

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

ملاحظة إلى مبرمجي لغة سي++:

يختلف استخدام ومفهوم كلمة struct بين اللغتين اختلافاً جنياً. ففي لغة سي++ يجوز تعريف الفصيلة بكلمة struct ، وليس هذا هو الحال في سي#.

ويستخدم المنشأ أساساً في اختزان الأنماط البسيطة التي تماثل في سلوكها الأنماط المبنية في اللغة مثل منشأ النقطة التي تحتوي على حقلين عدديين ، أو منشأ اللون ، أو المضلع. وبالرغم من أننا استخدمنا هدف النقطة في ضرب الأمثلة للفصائل ، ولكن المنشأ هو أنسب الأنماط لتمثيل النقطة حيث أنه لا يستهلك إلا حيزاً صغيراً في الذاكرة. ولو أننا أنشأنا مصفوفة كبيرة من النقط الممثلة في 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) ، سيلي الحديث عنها.
 - modifiers: مجموعة مناسبة من معذلات التوصل (Access new (modifiers أو الكلمة
 - identifier: اسم المنشأ
 - interfaces: وصلة بينية أو أكثر يتم تطبيقها بواسطة المنشأ
 - body: الجسم الذي 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 الفصيحة فإن المنشأ لا يجوز شحن حقوله مبدئياً ، أى أن العبارات التالية سوف تؤدي إلى رسالة خطأ من المترجم:

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

❖ يجوز أن يستخدم المنشأ أسلوب بناء أو أكثر ، ولكن لايجوز تعريف 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.

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

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

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

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

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

وتجب ملاحظة أنه بالرغم من أن المنشأ لا يدخل فى شجرة الوراثة ، ولكنه – بطريقة ضمنية – يرث الفصيلة الأساسية التى تمثل جنر شجرة الفصائل جميعاً وهى فصيلة الهدف (object).

مثال (7-1)

في هذا المثال نستخدم المنشأ في بناء الألوان الأساسية الثلاثة: الأحمر R ، والأزرق B ، الأخضر G . والألوان الثلاثة عبارة عن أعداد صحيحة. كما أننا قد 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 في 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)

فى هذا المثال نستخدم مجموعة من الخواص للتوصل إلى أعضاء المنشأ 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 للتعريف
        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 البديهي أنه يمكنك استخدام أسلوب بناء ذي ثلاثة بارامترات كبديل لهذه الطريقة.

تدريب (7-1)

من البديهي أنه يمكنك أن تضع كل محتويات البرنامج بداخل وعاء المنشأ ، تماماً كما هو الحال مع الفصائل. وفي هذه الحال فإن درجات التوصل إلى الأعضاء سوف تختلف عنها عندما يكون هناك وعاء للأعضاء ووعاء للأسلوب الرئيسي. جرب أن تعيد صياغة المثال السابق بحيث يقع بالكامل بداخل المنشأ. جرب أيضاً أن تمزج المثالين السابقين معاً بحيث يحتوى المنشأ على كل 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 تغييرات ، أما تمريرها بالقيمة فإنه يؤدي إلى تعديل النسخة الموجودة في الأسلوب أما المتغيرات الأصلية فتبقى على حالها. ولأن المنشأ من أنماط القيمة فإن أهدافه تخلق في حيز الذاكرة المسمى بالخرزنة (Stack) وعند تمريرها إلى الأساليب فإنها تمرر بالقيمة. أما الفصيلة فإن أهدافها تخلق في حيز الذاكرة المسمى بالكوم (Heap) وعند تمريرها إلى الأساليب فإنها تمرر كمرجع. والمثال التالي يوضح هذا المفهوم.

مثال (7-3)

في هذا المثال نعلن عن فصيلة MyClass و منشأ MyStruct. ويحتوى كل منهما على حقل عددي صحيح. ثم نبني هدفين ، الأول من الفصيلة MyClass والثاني من المنشأ MyStruct ، ونشحن كل حقل بالقيمة 555. ثم نمرر الهدفين إلى الأساليب MyMethod1 و MyMethod2 لتغيير قيمة الحقول 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)

عدّل 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)

القائمة نمط من أنماط القيمة يستخدم لتخزين مجموعة محددة من
البيانات 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 يلي الحديث عنها).
- modifiers: المعدّلات المسموح بها ، وهى تتكون من واحد أو أكثر من معدّلات 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 الإِسبوع:

```
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

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

ومن 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 هذا التخصيص يجعل العناصر التالية للعنصر Wed تحمل القيم 9, 10, 11, 12. أما العنصر الأول (Mon) فهو 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.

وعند التعامل 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 (7-4)

فى هذا المثال نعلن عن ثلاثة قوائم تختص بأيام الأسبوع ، وفصول السنة ، وتقديرات النجاح. ونلاحظ أننا قد استخدمنا الرخص المختلفة

التي 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 هي كلمات إضافية يمكن أن نعدل 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 الكلمات الاختيارية.

وهناك صفات معرفة في مكتبة دوت.نت ، كما أن هناك صفات مبتكرة تستطيع أن تبتكرها بنفسك وتضيفها إلى الإعلانات. وفي 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 التي لا تستخدم دوت.نت (أو الكود المحتوى على مؤشرات). ويمكنك أن تستدعي أحد دوال الكود غير المحكوم باستخدام الصفات. ولنفرض أنك ترغب في استدعاء الدالة النوافذية **MessageBoxA()** (وهي دالة صندوق الرسالة) الموجودة في ملف المكتبة:

C:\WINDOWS\system32\user32.dll

لكي تحقق ذلك اتبع الآتي:

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

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

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

• أضف الصفة DllImport قبل إعلان الدالة كالاتى:

```
[DllImport("user32.dll")]
```

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

إن هذه الصفة تخبر البرنامج أن الدالة المزمع استيرادها توجد فى المكتبة user32.dll. والصفة تمكنك من استيراد أى دالة من المكتبة المذكورة بين علامتى الاقتباس.

• أضف توجيهها لاستخدام الحيز:

System.Runtime.InteropServices

مثال (7-5)

فى هذا 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 التالي:



تنفيذ البرنامج (7-5)

تدريب (7-3)

استدع أحد دوال الدخل والخرج في لغة سي++ مثل `puts` من ملف المكتبة `msvcrt.dll` واكتب استخداما لها في برنامج.

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

استخدام الصفات مع المنشآت لمحاكاة المنشأ المشترك

يمكنك باستخدام الصفات أن تتحكم في وضع المنشأ في خلايا الذاكرة. وعلى سبيل المثال يمكنك باستخدام منشأ سي++ أن تحاكي المنشأ المشترك (Union) في لغة سي++ كما في هذا المثال:

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

```
[FieldOffset(0)] // الإزاحة رقم #0  
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();
    // اختزن عدد صحيح
    u.i = 13;
    Console.WriteLine("Integer = {0}", u.i);
    // اختزن عدد حقيقي
    u.d = 12.34;
    Console.WriteLine("Double = {0}", u.d);
    // اختزن لبنة
    u.c = (char)65;
    Console.WriteLine("Character = {0}", u.c);
    // اختزن بايت
    u.b = 127;
    Console.WriteLine("Byte = {0}", u.b);
  }
}
```

تنفيذ البرنامج:

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

تدريب (4-7)

من المعروف أن العدد الطويل (long) يخترن في حيز قدره ثمانية بايت. كما أن العدد الصحيح يخترن في أربعة بايت. معنى ذلك أنك تستطيع أن تختزن عددين من النوع الصحيح في 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 سبيل المثال ، لو كان العدد الصحيح الأول هو 5 وكان الثانى 7 ، فإن العدد الطويل يصبح:

700000005

أى أن العددين الصحيحين يحتلان البايث الأولى والخامسة. (لاحظ أن البايث الواحدة تمثل برقمين متتابعين.)