الوراثة

تعددية الأشكال (Polymorphism)

تسمح لغة البرمجة غرضية التوجه بأن يكون للغرض أكثر من شكل.

أنواع تعددية الأشكال

✓ الستاتيكية

وهذا النوع يتم التعرف عليه في compile time . #C تؤمن طريقتين هما

- ❖ **التحميل الزائد للتوابع** : هذا النوع يسمح بتعريف أكثر من تابع بنفس الاسم ضمن المجال ذاته ولكن باختلاف أنماط أو ترتيب أو عدد الوسطاء
- ➢ **التحميل الزائد للعمليات.**

✓ الديناميكية

يتم التعرف على هذا النوع أثناء RunTime. لتطبيق هذا النوع هناك مفهومين هما:

- ➢ **Virtual Method(Overriding)**

يمكن للصف الابن إعادة تعريف تابع ورثه من الأب حيث يكون التابع الجديد له نفس اسم ونفس وسطاء التابع الموجود في الأب ولكن باختلاف جسم التابع.
ولجعل الصف الأب يسمح بإعادة تعريف تابع لديه عند أبنائه نستخدم كلمة virtual امام هذا التابع.
المثال التّالي يوضّح ذلك

```
using System;
namespace PolymorphismApplication
{
   class Shape
   {
      protected int width, height;
      public Shape( int a=0, int b=0)
      {
         width = a;
         height = b;
      }
      public virtual int area()
      {
         Console.WriteLine("Parent class area :");
         return 0;
      }
   }
   class Rectangle: Shape
   {
      public Rectangle( int a=0, int b=0): base(a, b)
      {
      }
      public override int area ()
      {
         Console.WriteLine("Rectangle class area :");
         return (width * height);
      }
   }
   class Triangle: Shape
   {
      public Triangle(int a = 0, int b = 0): base(a, b)
      {    }
      public override int area()
      {
          Console.WriteLine("Triangle class area :");
          return (width * height / 2);
      }
   }
   class Caller
   {
       public void CallArea(Shape sh)
       {
           int a;
           a = sh.area();
           Console.WriteLine("Area: {0}", a);
       }
   }
   class Tester
   {
       static void Main(string[] args)
       {
           Caller c = new Caller();
           Shape s = new Shape(10,7);
           Rectangle r = new Rectangle(10, 7);
           Triangle t = new Triangle(10, 5);
         c.CallArea(s); Console.WriteLine("---------------------------------\n");
           c.CallArea(r); Console.WriteLine("---------------------------------\n");
           c.CallArea(t); Console.ReadKey();
       }   } }
```

لاحظ أننا وضعنا كلمة افتراضي أمام تابع حساب المساحة حتى نسمح للصف الابن بإعادة تعريفه.

قام الصف الابن بإعادة تعريف تابع حساب المساحة الذى ورثه من والده

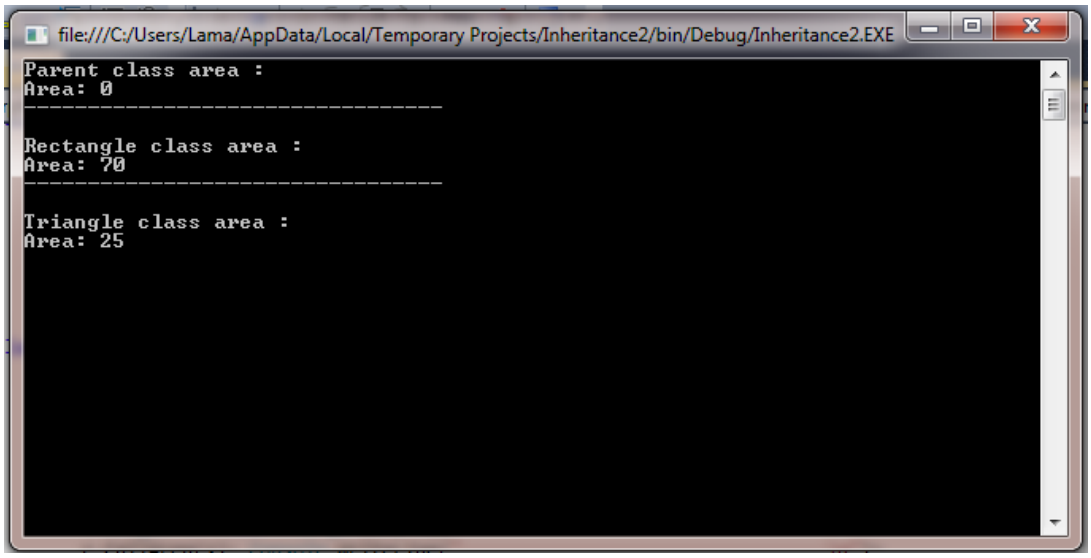
نلاحظ في المثال السابق أننا عرفنا بتابع main غرض من الصف caller حتى نستدعي الدالة CallArea والتي تأخذ كوسيط غرض من النمط shape –شكل- كما في التعليمة

```
c.CallArea(s);
```

حيث تقوم الدالة باستدعاء تابع حساب المساحة عند صف الأب.
ولكن في حال مررنا لها غرض من النمط مستطيل كما في التعليمة

```
c.CallArea(r);
```

فهذا صحيح أيضاً وعندها تقوم هذه الدالة باستدعاء تابع حساب المساحة الذي عرفه الصف الابن المستطيل.
ويكون خرج البرنامج السابق كالتالي



والآن لنقم ببعض التعديلات على البرنامج السابق في التابع main اكتب عوضاً عن التعليمة

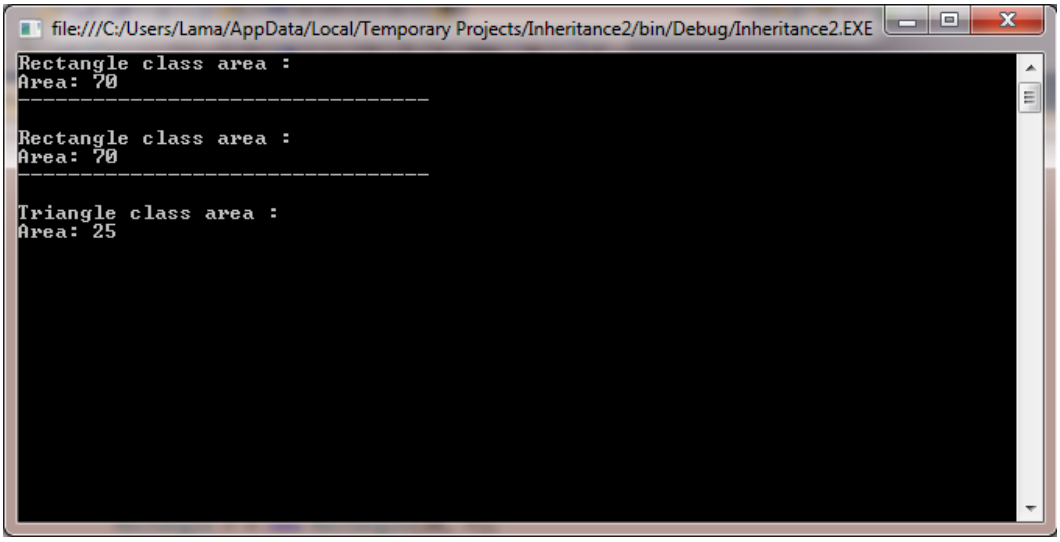
```
Shape s = new Shape(10,7);
```

التعليمة التالية

```
Shape s = new Rectangle(10,7);
```

نلاحظ أن التعليمة السابقة صحيحة فهو ضمنياً يجري عملية قصر. وعند تنفيذ البرنامج نلاحظ أنه يقوم باستدعاء تابع حساب المساحة للصف الابن (المستطيل).

ويكون خرج البرنامج كالتالي



ومن الضروري ملاحظة أنّ التعليمة التالية

```
Rectangle r = new Shape(10, 7);
```

هي تعليمة خاطئة ويعطيك البرنامج خطأ بأنه لا يمكنك التحويل من غرض شكل إلى غرض مستطيل.

- Abstract Class

تسمح الصفوف المجردة بتعريف صف جزئي يرث منه صفوف جديدة وتقوم بتعريف التوابع وأداءها. يحوي الصف المجرد على توابع مجردة تملك اسم ووسطاء ولكن بدون تحديد جسمها ويمكن أن يحوي أيضاً على توابع غير مجردة.

ملاحظات هامة

- لا يمكن إنشاء غرض من الصف المجرد
- لا يمكن أن نعرف تابع مجرد ضمن صف غير مجرد.
- عندما نعرّف صف على أنه sealed أي لا يمكن أن نرث منه. لذلك لا يمكن أن يكون صف ما abstract and sealed.

مثال

ليكن لدينا البرنامج التالي والذي يعرف صف الشكل على أنه مجرد و يحوي بداخلة تابع لحساب المساحة area() أيضاً هذا التابع مجرد بالإضافة إلى تابع آخر غير مجرد.

ويقوم الصف مستطيل الوارث من الصف شكل بإعادة تعريف التابع المجرد (لاحظ في حال حذفت كلمة override امام التابع area ينتج خطأ).

وأيضاً في حال قمنا بتعريف تابع مجرد في الصف مستطيل سينتج لدينا خطأ.

وتجنب أن تقوم في main بإنشاء غرض من الصف شكل كالتالي

```
Shape s = new Shape();
```

لأن ذلك أيضاً سيولد خطأ أنه لا يمكنك أن تعرف غرض من صف مجرد.

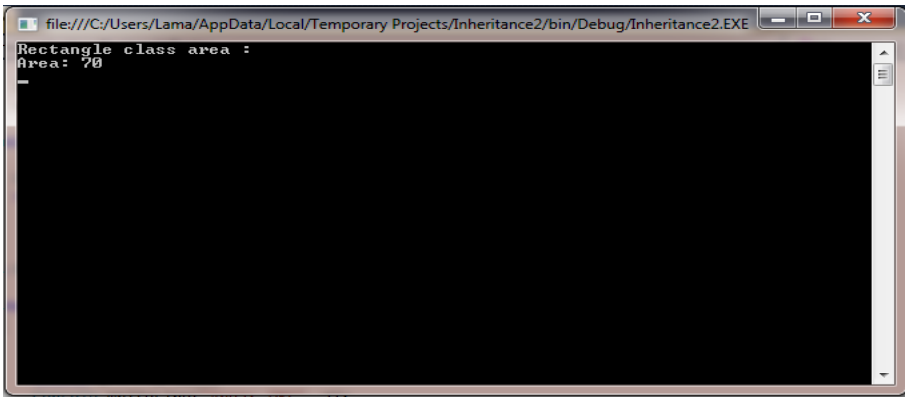
والآن بتطبيق البرنامج التالي

```
using System;
namespace PolymorphismApplication
{
   abstract class Shape
   {
      public abstract int area();
      public int area1() { return 0; }
   }
   class Rectangle:  Shape
   {

      private int length;
      private int width;
      public Rectangle( int a=0, int b=0)
      {
         length = a;
         width = b;
      }
      public override int area ()
      {
         Console.WriteLine("Rectangle class area :");
         return (width * length);
      }
   }

   class RectangleTester
   {
       static void Main(string[] args)
       {
           Rectangle r = new Rectangle(10, 7);
           double a = r.area();
           Console.WriteLine("Area: {0}", a);
           Console.ReadKey();
       }
   }
}
```

يكون الخرج كالتالي



إذا عدنا للبرنامج السابق وكتبنا أمام تعريف الصف shape كلمة sealed نلاحظ أنه ينتج خطأ لأنه لا يمكنك أن تجعل الصف المجرد أيضاً sealed أو static .

ملاحظات

a) كل الصفوف في C# ترث من الصف Object المعرف مسبقاً لديها.
b) يمكن أن نستخدم الكلمة is للتحقق فيما إذا كان غرض من صف ما مثال للتوضيح:

```
If (student1 is Human)
{
}
```