الباب الخامس
التعامل مع الفصائل

محتويات الباب

- **الفصائل** (Classes)
- **حيز الاسم** (Namespaces)
- **مستويات التوصل إلى الأعضاء** (Access Levels)
- **الخواص** (Properties)
- **الأعضاء الإستاتيكية** (Static Members)
- **الحقول والمتغيرات الثابتة** const
- **أساليب البناء** (Constructors)
- **الحقول المخصصة للقراءة** (Read-only Fields)
- **الوراثة** (Inheritance)
- **أساليب الهدم** (Destructors)

الفصائل (Classes)

إن الفصـــيلة هـــى أهـــم عناصر لغة سى# حيث أن جميع عناصر الـــبرنامج تقـــع بداخلها. وبخلاف اللغات الأخرى فإن مترجم اللغة لن يسمح لك باستخدام أية متغيرات أو أساليب خارج الفصيلة ، فضلاً عن أن المتغـــيرات العامة (Global Variables) ، كما ذكرنا ، ليس لها وجود فـــى لغـــة سى#. ونذكرك بأن الفصيلة كانت هى بداية استخدام فلسفة البرمجة الموجهة نحو الأهداف فى لغة سى++.

الإعلان عن الفصيلة

يـــتم الإعـــلان عن الفصيلة ، كما ذكرنا من قبل ، باستخدام الكلمة المفتاحية class كالآتى:

```
class MyClass
{
  // تطبيق الفصيلة
}
```

ويأتى تطبيق الفصيلة (محتوياتها) فى نفس الإعلان بين القوسين {}

وكما ذكرنا من قبل فإن الإعلان عن منشأ يتم بنفس الطريقة (مع الفارق أن المنشـــأ مـــن أنمـــاط **القـــيمة** بينما أن الفصيلة من أنماط **المرجع**). والإعـــلان الـــتالى يستخدم الكلمة المفتاحية struct لإعلان منشأ بالاسم MyStruct:

```
class MyStruct
```

```
{
  تطبيق المنشأ //
}
```

وسوف نناقش الفوارق بين المنشأ والفصيلة فى الأبواب القادمة.

شحن الحقول بالقيم الابتدائية (Initialization)

تعتبر الفصيلة هى الصورة المجردة للبيانات ، ويجوز أن نخلق منها أمثلة (Instances) أو أهدافاً (Objects). ففصيلة النقطة مثلاً تمثل موقعاً عاماً (x, y) ، أما النقطة (12, 25) أو النقطة (3, 4) فهى أهداف حية من فصيلة النقطة. أى أن الأهداف أمثلة حية تحتوى على البيانات الحقيقية. وهذا هو مثال لفصيلة نقطة تحتوى على حقلين للإحداثيات x, y:

```
class Point
{
  int x;
  int y;
}
```

كما يجوز شحن حقول الفصيلة بقيم ابتدائية عند الإعلان عنها بالطريقة الآتية:

```
class Point
{
  int x = 0;
  int y = 0;
}
```

وقد تم شحن الحقول العددية هنا بالقيم 0, 0 وهى القيم سابقة التعريف ، ومن الجائز استخدام أية قيم أخرى.

خلق الأهداف من الفصيلة (Instantiation)

عـــندما تخلـــق هدفاً من هذه الفصيلة فإنك تعلن عنه بإحدى الطرق الآتية:

```
Point myPoint;   // خلق الهدف
```

أو

```
Point myPoint = new Point();   // خلق الهدف مع شحن الحقول
```

يـــؤدى الإعـــلان الأول إلى خلق الهدف myPoint بدون شحنه بقيمة ابتدائـــية. والهـــدف غـــير المشـــحون لا يمكن استخدامه (مثل طباعة محتوياته أو التخصيص لحقوله).

أما الإعلان الثانى الذى يستخدم الكلمة المفتاحية new فإنه عند خلق الهـــدف myPoint يستدعى دالة البناء سابقة التعريف لشحن حقوله بالقيم سابقة التعريف مالم يكن قد سبق شحن حقول الفصيلة.

ويجوز خلق أكثر من هدف من فصيلة النقطة بنفس الطريقة:

```
Point p1 = new Point();
Point p2 = new Point();
```

وبهذا يمكنك أن تمنح كل هدف إحداثياته الخاصة مثل:

```
p1.x = 15;
p1.y = 22;
p2.x = 10;
```

```
p2.y = 12;
```

وكما نرى أن الحقول x, y قد ارتبطت هنا بأسماء الأهداف p1, p2.

وفـــى المـــثال التالى نرى صورة كاملة للبرنامج الذى يحتوى على الفصـــيلة والهدف المخلوق منها. وقد جاءت عملية خلق الهدف بداخل الأسلوب Main ، ومع ذلك يجوز أن تتم بداخل أى أسلوب آخر لكنها لا يجوز أن تتم بداخل الفصيلة نفسها.

مثال (1-5)

```
// Example 5-1.cs
// Instantiation

using System;

class Point
{
  int x;
  int y;

  static void Main()
  {
    // خلق هدف من الفصيلة
    Point p1 = new Point();
    // تخصيص قيم عددية لحقول الهدف
    p1.x=22;
    p1.y=25;
    // الطباعة
    Console.Write("x = {0}, y = {1}", p1.x, p1.y);
  }
}
```

تنفيذ البرنامج:

```
x = 22, y = 25
```

حيز الاسم (Namespace)

يمكـــنك أن تضع فصيلة أو أكثر بداخل حيز اسم. وتتأتى فائدة حيز الاسم عندما تستخدم أسماء متشابهة للفصائل. وكما هو الحال فى مكتبة دوت.نـــت فإن الفصائل تتبع حيزات أسماء مختلفة. ومن الجائز أن تجد فصيلة فى حيز ما تحمل نفس الاسم فى حيز آخر. وفى هذه الحال فإن التميـــيز بين الفصائل يتم باستخدام التأهيل الكامل للاسم الذى يبدأ باسم الحيز ، مثلاً:

```
MyNameSpace.Class1
YourNameSpace.Class1
```

ومـــن الجائـــز أيضاً أن تضع حيز اسم بداخل حيز اسم آخر. وفى هذه الحـــال فـــإنك تستطيع تميّز بين الفصائل المتشابهة فى الأسماء فى كل حيز ، مثلا:

```
MyParentNameSpace.MyNestedNamespace.MyClass1
MyParentNameSpace.MyClass1
```

أى أن حـــيز الاســـم هـــو وعـــاء لاحتواء الفصائل وعناصر البرنامج الأخرى وتمييزها عما يماثلها فى حيزات الاسماء الأخرى.

والآتى بعد مثال لفصيلة تقع بداخل حيز اسم (MyNameSpace) وفصيلة أخرى تحمل نفس الاسم تقع بداخل حيز اسم آخر (YourNameSpace). ويتم خلق الأهداف mc1 و mc2 من الفصيلتين وطباعة محتويات حقول كل هدف من داخل الأسلوب Mian بالفصيلة MyClass التابعة لحيز الاسم الأول (MyNameSpace).

مثال (5-2)

```
// Example 5-2.cs
// Namespaces example

namespace MyNameSpace
{
  using System;
  class MyClass
  {
    int field1 = 1;
    int field2 = 2;
    public void MyMethod()
    {
       Console.WriteLine("MyNameSpace.MyClass fileds:");
       Console.WriteLine("Contents of field1 = {0}", field1);
       Console.WriteLine("Contents of field2 = {0}", field2);
    }
    static void Main()
    {
       // Create an object of MyClassNameSpace.MyClass:
       MyClass mc1 = new MyClass();
       mc1.MyMethod();
       // Create an object of YourNameSpace.MyClass:
       // لاحظ استخدام الاسم المؤهل لخلق الهدف
       YourNameSpace.MyClass mc2 =
                        new YourNameSpace.MyClass();
        mc2.MyMethod();
    }
  }
}

namespace YourNameSpace
{
  using System;
  class MyClass
  {
    int field1 = 3;
    int field2 = 4;
    public void MyMethod()
```

```
    {
        Console.WriteLine("YourNameSpace.MyClass fileds:");
        Console.WriteLine("Contents of field1 = {0}", field1);
        Console.WriteLine("Contents of field2 = {0}", field2);
    }
  }
}
```

تنفيذ البرنامج:

```
MyNameSpace.MyClass fileds:
Contents of field1 = 1
Contents of field2 = 2
YourNameSpace.MyClass fileds:
Contents of field1 = 3
Contents of field2 = 4
```

ونلاحظ هنا أن البرنامج لا يستخدم إلاّ أسلوباً رئيسياً واحداً (Main). ولأن الأسلوب الرئيسى يقع بداخل حيز الاسم الأول MyNameSpace لذلك فإنه لم تكن هناك حاجة إلى تأهيل اسم الفصيلة تأهيلاً كاملاً عند إنشاء الهدف mc1. ولكن الحال لم يكن كذلك عند إنشاء الهدف mc2 من الفصيلة التى تقع بداخل حيز الاسم الثانى YourNameSpace

مستويات التوصل إلى الأعضاء (Access Levels)

من الشائع أن يقوم مبرمج ما بتصميم الفصيلة بينما يقوم باستخدامها أشخاص آخرون ، يمكننا أن نطلق عليهم اسم العملاء. ومن أهم خصائص كبسولة الفصيلة إمكانية حماية أعضائها بوضع قيود على التوصل إليها مباشرة بواسطة العملاء. وفى العادة يمكن للمبرمج أن يتوصل إلى الحقول فقط عن طريق أساليب الفصيلة.

ويـــتم تحديـــد مستوى التوصل للأساليب والحقول (والأنماط أيضاً) باستخدام الكلمات المفتاحية الأربعة الآتية:

- عامة: public
- خاصة: private
- محمية: protected
- داخلية: internal

ملاحظة:

يطلـــق على هذه المجموعة من الكلمات المفتاحية اسم "المُعدلات" حيث أنها تستخدم فى تعديل إعلان العضو أو النمط بإضافتها إلى الإعلان مثل: public int MyMethod() وتسمّى هذه المجموعة باسم معدّلات التوصل ، حيث أنها فئة جزئية من المعدلات فى لغة سى#.

وهناك خمسة درجات للحماية (أو مستويات للتوصل) موضحة بالجدول التالى.

جدول (1-4) درجات الحماية المختلفة

درجة الحماية	المعنى
public	تستخدم هذه الدرجة مع الأنماط والأعضاء. ولاقـــيود علـــى التوصل إلى العضو الذى يحمل درجة الحماية public.
private	تســـتخدم هـــذه الدرجة مع الأعضاء. ولا يمكـــن التوصـــل للأعضـــاء إلاّ بواسطة الأساليب الأعضاء فى الفصيلة.

protected	تستخدم هذه الدرجة مع الأعضاء. ويمكن التوصل للأعضاء التى تحمل هذه الدرجة بواسطة الأساليب الأعضاء فى الفصيلة أو فى فصيلة مشتقة منها.
internal	تستخدم هذه الدرجة مع الأنماط والأعضاء. ويمكن التوصل للأعضاء التى تحمل هذه الدرجة بواسطة الأساليب الأعضاء فى المجمع (Assembly).
protected internal	تستخدم هذه الدرجة مع الأعضاء. ويقتصر التوصل فى هذه الدرجة على الأساليب الأعضاء فى الفصيلة أو فى فصيلة مشتقة منها ، علاوة على أعضاء المجمع.

وفيما يلى نوضح درجات الحماية سابقة التعريف ودرجات الحماية المسموح باستخدامها مع أعضاء الأنماط المختلفة التى سنتعرف بها تباعاً فى فصول الكتاب.

جدول (2-4) درجات الحماية المسموح باستخدامها مع الأعضاء

تبعية العضو	درجة الحماية سابقة التعريف	درجات الحماية التى يمكن استخدامها
منشأ متعدد: enum	public	لا يكن
فصيلة: class	private	public protected internal private protected internal

وصلة بينية: interface	public	لا يكن
منشأ: struct	private	public internal private

- لاتوجد أى درجة حماية لحيز الاسم (Namespace).
- بالنسبة للفصائل التى لا توجد بداخل فصائل أخرى فإن درجة الحماية سابقة التعريف لها هى internal ومن الجائز أن تكون إما public أو internal.

وفى المثال التالى نستخدم فصيلة النقطة (Point) لعرض طريقة استخدام درجات الحماية المختلفة.

مثال (3-5)

```
// Example 5-3.cs
// Access Levels

using System;

class OuterClass
{
  // private لايسمح بدرجة الحماية الخاصة
  // إلاّ للفصائل التى تقع بداخل فصائل أخرى:
  private class Point
  {
    // public لاحظ أنه بدون المعدل العام
    // لايمكنك التوصل إلى الحقول الآتية:
```

```
    public int x;
    public int y;
  }

  class MainClass
  {
    static void Main()
    {
      Point p1 = new Point();
      p1.x = 33;
      p1.y = 22;
      Console.Write("x = {0}, y = {1}", p1.x, p1.y);
    }
  }
}
```

تنفيذ البرنامج:

```
x = 33, y = 22
```

ملاحظات على البرنامج السابق:

- يوجد فى البرنامج ثلاث فصائل: الفصيلة Point و الفصيلة MainClass ، والفصيلة التى تحتوى كليهما وهى الفصيلة OuterClass. وقد وضعنا الأسلوب الرئيسى فى فصيلة مستقلة بخلاف فصيلة النقطة حتى يظهر تأثير درجات الحماية. ولذلك فإنه بدون درجة الحماية public لحقول فصيلة النقطة لا يمكن التوصل إلى الحقول x, y. ولتجرب تغيير درجة حماية الحقول إلى private أو protected وشاهد الرسالة التى يرسلها إليك المترجم.

- أمـــا الفصـــيلة الخارجـــية OuterClass فقد استخدمناها كوعاء للفصـــائل الأخـــرى حتى نستطيع أن نمنح فصيلة النقطة درجة الحمايـــة private ، كمـــا هو موضح بالجدول السابق. ولو أنك حذفـــت الفصـــيلة الخارجية فسوف يعترض المترجم على كلمة private. ومـــع ذلك فإن درجة حماية الفصيلة الخارجية لا تؤثر على أداء البرنامج.

تدريب (1-5)

جـــرب تغيـــير درجات الحماية المختلفة للفصائل والأعضاء بالبرنامج السابق ، وشاهد النتائج.

الخواص (Properties)

هناك طريقة سهلة لحماية حقـــول الفـــصيلة وهى استخدام الخواص (Properties). والخـــواص ماهى فى الحقيقة إلاّ أساليب مثل الأساليب الأعضـــاء ، ولكـــنها تســـهل على العميل التوصل إلى حقول الفصيلة الخاصـــة (private) كمـــا لوكان يتعامل مع الحقول مباشرة. وقد كانت الطـــريقة التقلـــيدية المســـتخدمة فى لغة سى++ من قبل هى تخصيص مجموعة من الدوال الأعضاء للتوصل إلى الحقول الخاصة مثل:

```
Set_MyPoint_x();
Get_MyPoint_x();
```

والدالـــة الأولـــى وظيفـــتها هى التخصيص للحقل x والثانية هى قراءة محتويات الحقل x ، وهكذا.

أما طريقة الخواص فهى كالمثال الآتى:

```
private string name;
public string Name
{
  get
  {
    return name;
  }
  set
  {
    name = value;
  }
}
```

يعلن العضو "name" بدرجة الحماية الخاصة (private). وتعلن الخاصية بدرجة الحماية العامة (public) ، وهى هنا تحمل الاسم Name (ولك أن تســـميها ما تشاء). ونرى بداخل الخاصية كلمتين لهما معنى خاص هما set و get.(وهمـــا أسماء لأساليب مميزة). والأسلوب get يستخدم فى قـــراءة العضو الخاص (name) ، ولذلك فهو يحتوى على كلمة return الـــتى تـــرجع البـــيان المختزن فى هذا العضو. أما الأسلوب set فهو يســـتخدم لتغيير محتويات العضو بتخصيص القيمة value إليه. والكلمة value هى أيضا كلمة ذات معنى خاص ولا تستخدم إلاّ فى هذا المجال. هذا من ناحية تصميم الخاصية ، أما من ناحية استخدامها فكل ما عليك أن تستخدم اسمها (Name) كما لوكان اسم الحقل (name) ، وعلى سبيل المثال:

```
Name = "Mohamed Aly";  // للتخصيص للحقل
Console.Write(Name);   // لقراءة محتويات الحقل
```

وقـد يختار مصمم الفصيلة أن تكون الخاصية "للقراءة فقط" ، وفى هذه الحالة فإنها لا تحتوى على الأسلوب set بل يكتفى بالأسلوب get فقط.

يمكـنك مـع الأسلوبين set و get استخدام عرف مختلف فى كتابة الكود لتوفير حيز الكتابة كالآتى:

```
public string Name
{
  get {return name;}
  set {name = value;}
}
```

وفـى المـثال التالى نستخدم خاصيتين فى الفصيلة إحداهما للقراءة والكتابة والأخرى للقراءة فقط.

مثال (5-4)

```
// Example 5-4.cs
// Properties example.

using System;

public class Employee
{
  // إعلان الحقول الخاصة
  private string companyName = "Microsoft";
  private string name;

  // إعلان خاصية الاسم – قراءة وكتابة
  public string Name
  {
```

```
    get {return name;}
    set {name = value;}
  }

  // إعلان خاصية اسم الشركة – قراءة فقط
  public string CompanyName
  {
    get {return companyName;}
  }
}

public class MainClass
{
  public static void Main()
  {
    Employee emp = new Employee();
    emp.Name = "Hazem Abolrous";
    Console.WriteLine("Company Name: {0}",
                      emp.CompanyName);
    Console.WriteLine("Employee name: {0}", emp.Name);
  }
}
```

تنفيذ البرنامج:

```
Company Name: Microsoft
Employee name: Hazem Abolrous
```

الأعضاء الإستاتيكية (Static Members)

هـل لاحظـت فى المثال السابق أن اسم الشركة (companyName) لا يتغـير بحسب الموظف. فمهما كانت بيانات الموظفين فإن اسم الشركة الـتى يتـبعونها لا يتغـير. لذلك فإن اسم الشركة يمكن تمييزه بالكلمة المعدّلة **static** التى تمنحه صفة التبعية للفصيلة بحيث أننا لا نحتاج فى

هــذه الحالة إلى ربطه بهدف معين. بذلك يمكن تعديل إعلانات الحقول إلى:

```
private static string companyName = "Microsoft";
private string name;
```

وعند استدعاء الحقل الإستاتيكى نربطه مباشرة بالفصيلة كالآتى:

```
Console.WriteLine("Company Name: {0}",
                  Employee.CompanyName);
```

إن هذا الفارق يؤدى إلى تقسيم أعضاء الفصيلة إلى نوعين:

- أعضاء أمثلة (Instance members)
- أعضاء إستاتيكية (Static members)

ولا تقتصر الأعضاء الإستاتيكية على الحقول بل على الأساليب أيضاً.

والمثال التالى يحتوى على تعديل للمثال السابق لتحويل اسم الشركة إلى عضو إستاتيكى وكذلك الخاصية المرتبطة بها.

مثال (5-5)

```
// Example 5-5.cs
// Static properties example.

using System;

public class Employee
{
  // إعلان الحقول الخاصة
  private static string companyName = "T-Mobile";
  private string name;
```

```
  // إعلان خاصية الاسم – قراءة وكتابة
  public string Name
  {
    get {return name;}
    set {name = value;}
  }

  // إعلان خاصية اسم الشركة – قراءة فقط
  public static string CompanyName
  {
    get {return companyName;}
  }
}

public class MainClass
{
  public static void Main()
  {
    Employee emp = new Employee();
    emp.Name = "Sally Abolrous";
    Console.WriteLine("Company Name: {0}",
                      Employee.CompanyName);
    Console.WriteLine("Employee name: {0}", emp.Name);
  }
}
```

تنفيذ البرنامج:

```
Company Name: T-Mobile
Employee name: Sally Abolrous
```

ومن أمثلة الأساليب الإستاتيكية التى تعرفنا ببعضها من خلال الأمثلة ، أساليب الفصيلة Math مثل Math.Tan() و Math.Round() وهى جميعاً تستخدم مع الفصيلة مباشرة وبدون إنشاء أية أهداف. ومن الجدير بالذكر أن جميع أساليب الفصيلة Math أساليب إستاتيكية.

الحقول والمتغيرات الثابتة (const)

يستخدم المعدل **const** لإعلان نوعية من المتغيرات المحلية أو الحقول التى تتميز بالثبات بحيث لا يجوز تغييرها بعد إعلانها. وكمثال للأعضاء الثابتة التى مررنا بها فى الأمثلة عضو النسبة التقريبية PI بالفصيلة Math. إن هذا العضو لايمكن تغيير قيمته ، فضلاً عن أننا لن نحتاج إلى ذلك على الإطلاق. وعلى سبيل المثال فإن الفصيلة الآتية تحتوى على حقول ثابتة:

```
class MyClass
{
  public const int field1 = 3, field2 = 10;
}
```

ونلاحظ أنه قد تم شحن الحقول فى نفس الإعلان ، وهذا شرط أساسى فى إعلان الحقول أو المتغيرات الثابتة.

وكذلك الحال بالنسبة للمتغيرات المحلية. فلو أنك ، فى برنامجك ، استخدمت أحد ثوابت التحويل مثل ثوابت تحويل وحدات القياس فإنك من الأفضل أن تضيف المعدل const إلى إعلان المتغير كالأمثلة الآتية:

```
const double myDoubleConst = 3.1;
const int myIntConst = 25;
```

فى هذه الأحوال لا يمكنك تغيير أحد هذه المتغيرات بعبارات مثل:

```
int x = ++myIntConst;  //error
myIntConst = 26;  // error
```

وفى المثال التالى نستخدم الثابت convRatio للتحويل من كيلومتر إلى ميل.

مثال (5-6)

```
// Example 5-6.cs
// const example

using System;

public class TestClass
{
  public static void Main()
  {
    const double convRatio = 0.6211;  // Kilos to miles
    double miles = 10;
    double kilos = miles/convRatio;

    Console.WriteLine("{0} Mile(s) = {1:F2} Km", miles, kilos);
  }
}
```

تنفيذ البرنامج:

```
10 Mile(s) = 16.10 Km
```

أساليب البناء (Constructors)

من الجائز أن تتضمن الفصيلة أسلوباً أو أكثر لبناء الأهداف عند خلقها. وتسمى هذه النوعية من الأساليب الأعضاء بأساليب البناء. وهى تنقسم إلى عدة أنواع:

- أساليب بناء الأمثلة
- أساليب البناء الخاصة
- أساليب البناء الإستاتيكية

أساليب بناء الأمثلة (Instance constructors)

يعتـــبر هذا النوع هو الأكثر شيوعاً فى الاستخدام وقد يكتفى بتسميته بأساليب البناء. ويسمى أسلوب البناء باسم الفصيلة نفسها. ويتم استدعاء أسلوب البناء عند خلق هدف جديد مثل:

```
Point myPoint = new Point();
```

ويجوز أن تحتوى الفصيلة أيضاً على أسلوب بناء ذى بارامترات مثل:

```
Point myPoint = new Point(int x, int y);
```

وكما نلاحظ فى الأمثلة السابقة أنه إذا لم تتضمن الفصيلة أى أسلوب للبـــناء ، فـــإن إعـــلان الهدف يؤدى إلى استدعاء أسلوب البناء سابق التعريف ، وهو أسلوب بناء بدون بارامترات مثل Point(). ويؤدى هذا الأســـلوب إلـــى شـــحن الحقول بالقيم سابقة التعريف (وهى0 فى حالة الحقول العددية الصحيحة x, y).

مثال (5-7)

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

using System;

class Point
{
  public int x, y;

  // Default constructor: أسلوب بناء بدون بارامترات
  public Point()
  {
    x = 0;
```

```
    y = 0;
  }

  // A constructor with arguments: أسلوب بناء ذو بارامترات
  public Point(int x1, int y1)
  {
    x = x1;
    y = y1;
  }
}

class MyClass
{
  static void Main()
  {
    Point p1 = new Point();
    Point p2 = new Point(2,10);

    Console.WriteLine("First Point at: ({0}, {1})", p1.x, p1.y);
    Console.WriteLine("Second Point at: ({0}, {1})", p2.x, p2.y);
  }
}
```

تنفيذ البرنامج:

```
First Point at: (0, 0)
Second Point at: (2, 10)
```

ملاحظات على البرنامج السابق:

لاحظ أنه لايمكنك استخدام أسلوب بناء واحد (ذا بارامترات) فى البرنامج السابق اعتماداً على أن المترجم يمد تلقائياً بالأسلوب سابق التعريف. إن المترجم يوفر الأسلوب سابق التعريف فقط فى حالة عدم وجود أساليب بناء.

استخدام الكلمة المفتاحية this

كما فى لغة سى++ فإن الكلمة **this** تستخدم فى التوصل إلى الحقول من خلال أساليب البناء أو الأساليب الأعضاء عموماً. ففى المثال السابق كان من الممكن إعلان أسلوب البناء بالطريقة الآتية:

```
public Point(int x, int y)
{
  this.x = x;
  this.y = y;
}
```

وفى هذه الحالة يمكنك أن تستخدم نفس الأسماء لكل من البارامترات والحقول (x, y) وذلك لأن التعبير this.x يشير إلى الهدف الحالى ، أما المتغير x فهو يمثل حقل الفصيلة. ولذلك فإنه من البديهى أن الكلمة this لا تستخدم مع الأساليب الإستاتيكية.

بالرغم من أن الكلمة this عبارة عن مؤشر (كما يطلق عليها فى لغة سى++) ولكن لغة سى# تتجنب استخدام هذه الكلمة.

أساليب البناء الخاصة (Private Constructors)

تستخدم أساليب البناء الخاصة مع الفصائل التى تحتوى على أعضاء إستاتيكية فقط. ولا يسمح للفصائل الأخرى (فيما عدا الفصائل التى تقع بداخل الفصيلة نفسها) أن تخلق أمثلة من الفصيلة. أنظر هذا المثال:

```
public class MyClass
{
  private MyClass() {}  // أسلوب بناء خاص
  public static int companyName;
```

```
    public static int employmentDate;
}
```

وأسلوب البناء الخاص عبارة عن أسلوب فارغ (بلا محتويات) ، وظيفته الأساسية منع توليد أسلوب البناء سابق التعريف للفصيلة. كما نلاحظ أن الأسلوب يأخذ المعدّل private ؛ وهذا مجرد عرف لأن درجة الحماية سابقة التعريف هى private.

تدريب (5-2)

استخدم شريحة الكود السابقة فى برنامج وحاول استخدام العبارة التالية فى فصيلة أخرى وشاهد نتيجة الترجمة:

```
MyClass myObject = new MyClass();
```

أساليب البناء الإستاتيكية (Static Constructors)

يستخدم أسلوب البناء الإستاتيكى لشحن الفصيلة. وهو يستدعى قبل خلق أى هدف من الفصيلة وقبل استدعاء أى من أعضائها الإستاتيكية.

وفى المثال التالى نرى أسلوب البناء الإستاتيكى ()MyClass الذى يحتوى على عبارتين للطباعة ، كما نرى الأسلوب ()MyMethod الذى يحتوى على عبارة أخرى للطباعة. وعند استدعاء الأسلوب ()MyMethod نلاحظ أن أسلوب البناء الإستاتيكى قد تم استدعائه تلقائياً.

مثال(5-8)

```
// Example 5-8.cs
// Static constructor example
```

```
using System;

class MyClass
{
  // Static constructor: أسلوب البناء الإستاتيكى
  static MyClass()
  {
    Console.WriteLine("Hey, I am the static constructor! " +
                      "I am called automatically!");
  }

  public void MyMethod()
  {
    Console.WriteLine("Hi, I am MyMethod. I was called after " +
                      "the static constructor had been invoked!");
  }
}

class MainClass
{
  static void Main()
  {
    MyClass myObject = new MyClass();
    myObject.MyMethod();
  }
}
```

تنفيذ البرنامج:

```
Hey, I am the static constructor! I am called
automatically!
Hi, I am MyMethod. I was called after the static
constructor had been invoked!
```

الحقول المخصصة للقراءة (Read-only Fields)

تستخدم الكلمة المفتاحية **readonly** في تعديل إعلانات حقول الفصائل حيث تجعل الحقل مخصصاً للقراءة. والحقل المخصص للقراءة لا يجوز التخصيص له إلاّ أثناء الإعلان عنه أو باستخدام أسلوب بناء.

والإعلان التالى يحتوى على حقل مخصص للقراءة:

```
public readonly int readOnlyInt1 = 55;
```

كما أن أسلوب البناء التالى يخصص قيمة عددية لنفس الحقل:

```
public MyClass()
{
  readOnlyInt1 = 66;
}
```

والبرنامج التالى يحتوى على مثال كامل للحقول المخصصة للقراءة.

مثال (5-9)

```
// Example 5-9.cs
// readonly example

using System;

class MyClass
{
  public int myRegularInt;
  public readonly int readOnlyInt1 = 55; // Ok :إعلان وتخصيص
  public readonly int readOnlyInt2;

  public MyClass()
  {
    readOnlyInt2 = 66;  //   :بناء
  }

  public MyClass(int l, int m, int n)
  {
    myRegularInt = l;
```

```
    readOnlyInt1 = m;    // بناء:
    readOnlyInt2 = n;    // بناء:
  }
}
class MainClass
{
  static void Main()
  {
    MyClass obj1= new MyClass(11, 22, 33);   // OK :أسلوب بناء
    Console.WriteLine("obj1 fields are: {0}, {1}, {2}" ,
    obj1.myRegularInt, obj1.readOnlyInt1, obj1.readOnlyInt2);

    MyClass obj2 = new MyClass();
    obj2.myRegularInt = 44;   // OK :تخصيص لحقل عادى
    Console.WriteLine("obj2 fields are: {0}, {1}, {2}" ,
    obj2.myRegularInt, obj2.readOnlyInt1, obj2.readOnlyInt2);
  }
}
```

تنفيذ البرنامج:

```
obj1 fields are: 11, 22, 33
obj2 fields are: 44, 55, 66
```

ملاحظات على البرنامج السابق:

لاحـــظ أن عمليات التخصيص الجائزة قد تمت إما أثناء الإعلان عن المتغـــيرات المخصصة للقراء أو باستخدام أساليب البناء. وعلى سبيل المثال لو أنك استخدمت أياً من العبارات التالية بداخل الأسلوب الرئيسى فإن المترجم يعترض عليها:

```
obj2.readOnlyInt1 = 55;   // Error : تخصيص غير جائز
obj2.readOnlyInt2 = 66;   // Error: تخصيص غير جائز
```

ملاحظة:

إن الفارق الرئيسى بين الحقول المخصصة للقراءة (readonly) والحقول الثابتة (const) أن الأولى يمكن تغيير محتوياتها بالطرق المشروعة أما الثانية فلا يجوز تغيير قيمتها بعد إعلانها.

الوراثة (Inheritance)

يمكن للفصيلة فى لغة سى# أن ترث من فصيلة أخرى كالمثال الآتى:

```
class MyDerivedClass: MyBaseClass
{
   // تطبيق الفصيلة
}
```

فى هذا المثال نعلن عن الفصيلة MyDerivedClass وفى نفس الوقت نعلن عن اشتقاقها من الفصيلة MyBaseClass. وكما نرى فى الإعلان أن الوراثة تتم باستخدام الرمز ":" قبل اسم الفصيلة الموروثة. والوراثة تعنى أن الفصيلة المشتقة (الوارثة) تحتوى على جميع أعضاء فصيلة الأساس (الموروثة). فعلى سبيل المثال فإن فصيلة الموظف ترث من فصيلة الإنسان وتحتوى على جميع خصائصه بالإضافة إلى الخصائص المتفردة للموظف.

وفى لغة سى# ، بخلاف اللغات الأخرى ، لا يجوز وراثة أكثر من

فصيلة واحدة. بينما يمكن **تطبيق** أكثر من وصلة بينية (وتطبيق الوصلة البينـــية هو المناظر للوراثة فى الفصائل). وعند تطبيق وصلة بينية فإن الفصـــيلة ترث أعضاء الوصلة البينية ــ وسوف يلى الحديث عن ذلك عند مناقشة الوصلات البينية.

تستخدم أيضاً مصطلحات التخصص (Specialization) و التعميم (Generalization) للتعبـــير عـــن علاقات الوراثة. فعلى سبيل المثال يمكننا القـــول بـــأن فصـــيلة الـــبقرة **تخصص** من فصيلة الثدييات ، كما أن فصيلة الثدييات **تعميم** لفصيلة البقرة.

مثال (5-10)

```
// Example 5-10.cs
// Inheritance example

using System;

  class Citizen
  {
    string idNumber = "111-2345-H";
    string name = "Hani M. Ashoor";

    public void GetPersonalInfo()
    {
      Console.WriteLine("Name: {0}", name);
      Console.WriteLine("ID Card Number: {0}", idNumber);
    }
  }
  class Employee: Citizen
  {
    string companyName = "Technology Group Inc.";
    string companyID = "ENG-RES-101-C";
```

```
    public void GetInfo()
    {
      // Calling the base class GetPersonalInfo method:
      Console.WriteLine("Citizen's Information:");

      GetPersonalInfo();

      Console.WriteLine("\nJob Information:");
      Console.WriteLine("Company Name: {0}", companyName);
      Console.WriteLine("Company ID: {0}", companyID);
    }
  }

class MainClass {
  public static void Main()
  {
    Employee E = new Employee();
    E.GetInfo();
  }
}
```

تنفيذ البرنامج:

```
Citizen's Information:
Name: Hani M. Ashoor
ID Card Number: 111-2345-H

Job Information:
Company Name: Technology Group Inc.
Company ID: ENG-RES-101-C
```

ملاحظات على البرنامج السابق:

نلاحظ فى البرنامج السابق أن درجة الحماية للدوال الأعضاء هى درجة الحماية العامة public. وهذا ضرورى حتى يمكن التوصل إلى أعضاء الفصائل.

ملاحظة:

يستخدم المُعدّل **sealed** لمنع الوراثة من الفصيلة. وبالطبع فإنك لا تستطيع أن تستخدم الكلمة **abstract** مع الكلمة sealed فى نفس الإعلان. (سيلى شرح استخدام المُعدّل abstract فى الفصل السادس.)

أساليب الهدم (Destructors)

يتم إعلان أسلوب الهدم لفصيلة مثل MyClass بالصورة الآتية:

```
~MyClass()
{
  عبارات الهدم //
}
```

أى أن اسلوب الهدم يستخدم اسم نفس الفصيلة ولكنه يضيف إليه البادئة "~". ولا يمكنك استدعاء أساليب الهدم حيث أنها تستدعى تلقائياً بعد انتهاء عمل الأهداف. وفى حالة الوراثة فإنه يتم هدم الأهداف بدءاً من الأحفاد ثم الأبناء ثم الآباء وهكذا. كما تستدعى دوال الهدم ــ بالطبع ــ عندما ينتهى البرنامج من عمله.

والبرنامج التالى يوضح التسلسل فى استدعاء أساليب الهدم لمجموعة من الأهداف التى تتسلسل فى شجرة الوراثة.

مثال (11-5)

```
// Example 5-11.cs
// Destructor example

using System;
```

```
class Parent   // الآباء
{
  ~Parent()
  {
    Console.WriteLine("Calling the Parent destructor.");
  }
}

class Kid: Parent   // الأبناء
{
  ~Kid()
  {
    Console.WriteLine("Calling the Kid destructor.");
  }
}

class GrandKid: Kid   // الأحفاد
{
  ~GrandKid()
  {
    Console.WriteLine("Calling the Grandkid destructor.");
  }
}

public class MyClass
{
  public static void Main()
  {
    GrandKid myObject = new GrandKid();
  }
}
```

تنفيذ البرنامج:

```
Calling the Grandkid destructor.
Calling the Kid destructor.
Calling the Parent destructor.
```

ملاحظات على البرنامج السابق:

نلاحظ فى نتيجة البرنامج أن الهدف Grandkid ، وهو الهدف الحفيد ، قد تم هدمه أولاً ، ثم تلاه Kid و Parent.

بالرغم من أن أساليب الهدم لها دور رئيسى فى لغة سى++ حيث تستطيع أن تضمنها العبارات اللازمة لمسح الذاكرة التى سبق حجزها للمؤشرات. ولكنك فى لغة سى# لن تحتاج إلى ذلك لأن جامع القمامة يتولى عنك تنظيف الذاكرة تلقائياً بعد انتهاء عمل الأهداف المختلفة وخروجها من الصورة.

ومع ذلك فإنك تستطيع استخدام أسلوب الهدم لتنظيف الذاكرة من الموارد غير المحكومة مثل الملفات ووصلات الشبكات الإليكترونية وهذا يخرج عن مستوى هذا الكتاب ويمكنك الإطلاع على تفصيلات الموضوع بقراءة تفصيلات أسلوب دوت.نت **Dispose()** إما فى شاشة النجدة أو على موقع الوب:

http://msdn.microsoft.com/

هناك طريقة لإرغام جامع القمامة على بدء العمل باستخدام أسلوب دوت.نت **GC.Collect()** ولكن هذا لا يوصى به إذ أنه قد يؤدى إلى نتائج غير مرغوب فيها.