الصفوف المجرّدة والصفوف المحكمة
Abstract and Sealed Classes

الكلمات المفتاحية

الصفّ المجرّد، الصفّ المحكم.

ملخص الفصل

في هذا الفصل، يتمّ توضيح مفهوم الصفّ المجرّد وكيفية التصريح عنه وكيفية استخدامه. ويتضمّن الفصل شرحاً لكيفية منع الوراثة لأحد الصفوف، حيث يمكن منع وراثة صفّ بأكمله أو الاكتفاء بمنع تعديل سلوك إحدى طرائقه فقط.

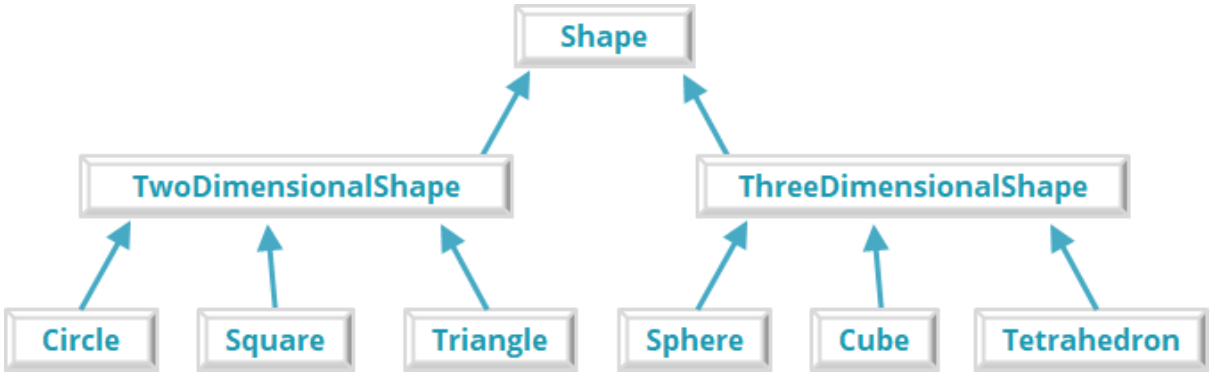
الأهداف التعليمية

يتعرّف الطالب في هذا الفصل على:

- الأعضاء المجرّدة وكيفية التصريح عنها
- الصفّ المجرّد
- الصفّ المحكم وكيفية التصريح عنه
- الطريقة المحكمة

مقدمة

في كثير من الأحيان، يكون من المفيد التصريح عن صفوف لن يتمّ إنشاء أغراض منها باستخدام الكلمة new على الرّغم من توفّر إمكانية التصريح عن أغراض منها. ويُستخدم الصفّ المجرّد Abstract Class عادة لتجميع الأعضاء المشتركة في هرمية في صف أساسي واحد. ففي هرمية الأشكال الهندسية، على سبيل المثال، يمكن تعريف الصفّ المجرّد Shape كرأس للهرمية كما هو موضّح فيما يأتي:



وفي كثير من الحالات، يمكن منع وراثة صفّ من قبل صفوف أخرى فيتمّ التصريح عنه بأنّه صفّ مختوم Sealed Class.

1. الصفوف المجرّدة

الصفّ المجرّد هو صفّ لا يمكن إنشاء أغراضٍ منه، ويتمّ التصريح عنه باستخدام الكلمة المفتاحية abstract. أمّا الطريقة المجرّدة، فلها توقيع فقط، أي أنّها لا تمتلك أية تعليمات، ويتمّ التصريح عنها باستخدام الكلمة المفتاحية abstract، وهي طريقة افتراضية ضمنياً. ويمكن أن تكون خاصّية ما مجرّدة إذا سُبِقَت بالكلمة abstract ولم تحوِ تنجيزاً له، ولكن لا يمكن للخاصّية الساكنة (المسبوقة بالمحدّد static) أن تكون مجرّدة. ولا يمكن لطريقة مجرّدة (أو خاصية مجرّدة) أن توجد إلّا في صفّ مجرّد، أي أنّه عند التصريح عن طريقة بأنّها مجرّدة، يجب التصريح عن الصفّ الحاوي للطريقة بأنّه مجرّد أيضاً. ومن الخطأ استخدام المحدّدات static وvirtual مع طريقة مجرّدة. ويمكن للصفّ أن يكون مجرّداً على الرّغم من عدم وجود أي عضو مجرّد من بين أعضائه، أي يمكن أن تكون جميع أعضاء الصفّ بما فيها الطرائق غير مجرّدة، ويكون الصفّ مجرّداً.

أمثلة توضيحية

مثال1:

في الرمّاز الآتي ، يضمّ الصفّ المجرّد BaseClass الطريقة المجرّدة AbstractMethod والخاصّيتين المجرّدتين X وY:

```
using System;
namespace AbstractClasses
{

    abstract class BaseClass // Abstract class
    {
        protected int _x = 100;
        protected int _y = 150;
        public abstract void AbstractMethod(); // Abstract method
        public abstract int X { get; }
        public abstract int Y { get; }
    }// end class DerivedClass
    class DerivedClass : BaseClass
    {
        public override void AbstractMethod()
        {
            _x++; _y++;
        }
        public override int X // overriding property
        {
            get { return _x + 10; }
        }
        public override int Y // overriding property
        {
            get { return _y + 10; }
        }
        static void Main()
        {
            DerivedClass o = new DerivedClass();
            o.AbstractMethod();
            Console.WriteLine("x = {0}, y = {1}", o.X, o.Y);
        }// end Main
```

```
    }//end class DerivedClass
}//end namespace AbstractClasses
```

```
Output: x = 111, y = 161 //
```

يرث الصفّ المشتقّ DerivedClass الصفّ المجرّد BaseClass ويتمّ ضمنه تنجيز الأعضاء المجرّدة باستخدام الكلمة override، وبذلك يمكن إنشاء أغراض من الصفّ المشتقّ كالغرض o الذي تمّ استخدامه لاستدعاء الطريقة AbstractMethod وكتابة قيم الخصائص X وY.

عند محاولة إنشاء أغراض من الصفّ المجرّد BaseClass باستخدام التعليمة:

```
BaseClass bc = new BaseClass ;
```

سنحصل على رسالة الخطأ الآتية والتي تفيد بعدم المقدرة على إنشاء أغراض من صفّ مجرّد:

Error	CS0144	Cannot create an instance of the abstract class or interface 'BaseClass'

ولا يمكن إنشاء أغراض من صفّ وارث لصفّ مجرّد إلّا إذا تمّ ضمنه تنجيز جميع الأعضاء المجرّدة. فإذا ألغينا تنجيز الخاصّية Y في الصفّ المشتقّ وحاولنا التنفيذ، سنحصل على رسالة الخطأ الآتية التي تشير إلى أنّنا لم نقم بتنجيزها:

Error	CS0534	'DerivedClass' does not implement inherited abstract member 'BaseClass.Y.get'

مثال2:

يمكن لصفّ مجرّد أن يرث صفّاً مجرّداً آخر، ولكن لا يعني ذلك أنّه أصبح بالإمكان إنشاء أغراض منه. لنعدّل الرمّاز السابق بحيث يرث الصفّ DerivedClass الصفّ المجرّد BaseClass2 والذي بدوره يرث الصفّ المجرّد BaseClass، ولنقم بتعديل الطريقة Main كما هو موضّح:

```
using System;
namespace AbstractClasses
{

    abstract class BaseClass // Abstract class
    {
```

```
        protected int _x = 100;
        protected int _y = 150;
        public abstract void AbstractMethod(); // Abstract method
        public abstract int X { get; }
        public abstract int Y { get; }
    }// end class DerivedClass

    abstract class BaseClass2 : BaseClass
    {
        protected int _z = 200;
        public abstract int Z { get; }
    }
    class DerivedClass : BaseClass2
    {
        public override void AbstractMethod()
        {
            _x++; _y++; _z++;
        }
        public override int X // overriding property
        {
            get { return _x + 10; }
        }
        public override int Y // overriding property
        {
            get { return _y + 10; }
        }
        public override int Z // overriding property
        {
            get { return _z + 10; }
        }
        static void Main()
        {

            DerivedClass o = new DerivedClass();
            o.AbstractMethod();
            Console.WriteLine("x = {0}, y = {1} , z = {2}", o.X, o.Y, o.Z);
        }// end Main
    }//end class DerivedClass
}//end namespace AbstractClasses
```

```
Output: x = 111, y = 161, z= 211
```

نلاحظ أنّه تمّ إنشاء الغرض o من الصفّ المشتقّ DerivedClass بنجاح، وتمّ استدعاء إحدى الطرائق التي قام بتنجيزها، كما تمّ استدعاء الخاصّيتين X وY من الصفّ المشتقّ الأوّل والخاصّية Z من الصفّ المشتقّ الثاني.

مثال3:

إن عدم التمكّن من إنشاء غرض من صفّ مجرّد لا يعني أنّه لا يمكن استخدام مرجع منه لأحد الأغراض المنتسخة من صفّ مشتقّ منه. ففي الرمّاز الآتي، تمّ تعريف الصف المجرّد Animal الحاوي على الطريقة المجرّدة FoodHabits، وتمّت وراثته من الصفّ Herbivores والصفّ Carnivores، وقام كلّ من الصفّين الوارثين بتنجيز الطريقة المجرّدة.

```
using System;
abstract class Animal
{
    public abstract void FoodHabits(); //empty
}// end class Animal
class Herbivores : Animal  //derived 1
{
    public override void FoodHabits()
    {
        Console.Write(this.ToString());
        Console.WriteLine(", they eat only plants");
    }
}// end class Herbivors
class Carnivores : Animal //derived 2
{
    public override void FoodHabits()
    {
        Console.Write(this.ToString());
        Console.WriteLine(", they eat animals");
    }
    public void PrintInfo()
    {
        Console.WriteLine("I am a Carnivore");
    }
```

```
}// end class Carnivors

class Tester
{
    static void Main()
    {
        Animal al;
        Carnivores c = new Carnivores();
        Herbivores h = new Herbivores();
        al = c;
        h.FoodHabits();
        c.FoodHabits();
        al.FoodHabits();
        // al.PrintInfo();
        Console.ReadKey();
    }
}// end class Test
```

وفي الطريقة Main الموجودة ضمن الصفّ Tester، تمّ إنشاء المرجع al من الصفّ المجرّد Animal والغرض c من الصفّ Carnivores والغرض h من الصفّ Herbivores. وتمّ إسناد الغرض c إلى المرجع al، ثمّ تمّ استدعاء الطريقة FoodHabits مع كلّ من c و h و al. وبعد التنفيذ، تمّ الحصول على الخرج الآتي:

```
Herbivores, they eat only plants
Carnivores, they eat animals
Carnivores, they eat animals
```

ونلاحظ أنّه تمّ استدعاء الطريقة FoodHabits باستخدام al، ولكن عند محاولة استدعاء الطريقة PrintInfo مع المرجع al ستظهر الرسالة الآتية:

Error	CS1061	'Animal' does not contain a definition for 'PrintInfo' and no accessible extension method 'PrintInfo' accepting a first argument of type 'Animal' could be found (are you missing a using directive or an assembly reference?)

والتي تفيد بأنّ الصفّ Animal لا يحتوي على تعريف للطريقة PrintInfo، أي أنّ al هو غرض من الصفّ Animal وليس غرضاً من الصفّ Carnivores. وبذلك يكون الصفّ المجرّد هو صفّ لا يمكن إنشاء أغراض منه باستخدام الكلمة المفتاحية new، ولكن يمكن إنشاء أغراض منه باستخدام مرجع لغرض من صفّ مشتقّ منه، أو عن طريق استخدام بانٍ لصفّ مشتقّ منه. فيمكن إنشاء الغرض al من الصفّ المشتقّ Animal باستخدام التعليمة:

```
Animal al = new Carnivores();
```

وسيستخدم هذا الغرض التنجيز المتاح في الصفّ Carnivores. ولو استبدلنا في الرمّاز السابق محتوى الطريقة Main بما يأتي:

```
Animal al = new Carnivores();
Herbivores h = new Herbivores();
h.FoodHabits();
al.FoodHabits();
```

وأعدنا التنفيذ، لحصلنا على الخرج الآتي:

```
Herbivores, they eat only plants
Carnivores, they eat animals
```

2. الصفوف والطرائق المحكمة

الصفّ المحكم Sealed Class هو صفّ لا يمكن الوراثة منه ولكن بإمكانه الوراثة من صفّ آخر ويتمّ التصريح عن ذلك باستخدام المحدّد sealed.

```
sealed class A  { }
class B : A { } //Error : 'B' cannot derive from sealed type 'A'
```

أي لا يمكن لصفّ محكم أن يكون صفّاً أساساً Base Class لأي صفّ آخر، ولا يمكن أن يكون صفّاً مجرّداً، أي لا يمكن استخدام المحدّدين sealed وabstract مع نفس الصفّ.
ويمكن للطرائق والخصائص والمفهرسات والأحداث المتجاوزة overriding لطريقة افتراضية في صفّ أساس أن تُسبَق بالمحدّد sealed قبل المحدّد override. وفي هذه الحالة، لا يمكن تجاوزها في الصفوف التي ترث الصفّ المعرّفة ضمنه، ولكن تبقى إمكانية استدعائها متاحة.

مثال:

في الرمّاز التالي، تمّ تعريف الصفّ BaseClass الحاوي على الطريقة الافتراضية Add التي تعيد مجموع قيمتي وسيطي دخل من النمط int. وتمّ تعريف الصفّ DerClass الوارث للصفّ السابق، ويتمّ ضمنه تجاوز الطريقة Add بحيث تصبح قادرة على جمع ثلاثة قيم من النمط int، وتمّ التصريح عنها بأنّها محكمة أي لا يمكن تجاوزها وتغيير سلوكها إذا ما تمّت وراثتها. وتمّ تعريف الصفّ DerClass2 الذي يرث الصفّ DerClass، وتمّ ضمنه تجاوز الطريقة المحكمة Add.

```
using System;
class BaseClass
{
    public virtual int Add(int x, int y)
    {
        return (x + y);
    }
}// end class BaseClass
class DerClass : BaseClass
{
    // sealed method
    public sealed override int Add(int x, int y)
    {
        return (x + 2 * y);
```

```
    }
    public int Add3(int xx, int yy, int zz)
    {
        return (zz + Add(xx, yy));
    }
}// end class DerClass

class DerClass2 : DerClass
{
    // Error: you cannot override sealed methods
    public override int Add(int x, int y)
    {
        return (x + y);
    }
    public new int Add3(int xx, int yy, int zz)
    {
        return (zz + Add(xx, yy));
    }
}// end class DerClass2

class Tester
{
    static void Main()
    {
        DerClass2 sealedCls = new DerClass2();
        int total = sealedCls.Add(4, 5);
        Console.WriteLine("Total = " + total.ToString());
        DerClass derClass = new DerClass();
        total = derClass.Add3(5, 6, 8);
        Console.WriteLine("Total = " + total.ToString());
        Console.ReadKey();
    }// end Main
}// end Tester
```

وفي الصفّ Tester، وضمن الطريقة Main، تمّ إنشاء الغرض dc من الصفّ DerClass2، ثمّ تمّ استدعاء الطريقة Add من أجل القيمتين (4) و(5). وعند التنفيذ نحصل على رسالة الخطأ:

```
'DerClass2.Add(int, int)': cannot override inherited member
'DerClass.Add(int, int)' because it is sealed
```

والتي تفيد بعدم السماح بتغيير سلوك الطريقة Add الموروثة من الصفّ DerClass لأنّها محكمة. أما إذا قمنا بإلغاء تجاوز الطريقة DerClass.Add، وأعدنا التنفيذ فلن تظهر أي رسالة خطأ وسنحصل على الخرج المتوقع الآتي:

```
Total = 9
```

يوجد نمط معطيات أكثر تجريداً من الصفّ المجرّد نفسه، وتكون طرائقه مجرّدة ضمنياً ويُعرف هذا النمط بالواجهة Interface، وستتمّ دراسته في الفصل القادم.

الأنشطة المرافقة

1. تقوم شركة بدفع الرواتب أسبوعياً لموظّفيها، وتضمّ أربعة أنواع من الموظّفين:
   - `Salaried Employee` وهو موظّف يتقاضى معاشاً ثابتاً أسبوعياً.
   - `Hourly Employee` وهو موظّف يُدفع له لقاء كلّ ساعة عمل. ويكون له تعويض إضافي حيث يُدفع له أجر ساعة ونصف عن كلّ ساعة عمل تزيد فوق الـ 40 ساعة أسبوعياً.
   - `Commission Employee` وهو موظّف يُدفَع له عمولة تتناسب مع مبيعاته.
   - `Salaried Plus Commission Employee` وهو موظّف يكون له معاش ثابت وعمولة على مبيعاته ويمكن أن يأخذ مكافأة مقدارها 10% من معاشه في بعض الحالات التي تقرّرها الشركة.

2. قم بإنشاء تطبيق برمجي بلغة C# للشركة، بحيث يتمّ تمثيل الموظّف باسمه وكنيته وراتبه، وبحيث يتمّ تخزين الموظّفين ضمن مصفوفة. ويجب أن يقوم التطبيق بالمهام الآتية:
   - إضافة شخص إلى المصفوفة.
   - طباعة معلومات كافّة الموظّفين الذين يزيد دخلهم الأسبوعي عن قيمة معينة.
   - طباعة معلومات كافّة الموظّفين الذين يتقاضون عمولات فقط.
   - طباعة معلومات كافّة الموظّفين بالساعة وطباعة عدد الساعات التي قام بها كل موظّف خلال الأسبوع الماضي.
   - طباعة المبلغ الممثل لمجموع العمولات التي تقاضاها الموظّفون خلال الأسبوع الماضي.

   ثمّ قم بإنشاء الصفّ `Tester` الحاوي على الطريقة `Main` من أجل اختبار التطبيق.

المراجع

1. https://docs.microsoft.com/en-us/dotnet/csharp/
2. "التصميم والبرمجة غرضية التوجّه"، الدكتور سامي خيمي، الإجازة في تقانة المعلومات، من منشورات الجامعة الافتراضية السورية، الجمهورية العربية السورية، 2018.