الوراثة
Inheritance

العنوان	رقم الصفحة
مقدمة	3
1. مفهوم الوراثة	4
2. الوراثة ومحددات الوصول	9
3. الطرائق الافتراضية والتجاوز	13
4. هرمية الوراثة	16
5. تجاوز الطريقة ToString	21
6. ملاحظات يجب توخيها عند استخدام مفهوم الوراثة	22
7. الأنشطة المرافقة	23

الكلمات المفتاحية

الصف الأساس، الصف المشتق، الطريقة الافتراضية، تجاوز الطرائق، الوراثة الأحادية.

ملخص الفصل

نوضح في هذا الفصل مفهوم الوراثة الذي يُعتبر من أهم أساسيات البرمجة غرضية التوجّه، والذي يدعم إعادة استخدام الرمّاز المصدري. ونبين العلاقات الممكنة بين أعضاء صفّ ما وأعضاء الصفوف التي ترتبط معه بعلاقة وراثة.

الأهداف التعليمية

يتعرّف الطالب في هذا الفصل على:

- مفهوم الصف الأساس والصف المشتق
- معامل الوصول protected
- استخدام البناة والهادمين أثناء الوراثة
- استخدام الكلمة المفتاحية base
- مفهوم الطريقة الافتراضية والتجاوز
- استخدام الطريقة ToString
- الفارق بين الوراثة المتعدّدة والوراثة الأحادية

مقدمة

تهدف الوراثة Inheritance إلى إعادة استخدام الرمّاز المصدري لصف ما (يُسمّى الصف الأب) من خلال إنشاء صف جديد (يُسمّى الصف الابن) يتمتّع بكافة مواصفات الصف الأب ويقوم بتخصيصها و/أو الإضافة عليها. وبذلك يمكن اختصار الزمن اللازم لإنشاء البرمجيات من خلال الاستفادة من مكوّنات برمجية متوفّرة لدينا.

1. مفهوم الوراثة

الوراثة هي علاقة أب/ابن بين صفين، يُسمّى الصف الأب فيها بالصف الأساس Base Class، بينما يُسمّى الصف الابن بالصف المشتقّ Derived Class. وعند تعريف أي صف فهو يرث ضمنياً الصف Object الذي يمثّل الجد الأكبر لعائلة الصفوف في منصة .NET. ومن ضمنها لغة C#، فالرمّاز المصدري:

```
class Shape
{
    public Shape()
    {
        System.Console.WriteLine("It is the Shape’s constructor");
    }
}
```

مكافئ للرمّاز المصدري الآتي:

```
class Shape : Object
{
    public Shape()
    {
        System.Console.WriteLine("It is the Shape’s constructor");
    }
}
```

حيث يشير المحرف (:) لعملية الوراثة. وعند كتابة صفٍ جديد، وعوضاً عن كتابة كامل عناصره، يمكن القيام بوراثة بعض العناصر من صف آخر

لإيضاح مفهوم الوراثة نعرف شكل الدائرة في مثال1 باستخدام هذا المفهوم ثم نعيد كتابة الرماز في مثال2 من دون استخدام هذا المفهوم.

مثال1:

```
using System;
// base class
class Shape
{
    protected int dimension;
    public Shape()
    {
        dimension = 1;
        Console.WriteLine("The Shape's Constructor");
    }
    public void SetDimention(int d) { dimension = d; }
    ~Shape()
    {
        Console.WriteLine("Executing Destructor of class Shape");
    }
}// end class Shape

// derived class (inherits from Shape, extends Shape)
class Circle : Shape
{
    int radius;
    public Circle()
    {
        radius = 1;
        Console.WriteLine("The Circle's Constructor");
    }
    public void SetRadius(int r) { radius = r; }
    public void WriteDimesion()
    {
        Console.WriteLine("The Circle's dimension is " + dimension);
    }
    ~Circle()
    {
        Console.WriteLine("Executing Destructor of class Circle");
    }
}// end class Circle
```

```
class Tester
{
    public static void Main()
    {
        // creating cr object, an instance from Circle
        Circle cr = new Circle();
        //Calling SetDimention() method
        cr.SetDimention(2);
        //Calling WriteDimesion() method
        cr.WriteDimesion();
        //Releasing Object cr
        cr = null;
        GC.Collect();
    }
}// end class Tester
```

يرث الصف Circle الحقل dimension والطريقة SetDimention من الصف Shape، ويضيف عليهما الحقل radius والطريقة SetRadius والطريقة WriteDimesion.

مثال2:

من دون استخدام مفهوم الوراثة، يتحتّم على المطوّر نسخ كافّة عناصر الصف Shape (طبعاً باستثناء الباني والهادم) ولصقها ضمن الصف Circle.

```
using System;
// base class
class Shape
{
    protected int dimension;
    public Shape()
    {
        dimension = 1;
        Console.WriteLine("The Shape's Constructor");
    }
    public void SetDimention(int d) { dimension = d; }
    ~Shape()
    {
        Console.WriteLine("Executing Destructor of class Shape");
    }
}// end class Shape

// class Circle without using inheritance
class Circle
{
    int dimension;
    int radius;
    // contractor
    public Circle()
    {
        radius = 1;
        System.Console.WriteLine("The Circle's Constructor");
    }// end contractor
    public void SetDimention(int d) { dimension = d; }
    public void SetRadius(int r) { radius = r; }
    public void WriteDimesion()
    {
        System.Console.WriteLine("The Circle's dimension is " + dimension);
    }// end WriteDimesion
```

```
     // destructor
    ~Circle()
    {
        Console.WriteLine("Executing Destructor of class Circle");
    }// end destructor
}// end class Circle

class Tester
{
    public static void Main()
    {
        // creating cr object, an instance from Circle
        Circle cr = new Circle();
        //Calling SetDimention() method
        cr.SetDimention(2);
        //Calling WriteDimesion() method
        cr.WriteDimesion();
        //Releasing Object cr
        cr = null;
        GC.Collect();
    }
}// end class Tester
```

ولكن يجب دوماً تجنّب هذه الطريقة لأنّ وجود نسختين أو أكثر من نفس الرمّاز المصدري سيجعل مسألة صيانة الرمّاز وتعديله مسألة صعبة ومحفوفة بالخطر، فقد يقوم مطوّر الرمّاز بتعديل إحدى نسخ طريقة ما وينسى تعديل نسخة الطريقة الموجودة في صف آخر، وهذا ما يجعل نفس الطريقة تمتلك سلوكين مختلفين عن طريق الخطأ.

2. الوراثة ومحدّدات الوصول

لقد تعرّفنا سابقاً على محدّد الوصول private الذي يُستخدم لحماية أحد عناصر الصف من الوصول إليه من قبل عناصر الصفوف الأخرى، وتعرّفنا على محدّد الوصول public الذي يشير إلى إمكانية استخدام العنصر من قبل عناصر الصف الذي ينتمي إليه العنصر ومن قبل عناصر الصفوف الأخرى التي يمكنها الوصول إلى صفه. ولكنّنا نحتاج في بعض الأحيان إلى حماية عنصر ما من الاستخدام من قبل جميع الصفوف مع المحافظة على إمكانية وراثته، ولحلّ هذا الأمر وفّرت لغة C# (كمثيلاتها من لغات البرمجة غرضية التوجّه) محدّد الوصول protected الذي يسمح باستخدام العنصر من قبل عناصر الصف الذي ينتمي إليه العنصر ومن قبل عناصر الصفوف المشتقة من هذا الصف.

مثال:

في مثال1 من الفقرة السابق، يتضمّن الصف shape الحقل dimension الذي محدّد الوصول إليه protected فلو تمّ حذف هذه الكلمة لأصبح محدّد الوصول إليه هو محدّد الوصول الافتراضي للحقل أي private، وهذا يعني عدم تمكّن عناصر الصف Circle من الوصول إلى هذا الحقل والتعامل معه.

بالعودة إلى الرمّاز السابق وبعد حذف كلمة protected منه والقيام بمحاولة تنفيذه سنحصل على رسالة الخطأ الآتية ولن يتمّ التنفيذ:

```
'Shape.dimension' is inaccessible due to its protection level
```

توضّح الرسالة أنّ الحقل dimension غير قابل للوصول إليه بسبب مستوى الحماية الذي يتمتّع به وتتمّ الإشارة إلى مكان الخطأ في السطر التالي:

```
System.Console.WriteLine("The Circle's dimension is " + dimension);
```

وبالنسبة لعناصر الصف الأساس التي محدّد الوصول إليها public، يبقى محدّد الوصول إليها public في الصف المشتقّ.

1.2. استخدام البناة والهادمين في أثناء الوراثة

أثناء استخدام الوراثة يمكن أن يحتوي كل من الصف الأساس والصف المشتقّ على بناةٍ وهادمين. ويجب التنويه إلى أنّه لا يتمّ توريث البناة، أي أنّ الصف Circle لا يرث باني Shape بل يمتلك بانٍ خاص.
أيضاً لا يتمّ توريث الهادمين. ويتمّ استدعاء الهادمين بشكل معاكس لاستدعاء البناة.

مثال:

في الرماز التالي المستخدم في مثال 1:

```
using System;
// base class
class Shape
{
    protected int dimension;
    public Shape()
    {
        dimension = 1;
        Console.WriteLine("The Shape's Constructor");
    }
    public void SetDimention(int d) { dimension = d; }
    ~Shape()
    {
        Console.WriteLine("Executing Destructor of class Shape");
    }
}// end class Shape

// derived class (inherits from Shape, extends Shape)
class Circle : Shape
{
    int radius;
    public Circle()
    {
        radius = 1;
        Console.WriteLine("The Circle's Constructor");
    }
    public void SetRadius(int r) { radius = r; }
    public void WriteDimesion()
    {
        Console.WriteLine("The Circle's dimension is " + dimension);
```

```
    }
    ~Circle()
    {
        Console.WriteLine("Executing Destructor of class Circle");
    }
}// end class Circle

class Tester
{
    public static void Main()
    {
        // creating cr object, an instance from Circle
        Circle cr = new Circle();
        //Calling SetDimention() method
        cr.SetDimention(2);
        //Calling WriteDimesion() method
        cr.WriteDimesion();
        //Releasing Object cr
        cr = null;
        GC.Collect();
    }
}// end class Tester
```

يحتوي الصف Tester على الطريقة Main التي يتمّ فيها إنشاء الغرض cr من الصف Circle واستدعاء الطريقة WriteDimesion. وعند التنفيذ، سيتمّ استدعاء الباني الافتراضي للصف الأساس shape ثم باني الصف المشتقّ Circle من أجل إنشاء الغرض cr. وبما أنّ الغرض cr هو غرض من الصف Circle وكذلك هو غرض من الصف Shape في نفس الوقت، لذلك يمكن استخدامه لاستدعاء الطريقة SetDimention() كما يلي:

```
cr.SetDimention(2);
```

ثمّ يتمّ تنفيذ الطريقة WriteDimesion() التي تطبع القيمة الجديدة للحقل dimension، وبعد ذلك يتمّ تحرير الغرض cr عن طريق إسناد القيمة null إليه واستدعاء الطريقة Collect من جامع النفايات GC (GarbageCollector). وبعد تنفيذ الرمّاز السابق تظهر النتيجة:

```
The Shape's Constructor
The Circle's Constructor
The Circle's dimension is 2
Executing Destructor of class Circle
Executing Destructor of class Shape
Press any key to continue...
```

نلاحظ في نتيجة التنفيذ، أنّه يتمّ استدعاء الهادمين بشكل معاكس لاستدعاء البناة. فعند تحرير الغرض cr، يتمّ أولاً استدعاء الهادم ~Circle الخاصّ بالصف المشتقّ Circle ثم استدعاء الهادم ~Shape الخاصّ بالصف الأساس Shape.

2.2. الباني الخاصّ والوراثة

إذا امتلك الصف بانٍ خاصّ private constructor فقط من دون امتلاك أي بانٍ آخر، فلا يمكن أن يرثه أي صف آخر. ويُستخدم الباني الخاصّ في الحالات التي لا نرغب بإنشاء أغراض من الصف الحاوي له خارج الصف. ونوضّح كيفية استخدامه في المثال الآتي:

```
class NoInstance
{
    // Private Constructor:
    private NoInstance() { }

    public static double pi = Math.PI;  //3.1415...
}//end class NoInstance
```

إنّ إضافة بانٍ من دون أي تعليمة، يمنع استخدام الباني الافتراضي الذي يتمّ إنشاؤه تلقائياً عند عدم تعريف بانٍ للصف. وعلى الرغم من اعتبار محدّد الوصول للباني هو private في حالة عدم كتابته، إلا أنّه من الأفضل كتابة محدّد الوصول private بشكل صريح.

ويُمكن استخدام الباني الخاصّ في حالة احتواء الصف على أعضاء ساكنة فقط، وفي مثل هذه الحالات من الأفضل التصريح عن الصف ككلّ أنّه ساكن.

3. الطرائق الافتراضية والتجاوز

الطريقة الافتراضية Virtual Method هي إحدى طرائق الصف الأساس التي يُسمح بتجاوزها Overriding (أي تغيير سلوكها) من قبل طريقة أخرى في الصف المشتقّ على أن يكون للطريقتين نفس التوقيع. يتمّ استخدام الكلمة virtual في توقيع الطريقة كمحدّد للإشارة على أنّها افتراضية، ولا يُستخدم المحدّد virtual برفقة أي من المحدِّدات: abstract وprivate وoverride وstatic.

مثال:

في الرمّاز التالي، نعرِّف ما يلي:

1. الصف Rectangle (مستطيل) ويحتوي على الحقلين length وwidth والطريقة GetArea التي تعيد مساحة المستطيل المساوية لجداء قيمتي الحقلين السابقين والطريقة الافتراضية Display التي تكتب على الشاشة قيمتي الحقلين ونتيجة استدعاء الطريقة السابقة.
2. الصف TableTop (غطاء الطاولة) المشتقّ من الصف Rectangle ويمتلك، إضافة للعناصر الموروثة، الحقل priceOneMeter المعبِّر عن سعر واحدة المساحة، والطريقة GetCost التي تعيد تكلفة غطاء الطاولة، والطريقة Display التي تجاوزت الطريقة الافتراضية في الصف الأساس والتي تحمل نفس التوقيع وتقوم هذه الطريقة بالطباعة على الشاشة معلومات الغرض الذي يقوم باستدعائها.
3. الصف Tester الذي يحتوي على الإجرائية Main التي يتمّ ضمنها إنشاء غرض من الصف TableTop باستخدام الباني الخاص به والذي يأخذ كوسائط دخل ثلاث قيم من أجل تهيئة الحقول الثلاثة length وwidth وpriceOneMeter.

```
using System;
namespace RectangleApplication {
class Rectangle {
    //member variables
    protected double length;
    protected double width;
    public Rectangle(double l, double w) {
    length = l;
    width = w;
        }
public double GetArea() {
    return length * width;
}
public virtual void Display() {                 // virtual method
    Console.WriteLine("Length: {0}", length);
```

```
    Console.WriteLine("Width: {0}", width);
    Console.WriteLine("Area: {0}", GetArea());
    }
}//end class Rectangle

 class Tabletop : Rectangle {
    private double priceOneMeter = 1;
    public Tabletop(double l, double w, double prc) : base(l, w) {
        priceOneMeter = prc;
    }
    public double GetCost( ) {
    double cost;
    cost = GetArea() * priceOneMeter;
    return cost;
        }
public override void Display() {              //overridden method
    //use base display and then show more data
    base.Display();
    Console.WriteLine("Cost: {0}", GetCost());
    }
}// end class Tabletop
class Tester  {
        static void Main( )  {
            Tabletop t = new Tabletop(4.5, 7.5, 10.0);
            t.Display();
            Console.ReadKey();
        }// end Main
    }// end class Tester
}//end namespace
```

نلاحظ أنّ باني الصف المشتقّ TableTop يستدعي باني الصف الأساس من خلال الكلمة المفتاحية base:

```
public Tabletop(double l, double w, double prc) : base(l, w) {
    priceOneMeter = prc;
}
```

ويقوم بتمرير قيم وسائط الدخل التي تُمرَّر إليه ثمّ يقوم بتهيئة الحقل الخاصّ به priceOneMeter باستخدام قيمة مُدخلة له.

وكذلك تقوم الطريقة المتجاوزة Display في الصف المشتقّ باستدعاء الطريقة الافتراضية Display من الصف الأساس باستخدام الكلمة المفتاحية base على النحو الآتي:

```
1.base.Display();
```

وعند تنفيذ الرمّاز المصدري السابق سنحصل على الخرج الآتي:

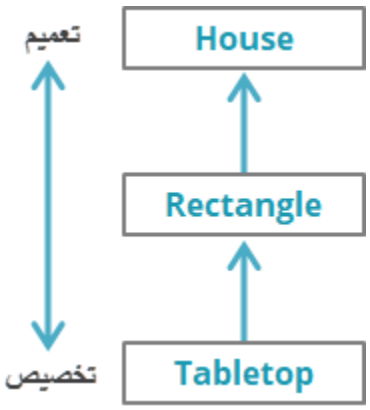
```
Length: 4.5
Width: 7.5
Area: 33.75
Cost: 337.5
Press any key to continue...
```

فيما يلي بعض الملاحظات حول تجاوز الطرائق

- يجب أن يكون للطريقة المتجاوِزة في الصف المشتقّ والطريقة التي تم تجاوزها في الصف الأساس نفس التوقيع أي نفس وسائط الدخل من حيث العدد والأنماط ونفس الاسم.
- لا يغير التجاوز محدد الوصول إلى الطرائق التي يتم تجاوزها.
- يمكن تجاوز الطرائق الافتراضية virtual والمجردة abstract والمتجاوِزة override.
- لا يمكن تجاوز الطرائق الساكنة static methods.
- لا يمكن تجاوز الطرائق الخاصّة (التي محدّد الوصول إليها private).
- يمكن تجاوز الخصائص properties والمفهرسات indexers (سيتمّ شرح المفهرسات في فصل قادم).

4. هرمية الوراثة

يمكن للصف المشتقّ أن يكون صفاً أساساً لصفوف أخرى، فالصف Rectangle هو صف أساس للصف Tabletop وهو في نفس الوقت صف مشتقّ من الصف Shape.



نسمّي الصف Rectangle بالصف الأساس المباشر Direct Base Class للصف Tabletop، ونسمّي الصف Shape بالصف الأساس غير المباشر Indirect Base Class.

وفي لغة C#، يمكن للصف الأساس أن يكون له أي عدد من الصفوف المشتقة، ولكن لا تسمح هذه اللغة بالوراثة المتعدّدة Multiple Inheritance. أي لا يمكن لصف مشتقّ أن يكون له أكثر من صفٍ أساس مباشر واحد، وهذا ما نسمّيه بالوراثة الأحادية Single Inheritance.

وعند تصميم الوراثة بين الصفوف، عادة ما يكون الصف الأساس أكثر عمومية من الصف المشتقّ مباشرة منه، والذي بدوره يكون أكثر عمومية من الصف المشتقّ مباشرة منه، وهكذا دواليك. أي كلّما انتقلنا في هرمية الوراثة عمقاً باتجاه الصفوف المشتقة كلما ازداد التخصيص، ويزداد التعميم في الاتجاه المعاكس.

ويرتبط الصف المشتقّ مع صف أساس يرث منه بالعلاقة "is-a"، والتي تعني أن أي غرض من الصف المشتقّ هو غرض من الصف الأساس.

مثال:

في الرمّاز الآتي، نعرِّف الصف Person الحاوي على الأعضاء الآتية:

- الحقل المحميّ age
- الخاصّية المحميّة Name
- الطريقة العامة SetAge
- الطريقة العامة WelocmMessage لطباعة رسالة ترحيب متضمنة معلومات الغرض الذي يستدعيها

ونعرِّف الصف Teacher المشتقّ من الصف Person ويحتوي على:

- الحقل الخاصّ subject
- بانٍ ذي ثلاثة وسائط دخل
- الطريقة العامة PrintInfo لطباعة معلومات الغرض الذي يستدعيها

```
using System;
namespace Inheritance
{
    using System;
    class Person
    {
        protected int age;
        protected string Name { set; get; }
        //method SetAge for updating the age value
        public void SetAge(int n)
        {
            age = n;
        }// end SetAge
        //method Great for printing Hello message on the screen
        public void WelocmMessage()
        {
            Console.WriteLine("Hello, I am a {0} ,  ", this.ToString());
            Console.WriteLine("My name is {0}, my age is {1} ", Name, age);
        }// end WelocmMessage
    }// end class Person

    //class Teacher which is derived from class Person
    class Teacher : Person
    {
        private string subject;
        // constructor
        public Teacher(int age, string sub, string name)
        {
            base.age = age;
            subject = sub;
            Name = name;
        }// end class constructor
        public void PrintInfo()
```

```
        {
            Console.WriteLine("Hello! I am a {0} , Teacher of {1}  ",
this.ToString(), subject);
            Console.WriteLine("My name is {0}, my age is {1} ", Name, age);
        }
    }// end class Teacher
    //create a test class called "StudentAndTeacherTest" that will contain "Main"
    class StudentAndTeacherTest
    {
        static void Main()
        {
            // create a Teacher (derived class object)
            // age = 23, subject = OOP, name = Saly
            Teacher myTeacher = new Teacher(23, "OOP", "Saly");
            // create a Person (base class object )
            Person p1 = new Person();
            // assign the Teacher object to the Person object
            // we can dot it because a Teacher is a Person
            p1 = myTeacher;
            // change the age of the Teacher object by changing the age Person
object
            p1.SetAge(30);
            // call WelocmMessage frrom Person object
            Console.WriteLine("\nCalling Person WelocmMessage:");
            p1.WelocmMessage();
            // call WelocmMessage from Teacher object
            Console.WriteLine("\nCalling Teacher WelocmMessage:");
            myTeacher.WelocmMessage();
            // call PrintInfo
            Console.WriteLine("\nCalling Teacher PrintInfo:");
            myTeacher.PrintInfo();
            // create a Person (base class object ) and
            // assign new Teacher (derived class object) to it
            Person p2 = new Teacher(25, "Math", "Khaled");
            Console.WriteLine("\nCalling Person WelocmMessage:");
            p2.WelocmMessage();
            // myTeacher = p2; //Not accepted,
            /*Error CS0266: Cannot implicitly convert type
'InheritanceHierarchy.Person'
             * to 'InheritanceHierarchy.Teacher'. An explicit conversion exists
```

```
            * (are you missing a cast?)
            */
        }// end Main
    }// end class StudentAndTeacherTest
}// end namespace Inheritance
```

نقوم بإنشاء الغرض myTeacher من الصف Teacher باستخدام باني الصف، ثمّ نقوم بإنشاء الغرض p1 من الصف Person، ونسند له الغرض myTeacher، وهذا ممكن لأنّ كل غرض من الصف Teacher هو غرض من الصف Person. وهو نفس السبب الذي يسمح لنا بإنشاء الغرض p2 من الصف Person باستخدام باني الصف Teacher. ثمّ نقوم بإسناد الغرض p2 إلى الغرض myTeacher، ونحاول التنفيذ فنحصل على رسالة الخطأ الآتية التي تفيد بأنّه لا يمكن التحويل ضمنياً بين نمط المعطيات Person ونمط المعطيات Teacher:

```
Cannot implicitly convert type 'Inheritance.Person' to 'Inheritance.Teacher'. An explicit conversion exists (are you missing a cast?)
```

وللقيام بعملية الإسناد السابقة، يجب القيام بتحويل قسري Casting بين نمط المعطيات Person ونمط المعطيات Teacher. ولذلك، نضيف التعليمات الآتية إلى نهاية الطريقة Main:

```
myTeacher = (Teacher)p2;// accepted
Console.WriteLine("\nCalling Teacher WelocmMessage:");
myTeacher.WelocmMessage();
```

وعند إعادة التنفيذ، نحصل على الخرج الآتي:

```
Calling Person WelocmMessage:
Hello, I am a Inheritance.Teacher ,
My name is Saly, my age is 30

Calling Teacher WelocmMessage:
Hello, I am a Inheritance.Teacher ,
My name is Saly, my age is 30

Calling Teacher PrintInfo:
Hello! I am a Inheritance.Teacher , Teacher of OOP
My name is Saly, my age is 30

Calling Person WelocmMessage:
```

```
Hello, I am a Inheritance.Teacher ,
My name is Khaled, my age is 25
```

5. تجاوز الطريقة ToString

توجد الطريقة ToString مع جميع الأغراض، وتُستخدم لإرجاع سلسلة محرفية string تضمّ معلومات عن الغرض الذي يستدعيها. وفي حال لم يتمّ تعديل سلوكها، تعيد الطريقة ToString الاسم الكامل لنمط الغرض الذي يستدعيها. ولتوضيح طريقة استخدامها.

مثال:

نعدِّل الرمّاز السابق بحيث نقوم بتجاوز الطريقة ToString لتعيد معلومات الغرض المطلوب طباعتها، ونستدعيها في جسم الطريقة PrintInfo كما يأتي:

```
public override string ToString()
{
    return string.Format("Hello! I am a {0} ,  my age is {1}, Teacher of {2}.",
Name, age, subject);
}
public void PrintInfo()
{
    Console.WriteLine(this.ToString());
}
```

وبعد إعادة التنفيذ، يتمّ الحصول على الخرج:

```
Calling Person WelocmMessage:
Hello, I am a Hello! I am a Saly ,  my age is 30, Teacher of OOP. ,
My name is Saly, my age is 30

Calling Teacher WelocmMessage:
Hello, I am a Hello! I am a Saly ,  my age is 30, Teacher of OOP. ,
My name is Saly, my age is 30

Calling Teacher PrintInfo:
Hello! I am a Saly ,  my age is 30, Teacher of OOP.

Calling Person WelocmMessage:
Hello, I am a Hello! I am a Khaled ,  my age is 25, Teacher of Math.
,
My name is Khaled, my age is 25
```

6. ملاحظات يجب توخيها عند استخدام مفهوم الوراثة

بالرغم من الفوائد العديدة للوراثة كإعادة استخدام الرمّاز المصدري واختصار حجمه، إلا أنّها فرضت على المبرمجين بعض الأمور التي يجب مراعاتها، وتتلخّص بما يأتي:

- يجب توخّي الحذر عند إجراء أي تغيير على أي صف، فيجب التأكد من أنّ التغيير لن يؤثّر على الصفوف المشتقة من الصف الذي تمّ تعديله.
- عند حدوث خطأ ما في تصميم الصف الأساس، سيتمّ توريث الخطأ إلى الصفوف المشتقة.
- في بعض الأحيان، قد يرث أحد الصفوف عناصر من صف آخر من دون الحاجة إليها، وعند إنشاء كلّ غرض منه سيتمّ حجز موارد لهذه العناصر من دون فائدة مرجوة منها.

وفي حال عدم مقدرة المبرمجين على مراعاة الملاحظات السابقة عند تصميم صف ما، يمكنهم منع وراثة الصف عن طريق التصريح عنه بأنّه مُحكَم Sealed.

ويجب عدم الخلط بين مفهومي التحميل الزائد للطرائق وتجاوز الطرائق، فالتحميل الزائد يُطبَق على الطرائق التي تنتمي لنفس الصف، أما التجاوز فيتم تطبيقه على طريقة موروثة من صف آخر. وكلٌ من المفهومين يمثل أحد أنماط تعدد الأشكال Polymorphism، والذي أحد أنماطه أيضاً إخفاء الطرائق Method Hiding الذي سيتم توضيحه في الفصل القادم.

الأنشطة المرافقة

التمرين الأول: صف الدائرة Circle وصف الأسطوانة Cylinder

1. أعد كتابة الصف `Circle` الذي قُمت بدراسته سابقاً.
2. قم بالتصريح عن الصف `Cylinder` والذي يرث الصف `Circle`.
3. أضف للصف أسطوانة الخاصّية `Height` الممثّلة لارتفاع الأسطوانة والتي يجب أن تكون موجبة القيمة.
4. عرِّف للصف أسطوانة بانياً افتراضياً يُسند القيمة 0 لكلّ من نصف القطر والارتفاع، ثم أضف بانياً يأخذ وسيطي دخل ويُسند قيمتيهما إلى نصف القطر والارتفاع.
5. أضف باني نسخ له وسيط دخل وحيد من النمط `Cylinder`.
6. ترث الأسطوانة الطريقة `Circumference` المُعبّرة عن محيط الأسطوانة (محيط الأسطوانة هو نفس محيط قاعدتها الدائرية الشكل).
7. قم بتجاوز `overriding` الطريقة `Area` لتصبح قابلة لحساب مساحة الأسطوانة.
8. أضف الطريقة `Volume` لحساب حجم الأسطوانة.
9. استخدم الطريقة `ToString` لطباعة معلومات الأسطوانة على الشاشة.
10. قم باختبار الصف السابق مع عدّة كائنات منه.

التمرين الثاني: صف المستطيل Rectangle وصف متوازي المستطيلات Cubod

1. أعد كتابة الصف `Rectangle` الذي قُمت بدراسته سابقاً.
2. قم بالتصريح عن الصف متوازي المستطيلات `Cuboid` والذي يرث من الصف `Rectangle`.
3. يكون لمتوازي المستطيلات خاصّية إضافية هي الارتفاع `Height` ويجب أن تكون موجبة.
4. يكون لمتوازي المستطيلات بانٍ افتراضي يُسند القيمة (0) لجميع أبعاده (الطول والعرض والارتفاع).
5. يكون لمتوازي المستطيلات بانٍ يأخذ ثلاثة وسائط دخل لتمرير قيم للأبعاد الثلاثة.
6. أضف بانياً يأخذ وسيطي دخل، الأول من النمط مستطيل والثاني يمثل الارتفاع.
7. أضف باني نسخ يأخذ وسيط دخل وحيد من النمط مستطيل.
8. يرث متوازي المستطيلات الطريقة محيط المستطيل `Perimeter` (محيط متوازي المستطيلات هو نفسه محيط المستطيل).
9. قم بتجاوز `overriding` الطريقة `Area` لتصبح قابلة لحساب مساحة متوازي المستطيلات.
10. أضف الطريقة `Volume` لحساب حجم متوازي المستطيلات.
11. استخدم الطريقة `ToString` لطباعة معلومات متوازي المستطيلات على الشاشة.
12. قم باختبار الصف السابق مع عدّة كائنات منه.

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.