الباب السادس

الـــدوال الأعـــضاء

محتويات الباب

- الدوال الأعضاء (Function Members)
- تعدد الأشكال (Polymorphism)
- التحميل الزائد للأساليب (Overloading)
- الفصائل والأساليب المجردة (Abstract classes and methods)
- إمرار البارمترات إلى الأساليب
- الطرق المختلفة لتمرير البارامترات
- الفهارس (Indexers)
- المؤثرات المبتكرة (User-defined operators)

الدوال الأعضاء (Function Members)

لعل بعضنا يتوقع أننا سوف ننتقل إلى الحديث عن لغة سى++ تحت عـــنوان الدوال الأعضاء ، ولا سيما بعد أن تعودنا على كلمة الأساليب فى لغة سى# ، وعلمنا أنها هى المناظر لكلمة الدوال فى سى++.

والحقيقة أن سى# تستخدم كلمة الدوال الأعضاء كاسم عام لأعضاء الفصيلة ، باستثناء الحقول ، والتى تشمل الآتى:

- الأساليب (Methods)
- الخواص (Properties)
- الأحداث (Events)
- الفهارس (Indexers)
- المؤثرات المبتكرة (User-defined operators)
- أساليب البناء (Constructors)
- أساليب الهدم (Destructors)

وقـــد تعرفنا ببعض هذه الأعضاء ، وسوف يلى الحديث عن الأعضاء الأخرى فى فصول الكتاب.

تعدد الأشكال (Polymorphism)

إن أحد مميزات الوراثة هى إعادة استخدام الكود ، فلو أنك كتبت فصيلة المواطن مثلاً ، فلا تحتاج إلى إنشائها مرة أخرى عندما تريد أن تنشىء

فصـــيلة الموظـــف أو الشـــرطى أو الفلاح. إن خصائص هذه الفصائل موجـــودة فـــى فصيلة المواطن وما عليك إلاّ أن ترث فصيلة المواطن وتضـــيف إليها خصائص الموظف أو الشرطى أو الفلاح فتحصل على الفصـــائل الجديـــدة بأقل مجهود. فالوراثة إذن توفر الجهد المبذول فى اختراع ما سبق اخترعه ، أو كما يقولون "قى إعادة اختراع العجلة".

أمـــا المـــيزة الثانية للوراثة فهى تعدد الأشكال. وتعدد الأشكال ظاهرة موجـــودة فـــى حياتنا اليومية. فأنت عندما تطلب رقم تليفون معين فإن العملية تنتهى بأن يضرب جرس التليفون لدى الطرف الآخر ، بصرف الـــنظر عن نوع التليفون أو نوع الجرس. فقد يكون التليفون موبايل (Mobile) وقـــد يكون تليفوناً تقليدياً من الطرازات القديمة التى كادت أن تـــنقرض. وقـــد يكون جرس التليفون تقليدياً ، وقد يكون نغمة موسيقية جمـــيلة يمكـــن ضبطها حسب الطلب. إن هذه الإختلافات لا تؤثر على النتيجة فشركة التليفونات مسئولة عن التعامل مع جميع هذه الأمثلة من ماكينات التليفون لتحقيق نتيجة واحدة وهى طلب المكالمة.

وفى مجال البرمجة فإننا نلتقى ــ على سبيل المثال ــ بالعديد من أنواع الـــنوافذ فى بيئة نوافذ ميكروسوفت. فهناك نافذة الرسالة التى تنقل إليك تعلـــيمة ما أو تحذيراً ما ، وهناك نافذة للرسم وأخرى للكتابة. ولو أنك كتبت أسلوباً لرسم النافذة بالاسم DrawWindow مثلاً ، فإنه يمكن ربطه بـــأهداف مختلفة تشكل أنواعاً مختلفة من النوافذ. وبحسب الهدف الذى يرتبط به الأسلوب يحقق نتائج مختلفة كالأمثلة الآتية:

```
myButton.DrawWindow();  // يؤدى إلى رسم زر
myMsg.DrawWindow();  // يؤدى إلى رسم نافذة رسالة
myPaintSaurface.DrawWindow();  // يؤدى إلى رسم نافذة للرسم
```

إن هذه الأهداف لا تتبع بالطبع فصيلة واحدة ولكنها تتبع فصائل مشتقة من نفس الفصيلة التى عرفنا بها الأسلوب DrawWindow.

الأساليب الافتراضية والأساليب الراكبة virtual, override

تتحقق عملية تعدد الأشكال السابقة بإعلان الأسلوب DrawWindow أسلوباً إفتراضياً باستخدام الكلمة المفتاحية **virtual** ، وذلك فى فصيلة الأساس ، أى:

```
public virtual void DrawWindow ()
{
  تعريف الأسلوب بفصيلة الأساس //
}
```

أما بالنسبة للفصائل التى ترث فصيلة الأساس فإنك تعلن بكل فصيلة أسلوباً بنفس الاسم (DrawWindow) ولكنك تستخدم معه الكلمة المفتاحية **override** ، مثل:

```
public override void DrawWindow()
{
  تعريف الأسلوب بأحد الفصائل المشتقة //
}
```

إن الأساليب التى تستخدم الكلمة المعدلة override يطلق عليها اسم الأساليب الراكبة ، لأنها تركب الأسلوب الافتراضى الذى يحمل نفس الاسم والتوقيع (نوعية وعدد البارامترات) ودرجة الحماية بالفصيلة المشتقة. وعند تعريف أسلوب راكب فإنه من حقك أن تغير فى وظيفته كما تشاء.

ويجوز للأسلوب الراكب أن يركب أياً من الأساليب الآتية:

- الأساليب الافتراضية (virtual)
- الأساليب المجردة (abstract)
- الأساليب الراكبة (override)

وسوف يلى شرح الأساليب المجردة فى الفقرة التالية.

ملاحظات:

لا يمكنك استخدام المعدّل virtual مع المعدلاّت الآتية:

static
abstract
override

كما لا يمكنك الجمع بين المعدل override والمعدلات الآتية:

static
abstract
virtual
new

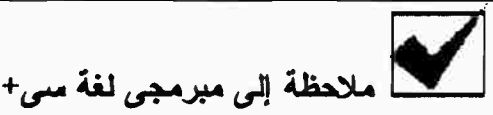
(سوف يلى الحديث عن المعدل abstract و new).

استدعاء أعضاء الفصيلة الموروثة base

فـــى أحـــوال كثيرة من أحوال الوراثة قد نحتاج إلى بناء هدف فصيلة الأســـاس أثناء بناء هدف الفصيلة المشتقة. إن هذا يتم باستخدام الكلمة المفتاحية **base** كالمثال الآتى:

```
public MyDerived(int x) : base(x)
{
  // ...
}
```

إن هذا الأسلوب هو أسلوب بناء الفصيلة MyDerived ، وقد استدعى من داخله أسلوب بناء فصيلة الأساس بهذه الطريقة.



ملاحظة إلى مبرمجى لغة سى++:

لاحـــظ أن هـــذه الطريقة فى بناء فصيلة الأســـاس تكـــافئ قـــائمة الشحن (Initialization list) المستخدمة فى لغة سى++.

وفـــى أحـــيان أخـــرى قد نرغب فى استدعاء أسلوب ما من فصيلة الأســـاس من داخل إحدى الفصائل المشتقة. فإذا كان الأسلوب قد ركبه أسلوب آخرمن الفصيلة المشتقة ، فإن استخدام الكلمة base هو الطريق إلى ذلك:

```
base.GetInformation();
```

إن هذه العبارة تستدعى الأسلوب الأصلى GetInformation بعد أن تم ركوبه بأسلوب يحمل نفس الاسم من أساليب الفصيلة المشتقة.

ملاحظة:

لا يجوز استخدام الكلمة base بداخل أسلوب إستاتيكى.

ركوب الأساليب الافتراضية بفصيلة الأساس

سوف نضرب هنا مثالاً بمساحات الأشكال ، وهو مثال سهل الإنشاء فـــى حدود ما تعلمناه حتى الآن من مفردات اللغة. إن فصيلة المساحات العامـــة تحـــتوى علـــى البعديـــن x, y وعلى أسلوب افتراضى لحساب المساحة:

```
public virtual double Area()
{
  return 0;
}
```

(لاحـــظ أن القـــيمة المرتجعة من هذا الأسلوب "0" لا قيمة لها لأن الأساليب الأخرى سوف تركبه وتغير معناه على أى حال).

ويـــرث فصيلة المساحات العامة بعض الفصائل الأخرى مثل فصيلة النقطة والدائرة والكرة والاسطوانة. والعامل المشترك بين هذه الفصائل وبين فصيلة المساحات العامة هى الأبعاد x, y. فالبعد x قد يمثل ضلع مـــربع أو نصف قطر دائرة أو كرة. والبعد y قد يكون طول المربع أو ارتفاع الاسطوانة. ووراثة هذه الأبعاد بصفة عامة يسهل بناء الفصائل المشتقة. وتحتوى كل فصيلة على أسلوب لحساب المساحة بنفس الاسم Area(). وهذا هو أسلوب مساحة الدائرة:

```
public override double Area()
{
  return Math.PI * x * x;  //  مساحة الدائرة ط نق٢
}
```

أما هذا فهو أسلوب مساحة الكرة:

```
public override double Area()
{
  return 4 * Math.PI * x * x;  //   مساحة الكرة = ٤ ط نق٢
}
```

وهذا هو أسلوب مساحة الاسطوانة:

```
public override double Area()
{
  return 2*Math.PI*x*x + 2*Math.PI*x*y;
                  //   مساحة الاسطوانة = ٢ ط نق٢ + ٢ ط نق • الارتفاع
}
```

ونلاحظ فى هذه الأساليب جميعاً أنها تركب أسلوب المساحة الافتراضى بفصيلة الأساس. أما مساحة النقطة فهى صفر وهى لا تحتاج إلى تعريف خاص لأنها تستخدم أسلوب البناء سابق التعريف وهو يخصص القيمة صفر لجميع الحقول.

بقيت نقطة هامة حتى يكتمل بناء المثال الذى نحن بصدده. إن علينا أن نبنى فصيلة الأساس عند بناء الفصائل المشتقة منها ، كالمثال الآتى:

```
public Cylinder(double r, double h): base(r, h)
{
}
```

فى هذا الأسلوب نستخدم الكلمة base فى استدعاء أسلوب بناء فصيلة الأساس قبل بناء الهدف من فصيلة الاسطوانة. والغرض من ذلك هو تمرير البارامترات r, h إلى الحقول المناظرة بفصيلة الأساس. وفيما يلى المثال الكامل لفصيلة المساحات والفصائل المشتقة منها.

مثال (6-1)

```
// Example 6-1.cs
```

```
// virtual Example
using System;

public class AreasClass  // فصيلة الأساس للمساحات العامة
{
  // Fields:  الحقول
  protected double x, y;
  // Constructors:  دوال البناء
  public  AreasClass()
  {
  }
  public AreasClass(double x, double y)
  {
    this.x = x;
    this.y = y;
  }
  // Methods:  أسلوب حساب وإرجاع المساحة
  public virtual double Area()
  {
    return 0;
  }
}
// فصيلة النقطة ــ تستخدم أسلوب البناء بدون بارامترات
class Point: AreasClass
{
  public Point(): base()
  {
  }
}
// فصيلة الدائرة ــ تستخدم نصف القطر فقط
class Circle: AreasClass
{
  public Circle(double r): base(r, 0) //  بناء فصيلة الأساس وفصيلة الدائرة
  {
  }
  public override double  Area()
```

```
  {
    // لإرجاع مساحة الدائرة ــ لاحظ أن الحقل x يمثل نصف القطر
    return Math.PI * x * x;
  }
}

class Sphere: AreasClass  // فصيلة الكرة
{
  public Sphere(double r): base(r, 0) // بناء فصيلة الأساس وفصيلة الكرة
  {
  }
  public  override double Area()
  {
    // x لاحظ أن نصف القطر هو الحقل
    return 4 * Math.PI * x * x;
  }
}

class Cylinder: AreasClass  // فصيلة الاسطوانة
{
  // بناء فصيلة الأساس وفصيلة الاسطوانة
  public Cylinder(double r, double h): base(r, h)
  {
  }
  public override double Area()
  {
    // x لاحظ أن نصف القطر هو الحقل
    return 2*Math.PI*x*x + 2*Math.PI*x*y;
  }
}

class MyClass
{
  public static void Main()
  {
    // استقبال الأرقام من لوحة الأزرار
    Console.Write("Please  enter the radius: ");
```

```
    double radius = Convert.ToDouble(Console.ReadLine());
    Console.Write("Please enter the height: ");
    double height = Convert.ToDouble(Console.ReadLine());

    // Create objects: خلق الأهداف
    Point myPoint = new  Point();
    Circle myCircle = new Circle(radius);
    Sphere myShpere = new Sphere(radius);
    Cylinder myCylinder = new Cylinder(radius,height);

    // Display results: طباعة النتائج (المساحات)
    Console.WriteLine("Area of your point    = {0:F2}",
                      myPoint.Area());
    Console.WriteLine("Area of your circle   = {0:F2}",
                      myCircle.Area());
    Console.WriteLine("Area of your sphere   = {0:F2}",
                      myShpere.Area());
    Console.WriteLine("Area of your cylinder = {0:F2}",
                      myCylinder.Area());
  }
}
```

تنفيذ البرنامج

عند تشغيل هذا البرنامج سوف يطلب منك إدخال عددين ، الأول يمثل نصف القطر للدائرة أو الكرة أو سطح الإسطوانة ، والثانى يمثل ارتفاع الإسطوانة. والآتى بعد عينة من تنفيذ البرنامج بإدخال العددين 4, 6 لنصف القطر والإرتفاع. ثم يوافيك البرنامج بالنتائج الموضحة.

```
Please enter the radius: 4
Please enter the height: 6
Area of your point    = 0.00     مساحة النقطة
Area of your circle   = 50.27    مساحة الدائرة
Area of your sphere   = 201.06   مساحة سطح الكرة
Area of your cylinder = 251.33   مساحة سطح الأسطوانة
```

تدريب (1-6)

ورّث فصيلة المواطن (Citizen) إلى فصيلة الموظف (Employee) كما فى المثال 10-5 ، ثم عدّل الكود باستخدم الدوال الافتراضية والدوال الراكبة للحصول على نفس المعلومات التى يوافينا بها البرنامج.

الفصائل والأساليب المجردة abstract

تستخدم الكلمة المفتاحية **abstract** فى إعلان الفصائل والأساليب والخواص المجردة. وتستخدم الفصيلة المجردة للوراثة فقط. ويتم إعلانها كالآتى:

```
abstract class MyBaseClass  // فصيلة مجردة
{
  public abstract void MyMethod(); // أسلوب مجرد
}
```

وتتميز الفصيلة المجردة بالخصائص التالية:

- ❖ لا يجوز خلق أهداف من الفصيلة المجردة.
- ❖ يجوز أن تحتوى على أساليب وخواص مجردة.
- ❖ عند توريث الفصيلة المجردة فلابد من تطبيق أساليبها وخواصها بالفصيلة المشتقة.

أما الأسلوب المجرد فهو عبارة عن أسلوب افتراضى (بطريقة ضمنية) وهو لا يوجد إلاّ بداخل الفصائل المجردة ولا يحتوى على أى

تطبـــيق كمـــا نرى فى الإعلان السابق. ويأتى تطبيق الأسلوب بداخل الأســـاليب الراكـــبة بالفصائل المشتقة. ولا يمكنك أن تستخدم الكلمات الآتية فى إعلان الأساليب المجردة:

- static
- virtual
- override

ملاحظة إلى مبرمجى لغة سى++:

إن الأسلوب المجرد يكافى الدالة الافتراضية النقية (Pure virtual function) فى لغة سى++.

أمـــا الخواص المجردة فهى تماثل الأساليب المجردة فى سلوكها فيما عدا طريقة الإعلان.

مثال (6-2)

```
// Example 6-2.cs
// Abstract Classes

using System;

// Abstract class: فصيلة مجردة
abstract class MyBaseClass
{
  // Fields: الحقول
  protected int number = 100;
  protected string name = "Mohamed Aly";

  // Abstract method: أسلوب مجرد
  public abstract void MyMethod();
```

```
  // Abstract properties: الخواص المجردة
  public abstract int Number
  { get; }
  public abstract string Name
  { get; }
}
// Inheriting the class: وراثة الفصيلة
class MyDerivedClass: MyBaseClass
{
  // Overriding properties: ركوب الخواص
  public override int Number
  {
    get { return number; }
  }
  public overri de string Name
  {
    get { return name; }
  }
  // Overriding the method: ركوب الأسلوب المجرد
  public override void MyMethod()
  {
    Console.WriteLine("Number = {0}", Number);
    Console.WriteLine("Name = {0}", Name);
  }
}

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

تنفيذ البرنامج:

```
Number = 100
Name = Mohamed Aly
```

تدريب (6-2)

علمنا أن الأساليب الراكبة (override) يجوز أن تركب أساليب أخرى راكبة. معنى ذلك أننا نستطيع إضافة أسلوباً راكباً جديداً بالاسم MyMethod() إلى فصيلة جديدة مشتقة من الفصيلة MyDerivedClass بالمثال السابق ، مثل:

```
class MySecondDerivedClass: MyDerivedClass
{
  public override void MyMethod()
  {
    // Implementation  تطبيق الأسلوب
  }
}
```

أضف هذا الأسلوب إلى البرنامج السابق مع كتابة تطبيق مناسب له ثم استدعه من داخل الأسلوب الرئيسى للتحقق من النتيجة.

التحميل الزائد للأساليب (Overloading)

إن التحميل الزائد يعنى أنه يمكنك استخدام نفس الاسم لأكثر من أسلوب واحد ، وعلى المترجم أن يتولى التعرف على الأسلوب المستدعى من نوعية وعدد البارامترات المستخدمة فى الاستدعاء (التوقيع).

وقد يكون من المناسب استخدام التحميل الزائد للأساليب إذا كان العمل الذى تقوم به الأساليب جميعاً متطابق. ولو أنك مثلاً أردت إنشاء أسلوب

لتربـــيع الأعداد الصحيحة وآخر لتربيع الأعداد الحقيقية ، فإنك تستطيع استخدام نفس الاسم لكل من الأسلوبين ، مثل:

```
int SquareIt(int x)
double SquareIt(double f)
```

فإذا كتبت الاستدعاء التالى:

```
SquareIt(3.25);
```

فـــإن المترجم سوف يستدعى الأسلوب SquareIt(double f) على أساس أن البارامتر المستخدم عدد حقيقي. ولو أنك كتبت الاستدعاء:

```
SquareIt(44);
```

فـــإن الأسلوب SquareIt(int x) هو الذى يتم استدعاؤه ، على أساس أن البارامتر عدد صحيح.

وفـــى عملـــية التحمـــيل الـــزائد للأســـاليب فإن نمط الأسلوب (القيمة المرتجعة) لا يمثل أية أهمية. فمن الجائز أن تتشابه جميع الأساليب فى الاســـم والنمط كالمثال التالى ، ولكن المترجم لا زال يفرق بينها وبين بعضها على أساس البارامترات المستخدمة فى الاستدعاء:

```
void SquareIt(int x)
void SquareIt(double f)
```

ومـــن أهـــم فوائـــد التحميل الزائد هو التحميل الزائد للمؤثرات بغرض ابتكار وظائف جديدة للمؤثرات لاستخدامها مع الأهداف ، كما سيلى فى الفقرات القادمة.

مثال (6-3)

فـــى هـــذا المثال استخدمنا ثلاثة أساليب بنفس الاسم MyMethod ولكنها تخـــتلف فى البارامتر المستخدم. وكما نرى فى نتيجة تنفيذ البرنامج أن الأسلوب المناسب قد تم استدعاؤه فى كل حالة.

```
// Example 6-3.cs
// Overloading methods

using System;

class MyClass
{
  void MyMethod(string s1)
  {
    Console.WriteLine(s1);
  }

  void MyMethod(int m1)
  {
    Console.WriteLine(m1);
  }

  void MyMethod(double d1)
  {
    Console.WriteLine(d1);
  }

  static void Main()
  {
    string s = "This is my string";
    int m = 134;
    double d = 122.67;

    MyMethod(s);
    MyMethod(m);
    MyMethod(d);
  }
}
```

تنفيذ البرنامج:

```
This is my string
134
122.67
```

إمرار البارامترات إلى الأساليب

عندما نقوم بإمرار المتغيرات كبارامترات إلى الأساليب المختلفة فإنه من اللازم أن تكون على دراية بما يحدث فى الخلفية لهذه المتغيرات. فقد تقوم بإنشاء أسلوب ما لتغيير قيمة متغير ، ثم تكتشف فى النهاية أن التغيير كان محلياً بداخل الأسلوب وأن قيمة المتغير لم تتغير بصفة نهائية. ومن الأمثلة المشهورة لهذه المسألة دالة تبديل قيمتى متغيرين (Swap(x,y)). وفى لغة سى# يمكنك تمرير البارامترت بالقيمة (وهذه هى الطريقة سابقة التعريف) أو بالمرجع. وتمرير البارمترات بالمرجع يجعل التغيرات التى تجرى على البارامترات تغيرات نهائية. ويتم تمرير البارامترات بالمرجع باستخدام الكلمة **ref**.(سيلى أيضاً شرح الكلمات out و param.)

وفى الفقرات التالية سوف نعرض أمثلة للحالات التى ينجح فيها تغيير قيمة المتغيرات بواسطة الأساليب الأعضاء والحالات الأخرى التى تفشل فيها عملية التغيير.

مثال (4-6) تبديل قيمة متغيرين ــ محاولة ناجحة

فى هذا المثال استخدمنا الأسلوب ()Swap لتبديل محتويات المتغيرين x, y وقد استخدمنا الكلمة ref فى كل من إعلان الأسلوب واستدعاء

الأســـلوب. وكمـــا نرى فى نتيجة التنفيذ أن البرنامج قد نجح فى عملية التبديل.

```
// Example 6-4.cs
// Swap method example

using System;

class MyClass
{
  static void Swap(ref int x, ref int y)  //    تمرير مرجع إلى البارامترات
  {
    int temp = x;
    x = y;
    y = temp;
  }
  static  void Main()
  {
    int x = 25;
    int y = 33;
    //    طباعة المحتويات قبل التبديل
    Console.WriteLine ("Before swapping: x={0}, y={1}", x, y);
    Swap(ref x, ref y);  //    تمرير مرجع إلى البارامترات
    //  طباعة المحتويات بعد التبديل
     Console.WriteLine ("After swapping:  x={0}, y={1}", x, y);
  }
}
```

تنفيذ البرنامج:

```
Before swapping: x=25, y=33        قبل التبديل
After swapping:  x=33, y=25        بعد التبديل
```

مثال (6-5) تبديل قيمة متغيرين ـ محاولة فاشلة

أمـــا فـــى المـــثال التالى فتستطيع أن ترى كيف أن تمرير البارامترات بالقـــيمة لا يؤدى إلى النتيجة المطلوبة. وقد طبعنا قيمة المتغيرين x, y مـــن داخـــل الأســـلوب الرئيسى ومن داخل الأسلوب Swap() لمتابعة الأحداث الجارية على المتغيرات.

```
// Example 6-5.cs
// Swap method example

using System;

class MyClass
{
  static void Swap(int x, int y)
  {
    int temp = x;
    x = y;
    y = temp;
    //  طباعة المحتويات بداخل أسلوب التبديل
    Console.WriteLine ("Values inside the method: x={0},
y={1}", x, y);
  }
  static void Main()
  {
    in t x = 25;
    int y = 33;
    //  طباعة المحتويات قبل التبديل
    Console.WriteLine ("Before swapping: x={0}, y={1}", x, y);
    Swap(x,y);
    //  طباعة المحتويات بعد التبديل
    Console.WriteLine ("After swapping:  x={0}, y={1}", x, y);
  }
}
```

تنفيذ البرنامج:

```
Before swapping: x=25, y=33                قبل التبديل
Values inside the method: x=33, y=25       من داخل أسلوب التبديل
After swapping:  x=25, y=33                بعد التبديل (لا تغيير)
```

ملاحظات على البرنامجين السابقين:

لمـــاذا فشلت عملية التبديل فى هذا المثال؟ ولماذا نجحت فى المثال السابق؟ إن ما حدث أن التبديل قد تم بالفعل بداخل الأسلوب Swap ولكن المتغيرات التى استخدمت فى التبديل هى مجرد نسخة من المتغيرات x, y. أمـــا فـــى المثال السابق فقد مرّرنا عناوين خلايا الذاكرة التى تمثل المتغـــيرات ولذلك نجحت عملية التبديل. وهذا هو نفس الأسلوب المتبع فى عملية التبديل باستخدام المؤشرات فى لغة سى++.

تدريب (3-6)

اكتب أسلوباً لتبديل محتويات متغيرين حرفيين مثل:

```
s1 = "Mohamed";
s2 = "Aly";
```

ثم استدع الأسلوب واطبع النتيجة. وفيما يلى صورة من النتيجة المتوقعة لتنفيذ البرنامج:

```
Before swapping: s1 = Mohamed, s2 = Aly    محتويات المتغيرات قبل الاستدعاء
Inside the swap method: s1 = Aly, s2 = Mohamed    بداخل الأسلوب
After swapping: s1 = Aly, s2 = Mohamed     محتويات المتغيرات بعد الاستدعاء
```

الطرق المختلفة لتمرير البارامترات ref, out , params

هـناك أكـثر مـن طـريقة لإعلان بارامترات الدوال ، بحيث تحتفظ الأسـاليب بالتغيرات الحادثة فى المتغيرات عند رجوع الأسلوب وتسليم دفـة التحكم إلى الأسلوب الذى تم منه الاستدعاء. وكما رأينا فى الفقرة السابقة أن الكلمة ref تستخدم لتمرير المتغيرات إلى الأساليب بالمرجع. فـى الحقـيقة أن هناك ثلاث كلمات يمكنك استخدامها مع البارامترات لـتمرير البارامـترات بالمـرجع وهـى تختلف فيما بينها فى طريقة الاستخدام:

- ref
- out
- params

استخدام الكلمة ref

أمـا الكلمـة الأولى ref فهى تتطلب شحن المتغيرات بقيمة ابتدائية قبل تمريرها إلى الأساليب فى صورة بارامترات ، وإلاّ فإن الترجمة لن تتم. وعلى سبيل المثال:

```
string myVariable = "This my string"; // شحن المتغير قبل الاستدعاء
MyMethod(ref myVariable); // استدعاء الأسلوب
```

وعـند رجوع هذا الأسلوب فإن المتغير myVariable سوف يحتفظ بأى تغير يجرى عليه بداخل الأسلوب MyMehod.

استخدام الكلمة out

أمـا الكلمـة out فهى لاتتطلب شحن المتغيرات مبدئياً ، ولكنها تتطلب

شـــحنها بداخل الأسلوب المستقبل للبارامترات (MyMethod). وهذا مثال للأسلوب MyMehod:

```
static void MyMethod(out string  myVariable)
{
  myVariable = "This my string";
}
```

وعند استخدام هذا الأسلوب فإنه يُستدعى باستخدام الكلمة out:

```
MyMethod(out myVariable);  // استدعاء الأسلوب
```

ومن الجائز أن يستخدم الأسلوب أكثر من بارامتر واحد مثل:

```
YourMethod(out int x, out int y, out int z)  {}
```

ويتم الاستدعاء بنفس الطريقة ، أى:

```
YourMethod(out m, out n, out l);
```

مـــن الجدير بالذكر أن اختلاف الأساليب فى الكلمة المستخدمة مع البارامترات (فقط) لا يكفى لتحميل الأسلوب تحميلاً زائداً.

مثال (6-6)

يوضـــح المثال التالى كيفية استخدام البارامتر out فى برنامج حيث نرى أن الأسلوب MyMethod يقوم بشحن بارامترات المصفوفة التى تم إعلانهـــا فى الأسلوب الرئيسى Main. ومن الجدير بالذكر أنه ليس من المطلـــوب شـــحنها بداخل الأسلوب الرئيسى لأن الأسلوب MyMethod سيقوم بإعادة شحنها وبالأبعاد المطلوبة.

```
// Example 6-6.cs
// out Example
```

```
using System;

public class MyClass
{
  public static void MyMethod(out int[] warList)
  {
    warList = new int[] {48, 56, 67, 73};  //    شحن المصفوفة
  }

  public static void Main()
  {
    int[] myarray;  //  الإعلان بدون شحن
    MyMetho d(out myarray);
    for (int i = 0 ; i < myarray.Length ; i++)
      Console.Write("{0} ", myarray[i]);
  }
}
```

تنفيذ البرنامج:

```
48 56 67 73
```

ملاحظات على البرنامج السابق:

لاحظ في هذا البرنامج أنه بالرغم من أن المصفوفة قد تم شحنها بداخل الأسلوب MyMethod ولكننا قمنا بطباعة محتوياتها من الأسلوب الرئيسي. وهذا لا يمكن تحقيقه بدون الكلمة out. لاحظ أيضاً أن استخدام الكلمة static ليست ضرورية ، وقد استخدمناها هنا حتى نتمكن من استدعاء الأسلوب مباشرة بدون الحاجة إلى خلق الأهداف.

تدريب (6-4)

أجر التعديلات اللازمة على البرنامج السابق بحيث:

- تستخدم أسلوباً غير استاتيكى
- تستخدم عدة بارامترات من النمط الصحيح بدلاً من المصفوفة

ثـــم أضف أسلوبا آخر بنفس الاسم ونفس أنماط البارامترات مع تغيير الكلمة out إلى ref ، ثم اختبر التحميل الزائد للأسلوبين (قم بتغيير عدد البارامترات إن لزم الأمر).

استخدام الكلمة param

أمـــا الكلمـــة params فهى تستخدم مع المصفوفات. وهى تسمح لك بإرسال أى عدد من البارامترات إلى الأسلوب المستدعى ، بدون الحاجة إلـــى إعلانها بالضرورة فى مصفوفة. والكلمة param مطلوبة فقط فى إعـــلان الأسلوب المستدعى ، ولكنها لا تستخدم فى الاستدعاء ، وعلى سبيل المثال الأسلوب:

```
static void MYMethod(params object[] myObjArray)
```

يمكن استدعاؤه بالطريقة الآتية:

```
MyMethod(123, 'A', "My original string")
```

فالبارامـــترات المســـتخدمة هنا من النوع object بمعنى أنها يجوز أن تكون تشكيلة من الأنماط المختلفة التى تنحدر من النمط object.

كذلك فإن الأسلوب التالى:

```
static void MYMethod(params int[] myIntArray)
```

يمكـــنه فقـــط أن يســـتقبل مجموعـــة من الأعداد الصحيحة التى تشكل مصفوفة من النوع الصحيح (بصرف النظر عن طولها) مثل:

```
MyMethod(2, 4, 7)
```

ومـــن الجدير بالذكر أن الأسلوب الذى يستخدم الكلمة param لا يجوز أن يستخدام أكثر من بارامتر واحد.

مثال (6-7)

فـــى هـــذا المثال نقدم استخدام الكلمة param فى إعلان بارامترات الأســـلوب المســـتدعى. كمـــا نقدم التحميل الزائد للأسلوب بتغيير نوع البارامترات المستخدمة.

```
// Example 6-7.cs
// params and overloading example

using System;

public class MyClass
{

  public void MyMethod(params int[] myIntArray)
  {
    // Changing one element:
    myIntArray[1]= 555;

    Console.WriteLine("My integer list:");
    for ( int i = 0 ; i < myIntArray.Length ;    i++ )
      Console.WriteLine(myIntArray[i]);
    Console.WriteLine();
  }

  public void MyMethod(params object[] myObjArray)
  {
    // Changing one element:
```

```
    myObjArray[2] = "This a new string";
  }
}

class MainClass
{
  static void M ain()
  {
    object[] myObjList =
        new object[] {123, 'A', "My original string"};

    MyClass mc = new MyClass();

    mc.MyMethod(11, 22, 33);  //   استخدام بارامترات عددية
    mc.MyMethod(myObjList);  //   استخدام مصفوفة أهداف

     Console.WriteLine("My object list:");
    for ( int i = 0 ; i < myObjList.Length ; i++ )
      Console.WriteLine(myObjList[i]);
  }
}
```

تدريب (5-6)

اكتب أسلوباً يمكنه استقبال البارمترات بأى من الطرق الآتية:

```
mc.MyMethod(11, 22, 33);
mc.My Method(45.33, 'A', "My string");
```

الفهارس (Indexers)

تستخدم الفهارس فى التعامل مع الفصيلة كما لوكانت مصفوفة (أو مجموعة "Collection") باستخدام الأقواس []. وتشبه الفهارس الخواص فى أنها تستخدم الأساليب get و set فى التعبير عن سلوك الفهرست.

ويعلن الفهرست كالآتى:

```
indexer-type this [parameter-type parameter]
{
  get {};
  set {};
}
```

حيث:

- ❖ indexer-type: نمط الفهرست
- ❖ parameter-type: نمط البارامتر
- ❖ parameter: بارامتر أو قائمة من البارامترات

وتشير الكلمة this إلى الهدف الذى يظهر بداخله الفهرست. وبالرغم من أن الفهرست لا يحمل اسماً ما ، ولكنه يتم تمييزه بتوقيعه أى بعدد ونوع البارمـترات التى يستخدمها. ويجوز تعديل إعلان الفهرست إما بالكلمة new أو بأحد معدلات التوصل.

مثال (6-8)

فـى هـذا المثال نعلن عن مصفوفة وفهرست من النمط string وقد اسـتخدمنا الفهرسـت فـى التوصل إلى أعضاء الهدف s كما لو كانت أعضاء مصفوفة ، أى باستخدام s[1] و s[2] إلى آخره.

```
// Example 6-8.cs
// Indexer example

using System;

class MyClass
{
```

```
  private string[] myArray = new string[10];
  // Indexer declaration:  إعلان الفهرست
  public string this[int index]
  {
    get
    {
        return myArray[index];
    }
    set
    {
        myArray[index] = value;
    }
  }
}

public class MainClass
{
  public static void Main()
  {
    MyClass s = new MyClass();
    // #1, #2  استدعاء الفهرست لشحن الأعضاء
    s[1] = "Aly";
    s[2] = "Taha";

    for (int i=0; i<5; i++)
    {
      Console.WriteLine("{0}={1}", i, s[i]);
    }
  }
}
```

تنفيذ البرنامج:

```
0=
1=Aly
2=Taha
3=
4=
```

ملاحظات على البرنامج السابق:

- من الشائع أن تستخدم الأساليب get و set فى اختبار حدود الفهرست منعاً لحدوث أخطاء ، وذلك مثل:

```
if (!(index < 0 || index >= 10))
// ...
```

- يجوز أن تتضمن الوصلات البينية (interfaces) فهارس كما الفصائل ، ويتم إعلانها بنفس الطريقة فيما عدا أن الوصلات البينية لا تستخدم المعدّلات ، كما أن الأساليب set و get لا يتم تطبيقها إلاّ فى الفصائل. والآتى بعد مثال لهذه الأساليب عند استخدامها مع الوصلات البينية:

```
string this[int index]
{
  get;
  set;
}
```

المؤثرات المبتكرة (User-defined operators)

باستخدام خاصية التحميل الزائد يمكنك أن تبتكر وظائف جديدة لبعض المؤثرات. وقد تنشأ الحاجة إلى ذلك عند التعامل مع الأهداف من الفصائل والمنشآت ، فقد ترغب مثلاً فى اختبار التساوى بين هدفين باستخدام مؤثر التساوى (==). فى هذه الحال فإنك تعيد تعريف المؤثر فى برنامجك حتى تكسبه هذه الوظيفة المبتكرة.

ويتم تحميل المؤثرات باستخدام الكلمة المفتاحية operator كالمثال الآتى لتحميل المؤثر + بحيث يصلح لجمع الأعداد المركبة (Complex) وهى

الأعـــداد التى تتكون من جزء حقيقى (Real) وآخر تخيّلى (Imaginary) مثل العدد 21i + 34:

```
public static CompNum operator+( CompNum n1+CompNum n2)
{
  // تطبيق التعريف الجديد //
}
```

ونلاحـــظ هـــنا أن المؤثـــر المرغوب فى إعادة تعريفه يأتى تالياً لكلمة operator مـــثل operator+ أو operator== وهكـــذا. أمـــا الأســـلوب المســـتخدم فهو دائماً إستاتيكى. والجدول التالى يحصر المؤثرات التى يجـــوز تحميلها تحميلاً زائداً. ونلاحظ أن بعض المؤثرات ثم تصنيفها على أنها فردية (Uniary) بمعنى أنها تؤثر على معامل واحد مثل ++ ، وتـــم تصنيف البعض الأخر كمؤثرات ثنائية (Binary) وهى التى تؤثر على معاملين مثل مؤثرات الضرب والقسمة.

جدول (1-6) المؤثرات التى يجوز تحميلها تحميلاً زائداً

المؤثرات	ملاحظة على التحميل الزائد
+, -, !, ~, ++, --, true, false	يجوز تحميلها كمؤثرات فردية
+, -, *, /, %, &, |, ^, <<, >>	يجوز تحميلها كمؤثرات ثنائية
==, !=, <, >, <=, >=	أنظر الملاحظة أسفل الجدول
&&, ||	لا يجوز تحميلها مباشرة ، حيث أنها يتم تعرـــيفها من خلال تحميل المؤثرات & و |
+=, -=, *=, /=, %=, &=, |=, ^=, <<=, >>=	لا يجوز تحميل المؤثر = ولكن المؤثر =+ يتم تعريفه مع تحميل المؤثر +

=, ., ?:, ->, new, is, sizeof, typeof	لا يجوز تحميلها

ملاحظة: يشترط فى لغة سى# ، بخلاف لغة سى++ ، أن تقوم بتحميل المؤثـــرات فـــى أزواج. بمعنى أنه إذا قمت بتحميل المؤثر == فعليك أيضاً أن تقوم بتحميل المؤثر المناظر =!. وكذلك الحال بالنسبة للمؤثر < ونظيره > ، وبالنسبة للمؤثر =< ونظيره =>.

مثال (6-9)

فـــيما يلى نقدم المثال الكامل لتعريف مؤثر جمع الأعداد المركبة حيث نستخدم العددين 15+33i و 10+12i ونطبع نتيجة الجمع فى صورة عدد مركب: 25 + 45i

```
// Example 6-9.cs
// Overloading operators

using System;

public class CompNum
{
  public int real;
  public int imag;

  // أسلوب البناء
  public CompNum(int r, int i)
  {
    real = r;
    imag = i;
  }

  // إعلان المؤثر المحمّل تحميلاً زائداً
  public static CompNum operator+(CompNum c1, Comp  Num c2)
  {
```

```
    // القيمة المرتجعة من عملية الجمع
    return new CompNum(c1.real + c2.real, c1.imag + c2.imag);
  }

  static void Main()
  {
    // إعلان الأعداد المركبة كأهداف من الفصيلة
    CompNum n1 = new CompNum (15, 33);
    CompNum  n2 = new CompNum (10, 12);

    // اجمع الهدفين وخصص الناتج إلى عدد مركب
    CompNum sum = n1 + n2;

    // طباعة العددين بالصورة المناسبة
    Console.WriteLine("Num1 = {0} + {1}i",n1.real, n1.imag);
    Console.WriteLine("Num2 = {0} + {1}i",n2.  real, n2.imag);

    // طباعة مجموع العددين
    Console.WriteLine("Sum  = {0} + {1}i",sum.real, sum.imag);
  }
}
```

تنفيذ البرنامج:

```
Num1 = 15 + 33i
Num2 = 10 + 12i
Sum  = 25 + 45i
```

ركوب الأسلوب ToString

إن الأسلوب ToString من أهم أساليب مكتبة دوت.نت حيث أنه ، كما ذكرنا فى الباب الثالث ، يعمل فى خلفية البرامج عند استخدام أسلوب الطباعة Console.Write أو Console.WriteLine. ومن الجائز استخدامه صراحة مثل:

```
Console.Write(myVariable.ToString());
```

ولكننا تعودنا أن نكتب هذه العبارة كالآتى:

```
Console.Write(myVariable);
```

والأسلوب ToString عبارة عن أسلوب راكب وقد تم تعريفه فى الفصيلة System.AppDomain كالآتى:

```
public override string ToString();
```

وقد يتطلب الأمر فى بعض الحالات أن نركب الأسلوب ToString لتغيير سلوكه ، ولا سيما فى التطبيقات التى نتعامل فيها مع الأهداف كما لوكانت متغيرات عادية. وتحميل المؤثرات هو أحد هذه التطبيقات.

ولنبدأ بمثال بسيط لتغيير سلوك أسلوب الطباعة بحيث أنه عندما يطبع هدفاً ما فإنه يطبع الرسالة "Hello there!". وهذا هو الأسلوب فى صورته الجديدة:

```
using System;
class MyClass
{
  public override string ToStrin g()  // ToString ركوب الأسلوب
  {
    return("Hello there!");
  }
  static void Main()
  {
    MyClass s = new MyClass();  //  إعلان الهدف
    Console.WriteLine(s);  //  طباعة الهدف
  }
}
```

إن هذا البرنامج سوف يطبع الرسالة "Hello there!" نتيجة لتغيير سلوك الأسلوب ToString.

ولـــو كان الهدف يحتوى على حقول كما فـــى فـــــصيلة النقطة مثلاً ، فإنـــه يلـــــزم اســـتخدام أســـــلـــوب التحويل إلى الصيغة الحرفية (String.Format) عند ركوب الأسلوب كالآتى:

```
return (String.Format("{0}, {1}", x, y));
```

وهذه هى الصورة الجديدة لطباعة هدف من فصيلة النقطة:

```
using System;
class Point
{
  public int x=5, y=10;
  public override string ToString()
  {
    // ركوب الأسلوب مع تعديل الفورمات
    return (String.Format("{0}, {1}", x, y));
  }
  static void Main()
  {
    Point p1 = ne  w Point();
    Console.WriteLine(p1);  // طباعـــة الهـــدف
  }
}
```

عند تنفيذ هذا البرنامج يعطى النتيجة: 5, 10

تدريب (6-6)

عدل المثال السابق (6-9) بحيث تطبع العدد المركب كهدف ، أى:

```
Console.WriteLine("Sum = ", Sum);
```