الواجهات
Interfaces

الكلمات المفتاحية

الوراثة المتعدّدة، الوراثة الأحادية، الواجهة.

ملخص الفصل

خُصّص الفصل الحالي لشرح مفهوم الواجهة ولتوضيح الفرق بينه وبين مفهوم الصفّ المجرّد.

الأهداف التعليمية

يتعرّف الطالب في هذا الفصل على:

- مفهوم الوراثة المتعدّدة
- الواجهات وكيفية تعريفها
- الفارق بين الواجهة والصفّ المجرّد

مقدمة

لا تدعم لغة C# الوراثة المتعدّدة Multiple Inheritance، أي أنّها لا تسمح بأن يرث صفّ ما مباشرة من عدّة صفوف في آن واحد. ولحلّ هذه المسألة، يتمّ استخدام الواجهات Interfaces، حيث يمكن لصفّ ما أن يرث مباشرة عدّة واجهات معاً على الرّغم من عدم السماح له بالوراثة المباشرة إلّا من صفّ واحد فقط، وتُسمّى عملية وراثة صفّ لواجهة بالتنجيز Implementation.

1. مفهوم الواجهة

في النسخ التي تسبق النسخة C# 8.0، الواجهة هي تجمّع منطقي لعدّة سلوكيات مع بعضها البعض، وتتمثّل هذه السلوكيات بالطرائق والخصائص والمفهرسات والأحداث. وتحتوي الواجهة على تواقيع signatures هذه المكونات ونمط الخرج الذي تعيده فقط، ولم يكن من المسموح كتابة كتل التعليمات التي تحدّد دور المكوّنات، كما لم يكن مسموحاً احتواء الواجهة على حقول. ويُترَك للصفّ الذي سيقوم بتنجيزها توصيف مهمّة كلّ مكوّن من مكوّناتها، أي أنّ مهمّة الواجهة تقتصر على الإشارة إلى مهام من دون تحديد كيفية القيام بها. وعندما تقوم عدّة صفوف بتنجيز نفس الواجهة، يحدِّد كلّ منهم الأعمال المختلفة لمكوّناتها وفقاً لاحتياجاته. فيقوم كلّ صف باستكمال جميع المكوّنات غير المكتملة من خلال منح كلّ مكوّن من المكوّنات مجموعة التعليمات الخاصّة به مع الالتزام بالتواقيع التي تفرضها الواجهة.

ابتداءً من النسخة C# 8.0، يُسمَح لمكوّنات الواجهة أن تحتوي على كتل تعليمات، وهذا ما يُسمّى بالتنجيز الافتراضي default implementation، الذ ي يتم استخدامه في حال عدم تنجيز المكوّن المنجز افتراضياً من قِبَل صفّ أو بنية. أصبح من الممكن للواجهات أن تحتوي على: ثوابت ومعاملات وبانٍ ساكن وأنماط متداخلة و مكوّنات ساكنة من حقول وطرائق وخصائص ومفهرسات وأحداث. كما يمكن استخدام الأعضاء بشكل صريح مع اسم الواجهة واستخدام محدّدات وصول بشكل صريح (محدّد الوصول الافتراضي public).

مثال1:

يوضّح المثال التالي كيفية التصريح عن الواجهة IMyInterface وكيفية تنجيزها من قِبَل الصفّ InterfaceImplementer.

```
using System;
namespace Interfaces
{
    interface IMyInterface
    {
        void MethodToImplement(); // Method signature, no assess specifier
                                  // There is no body
    }// End interface IMyInterface

    // Interface Implementation
    class InterfaceImplementer : IMyInterface
    {
        public void MethodToImplement()
        {      // Method body
```

```
            // Method Implementation
            Console.WriteLine("I am the body of MethodToImplement");
        }// End MethodToImplement

        // Main Method
        public static void Main()
        {
            //Creating an object from the class
            InterfaceImplementer iobj = new InterfaceImplementer();
            //IMyInterface iogjI = new IMyInterface();
            iobj.MethodToImplement();
            Console.ReadKey();
        }// End Main
    }// End class InterfaceImplementer
}// End namespace Interfaces
```

وبعد التنفيذ، يظهر الخرج الآتي:

```
I am the body of MethodToImplement
```

بالاطلاع على الرمّاز السابق، يمكن ملاحظة ما يأتي:

- تمّ التصريح عن الواجهة IMyInterface باستخدام الكلمة المفتاحية interface، وتمّ استخدام الحرف I في بداية اسم الواجهة، وهذا الأمر ليس إلزامياً بل اصطلاحاً للدلالة على أنّ الاسم يشير إلى واجهة تحتاج إلى تنجيز أعضائها.
- تمّ الاكتفاء بكتابة توقيع الطريقة MethodToImplement() فقط ضمن الواجهة IMyInterface
- لم يتمّ ذكر أي محدّد وصول إلى الطريقة MethodToImplement() لأنّ جميع عناصر الواجهة لها المحدّد public ضمنياً، وعند محاولة كتابة محدّد وصول في السطر 4، قد نحصل على رسالة خطأ (في بعض النسخ التي تسبق C#7.3).
- في توقيع الطريقة MethodToImplement()، لم يتمّ ذكر أي من الكلمات المفتاحية التالية: static,virtual, abstract, sealed لأنه لا يُسمَح بذلك.
- قام الصفّ InterfaceImplementer بوراثة الواجهة وتنجيز الطريقة MethodToImplement() عن طريق كتابة جسم (body) لها.
- تمّ إنشاء الغرض iobj من الصفّ InterfaceImplementer، وتمّ استخدامه لاحقاً لاستدعاء الطريقة MethodToImplement() المسؤولة عن كتابة النصّ الناتج عن عملية التنفيذ.
- لو حاولنا إنشاء غرض من الواجهة، لحصلنا على رسالة الخطأ التالية:

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

والتي تفيد بأنّه لا يمكن القيام بذلك.

مثال2:

وراثة صفّ لصفّ مجرّد ولواجهتين بشكل مباشر:

```
using System;
namespace InterfaceApplication
{

    abstract class Shape
    {
        protected int side;
        public abstract void SetSide(int s);
    } // End class Shape

    public interface ICost
    {
        int GetCost(int area);
    } // End Interface ICost

    public interface IVolume
    {
        int GetVolume(int volume);
    } // End Interface IVolume

    // Derived class
    class Square : Shape, ICost, IVolume
    {
        public override void SetSide(int s) { side = s; }
        public int GetArea() { return (side * side); }
        public int GetCost(int area) { return area * 10; }
        public int GetVolume(int area) { return area * side; }
    } // end class Square
```

```
    class Tester
    {
        static void Main()
        {
            Square sq = new Square();
            int area;
            sq.SetSide(5);
            area = sq.GetArea();
            // Print the area of the object.
            Console.WriteLine("The area is: {0}", sq.GetArea());
            Console.WriteLine("The cost is: {0}", sq.GetCost(area));
            Console.WriteLine("The volume is: {0}", sq.GetVolume(area));
            Console.ReadKey();
        } // end Main
    } // End class Tester
}//End namespace Application
```

يقوم الصفّ Square بوراثة الصفّ المجرّد Shape الحاوي على الطريقة المجرّدة SetSide ويقوم بتنجيزها بحيث تُسند قيمة مدخلة لها إلى الحقل side. ويرث الصفّ Square أيضاً الواجهة ICost الحاوية على توقيع الطريقة GetCost والواجهة IVolume الحاوية على توقيع الطريقة GetVolume، ويقوم بتنجيز كلتا الطريقتين GetCost وGetVolume.

وفي الطريقة Main التابعة للصفّ Tester، تمّ استخدام الغرض sq من الصفّ Square لاستدعاء الطريقة SetSide من أجل القيمة (5)، وتمّ استخدام الطريقة GetArea لحساب المساحة area. وبعد التنفيذ، ظهر الخرج الآتي:

```
The area is: 25
The cost is: 250
The volume is: 125
```

مثال3:

على الرغم من أنّ لغة C# لا تسمح بعلاقة وراثة بين البنى، إلّا أنّه يمكن لبنية struct أن تقوم بتنجيز واجهة أو أكثر، ويشكل الرمّاز الآتي أحد الأمثلة على ذلك:

```
using System;
namespace Interfaces
{
    interface ISampleInterface
    {
        // string lastName;// Instance fields are not permitted
        static string name; // static field only
        public void SampleMethod();// Using explicit access specifier
        public string GetName()
        {
            return ("\n\nThis is the default
       implementation of GetName() done by the interface,\nThe name is: "
                           + name);
                       }
}
struct ImplementationStruct : ISampleInterface
{
    // Struct constructor
    public ImplementationStruct(string str)
    {
        ISampleInterface.name = str; //Explicit interface member
    }
    // Explicit interface member implementation:
    void ISampleInterface.SampleMethod()
    {
        // Method implementation.
        Console.WriteLine("This is the implementation done by

         the struct, \nThe name is: " + ISampleInterface.name);
        }
        // Implementing the interface concrete method
     /* public string GetName() { return ("\nThis is the new implementation
       of GetName() " + "\nHello " + ISampleInterface.name); }
     */
```

```
    }// end struct ImplementationStruct

// defining the class Tester having the Main method

class Tester
{
    static void Main()
    {
        // Declaring an interface instance
        ISampleInterface itfs = new ImplementationStruct("My name");

        // Calling the member
        itfs.SampleMethod();
        Console.WriteLine(itfs.GetName());
        Console.ReadKey();
    }// end Main
}// end class Tester
```

بعد تنفيذ الرمّاز السابق، نحصل على الخرج الآتي:

```
This is the implementaion done by the struct,
The name is: My name

This is the default implementaion of GetName() done by the
interface,
The name is: My name
```

نقوم بتنجيز الطريقة GetName ضمن البنية، ونعيد تنفيذ البرنامج فنحصل على الخرج الآتي:

```
This is the implementaion done by the struct,
The name is: My name

This is the new implementaion of GetName()
Hello My name
```

نلاحظ أنّ التنجيز الجديد ألغى التنجيز الافتراضي.

وعلى الرّغم من أنّه أصبح بالإمكان للواجهة أن تضمّ حقولاً ساكنة، لا يمكن لها احتواء حقولٍ منتسخة (غير ساكنة).

2. التجريد والواجهات

يمكن اعتبار الواجهة صفّاً جميع عناصره مجرّدة، أي أنّها أكثر تجريداً من الصفّ المجرّد نفسه. ولكن لا يتمّ استخدام الكلمة المفتاحية `abstract` عند التصريح عنها أو عن أيٍ من مكوّناتها. ولتوضيح الفارق بين المفهومين، نذكر فيما يأتي أوجه الشبه ونقاط الاختلاف بينهما:

- كلاهما يحتوي على عناصر مجرّدة.
- يحتوي الصفّ المجرّد على عضو مجرّد واحد أو أكثر، ويمكن للصفّ المجرّد ألا يحوي أي عضوٍ مجرّد.
- جميع عناصر الواجهة مجرّدة، ولكن يمكن في بعض الأحيان أن تحوي عناصر لها تنجيز افتراضي يمكن استخدامه عند عدم توافر تنجيز في الصفّ الوارث لها.
- يجب استخدام الكلمة المفتاحية `abstract` مع اسم الصفّ المجرّد ومع اسم أي عضو مجرّد فيه، أمّا عناصر الواجهة فهي مجرّدة ضمنياً، ولا يجب ذكر الكلمة المفتاحية `abstract` مع اسمها أو مع أي من أعضائها.
- يمكن للواجهة أن ترث واحدة أو أكثر من الواجهات.
- يمكن للصفّ المجرّد أن يرث صفاً مجرّداً آخر.
- يمكن للواجهة أن تتمّ وراثتها من قبل `struct` ولكن لا يمكن وراثة صفّ مجرّد من قِبَل `struct`.
- يمكن لصفّ ما أن يرث عدّة واجهات ولكن لا يمكنه وراثة إلّا صفّ واحد (قد يكون مجرّداً أم لا)، وفي حال وراثة صفّ واحد وعدّة واجهات، يجب ذكر اسم الصفّ في بداية التصريح عن الوراثة ثمّ أسماء الواجهات.
- يمكن للواجهة أن تحتوي على بناة وحقول ساكنين `static`، ولا يمكنها احتواء حقول منتسخة.
- لا يمكن للواجهة أن تحتوي على بناة غير ساكنين ولا على هادمين، بينما الصفّ المجرّد فيمكنه احتواء بناة غير ساكنين وهادمين.

3. التنجيز الضمني والتنجيز الصريح

عندما يقوم صفّ ما بوراثة واجهتين تمتلكان طريقتين متطابقتين (لهما نفس التوقيع)، نكون أمام خيارين:

- إمّا أن يقوم الصفّ بتنجيز طريقة واحدة فقط لأنّه يرغب في أن تسلك الطريقتين في الواجهتين نفس السلوك، وعندها نسمّي التنجيز ضمنياً Implicit Implementation.
- أو أن يقوم الصفّ بتنجيز الطريقتين لأنّه يرغب في أن تسلك كلّ طريقة سلوكاً مختلفاً عن الأخرى، وعندها نسمّي التنجيز صريحاً Explicit Implementation، ولا بدّ في هذه الحالة من أن يُسبَق اسم كلّ طريقة باسم الواجهة التي تنتمي إليها لإزالة الغموض الذي قد يقع أثناء عملية الترجمة.

مثال

```
using System;
namespace ImplicitExplicit
{
    interface IControl
    {
        void Paint();
    }// End interface IControl

    interface ISurface
    {
        void Paint();
    }// End interface ISurface

    class SampleClass : IControl, ISurface
    {
        // Both ISurface.Paint and IControl.Paint call this method.
        public void Paint()
        {
            Console.WriteLine("Paint method in SampleClass");
        }// End Paint
    }// End class SampleClass

    class Test
    {

        static void Main()
```

```
        {
            SampleClass sc = new SampleClass(); // Creating sc object
            IControl ctrl = sc;                 //creating interface ctrl
            ISurface srfc = sc;                 // creating interface srfc
                                                // The following lines all call
the same method.
            sc.Paint();
            ctrl.Paint();
            srfc.Paint();
            Console.ReadKey();
        }// End Main
    }// End class Test
}// End namespace ImplicitExplicit
```

نلاحظ أنّ توقيع الطريقة Paint() في الواجهة IControl يتطابق مع توقيع الطريقة Paint() في الواجهة ISurface، وأنّ الصفّ SampleClass يرث الطريقتين ولكنّه قام بتنجيز طريقة واحدة فقط من دون تحديد تبعيّة هذه الطريقة، ولذلك يمكن اعتبار أنّ هذا التنجيز يتبع ضمنياً لكلتا الواجهتين. وللتحقّق من أنّ للطريقتين نفس السلوك في كلتا الواجهتين، ننشئ الغرض sc من الصفّ SampleClass، ثمّ نستخدم هذا الغرض لإنشاء الواجهة ctrl من النمط IControl والواجهة srfc من النمط ISurface. وبعد ذلك، نستدعي الطريقة Paint()، مرّة باستخدام الغرض sc ومرّة باستخدام الواجهة ctrl وأخرى باستخدام الواجهة srfc. وبعد التنفيذ، نحصل على الخرج الآتي:

```
Paint method in SampleClass
Paint method in SampleClass
Paint method in SampleClass
```

أي أنّه سواء استدعينا الطريقة Paint() باستخدام الغرض sc أو أي من الواجهتين ctrl, srfc سنحصل على نفس النتيجة لأنّ هذه الطريقة تمتلك سلوكاً واحداً فقط.

لنقم ببعض التعديلات على المثال السابق ونستبدل الصفّ SampleClass (الأسطر من 11 وحتى 17) بما يأتي:

```
class SampleClass : IControl, ISurface
{
    void IControl.Paint()
    {
        System.Console.WriteLine("Paint from IControl");
    }
    void ISurface.Paint()
    {
        System.Console.WriteLine("Paint from ISurface");
    }
}// End class SampleClass
```

لقد قام الصفّ بتنجيز الطريقة Paint مرّتين وقام بذكر اسم الواجهة التي تتبع لها الطريقة في كلّ مرّة بشكل صريح

وعند معاودة تنفيذ البرنامج سنحصل على رسالة الخطأ التالية الناتجة عن التعليمة ;()sc.Paint.

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

أي أنّه في حالة التنجيز الصريح لطريقة، لا يمكن استخدام غرض من الصفّ الذي قام بالتنجيز مباشرة لاستدعاء الطريقة، ولذلك نقوم بحذف التعليمة المسببة للخطأ، ونعيد تنفيذ البرنامج فتختفي رسالة الخطأ وتظهر العبارات الآتية على الشاشة:

```
Paint from IControl
Paint from ISurface
```

وفي كثير من الحالات، قد تحدث أخطاء غير متوقّعة تؤدّي إلى توقف البرنامج عن التنفيذ على الرّغم من عدم وجود أي خطأ قواعدي في كتابة رمّازه المصدري. ويعود السّبب في ذلك لحدوث حالات استثنائية أثناء التشغيل مرتبطة في أغلب الأحيان بالبيئة التي يتمّ تنفيذ البرنامج فيها. فيمكن أن يحدث انقطاع الاتصال مع أحد المخدّمات، أو فشل الاتصال بقاعدة معطيات، أو فقدان أحد الملفات اللازمة لعمل البرنامج. ويجب على المبرمج

أن يتنبّأ بمثل هذه الحالات الاستثنائية ويقوم بمعالجتها. ويوضّح الفصل القادم أهمّ الاستثناءات التي قد تحدث وكيفية معالجتها.

الأنشطة المرافقة

تُستخدَم الواجهة ICarbonFootprint مع صفوف مختلفة تتميّز أغراضها بأنّها تقوم بتلويث الهواء بغاز الكربون. قم بتعريف الواجهة ICarbonFootprint والتي تحوي الطريقة المجرّدة GetCarbonFootprint لحساب معدّل الكربون المنبعث من أغراض من الصفوف: بناء Building، سيارة Car، سفينة Boat. وقم بكتابة الطريقة GetCarbonFootprint لكلّ من هذه الصفوف والتي تضمّ حسابات بسيطة على بعض الخصائص مثل معدّل استهلاك الوقود. ثمّ أنشئ الصفّ Tester الحاوي على الطريقة Main من أجل اختبار ما قمت بتعريفه.

المراجع

1. https://docs.microsoft.com/en-us/dotnet/csharp/.
2. Dan Clark: Beginning C# Object-Oriented Programming, Berkeley, CA, Apress, 2013.
3. "التصميم والبرمجة غرضية التوجّه"، الدكتور سامي خيمي، الإجازة في تقانة المعلومات، من منشورات الجامعة الافتراضية السورية، الجمهورية العربية السورية، 2018.