الباب السابع

المنشآت والقوائم

(Structs & Enumerations)

محتويات الباب

- نمط المنشأ (Structs)
- تمرير أهداف المنشأ وأهداف الفصيلة
- نمط القائمة (Enumerations)
- استخدام القوائم
- مقدمة عن الصفات (Attributes)

نمط المنشأ (Structs)

بالرغم من أن الكلمة struct هى أساساً اختصار الكلمة structure ولكنها أصبحت من مفردات لغات البرمجة المتعارف عليها ، بالرغم من أنها لم تدخل قاموس اللغة الإنجليزية بعد. وفى لغة سى# فإن المنشأ (struct) عبارة عن نمط مبتكر مماثل للفصيلة ولكنه ينتمى إلى أنماط القيمة (Value Types).

ملاحظة إلى مبرمجى لغة سى++:

يختلف استخدام ومفهوم كلمة struct بين اللغتين اختلافاً جذرياً. ففى لغة سى++ يجوز تعريف الفصيلة بكلمة struct ، وليس هذا هو الحال فى سى#.

ويستخدم المنشأ أساساً فى اختزان الأنماط البسيطة التى تماثل فى سلوكها الأنماط المبنية فى اللغة مثل منشأ النقطة التى تحتوى على حقلين عدديين ، أو منشأ اللون ، أو المضلع. وبالرغم من أننا استخدمنا هدف النقطة فى ضرب الأمثلة للفصائل ، ولكن المنشأ هو أنسب الأنماط لتمثيل النقطة حيث أنه لا يستهلك إلاّ حيزاً صغيراً فى الذاكرة. ولو أننا أنشأنا مصفوفة كبيرة من النقط الممثلة فى فصائل ، فإن سعة الذاكرة المطلوبة سوف تكون كبيرة حيث أن كل نقطة يلزمها مرجع يشير إليها.

إعلان المنشأ

لإعلان المنشأ تستخدم الكلمة struct بالصورة الآتية:

[attributes] [modifiers] **struct** identifier [:interfaces] body [;]

حيث:

- attributes: صفات اختيارية (Attributes) ، سيلى الحديث عنها.
- modifiers: مجموعة مناسبة من معدّلات التوصل (Access modifiers) أو الكلمة new
- identifier: اسم المنشأ
- interfaces: وصلة بينية أو أكثر يتم تطبيقها بواسطة المنشأ
- body: الجسم الذى يحتوى على أعضاء المنشأ

والآتى بعد مثال لمنشأ يستخدم درجة الحماية العامة:

```
public struct MyStruct
{
  // أعضاء المنشأ
}
```

استخدام المنشأ

عند استخدام المنشآت تجب مراعاة النقاط التالية:

❖ بخلاف الفصيلة فإن المنشأ لا يجوز شحن حقوله مبدئياً ، أى أن العبارات التالية سوف تؤدى إلى رسالة خطأ من المترجم:

```
struct MyStruct
{
  int x = 0;   // error
  int y = 0;   // error
}
```

❖ يجـــوز أن يســـتخدم المنشأ أسلوب بناء أو أكثر ، ولكن لايجوز تعـــريف أسلوب بناء سابق التعريف (بدون بارامترات). أى أن أسلوب البناء التالى غير جائز:

```
struct MyStruct
{
  // ...
  MyStruct() {};  // error
}
```

ويمد المترجم دائماً بأسلوب البناء سابق التعريف لشحن أعضاء المنشأ بالقيمة الابتدائية.

❖ مـــن الجائز عند خلق الأمثلة من المنشأ أن تستخدم المؤثر new كما فى العبارة:

```
MyStruct ms = new MyStruct();
```

ويمكـــنك إعلان الهدف بدون استخدام كلمة new ، مع ملاحظة أنك لا تستطيع استخدام المنشأ قبل شحن الأعضاء.

❖ ومن خصائص المنشأ أنه لا يرث ولا يورّث ، ولكنه يجوز أن يُطبق وصلة بينية أو أكثر. كالمثال الآتى:

```
struct MyStruct: IMyInterface
{
  // ...
}
```

وتجـــب ملاحظـــة أنه بالرغم من أن المنشأ لا يدخل فى شجرة الوراثـــة ، ولكنه ـــ بطريقة ضمنية ـــ يرث الفصيلة الأساسية التى تمثل جـــذر شـــجـــرة الفصائل جميعاً وهى فصيلة الهدف (object).

مثال (7-1)

فـــى هذا المثال نستخدم المنشأ فى بناء الألوان الأساسية الثلاثة: الأحمر R ، والأزرق B ، الأخضـــر G . والألـــوان الـــثلاثة عبارة عن أعداد صـــحيحة. كمـــا انـــنا قـــد استخدمنا الفصيلة MyClass لبناء الأهداف واستدعاء الأساليب بها.

```
// Example 7-1.cs
// struct example

using System;

public struct Color
{
  // Fields:الحقول الخاصة
  private int r;
  private  int g;
  private int b;

  // Constructor: أسلوب بناء ذو بارامترات
  public Color(int r, int g, int b)
  {
   this.r = r;
   this.g = g;
   this.b = b;
  }

  // Override the method ToString(): ركوب أسلوب التحويل
  public override string ToString()
  {
    return (String.Format("Red = {0}, Green = {1}, Blue = {2}",
r, g, b));
  }
}

class MyClass
```

```
{
  static void Main()
  {
    // Declare objects:إعلان الأهداف
    Color c1 = new Color();
    Color c2 = new Color(100, 100, 0);
    // Print objects:طباعة الأهداف
    Console.WriteLine("The first object:");
    Console.WriteLine("The colors are: {0}", c1);
    Console.WriteLine("The second object:");
    Console.WriteLine("The colors are: {0}", c2);
  }
}
```

تنفيذ البرنامج:

```
The first object:
The colors are: Red = 0, Green = 0, Blue = 0
The second object:
The colors are: Red = 100, Green = 100, Blue = 0
```

ملاحظات على البرنامج السابق:

نلاحظ فى هذا المثال أننا قد بنينا هدفين من المنشأ هما c1, c2. الأول يستخدم أسلوب البناء سابق التعريف ، ولذلك فإن جميع أعضائه كانت تحتوى على القيمة 0. أما الثانى فقد تم بناؤه باستخدام البارمترات 100, 100, 0 التى خصصت هذه الأعداد إلى الألوان الثلاثة R, G , B بالترتيب.

نلاحظ أيضاً طباعة الأهداف مباشرة بالعبارة:

```
Console.WriteLine("The colors are: {0}", c1);
```

وقد استلزم ذلك ركوب أسلوب التحويل ToString لتغيير سلوكه كما رأينا فى الباب السابق.

مثال (7-2)

فـــى هـــذا المـــثال نستخدم مجموعة من الخواص للتوصل إلى أعضاء المنشـــأ Color. ويجوز استخدام الخواص مباشرة بالتخصيص لها ، كما يجوز استدعاء أساليب البناء كطريق بديل.

```
// Example 7-2.cs
// Using properties with structs

using System;

public struct Color
{
  // Fields:
  private int r;
  private  int g;
  private int b;

  // Properties: استخدام الخواص
  public int R
  {
    get
    {
      return r;
    }
    set
    {
      r = value;
    }
  }
  public int G
  {
    get
    {
      return g;
    }
    set
    {
      g = value;
```

```
    }
  }
  public int B
  {
    get
    {
      return b;
    }
    set
    {
      b = value;
    }
  }

  // Override ToString():
  public override string ToString()
  {
    return (String.Format("Red = {0}, Green = {1}, Blue = {2}",
R, B, G));
  }
}

class MyClass
{
  static void Main()
  {
    Color c1 = new Color();
    // طباعة الهدف سابق التعريف
    Console.WriteLine("The colors are: {0}", c1);
    c1.R = 100;
    c1.B = 100;
    c1.G = 0;
    // طباعة الهدف بعد تغيير الخواص
    Console.WriteLine("The colors are: {0}", c1);
  }
}
```

تنفيذ البرنامج:

```
The colors are: Red = 0, Green = 0, Blue = 0
The colors are: Red = 100, Green = 100, Blue = 0
```

ملاحظات على البرنامج السابق:

لاحظ أننا قد استخدمنا هنا هدفاً واحداً تم إنشاؤه باستخدام أسلوب البناء سابق التعريف. ثم استخدمنا هنا مجموعة من الخواص لتمثيل كل لون وهذا قد سمح لنا بالتخصيص لهذه الخواص مباشرة مثل:

```
c1.R = 100;
```

وقد طبعنا أعضاء الهدف بالقيم سابقة التعريف ثم طبعنا الأعضاء بعد تغيير قيمتها عن طريق الخواص.

ومن البديهى أنه يمكنك استخدام أسلوب بناء ذى ثلاثة بارامترات كبديل لهذه الطريقة.

تدريب (1-7)

من البديهى أنه يمكنك أن تضع كل محتويات البرنامج بداخل وعاء المنشأ ، تماماً كما هو الحال مع الفصائل. وفى هذه الحال فإن درجات التوصل إلى الأعضاء سوف تختلف عنها عندما يكون هناك وعاء للأعضاء ووعاء للأسلوب الرئيسى. جرب أن تعيد صياغة المثال السابق بحيث يقع بالكامل بداخل المنشأ. جرب أيضاً أن تمزج المثالين السابقين معاً بحيث يحتوى المنشأ على كلٍ من أساليب البناء والخصائص.

تمرير أهداف المنشأ وأهداف الفصيلة

رأينا من قبل نتيجة تمرير المتغيرات كبارامترات إلى الأساليب بالقيمة أو بالمرجع. وعلمنا أن تمريرها بالمرجع يمكن أن يجعل المتغيرات

تحتفظ بما يجرى عليها من تغيرات ، أما تمريرها بالقيمة فإنه يؤدى إلى تعديل النسخة الموجودة فى الأسلوب أما المتغيرات الأصلية فتبقى على حالهـــا. ولأن المنشأ من أنماط القيمة فإن أهدافه تخلق فى حيز الذاكرة المســـمى بالخـــزنة (Stack) وعـــند تمريرها إلى الأساليب فإنها تمرر بالقيمة. أما الفصيلة فإن أهدافها تخلق فى حـــيز الذاكرة المسمى بالكوم (Heap) وعند تمريرها إلى الأساليب فإنها تمرر كمرجع. والمثال التالى يوضح هذا المفهوم.

مثال (7-3)

فـــى هذا المثال نعلن عن فصيلة MyClass و منشأ MyStruct. ويحتوى كل منهما على حقل عددى صحيح. ثم نبنى هدفين ، الأول من الفصيلة MyClass والثانى من المنشأ MyStruct ، ونشحن كل حقل بالقيمة 555. ثم نمرر الهدفين إلى الأساليب MyMethod1 و MyMethod2 لتغيير قيمة الحقـــول بهما إلى القيمة 100. ولكن النتيجة توضح أن التغيير قد جرى فقط على حقل الفصيلة دون المنشأ.

```
// Example 7-3.cs
// Passing struct & class objects to methods

using System;

class MyClass
{
  public int classField;
}

struct MyStruct
{
```

```
  public int structField;
}

class MainClass
{
  public static void MyMethod1(MyStruct s)
  {
    s.structField = 100;
  }
  public static void MyMethod2(MyClass c)
  {
    c.classField = 100;
  }

  static void Main()
  {
    // Create class and struct objects:
    MyStruct sObj = new MyStruct();
    MyClass cObj = new MyClass();

    // Initialize the values of struct and class objects:
    sObj.structField = 555;
    cObj.classField = 555;

    // Print results:
    Console.WriteLine("Results before calling methods:");
    Console.WriteLine("Struct member = {0}", sObj.structField);
    Console.WriteLine("Class member = {0}\n", cObj.classField);

    // Change the values through methods:
    MyMethod1(sObj);
    MyMethod2(cObj);

    // Print results:
    Console.WriteLine("Results after calling methods:");
    Console.WriteLine("Struct member = {0}", sObj.structField);
    Console.WriteLine("Class member = {0}", cObj.classField);
  }
}
```

تنفيذ البرنامج:

```
Results before calling methods:
Struct member = 555
Class member = 555

Results after calling methods:
Struct member = 555   لم تتغير القيمة
Class member = 100    تغيرت القيمة
```

تدريب (7-2)

عـدّل نمط الحقل بكل من المنشأ والفصيلة ليكون string بدلاً من int ، وتحقق من أن نمط الحقل (سواء كان نمط قيمة أو مرجع) لا يؤثر على النتيجة.

نمط القائمة (Enumerations)

القائمـة نمـط مـن أنماط القيمة يستخدم لتخزين مجموعة محددة من البـيانات الثابتة مثل أيام الأسبوع أو شهور العام أو تقديرات الامتحان وهكذا.

إعلان القائمة

يتم إعلان القائمة باستخدام الكلمة **enum** بالصورة الآتية:

```
[attributes] [modifiers] enum identifier [:base-type]
{enumerator-list};
```

حيث:

- attributes: صفات اختيارية تكتب بين الأقواس المربعة "[]" (سوف يلى الحديث عنها).
- modifiers: المعدّلات المسموح بها ، وهى تتكون من واحد أو أكثر من معدلاّت التوصل ، أو الكلمة new.
- identifier: اسم القائمة
- base-type: نمط أعضاء القائمة. وقد يكون أياً من الأنماط الصحيحة فيما عدا نمط اللبنة (char). والنمط سابق التعريف هو int.
- enumerator-list: عناصر القائمة ، ويفصل كل عنصر عن الآخر بفاصلة ، ويجوز أن يحتوى أى عنصر على تخصيص بقيمة ما.

استخدام القوائم

الآتى بعد مثال لقائمة تحتوى على أيام الإسبوع:

```
enum WeekDays {Mon, Tue, Wed, Thu, Fri, Sat, Sun};
```

إن العناصر المتتابعة فى القائمة هى مجرد أسماء مبتكرة لاتعنى شيئاً بالنسبة للكومبيوتر ، ولكن هناك نمط يوجد فى خلفية العناصر وهو النمط int بصورة سابقة التعريف. بمعنى أن أعضاء القائمة فى الحقيقة

هـــى: 0, 1, 2, 3, 4, 5, 6. أى أن قيمة المتغير Mon هى 0 ، والمتغير Tue هى 1 ، وهكذا.

ومـــن الجائز إرغام هذه القيم المتسلسلة أن تبدأ من قيمة معينة مثل 1 ، وذلك بالتخصيص لأول عنصر فى القائمة. أى:

```
enum WeekDays {Mon=1, Tue, Wed, Thu, Fri, Sat, Sun};
```

كما يجوز التخصيص لأى عنصر بالقائمة ، وهذا يؤثر على ما يليه من العناصر ، مثل:

```
enum WeekDays {Mon, Tue, Wed=8, Thu, Fri, Sat, Sun};
```

إن هـــذا التخصيص يجعل العناصر التالية للعنصر Wed تحمل القيم 9, 10, 11, 12. أمـــا العنصـــر الأول (Mon) فهـــو لا زال يحمل القيمة 0 والثانى (Tue) يحمل القيمة 1.

وللتعبير عن أحد العناصر خارج القائمة فإنه يلزم تأهيله باسم القائمة ، مثل: WeekDays.Mon و WeekDays.Fri وهكذا.

وعـــند الـــتعامل مع عناصر القائمة كمتغيرات فمن اللازم إجراء عملية تحويل صريح إلى أى نمط صحيح (Integral type) ، مثل:

```
int Monday = (int) WeekDays.Mon;
long Monday = (long) WeekDays.Mon;
```

مثال (4-7)

فـــى هـــذا المثال نعلن عن ثلاثة قوائم تختص بأيام الأسبوع ، وفصول الســـنة ، وتقديرات النجاح. ونلاحظ أننا قد استخدمنا الرخص المختلفة

التى تسمح بها القواعد ، لمجرد ضرب المثال.

```
// Example 7-4
// enum example

using System;

// Declarations of enumerations:
enum WeekDays {Mon, Tue, Wed, Thu, Fri, Sat, Sun};
enum Seasons {Summer=1, Fall, Winter, Spring};
enum Grades {Pass=65, Good=75, VeryGood=85, Distinct=100};

class MyClass
{
  static void Main()
  {
    int sunday = (int) WeekDays.Sun;
    short summer = (short) Seasons.Summer;
    byte vGood = (byte) Grades.VeryGood;
    Console.WriteLine("Monday = {0}", sunday);
    Console.WriteLine("Summer = {0}", summer);
    Console.WriteLine("Very Good = {0}", vGood);
  }
}
```

تنفيذ البرنامج:

```
Monday = 6
Summer = 1
Very Good = 85
```

مقدمة عن الصفات (Attributes)

ربما يتعدى الحديث عن الصفات حدود هذا الكتاب ولكننا مع ذلك سوف نقدم جرعة خفيفة كمقدمة لهذا الموضوع.

الصفات هى كلمات إضافية يمكن أن نعدّل بها إعلانات الأنماط والأساليب والخواص إلى آخره ، وذلك بغرض إكسابها سلوكاً بعينه. وتكتب الصفات بين قوسين مربعين ، مثل:

```
[StructLayout (LayoutKind.Sequential)]
```

لاحظ أن الأقواس المربعة جزء من الصفات. لا تخلط بينها وبين أقواس الكلكمات الاختيارية.

وهناك صفات معرفة فى مكتبة دوت.نت ، كما أن هناك صفات مبتكرة تستطيع أن تبتكرها بنفسك وتضيفها إلى الإعلانات. وفى هذه الفقرة سوف نلقى نظرة على أهم استخدامات الصفات بمكتبة دوت.نت.

استدعاء دوال الكود غير المحكوم (Import Unmanaged Code)

كما ذكرنا من قبل أن الكود غير المحكوم هو كود اللغات التى لا تستخدم دوت.نت (أو الكود المحتوى على مؤشرات). ويمكنك أن تستدعى أحد دوال الكود غير المحكوم باستخدام الصفات. ولنفرض أنك ترغب فى استدعاء الدالة النوافذية **MessageBoxA()** (وهى دالة صندوق الرسالة) الموجودة فى ملف المكتبة:

C:\WINDOWS\system32\user32.dll

لكى تحقق ذلك اتبع الآتى:

- أعلن عن الدالة المطلوبة باستخدام الكلمات **extern** و **static** أى:

```
static extern int MessageBoxA(int h, string m, string c, int type);
```

ملاحظـــة: إذا كنت لا تتذكر صيغة هذه الدالة اكتب البرنامج باستخدام الإستوديو المرئى وسوف يساعدك فى بناء الدالة.

- أضف الصفة DllImport قبل إعلان الدالة كالآتى:

```
[DllImport("user32.dll")]
static extern int MessageBoxA(int h, string m, string c, int type);
```

إن هـــذه الصـــفة تخـــبر البرنامج أن الدالة المزمع استيرادها توجد فى المكتـــبة user32.dll. والصـــفة تمكنك من استيراد أى دالة من المكتبة المذكورة بين علامتى الاقتباس.

- أضف توجيهاً لاستخدام الحيز:

System.Runtime.InteropServices

مثال (7-5)

فـــى هـــذا المثال نرى تطبيقاً لاستيراد الدالة MessageBoxA من المكتبة user32.dll.

```
// Example 7-5.cs
// Pinvoke example

using System.Runtime.InteropServices;

class PlatformInvokeTest
{
  [DllImport("user32.dll")]
  static extern int MessageBoxA(int h, string m, string c, int type);
  static int Main()
  {
    return MessageBoxA(0, "Hello World!", "My Message Box", 0);
  }
}
```

تنفيذ البرنامج:

عند تنفيذ هذا البرنامج فإنه يفتح نافذة رسالة مكتوب فيها العبارة " Hello World!" وتحمل الاسم My Message Box. أنظر الشكل التالى:

تنفيذ البرنامج (5-7)

تدريب (7-3)

استدع أحد دوال الدخل والخرج فى لغة سى++ مثل **puts** من ملف المكتبة msvcrt.dll واكتب استخداما لها فى برنامج.

(ملاحظة: تستخدم الدالة puts لطباعة الحرفى المستخدم كبارامتر لها مثل puts(myString).)

استخدام الصفات مع المنشآت لمحاكاة المنشأ المشترك

يمكنك باستخدام الصفات أن تتحكم فى وضع المنشأ فى خلايا الذاكرة. وعلى سبيل المثال يمكنك باستخدام منشأ سى# أن تحاكى المنشأ المشترك (Union) فى لغة سى++ كما فى هذا المثال:

```
[StructLayout(LayoutKind.Explicit)]
public struct UnionStruct
{
   [FieldOffset(0)]    // الإزاحة رقم #٠
   public int i;
```

```
    [FieldOffset(0)]   // الإزاحة رقم #٠
    public double d;
}
```

فى هذا الإعلان استخدمنا صفتين هما:

StructLayout(LayoutKind.Explicit)
FieldOffset(0)

وبذلك أمكننا أن نضع فى نفس خلية الذاكرة (ذات الإزاحة صفر) عددين أحدهما من النمط الصحيح (int) والآخر من النمط الحقيقى مضاعف الدقة (double). وبالطبع فإن حجم كل عدد يختلف عن الآخر ولكنهما يبدآن من نفس العنوان FieldOffset(0). وفى إمكانك بالطبع أن تستخدم عناوين مختلفة مثل FieldOffset(2) أو FieldOffset(5) لاختزان الأعداد. ومن الجائز استخدام نفس العنوان لاختزان أى عدد من المتغيرات المختلفة (من أنماط القيمة) طالما أنك لن تستخدمها فى نفس الوقت. وهذه هى فكرة المنشأ المشترك.

مثال (7-6)

فى المثال التالى نضيف أنماط اللبنة والبايت إلى المنشأ المشترك الذى قدمناه فى الفقرة السابقة.

```
// Example 7-6.cs
// Union simulation

using System;
using System.Runtime.InteropServices;

[StructLayout(LayoutKind.Explicit)]
public struct UnionStruct
```

```
{
  [FieldOffset(0)]
  public int i;
  [FieldOffset(0)]
  public double d;
  [FieldOffset(0)]
  public char c;
  [FieldOffset(0)]
  public byte b;
}

class MyClass
{
  static void Main()
  {
    UnionStruct u = new UnionStruct();
    // اختزان عدد صحيح
    u.i = 13;
    Console.WriteLine("Integer = {0}", u.i);
    // اختزان عدد حقيقى
    u.d = 12.34;
    Console.WriteLine("Double = {0}", u.d);
    // اختزان لبنة
    u.c = (char)65;
    Console.WriteLine("Character = {0}", u.c);
    // اختزان بايت
    u.b = 127;
    Console.WriteLine("Byte = {0}", u.b);
  }
}
```

تنفيذ البرنامج:

```
Integer = 13
Double = 12.34
Character = A
Byte = 127
```

تدريب (7-4)

مـــن المعـــروف أن العـــدد الطويل (long) يختزن فى حيز قدره ثمانية بايـــت. كمـــا أن العدد الصحيح يختزن فى أربعة بايت. معنى ذلك أنك تستطيع أن تختزن عددين من النوع الصحيح فى نفس حيز الذاكرة الذى تختزن به عدداً طويلاً. اكتب البرنامج الذى يمثل هذه الحالة مع طباعة العدد الطويل بالكود السداسى عشر لتوضيح محتويات كل بايت. وعلى ســـبيل المثال ، لو كان العدد الصحيح الأول هو 5 وكان الثانى 7 ، فإن العدد الطويل يصبح:

700000005

أى أن العدديـــن الصحيحين يحتلان البايت الأولى والخامسة. (لاحظ أن البابت الواحدة تمثّل برقمين متتابعين.)