برمجة -٣-

المحاضرة ١+٣

م. تغريد حرفوش

الكلمة المفتاحية Base

- تستخدم هذه الكلمة المفتاحية لتحديد باني الاب الذي سيتم استدعاؤه في الصف الابن
- ضمنيا الباني في الصف الابن سؤاء كان بوسطاء او بدون وسطاء يستدعي باني الاب بدون وسطاء
- اذا اردنا اجبار الباني في الصف الابن على استدعاء باني الاب مع وسطاء يجيب استخدام الكلمة المفتاحية Base
- مثال:

```
class SubClass : BaseClass
  {
     SubClass(int id)
        : base()
     {intsructions}
}
```

استدعاء اجباري لباني الاب بدون وسطاء

```
class SubClass : BaseClass
  {
     SubClass(int id)
        : base(id)
     { intsructions}
  }
```

استدعاء اجباري لباني الاب مع وسطاء

- محدد الوصول protected أي اننا نستطيع الوصول الى كل اعضاء الصف الاب من الابن
- مثال:

```
class A
{
protected void DoWork()
{
Console.WriteLine(" protected ");
}
}
class Test
{
static void Main(string[] args)
{
A a = new A();
a.DoWork();
}
}
```

Sealed

عندما تسبق هذه الكلمة اسم الصف هذا يعني انه لا يمكن توريثه.

```
sealed class A
{
//...
}
class B : A
{
//...
}
```

Override-Virtual

- اذا كان لدى الاب طريقة ما واردنا ان تنفذ الطريقة عند الابن نحتاج هنا لتجاهل الطريقة الموجودة عند الاب حتى نحقق ذلك يجب ان نكتب الكلمة المفتاحية Virtual عند الطريقة الموجودة عند الصف الاب و نضع الكلمة Override عند الطريقة الموجودة عند الصف الابن التي سيتم تنفيذها بالفعل.

مثال

```
class Shape
{
public virtual void Draw()
{
Console.WriteLine("Drawing Shape...");
}
}
class Circle : Shape
{
public override void Draw()
{
Console.WriteLine("Drawing Circle...");
}
}
static void Main()
{
Shape theShape = new Circle();
theShape.Draw();
}
```

الخرج

```
Drawing Circle...
```

- يمكن استخدام أكثر من overrideلطريقة virtualواحدة وذلك اذا كان هناك أكثر من صف يرثوا من صف ما
- ويكون نتيجة التنفيذ حسب نوع الغرض المنشأ وليس حسب نوع المؤشر .
- - يمكن استخدام الكلمة المفتاحية Newعند الطريقة في الصف الابن التي لها نفس اسم الطريقة في الصف الأب
- للدلالة على أنه لا يوجد تعددية أشكال .

تعددية الاشكال

- هي التحويل من نمط الى اخر
- الطريقة الاولى هي طريقة القصر العادية CAST
- أي استخدام نمط الصف الاب لانشاء غرض من نمطالصف الابن.

مثال:

```
class A
{
public void MethodA()
{
}
Advanced Programming 1
26
}
class B : A
{
public void MethodB()
{
}
}
```

وبالتالي فانه من الممكن في #C أن نكتب :

```
A a = new B();
```

وهنا يجب أن نعامل هذا الغرض كغرض من الصف الأب وليس الابن

أي أنه يمكننا أن نصل الى أعضاء الصف الأب مثلاً :

```
a.MethodA();
```

ولا يمكن أن نصل الى أعضاء الصف الابن أي :

```
a.MethodB(); // error
```

الطريقة الثانية هي استخدام نفس الاسم للطريقة الموجودة في الصف الاب والصف الابن

```
class Shape
{
public void Draw()
{
Console.WriteLine("Drawing Shape...");
}
}
class Circle : Shape
{
public void Draw()
{
Console.WriteLine("Drawing Circle...");
}}
```

الآن لو قمنا بتعريف غرض كما يلي :

```
static void Main()
{
Shape theShape = new Circle();
theShape.Draw();
}
```

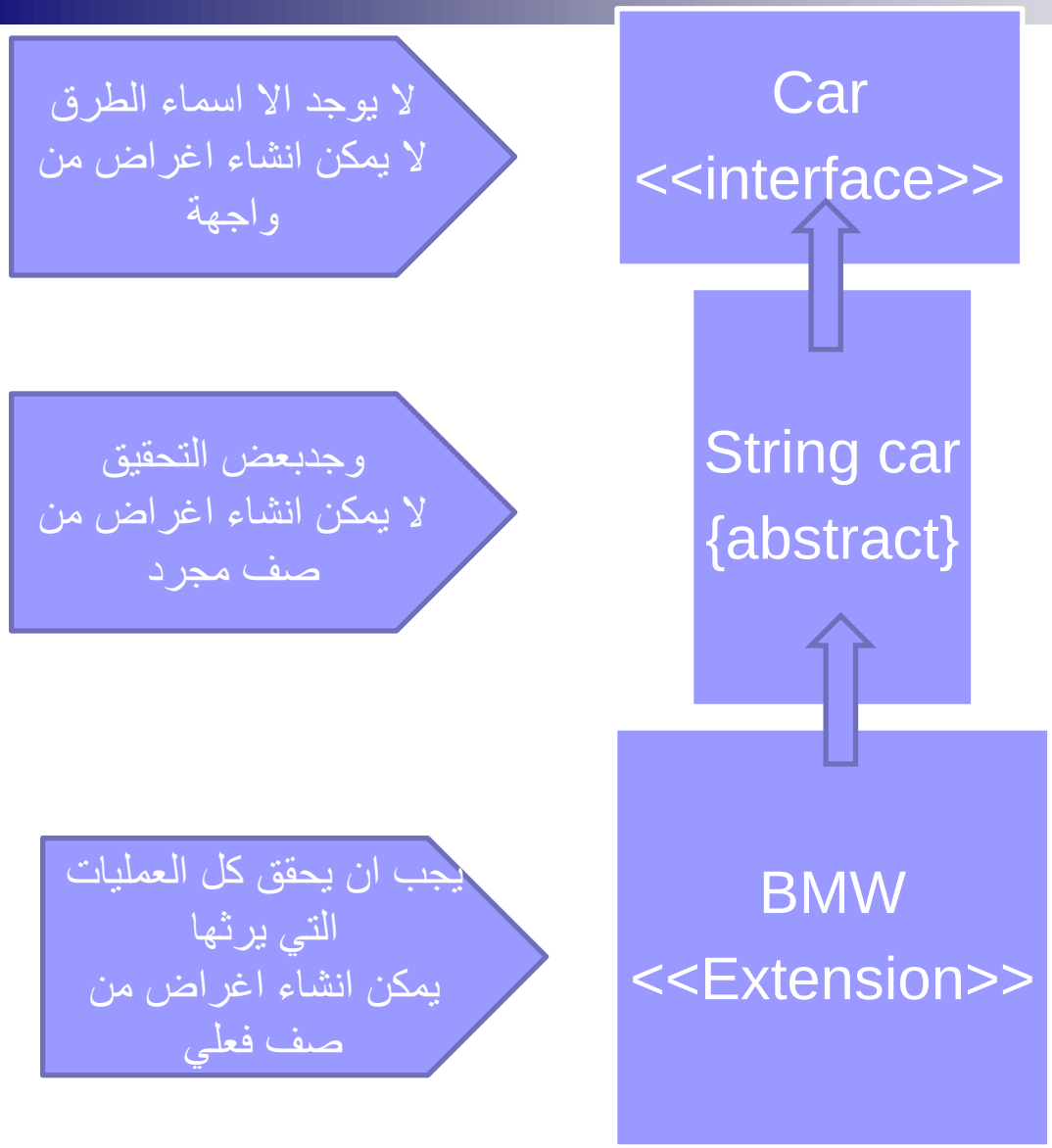
الخرج

```
Drawing Shape...
Press any key to continue . . .
```

الواجهات
intefaces

- في الوراثة يمكن للصف الاب ان يكون له اكثر من ابن وللابن اكثر من ابن
- اما ان يرث الابن من اكثر من اب فهي غير ممكنة
- الواجهات قامت بحل المشكلة وسمحت بالوراثة المتعددة
- هي اسماء metods فقط
- لاتحتوي على أي تحقيق أي لا يمكن كتابة أي كود بداخلها فهي تتكون من تصريح فقط دون كتابة تفصيلات
- لا يمكن انشاء غرض منها
- لا يمكن للواجهة الا ان تورث
- لا يمكن ان تحتوي طرق الواجهة على اجسام (Method body)
- يمكن للصف الواحد ان يحقق اكثر من واجهة

- عندما يقوم صف بالوراثة من واجهة نقول انه يقوم بتحقيق هذه الواجهة Implement
- عندما يقوم صف بالوراثة من صف مجرد او عادي نقول انه يقوم بالتمديد Extension
- الصف يحقق اكثر من واجهة ولا يمدد الا صف واحد
- أن الواجهة تتضمن خصائص و دوال مجردة و يعاد تحقيقها في الصفوف التي تحقق هذه الواجهة.
- لا يمكننا كتابة محدد الوصول access modifier للعناصر الموجودة ضمن الواجهة و ذلك لأنها ضمنيا معرفة على أنها عامة public ولكن هذا لا يعني اننا يمكن ان نكتب محدد الوصول public
- يفضل البد باسم الواجهة بالمحرف I


لا يوجد الا اسماء الطرق
لا يمكن انشاء اغراض من
واجهة
Car
<<interface>>
وجدبعض التحقيق
لا يمكن انشاء اغراض من
صف مجرد
String car
{abstract}
يجب ان يحقق كل العمليات
التي يرثها
يمكن انشاء اغراض من
صف فعلي
BMW
<<Extension>>

التصريح عن الواجهات

- بنفس طريقة التصريح عن صف ولكن مع استخدام الكلمة المحجوزة interface

interface *interface-name*

{

Return-type property-name {set; get;}

Return-type method-name (parameter-list);

}

```
interface grandf
{
string Name( );
}
interface father
{
int old(int year);
}
class son: grandf, father
{ ...
}
```

```
interface IShape
{
    int X { set; get; } //property
  int Y { set; get; } //property
  double surface();     //method

}
```

```
class Circle : IShape
   {
       int r;
       int x;
       int y;

       public Circle(int r,int x,int y) {
           this.x = x;
           this.y = y;
           this.r = r; }

       public int R {
           set { r = value; }
           get { return r; } }
```

```
public  int X
      {
          set { x = value; }
          get { return x; }
      }
      public  int Y
      {
          set { y = value; }
          get { return y; }
      }

      public double surface()
      {
          return 2 * Math.PI * r;
      }
```

```
class Program
   {
       static void Main(string[] args)
       {
           Circle c = new Circle(6, 1, 2);
   Console.WriteLine("the surface of your  circle="+c.surface());



       }

   }
```

```
the surface of your circle=37.6991118430775
```

يمكننا التعامل مع الواجهات كأي نمط معطيات يمكن أن نعرف منه متحولات و بارمترات و لكن لا يمكن أن ننشأ غرض Instance من الواجهة لأنها ينطبق عليها مفهوم الصف المجرد ، إذاً ما الحل ؟؟!!

يمكننا أن ننشأ مؤشر من الواجهة و إسناده إلى غرض من الصف الذي يحقق هذه الواجهة :

```
IShape iCircle=new Circle(4,3,1);
```