء النموذج:

- قم بالنقر على العقدة Title في نافذة مصدر البيانات Data Sources لإظهار القائمة المنسدلة واختر Details.

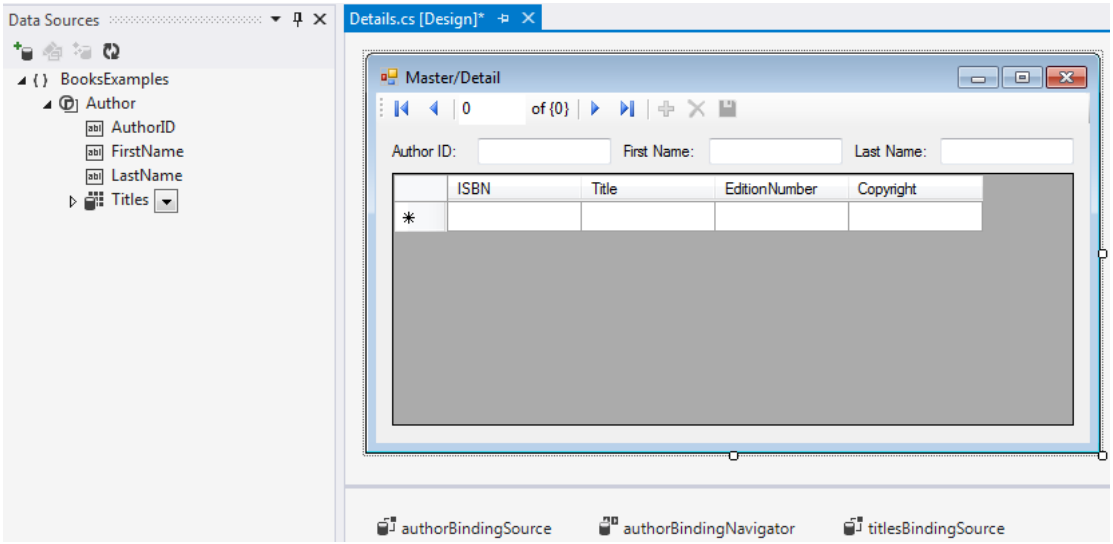


- ثم قم بسحب العقدة Author إلى النموذج.

- سيتم إضافة ثلاثة صناديق نص إلى النموذج لإظهار بيانات المؤلفين وعنصر الربط authorBindingSource وعنصر التجوال authorBindingNavigator.



- قم بسحب العقدة Titles الموجودة تحت العقدة Author إلى النموذج.
- سيتم إضافة عنصر الربط titleBindingSource إلى النموذج مع شبكة لعرض بيانات الكتب DataGridView.



نقوم بكتابة الكود التالي:

```
1  // Details.cs
2  // Using a DataGridView to display details based on a selection.
3  using System;
4  using System.Data.Entity;
5  using System.Linq;
6  using System.Windows.Forms;
7  namespace MasterDetail
8  {
9  public partial class Details: Form
10 {
11 public Details()
12 {
```

```
13  InitializeComponent();
14 } // end constructor
15 // Entity Framework DBContext
16 BooksExamples.BooksEntities dbcontext =
17   new BooksExamples.BooksEntities();
18 // initialize data sources when the Form is loaded
19 private void Details_Load( object sender, EventArgs e )
20 {
21 // load Authors table ordered by LastName then FirstName
22 dbcontext.Authors
23 .OrderBy( author => author.LastName )
24 .ThenBy( author => author.FirstName )
25 .Load();
26 // specify DataSource for authorBindingSource
27 authorBindingSource.DataSource = dbcontext.Authors.Local;
28 } // end method Details_Load
29 } // end class Details
30 } // end namespace MasterDetail
```

- نقوم في الأسطر 16 و17 بإنشاء الغرض `dbcontext`.
- نقوم في الأسطر 23 إلى 25 بتحميل تسجيلات جدول المؤلفين في الذاكرة مرتبة وفق الاسم الأخير ومن ثم وفق الاسم الأول.
- يقوم السطر 27 بإسناد dbcontext.Authors.Local إلى `authorBindingSource.DataSource` مما يُحقق الربط مع عناصر التحكم.
- يتم بشكل آلي ربط titlesBindingSource.DataSource مع كتب المؤلف الحالي مما يُظهر كتب المؤلف المحدد الحالي في الشبكة.

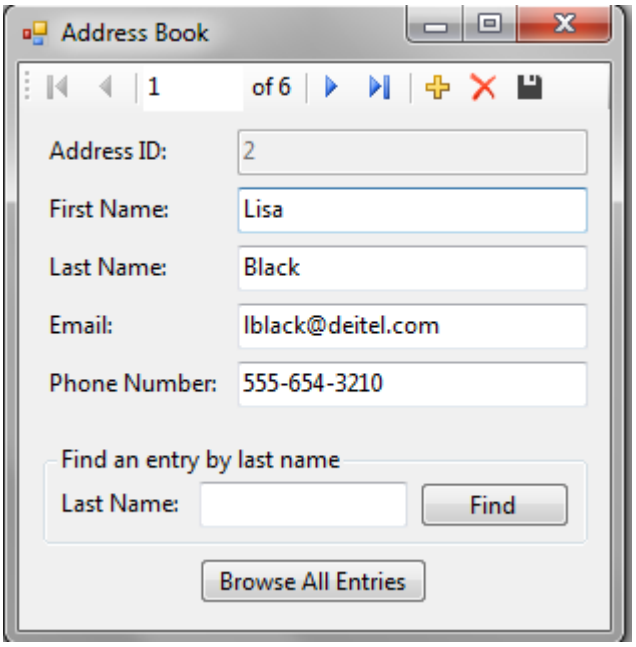
مثال تطبيقي: دفتر عناوين Address Book

نقوم في هذا المثال بإنشاء تطبيق دفتر عناوين AddressBook يسمح للمستخدمين بما يلي على قاعدة البيانات AddressBook.mdf:

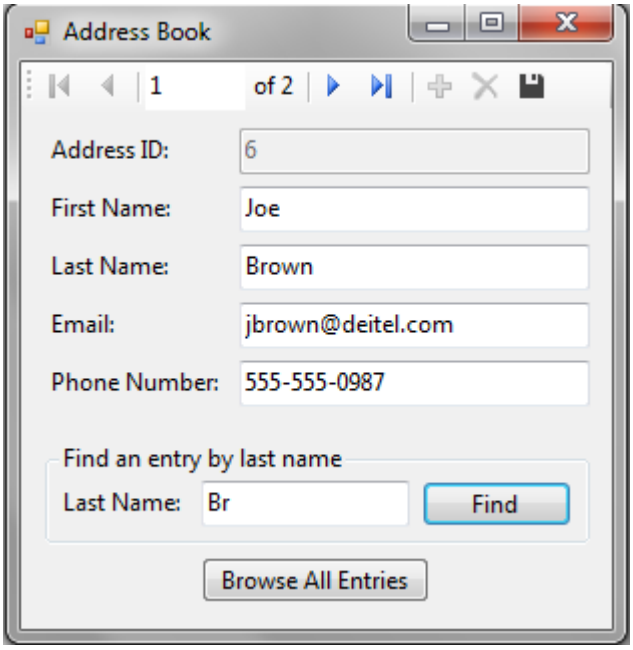
- إضافة شخص جديد.
- البحث عن الأشخاص الذين يبدأ اسمهم ببعض الحروف.
- تعديل بيانات شخص.
- حذف بيانات شخص.

يقوم التطبيق بعرض البيانات من خلال صناديق نص. كما سيسمح شريط التنقل BindingNavigator في الأعلى للتنقل بين التسجيلات والإضافة والحذف.

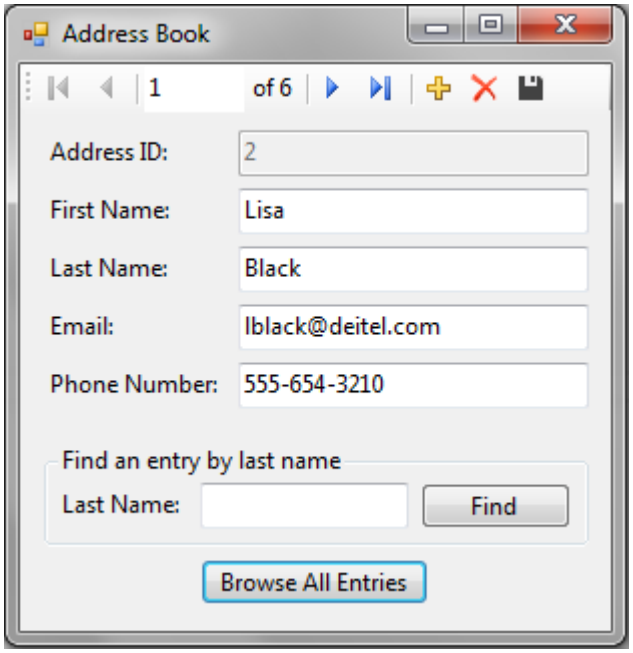
- يُمكنك استخدام شريط التنقل للتنقل بين التسجيلات:



- أدخل بعض الأحرف في صندوق النص أسفل النافذة ثم انقر على زر البحث Find لإظهار الأشخاص الذين يبدأ اسمهم الأخير بهذه الأحرف.



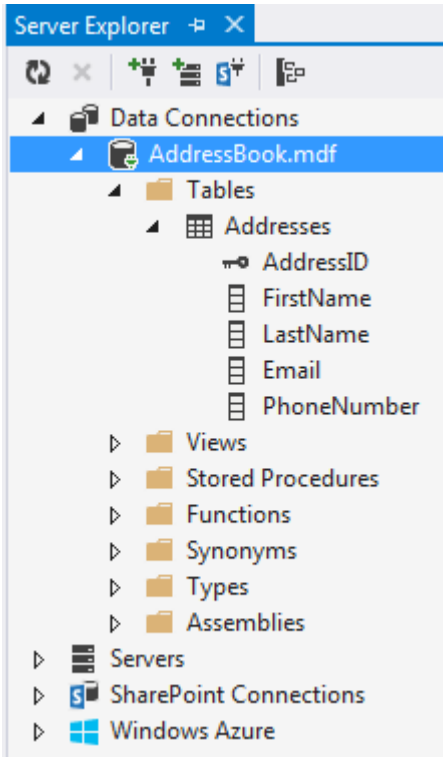
- انقر على الزر معاينة الجميع Browse All Entries للعودة إلى إظهار جميع التسجيلات:



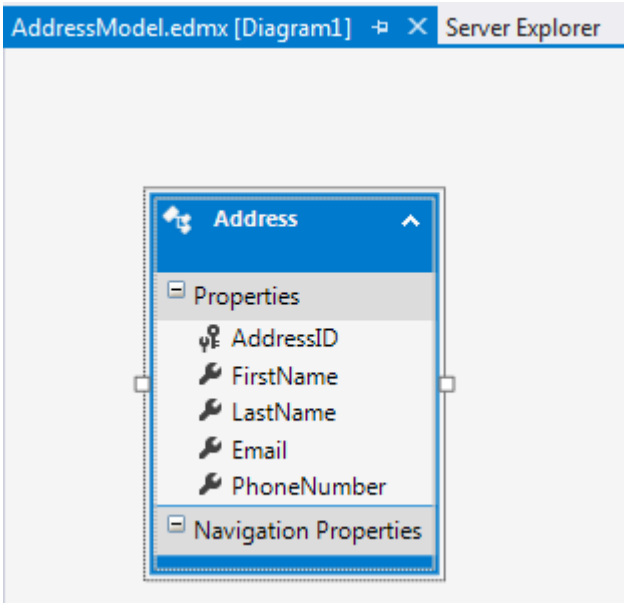
مراحل العمل

الخطوة الأولى: إنشاء مكتبة الصف التي تحوي نموذج كائن البيانات:

- أنشئ كما تعلمت سابقاً مكتبة الصف AddressExample والتي ستحوي نموذج كائن البيانات للتعامل مع قاعدة البيانات AddressBook.mdf.
- تحوي قاعدة البيانات جدولاً واحداً Addresses:



- أنشئ نموذج كائن البيانات وأضف إليه الجدول السابق وسمّه AddressModel.edmx.

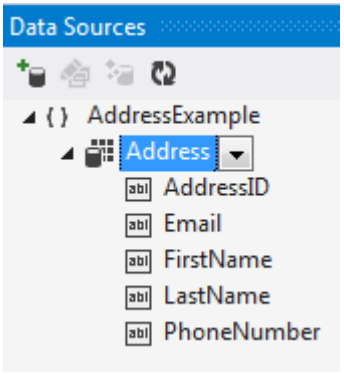


الخطوة الثانية: إنشاء تطبيق ويندوز Windows Forms Application:

- أنشئ تطبيق ويندوز ضمن نفس الحل AddressExample وسمّه AddressBook.
- قم بتسمية ملف النموذج Contacts.cs. وعدّل خاصية النص للنموذج إلى Address Book.
- اجعل المشروع AddressBook هو مشروع البداية startup project.

الخطوة الثالثة: إضافة الغرض Address كمصدر بيانات:

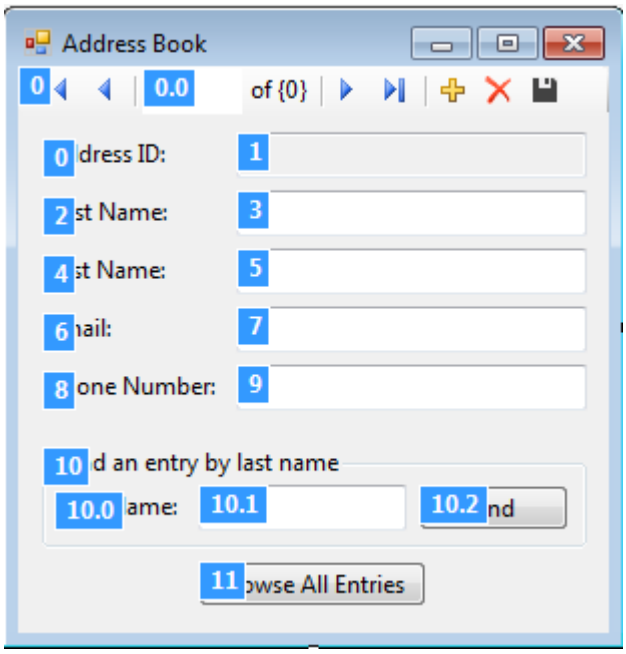
- قم بفتح نافذة مصادر البيانات وأضف الغرض Address.



- قم بإضافة المراجع اللازمة وسلسلة الاتصال كما تعلمت سابقاً.

الخطوة الرابعة: عرض تفصيل كل تسجيلة:

- في عرض تصميم النموذج، انقر على العقدة Address في نافذة مصادر البيانات واختر Details من القائمة المنسدلة ثم قم بسحب العقدة Address إلى النموذج.
- سيتم بشكل تلقائي إنشاء BindingNavigator وإضافة أزواج من اللصاقات وصناديق النص موافقة لأعمدة الجدول بترتيب الحقول أبجدياً.
- قم بتعديل أماكن توضع عناصر التحكم وانتبه بعدها لضبط خاصية ترتيب الجدوله لها.
- يُمكنك اختيار VIEW → Tab Order ومن ثم النقر على صناديق النص من الأعلى للأسفل لتحديد ترتيب الجدولة.



الخطوة الخامسة: إضافة العناصر اللازمة للبحث:

قم بإضافة صندوق نص وسمّه findTextBox وزر أمر للبحث findButton ضمن صندوق مجموعة GroupBox.

أضف أيضاً زر أمر browseAllButton تحت صندوق المجموعة للسماح للمستخدم باستعراض جميع التسجيلات وضع خاصية النص Browse All Entries.

يكون الكود الموافق:

```
/ Contact.cs
// Manipulating an address book.
using System;
using System.Data;
using System.Data.Entity;
using System.Data.Entity.Validation;
using System.Linq;
using System.Windows.Forms;
namespace AddressBook
{
public partial class Contacts: Form
{
public Contacts()
{
InitializeComponent();
} // end constructor
// Entity Framework DBContext
private AddressExample.AddressBookEntities dbcontext = null;
// fill our addressBindingSource with all rows, ordered by name
private void RefreshContacts()
{
// Dispose old DBContext, if any
if ( dbcontext != null )
  dbcontext.Dispose();
// create new DBContext so we can reorder records based on edits
dbcontext = new AddressExample.AddressBookEntities();
// use LINQ to order the Addresses table contents
// by last name, then first name
dbcontext.Addresses
.OrderBy( entry => entry.LastName )
.ThenBy( entry => entry.FirstName )
.Load();
// specify DataSource for addressBindingSource
addressBindingSource.DataSource = dbcontext.Addresses.Local;
addressBindingSource.MoveFirst(); // go to first result
findTextBox.Clear(); // clear the Find TextBox
} // end method RefreshContacts
// when the form loads, fill it with data from the database
private void Contacts_Load( object sender, EventArgs e )
{
 RefreshContacts(); // fill binding with data from database
} // end method Contacts_Load
// Click event handler for the Save Button in the
// BindingNavigator saves the changes made to the data
private void addressBindingNavigatorSaveItem_Click(
object sender, EventArgs e )
{
```

```
48  Validate(); // validate input fields
49  addressBindingSource.EndEdit(); // complete current edit, if any
50 // try to save changes
51 try
52 {
53  dbcontext.SaveChanges(); // write changes to database file
54 } // end try
55 catch ( DbEntityValidationException )
56 {
57  MessageBox.Show( "Columns cannot be empty",
58   "Entity Validation Exception" );
59 } // end catch
60 RefreshContacts(); // change back to initial unfiltered data
61 } // end method addressBindingNavigatorSaveItem_Click
62 // use LINQ to create a data source that contains only people
63 // with last names that start with the specified text
64 private void findButton_Click( object sender, EventArgs e )
65 {
66 // use LINQ to filter contacts with last names that
67 // start with findTextBox contents
68 var lastNameQuery =
69  from address in dbcontext.Addresses
70  where address.LastName.StartsWith( findTextBox.Text )
71  orderby address.LastName, address.FirstName
72  select address;
73 // display matching contacts
74 addressBindingSource.DataSource = lastNameQuery.ToList();
75 addressBindingSource.MoveFirst(); // go to first result
76 // don't allow add/delete when contacts are filtered
77 bindingNavigatorAddNewItem.Enabled = false;
78 bindingNavigatorDeleteItem.Enabled = false;
79 } // end method findButton_Click
80 // reload addressBindingSource with all rows
81 private void browseAllButton_Click( object sender, EventArgs e )
82 {
83  // allow add/delete when contacts are not filtered
84  bindingNavigatorAddNewItem.Enabled = true;
85  bindingNavigatorDeleteItem.Enabled = true;
86  RefreshContacts(); // change back to initial unfiltered data
87 } // end method browseButton_Click
88 } // end class Contacts
89 } // end namespace AddressBook
```

الطريقة RefreshContacts:

كما وضحنا في أمثلة سابقة يجب ربط عنصر التحكم addressBindingSource والذي يتحكم بعناصر الواجهة مع dbcontext الذي يتفاعل مع قاعدة البيانات. قمنا في هذا المثال بالتصريح عن dbcontext في السطر 18. إلا أننا نقوم بإنشائه وتفعيل الربط في الطريقة RefreshContacts (الأسطر 20 إلى 37) والتي يتم استدعاؤها في العديد من الطرق الأخرى.

يتم أولاً فحص الكائن dbcontext فيما إذا كان null أم لا. في حال كونه ليس null، يتم استدعاء الطريقة Dispose عليه ومن ثم إنشائه من جديد. نقوم بذلك بهدف إعادة ترتيب السجلات فقد يكون المستخدم قام بتعديل الأسماء (حيث ستبقى التسجيلات في نفس ترتيبها الأولي). تقوم الأسطر 30 و 31 بترتيب الأغراض

وفق الاسم الأخير ومن ثم وفق الاسم الأول. ثم يتم تحميل الأغراض في الذاكرة. يتم في السطر 34 الربط مع بيانات الذاكرة.

يقوم السكر 35 بالانتقال إلى أول غرض.

الطريقة Contacts_Load:

يتم في هذه الطريقة استدعاء الطريقة السابقة RefreshContacts.

الطريقة addressBindingNavigatorSaveItem_Click:

يتم في هذه الطريقة حفظ التعديلات في قاعدة البيانات.

الطريقة findButton_Click:

تستخدم هذه الطريقة استعلام LINQ للبحث عن الأشخاص الذين تبدأ أسمائهم بالأحرف المدخلة في صندوق البحث.

لايُمكنك ربط نتائج استعلام LINQ إلى BindingSource's DataSource مباشرة. ولذلك يستخدم السطر 74 الطريقة ToList للحصول على تمثيل قائمة List لنتائج الاستعلام.

عندما يتم الربط مع قائمة، يقوم dbcontext بملاحقة تعديلات تسجيلات القائمة فقط. وإن أي تسجيلات مضافة أو محذوفة من البيانات المفلترة سوف تُفقد. ولذلك نقوم بإلغاء تفعيل كل من الأزرار Add new و Delete أثناء عملية التصفية.

الطريقة browseAllButton_Click:

تقوم هذه الطريقة باستدعاء الطريقة RefreshContacts وإعادة تفعيل الأزرار Add new و Delete.