أكاديمية المبرمجين العرب

الإكليل (الجزء الثاني)

المختصر المفيد في البرمجة الكائنية التوجه

الحقيبة البرمجية

خالد السعداني

www.mobarmijoun.com

فهرس المواضيع

تمت كتابة الشفرات على نسخة الفيجوال ستوديو 2015 الاستعراضية Visual Studio Ultimate Preview 2015 والأكواد صالحة لكل النسخ بدء من الإصدار 2003 من برنامج الفيجوال ستوديو إلى غاية آخر إصدار بإذن الله.

تقديم:

الحمد للّه ابتداء وانتهاء والصلاة والسلام على سيدنا محمد وعلى آله وصحبه ثم أما بعد:

في نسخة الدوت نيت من الفيجوال بيسك التي ظهرت في بدايات عام 2002 تم إقحام نمط البرمجة كائنية التوجه كنمط برمجي أساسي في البنية الجديدة لهذه اللغة، الشيء الذي أدى إلى إضافة مجموعة من المفاهيم الجديدة المرتبطة بهذا النمط البرمجي مثل الفئات Classes والكائنات Objects والمشيدات Constructors والمهدمات Destructors والخصائص Properties والوظائف Methods والتغليف Encapsulation والوراثة Inheritance و تعدد الأشكال Polymorphisme، وإعادة التعريف Overriding، والتظليل Shadowing وغير ذلك من المفاهيم المتعلقة بالبرمجة الشيئية.

كثرة المفاهيم الكائنية وسهولة الخلط بين مسمياتها مثل Overloads و Overrides و Overridable يتسبب في جعلها عصية على الفهم، وإن تم استيعابها تكون عصية على التذكر، لذلك سنحاول في هذا الكتيب الصغير أن نتحدث عن البرمجة الكائنية التوجه من البداية ثم نتدرج من مفهوم إلى آخر بأسلوب سلس.

لمن يوجه هذا الكتيب ؟

هذا الكتيب موجه لمبرمجي لغة الفيجوال بيسك (الإصدار القديم) الذين يريدون الولوج إلى نمط البرمجة بالكائنات في نسخة الدوت نيت من الفيجوال بيسك، وموجه كذلك إلى مبرمجي لغة الفيجوال بسيك دوت نيت الذين يريدون تفصيلا وتوضيحا للمفاهيم الرئيسية في البرمجة الكائنية، وكمتطلبات قبلية على المبرمج أن يكون على دراية بأساسيات لغة الفيجوال بيسك وخاصة كيفية الإعلان عن الإجراء ات والدوال وطريقة النداء عليها.

ماهي البرمجة الكائنية التوجه OOP؟

البرمجة الكائنية التوجه Object Oriented Programming هي أسلوب برمجي يمكننا من كتابة الشفرات على شكل فئات Classes وكائنات Objects، ليصبح الكود شبيها بالحياة الواقعية حيث يصبح النوع الشامل عبارة عن فئة Class وكل نسخة من هذا النوع تصبح عبارة عن Object، بينما الدوال والإجراء ات تصبح عبارة عن وظائف Methods لهذه الكائنات.

تعالوا بنا نبسط هذه الكلمات، الفئة Class عبارة عن تمثيل شامل لنوع معين، فلو أخذنا على سبيل المثال "سيارة Car" فهي تمثيل شامل لنوع معين، لكن حينما نحدد نوعا معينا كأن نقول "ميرسيدس" فنحن هنا نتكلم عن كائن Object مستنسخ من الفئة Car.

لترسيخ الفكرة في أذهاننا فكر في أي تمثيل عام لكائن معين، خذ "الحاسوب" مثلا، أليس الحاسوب نمطا عاما لمجموعة من الأجهزة التي تشترك في نفس الخصائص والوظائف؟ أليست كل الأجهزة التي نسميها حواسيبا لها نوع وشكل معين؟ أليس من وظائف هذه الأجهزة "الاشتغال والتوقف وتخزين ومعالجة البيانات"؟ إذن يمكننا أن نعتبر الحاسوب فئة، لأنه يقدم نوعا قائما بذاته لمجموعة من الكائنات التي تؤدي نفس الدور وتكتسي نفس الخصائص، بنفس القياس لو حددنا حاسوبا بعينه: خذ مثلا " Compaq " فنحن هنا خرجنا من التعميم ودخلنا في التخصيص إذن نحن بدأنا نتكلم عن الكائنات Objects.

في لغة الفيجوال بيسك دوت نيت، نستطيع إنشاء فئة عبر كتابة الكلمة المحجوزة Class متبوعة باسم الفئة، فلو أردنا إسقاط الكلمات السابقة تطبيقيا فإن كود إنشاء الفئة "حاسوب CLS_COMPUTER" كما يلي:

```
Class CLS_COMPUTER

End Class
```

كل فئة قد تتكون من ثلاثة أجزاء:

الخصائص Properties: وهي مواصفات الفئة وخصائصها ومميزاتها

الوظائف Methods: الأعمال والسلوكيات التي تقوم بها الفئة

الأحداث Events: أفعال مرتبطة بالكائنات تؤدي عند حصولها إلى تنفيذ وظائف معينة.

الخصائص Properties:

وهي ما تتميز به الفئة من خصائص ومميزات، على سبيل المثال حينما نتحدث عن الفئة "حاسوب Computer" فيمكننا اعتبار لون الحاسوب، ونوعه، وتاريخ تصنيعه، وثمن بيعه من الخصائص المميزة لهذه الفئة، لذلك يمكننا كتابتها داخل الفئة على شكل متغيرات تسمى في عرف البرمجة الكائنية بالحقول Fields أو Attributes.

في المثال الآتي قمنا بكتابة الحقول المميزة لفئة الحاسوب:

```
Class CLS_COMPUTER
    'Fields / Attributes
    Public Model As String
    Public Price As Decimal
    Public Date_of_Manufacture As DateTime
End Class
```

لاحظ أننا قبل كتابة أسماء حقول الفئة، وضعنا الكلمة Public التي تنتمي إلى أسرة محددات الوصول Access Modifiers، وسميت كذلك لأنها تمكننا من تعريف مجال الوصول إلى هذا الحقل. في لغة الفيجوال بيسك دوت نيت تتكون أسرة محددات الوصول من أربعة كلمات وهي كما يلي:

محدد الوصول	شرحه
Private	وتعني أن الحقل سيبقى معروفا فقط على مستوى الفئة التي تحتويه، ولو أردت استعماله خارج مجال الفئة فلن يمكنك ذلك لأن محدد الوصول إليه خاص Private.

Public	إذا أعلنا عن متغير أو وظيفة أو أي عنصر من عناصر الفئة بالكلمة Public، فهذا يعني أنه يمكننا الوصول إليه داخل الفئة التي تحتويه وخارجها كذلك، بل وخارج المشروع الذي توجد فيه هذه الفئة أيضا، بمعنى لو لدي مجموعة من المشاريع وواحد منها يوجد به عنصر معلن عنه بمحدد الوصول Public فيعني ذلك أنه يمكنني استخدامه داخل كافة المشاريع الأخرى.
Friend	مثلها مثل Public تسمح لنا بالوصول إلى العنصر من داخل وخارج الفئة التي تحتويه، لكن مجال الوصول يتوقف عند المشروع الذي يضم هذه الفئة، بحيث لو أردت استخدام هذا العنصر داخل مشروع آخر فلن يمكنني ذلك على خلاف محدد الوصول Public.
Protected	هذه الكلمة تعني أن العنصر المعلن عنه بها يمكننا الوصول إليه فقط على مستوى الفئة التي تحتويه وكذلك الفئات المشتقة منها (سنتطرق إلى مفهوم الاشتقاق / الوراثة في فصل قادم)

لاستنساخ كائن من هذه الفئة يكفي أن نعلن عنه بنفس الطريقة التي نعلن بها عن المتغيرات مع استبدال نوع البيانات باسم الفئة مع ضرورة كتابة الكلمة New كما يلي:

```
    'Instanciate new object from Computer Class
    Dim Computer As New CLS_COMPUTER
```

بعد ذلك يمكننا إسناد القيم لحقول الفئة أو استعراضها كما يلي:

```
    'Instanciate new object from Computer Class
    Dim Computer As New CLS_COMPUTER

        'إسناد القيم إلى الحقول
        Computer.Model = "Acer"
        Computer.Price = 345.5
        Computer.Date_of_Manufacture = New DateTime(2006, 12,
24)


        'الحصول على قيم الحقول
        MessageBox.Show(String.Format("{0},{1},{2}",
                                      Computer.Model,
                                      Computer.Price,
Computer.Date_of_Manufacture))
```

يوجد في البرمجة الكائنية مفهوم مهم يسمى Encapsulation والذي نستطيع الاصطلاح عليه بالتغليف، ويعني هذا المفهوم أن نخفي الكود الخاص بالفئة عند التعامل مع كائنات مستنسخة منها، فبدل أن يستعرض المبرمج الثاني أكواد فئاتنا يصبح قادرا فقط على مشاهدة أسماء الوظائف والخصائص فقط، لذلك ينبغي أن نعلن دائما عن حقول الفئة بمحدد الوصول Private لكي لا يراه المبرمج الثاني، ونستعيض عن اسم الحقل بمفهوم جديد يسمى الخصائص Properties تكون عامة الوصول Public بينما تصبح الحقول خاصة الوصول Private.

كل حقل من حقول الفئة علينا إنشاء خاصية له حسب دوره، فإن كان يعطي معلومة فقط أعلنا عن الخاصية من نوع ReadOnly فقط، أما إن كان يستقبل القيمة فقط أعلنا عن الخاصية من نوع WriteOnly، وإلا فإننا في الوضع العادي نستغني عن الكلمتين معا، ونعلن عن الخاصية بشطريها.

كل خاصية مكونة من جزءين، جزء يمكننا من الحصول على القيمة المحفوظة في الحقل ويسمى Getter، وجزء يمكننا من إسناد قيمة ما لهذا الحقل ويسمى Setter.

لإنشاء خاصية لحقل ما، ينبغي أن نكتب محدد الوصول أولا (والأفضل أن يكون Public احتراما لدور الخصائص وهو تغليف الحقول) ثم الكلمة المحجوزة Property متبوعة باسم الخاصية وبنوعها (نفس نوع الحقل المراد إنشاء الخاصية له)، وهذا مثال على إنشاء خصائص لكافة حقول الفئة "حاسوب CLS_COMPUTER"

```
Class CLS_COMPUTER
    'Fields / Attributes
    Private Model As String
```

```
    Private Price As Decimal
    Private Date_of_Manufacture As DateTime

    'Properties الخصائص
    Public Property Model_Property As String
        Get
            Return Model
        End Get
        Set(value As String)
            Model = value
        End Set
    End Property

    Public Property Price_Property As Decimal
        Get
            Return Price
        End Get
        Set(value As Decimal)
            Price = value
        End Set
    End Property

    Public Property Date_of_Manufacture_Property As Date
        Get
            Return Date_of_Manufacture
        End Get
        Set(value As Date)
            Date_of_Manufacture = value
        End Set
    End Property

End Class
```

لو دققت جيدا في الكود أعلاه، ستلاحظ أن الخصائص معلن عنها بمحدد الوصول Public، بينما الحقول أصبحت Private، وهذا وجه من وجوه التغليف Encapsulation

بحيث يتم حماية بيانات الكائن داخل الفئة ولا يستطيع المبرمج الثاني التعامل معها إلا من خلال الخصائص والوظائف.

وستلاحظ كذلك أن لكل خاصية نوع بيانات من نفس نوع الحقل الذي تمثله، وأن كل خاصية مكونة من جزءين، الجزء الأول عبارة عن Getter يمكننا من الحصول على قيمة الحقل، والجزء الثاني عبارة عن Setter يمكننا من إسناد القيمة إليه.

تمكننا الخصائص من قراءة و إسناد القيم لحقول الفئة، بحيث يمكننا التعامل مع الكائن المستنسخ من الفئة "حاسوب CLS_COMPUTER" كما يلي:

```
    'Instanciate new object from Computer Class
    Dim Computer As New CLS_COMPUTER

        'Setting Values
        Computer.Model_Property = "Acer"
        Computer.Price_Property = 345.5
        Computer.Date_of_Manufacture_Property = New
DateTime(2006, 12, 24)


        'Getting Values
        MessageBox.Show(String.Format("{0},{1},{2}",
Computer.Model_Property,
Computer.Price_Property,
Computer.Date_of_Manufacture_Property))
```

يمكننا فصل الجزءين المكونين للخاصية عن بعضهما البعض، أو الاكتفاء بأحدهما حسب طبيعة الحقل، فلو أردنا أن يكون أحد الحقول للقراءة فقط نستغني عن الجزء Set ونعلن عن الخاصية بالكلمة المحجوزة ReadOnly كما يلي:

```
Public ReadOnly Property GetModel As String
    Get
        Return Model
    End Get
End Property
```

وبالمقابل، يمكننا جعل أحد الحقول للكتابة فقط بحيث نسند له القيمة دون الحاجة إلى قراءة قيمته عبر استخدام الكلمة WriteOnly كما يلي:

```
Public WriteOnly Property SetModel As String
    Set(value)
        Model = value
    End Set
End Property
```

الوظائف Methods:

وهي المهام والعمليات التي تقوم بها الفئة، وتكتب على شكل إجراءات ودوال حسب الدور الذي تؤديه.

ونحن نتحدث عن فئة الحاسوب يمكننا اعتبار العمليات التي يقدمها الحاسوب، مثل الاشتغال والتوقف وإعادة الاشتغال، وتخزين ومعالجة البيانات بمثابة وظائف Methods.

```
Class CLS_COMPUTER
    'Fields / Attributes
    Private Model As String
    Private Price As Decimal
    Private Date_of_Manufacture As DateTime

    'Properties الخصائص
    Public Property Model_Property As String
        Get
            Return Model
        End Get
        Set(value As String)
            Model = value
        End Set
    End Property

    Public Property Price_Property As Decimal
        Get
            Return Price
        End Get
        Set(value As Decimal)
            Price = value
        End Set
    End Property
```

```
    Public Property Date_of_Manufacture_Property As Date
        Get
            Return Date_of_Manufacture
        End Get
        Set(value As Date)
            Date_of_Manufacture = value
        End Set
    End Property

    'Methods الوظائف
    Public Sub Run()
        'Do something
    End Sub
    Public Sub ShutDown()
        'Do something
    End Sub
    Public Sub Restart()
        'Do something
    End Sub
    Public Function ProcessData() As DataType
        'Return something
    End Function
    Public Sub StoreData()
        'Do something
    End Sub

End Class
```

يمكننا استدعاء الوظائف بنفس طريقة استدعاء الخصائص، بحيث نستطيع الوصول إليها من خلال اسم الكائن المستنسخ من الفئة، كما يلي:

```
'Instanciate new object from Computer Class
Dim Computer As New CLS_COMPUTER

    'Calling Class Methods
    Computer.Run()
    Computer.StoreData()
    Dim Result = Computer.ProcessData()
    Computer.Restart()
    Computer.ShutDown()
```

المشيدات Constructors:

المشيدات هي إجراء ات نكتبها داخل الفئة من أجل تشييد الكائن عند عملية استنساخه عبر إسناد قيم بدئية له، أو القيام بعمليات معينة أثناء عملية استنساخ الكائن، لأنها أول ما ينفذ عند إنشاء الكائن.

لإنشاء مشيد في لغة الفيجوال بيسك، يكفي أن نكتب الكلمة New() بعد صيغة إنشاء الإجراءات التي هي Sub كما يلي:

```
Sub New()
End Sub
```

في المثال أعلاه أنشأنا مشيدا بالصيغة الافتراضية، بمعنى أنه حتى لو لم نقم بكتابته بذلك الشكل فلا مشكلة لأنه موجود ضمنيا.

للإعلان عن كائن من الفئة التي تحتوي على مشيد بهذه الصيغة الافتراضية نكتب ما يلي:

```
Dim myComputer As New CLS_COMPUTER()
```

الأقواس فارغة في نهاية اسم الفئة تدل على أن المشيد لا ينتظر أية برامترات.

يمكننا إنشاء مشيدات حسب حاجتنا، كأن نقوم بتمرير قيم بدئية لحقول الفئة عند استنساخ كائن منها كما يعرض لنا المثال التالي:

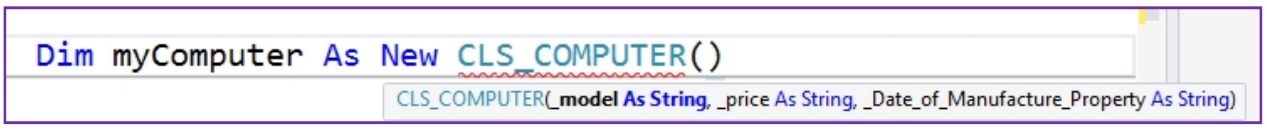
```
    'Constructors المشيدات
    Sub New(_model As String, _price As String,
_Date_of_Manufacture As String)
        Model = _model
        Price = _price
        Date_of_Manufacture = _Date_of_Manufacture
    End Sub
```

للتمييز بين حقول الفئة وبين البرامترات الوافدة للمشيد نقوم بتغييرات طفيفة كأن نضيف رمز الأندرسكور (_) أمام اسم البرامتر كما في الحالة أعلاه.

بالنسبة لحقول الفئة وجميع عناصرها من خصائص ووظائف فيمكننا كتابة الكلمة Me ثم نقطة بعدها وستظهر لنا كلها ونختار منها ما نشاء، كما يوضح لنا المثال التالي الذي يؤدي نفس دور المثال السابق:

```
    'Constructors المشيدات
    Sub New(_model As String, _price As String,
_Date_of_Manufacture As String)
        Me.Model = _model
        Me.Price = _price
        Me.Date_of_Manufacture = _Date_of_Manufacture
    End Sub
```

الآن أصبح بإمكاننا إسناد قيم بدئية عند استنساخ أي كائن من الفئة CLS_COMPUTER بحيث سيطلب منا إدخال قيم البرامترات كما تعرض الصورة التالية:



عند إنشاء أي كائن جديد، المفروض أن يكون على النسق التالي:

```
    Dim myComputer As New CLS_COMPUTER("Compaq", 888.6, New
DateTime(2014, 08, 16))
```

يمكنك إنشاء المشيدات بالعدد الذي تريد، احرص فقط أن تكون البنية Signature مختلفة، والمقصود بالبنية أن تكون البرامترات مختلفة في كل المشيدات إما من حيث عددها، أو من حيث نوع بياناتها، أو من حيث ترتيبها، فكل هذه الأشكال من المشيدات مقبولة:

```
    'المشيد الأول
    Sub New(_model As String, _price As String,
_Date_of_Manufacture As String)
        Me.Model = _model
        Me.Price = _price
        Me.Date_of_Manufacture = _Date_of_Manufacture
    End Sub

    'المشيد الثاني
    Sub New()
        Me.Model = "Undefined"
        Me.Price = 00.0
        Me.Date_of_Manufacture = Date.Now
    End Sub

    'المشيد الثالث
    Sub New(ByVal computer As CLS_COMPUTER)
        Me.Model = computer.Model
        Me.Price = computer.Price
        Me.Date_of_Manufacture = computer.Date_of_Manufacture
    End Sub
```

المشيد الأول يقوم باستقبال مجموعة من البرامترات ويسندها إلى حقول الفئة، بينما يقوم المشيد الثاني بإسناد قيم بدئية دون الحاجة إلى استخدام برامترات بحيث القيم محددة مسبقا، أما المشيد الثالث والأخير، فإنه يقوم باستقبال كائن من نفس نوع الفئة ومن خلال بياناته سيتم إنشاء الكائن الجديد.

الآن تعال بنا نرى مثالا على إنشاء ثلاث كائنات من هذه الفئة باستخدام هذه المشيدات الثلاثة:

```
    'نموذج على لستخدم المشيد الأول
    Dim Computer1 As New CLS_COMPUTER("Toshiba", 755.1, New
DateTime(2012, 07, 27))
```

في المثال أعلاه سيتم إسناد القيم الممررة إلى حقول الكائن Computer1.

```
    'نموذج على لستخدم المشيد الثاني
    Dim Computer1 As New CLS_COMPUTER("Toshiba", 755.1, New
DateTime(2012, 07, 27))

    Dim Computer2 As New CLS_COMPUTER()
```

في المثال أعلاه سيتم إسناد القيم الأساسية التي ذكرناها داخل المشيد إلى حقول الكائن Computer2.

```
    'نموذج على لستخدم المشيد الثالث
    Dim Computer1 As New CLS_COMPUTER("Toshiba", 755.1, New
DateTime(2012, 07, 27))

    Dim Computer2 As New CLS_COMPUTER()

    Dim Computer3 As New CLS_COMPUTER(Computer1)
```

في المثال أعلاه، سيتم بناء الكائن Computer3 بنفس البيانات التي تم إسنادها للكائن Computer1 لأننا مررناه على شكل برامتر إليه.

ملحوظة مهمة:

إذا أنشأت مشيدا واحدا فقط داخل فئة معينة له برامترات، فعند كتابتك لهذا الاستنساخ سيعتبر خطأ:

```
Dim myComputer As New CLS_COMPUTER()
```

لأن عليك إنشاء مشيد فارغ ليسمح بهذا النوع من الاستنساخ، أي عليك إضافة مشيد داخل الفئة بما يلي بالضبط:

```
    Sub New()

    End Sub
```

المهدمات Destructors:

المهدمات تقوم بنقيض الدور الذي تقوم به المشيدات، فإن كانت هذه الأخيرة تقوم ببناء الكائن أثناء عملية الاستنساخ من الفئة، فإن المهدمات تقوم بإنهاء الكائن وختم مهامه عبر تحرير الذاكرة الرئيسية منه أثناء خروج البرنامج من مجال الوصول Scope إلى هذا الكائن.

علما أننا لسنا مطالبين بكتابة أي كود من أجل تأليف المهدمات لأن دورها أصلا منوط بآلية التنظيف التلقائي Automatic Garbage Collector التي توفرها CLR (Common Language Runtime)، هذه الأخيرة التي تعتبر مثل الآلة الافتراضية التي نشغل عليها برامج الدوت نيت، فكلما انتهى كائن ما تقوم CLR بتحرير الذاكرة منه عبر آلية التنظيف التلقائي GC.

لكن إن أردنا أن ننفذ كودا معينا عند انتهاء مهام أي كائن، كأن نقطع اتصال، أو نخطر كائنات أخرى بأن الكائن لم يعد موجودا على الذاكرة، أو غير ذلك من العمليات، فيمكننا أن نعيد تعريف الإجراء Finalize() الذي يمكننا من اقتناص عملية إنهاء الكائن ونكتب فيه ما نشاء من كود كما يلي:

```
Protected Overrides Sub Finalize()
    MsgBox("تم إنهاء الكائن")
End Sub
```

الوراثة Inheritance:

الوراثة هي عملية برمجية من خلالها نقوم بتوريث جميع عناصر فئة معينة تسمى الفئة الرئيسية Base Class لفئة ثانية تسمى فئة مشتقة Derived Class، وصيغة الوراثة في لغة الفيجوال بيسك تكون عبر استخدام الكلمة Inherits.

لنفترض أن لدينا الفئة "شخص CLS_PERSON" مكونة من العناصر التالية:

```
Class CLS_PERSON
    Private name As String
    Private age As Integer

    Public Property Name_Property As String
        Get
            Return name
        End Get
        Set(value As String)
            name = value
        End Set
    End Property

    Public Property Age_Property As Integer
        Get
            Return age
        End Get
        Set(value As Integer)
            age = value
        End Set
    End Property

    Sub New()

    End Sub
```

```
    Sub New(_name As String, _age As Integer)
        Me.name = _name
        Me.age = _age
    End Sub

    Sub Speak()
        MsgBox("أتكلم أنا")
    End Sub
    Sub Work()
        MsgBox("أعمل أنا")
    End Sub
End Class
```

الفئة "شخص CLS_PERSON" تتكون من حقلين اثنين هما الاسم name والعمر age، وتحتوي على خاصية لكل منهما، إضافة إلى مشيدين أحدهما فارغ، والآخر يستقبل برامترات من أجل إسناد قيم بدئية لحقول الفئة أثناء عملية إنشاء كائن منها، وتحتوي كذلك على وظيفتين هما Speak() و Work().

لنفترض أننا بحاجة إلى فئة جديدة في مشروعنا خاصة بالموظفين، كما نعلم جميعا، كل موظف له اسم وله عمر إضافة إلى حقول جديدة، أضف إلى ذلك أن الموظف هو في الأصل شخص، أي أنه يقوم بنفس الوظائف الموجودة في الفئة "شخص CLS_PERSON"، لذلك بدل أن نعيد تأليف الفئة "موظف CLS_EMPLOYEE" من الأول نقوم بكل بساطة باشتقاقها/وراثتها من الفئة "شخص CLS_PERSON" وتلقائيا سيصبح لها نفس العناصر الموجودة بها.

```
    Class CLS_EMPLOYEE
        Inherits CLS_PERSON

    End Class
```

الآن صار لدي صلاحية استخدام جميع العناصر الواردة في الفئة الرئيسية من خصائص ووظائف وغيرها لأن الفئة CLS_EMPLOYEE أصبحت فئة مشتقة من الفئة CLS_PERSON.

بمعنى لو أنني أنشأت كائنا من الفئة CLS_EMPLOYEE، فمباشرة يمكنني استدعاء عناصر الفئة الأم لأنه تم توريثها كاملة للفئة المشتقة CLS_EMPLOYEE كما يعرض لنا المثال التالي:

```
Dim Employee As New CLS_EMPLOYEE()
Employee.Name_Property = "السعداني خالد"
Employee.Age_Property = 25
Employee.Speak()
Employee.Work()
```

يمكننا إضافة الحقول والمشيدات وباقي العناصر التي تميز الفئة المشتقة بشكل عادي كما يوضح لنا المثال التالي:

```
Class CLS_EMPLOYEE
    Inherits CLS_PERSON
    Private salary As Decimal

    Public Property Salary_Property As Decimal
        Get
            Return salary
        End Get
        Set(value As Decimal)
            salary = value
        End Set
    End Property
```

```
End Class
```

المشيدات كذلك يمكننا إنشاؤها بنفس الطريقة العادية، بحيث الحقول الأساسية مثل name و age، قد تم مسبقا إعدادها في مشيد داخل الفئة الأم بالشكل التالي:

```
'CLS_PERSON Class
Sub New(_name As String, _age As Integer)
    Me.name = _name
    Me.age = _age
End Sub
```

فكل ما علينا القيام به في الفئة المشتقة هو استدعاء المشيد الرئيسي في الفئة الأم عبر الكلمة MyBase، ثم نضيف الحقول الجديدة التي توجد بالفئة المشتقة كما يعرض المثالان الآتيان:

النداء على المشيد الفارغ من الفئة الرئيسية:

```
Sub New()
    MyBase.New()
End Sub
```

النداء على المشيد المصحوب ببرامترات من الفئة الرئيسية:

```
    Sub New(_name As String, _age As Integer, _salary As
Decimal)
        MyBase.New(_name, _age)
        Me.salary = _salary
    End Sub
```

إعادة التعريف Overriding:

المقصود بإعادة التعريف في البرمجة الكائنية التوجه، أن نعيد صياغة بعض عناصر الفئة الرئيسية بما يتوافق ويتناسب مع الفئة المشتقة، على سبيل المثال يوجد لدينا في الفئة الرئيسية CLS_PERSON وظيفة اسمها ()Work، المفترض أن يتم إعادة تعريفها لتتناسب مع الفئة CLS_EMPLOYEE لأننا أصبحنا نعلم طبيعة عمله على عكس الغموض الموجود في الفئة الرئيسية.

مثال آخر لتتضح لنا الرؤية، لنفترض أن الفئة الرئيسية CLS_PERSON تحتوي على إجراء يعرض معلومات الشخص كما يلي:

```
'CLS_Person Class
Sub DisplayData()
    MsgBox("الاسم: " & name)
    MsgBox("العمر: " & age)
End Sub
```

في الفئة المشتقة CLS_EMPLOYEE وجب أن نضيف لهذا الإجراء معلومة جديدة وهي الراتب Salary، لذلك صار إلزاما علينا إعادة تعريفها لتتناسب مع الفئة المشتقة.

لكي نتمكن من إعادة تعريف وظيفة معينة على مستوى الفئات المشتقة، علينا الإعلان عنها في الفئة الرئيسية بالكلمة المحجوزة Overridable، دلالة على أن هذه الوظيفة قابلة لإعادة التعريف، أي أن كود الوظيفة ()DisplayData في الفئة الرئيسية CLS_PERSON سيصبح كما يلي:

```
'CLS_PERSON Class
Overridable Sub DisplayData()
    MsgBox("الاسم: " & name)
    MsgBox("العمر: " & age)
End Sub
```

الوظيفة DisplayData() أصبحت الآن قابلة لإعادة التعريف على مستوى الفئات المشتقة، يكفي أن نكتب الكلمة Overrides أمام الوظيفة وسيتم اعتمادها أثناء إنشاء أي كائن من الفئة المشتقة، وصيغة إعادة التعريف كما يلي:

```
Public Overrides Sub DisplayData()
    MyBase.DisplayData()
    MsgBox("الراتب: " & salary)
End Sub
```

السطر الأول من الوظيفة يقوم بجلب محتواها من الفئة الرئيسية عبر الكلمة MyBase التي رأيناها مع وراثة المشيدات، ثم في السطر الثاني قمنا بعرض الراتب، لو أتينا الآن وأنشأنا كائن جديد من الفئة CLS_EMPLOYEE واستدعينا الوظيفة DisplayData()، فستكون النتيجة إظهار ثلاثة رسائل الأولى لعرض الاسم والثانية لعرض السن والثالثة لعرض الراتب.

في حالتنا هذه إعادة تعريف الوظيفة DisplayData() هو اختياري ويمكننا الاستغناء عن ذلك، في بعض الحالات نريد أن تكون إعادة تعريف بعض الوظائف إلزامية على مستوى الفئة البنت، لعمل ذلك وجب علينا استبدال الكلمة Overridable في الوظيفة الموجودة بالفئة الرئيسية بالكلمة MustOverride، علما أنه لا يمكننا الإعلان عن

الوظائف بهذه الكلمة إلا إن كانت الفئة الرئيسية فئة مجردة Abstract Class، ماهي الفئة المجردة؟ هذا ما سنتعرف عليه في الفصل التالي.

الفئات المجردة Abstract Classes:

هذا النوع من الفئات يمثل فئات لا يمكن استنساخ أي كائنات منها، ويكون دورها فقط هو السماح باشتقاق فئات منها، فمثلا لو أردنا أن نحصر دور الفئة CLS_PERSON في الوراثة فقط ومنع استنساخ أي كائن منها، نقوم بكتابة الكلمة المحجوزة أما الكلمة MustInherit كما يلي:

```
MustInherit Class CLS_PERSON
    Private name As String
    Private age As Integer

    'CLS_PERSON Class
    Overridable Sub DisplayData()
        MsgBox("الاسم: " & name)
        MsgBox("العمر: " & age)
    End Sub
    Public Property Name_Property As String
        Get
            Return name
        End Get
        Set(value As String)
            name = value
        End Set
    End Property

    Public Property Age_Property As Integer
        Get
            Return age
        End Get
        Set(value As Integer)
            age = value
        End Set
    End Property
```

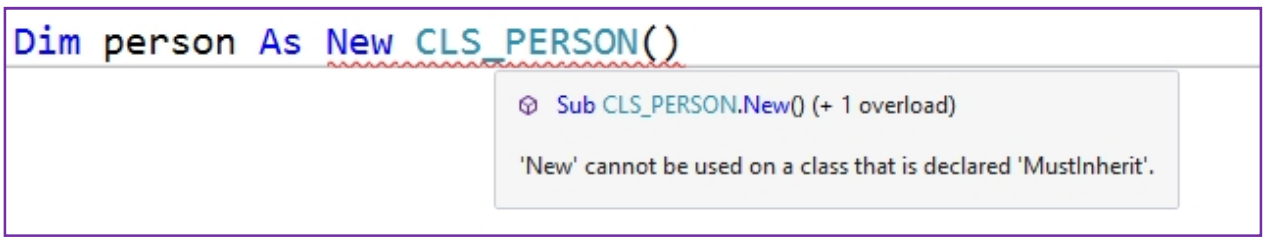
```
    Sub New()

    End Sub

    'CLS_PERSON Class
    Sub New(_name As String, _age As Integer)
        Me.name = _name
        Me.age = _age
    End Sub

    Sub Speak()
        MsgBox("أتكلم أنا")
    End Sub
    Sub Work()
        MsgBox("أعمل أنا")
    End Sub
End Class
```

الآن لو أتيت لأستنسخ كائنا من الفئة CLS_PERSON فلن يسمح لي بذلك لأنها أصبحت مجردة ودورها هو التوريث فقط.



كما تلاحظ معي في رسالة الخطأ أعلاه، فلا يمكن استنساخ الكائنات من الفئات المجردة أي المعلن عنها بالكلمة MustInherit.

كل فئة مشتقة الآن من الفئة CLS_PERSON، صار بإمكانها إعادة تعريف الوظيفة DisplayData()، حتى داخل الفئات الفرعية الممكن اشتقاقها من الفئات المشتقة

يمكننا إعادة تعريف هذه الوظيفة لأننا أعلنا عنها بالكلمة Overridable في الفئة الرئيسية.

إذا أردنا توقيف عملية إعادة التعريف في إحدى الفئات المشتقة وجب علينا استخدام الكلمة NotOverridable وبالتالي لن تستطيع أية فئة فرعية مشتقة إعادة تعريف هذه الوظيفة.

علما أن الكلمة NotOverridable لا يمكن كتابتها إلا داخل الفئات المشتقة وليس داخل الفئة الرئيسية كما يوضح المثال التالي:

```
'CLS_Person Class
Overridable Sub DisplayData()
    MsgBox("الاسم: " & name)
    MsgBox("العمر: " & age)
End Sub

'CLS_EMPLOYEE Class
NotOverridable Overrides Sub DisplayData()
    MyBase.DisplayData()
    MsgBox("الراتب: " & salary)
End Sub
```

الآن لو أتيت ووضعت في الوظيفة DisplayData() الكلمة MustOverride بدل Overridable فإنه سيصبح إجباريا علي إعادة تعريفها على مستوى الفئات البنات على خلاف الكلمة Overridable التي تعني أن إعادة التعريف أمر اختياري.

مع العلم أن استخدام الكلمة MustOverride أمام اسم الوظيفة يقتضي إلغاء كافة الأوامر الخاصة بالوظيفة الرئيسة لأنه سيتم إعادة تعريفها كما يلي:

```
'CLS_Person Class
MustOverride Sub DisplayData()
```

لاحظ أننا لم نكتب End Sub في نهاية الإجراء لأننا سنعيد تعريفه في الفئات المشتقة.

بناء على ذلك فحتى كتابة الأمر التالي MyBase.DisplayData() في الفئات المشتقة يعتبر خطأ لأن الإجراء معلن عنه فقط في الفئة الرئيسية ولا يؤدي أي دور هناك.

تعدد الأشكال Polymorphisme:

نتحدث عن مفهوم تعدد الأشكال حينما يكون لدينا متغير من فئة مشتقة يمكن إسناده إلى متغير من فئة رئيسية، كما يعرض لنا المثال التالي:

```
Dim Person As CLS_PERSON()
Person = New CLS_EMPLOYEE()
```

لاحظ أن المتغير Person من نوع الفئة CLS_PERSON ويؤشر إلى كائن من الفئة CLS_EMPLOYEE، أي أنه يمكننا إسناد أي نوع فرعي لنوع رئيسي وهذا هو مضمون مفهوم تعدد الأشكال، وفيما يلي مثال آخر لتتضح الفكرة في أذهاننا:

```
Sub ChangeInfos(Person As CLS_PERSON)

End Sub

Sub Main()
    Dim P As New CLS_PERSON
    Dim Emp As New CLS_EMPLOYEE

    ChangeInfos(P)
    ChangeInfos(Emp)
End Sub
```

لاحظ معي جيدا أن الإجراء ChangeInfos() ينتظر برامترا من نوع الفئة الرئيسية CLS_PERSON، ومع ذلك عند استدعاء هذا الإجراء نستطيع أن نمرر له متغير من نوع الفئة CLS_EMPLOYEE، وكذلك متغيرات من نوع فئات مشتقة منها، وهذا هو تعدد

الأشكال Polymorphisme ويسمى هذا النوع من تعدد الأشكال بالأنواع الفرعية المتعددة الأشكال SubTyping Polymorphisme.

زيادة التحميل Overloading:

يعتبر هذا المفهوم وجها آخر من وجوه مفهوم تعدد الأشكال، ويعني زيادة التحميل Overloading أنه يمكننا إنشاء مجموعة من الوظائف بنفس الاسم مع اختلاف بنية كل وظيفة عن مثيلاتها (بنية الوظيفة Signature تتمثل في عدد وترتيب وأنواع بيانات البرامترات ولا تشمل نوع الإرجاع)

وقد مر معنا هذا المفهوم ضمنيا حينما كنا نتحدث عن المشيدات، بحيث أنشأنا أكثر من مشيد داخل نفس الفئة لكن ببنية برامترات مختلفة، نفس الكلام يمكننا إسقاطه على الوظائف، وفيما يلي مثال على مفهوم زيادة التحميل Overloading في لغة الفيجوال بيسك دوت نيت:

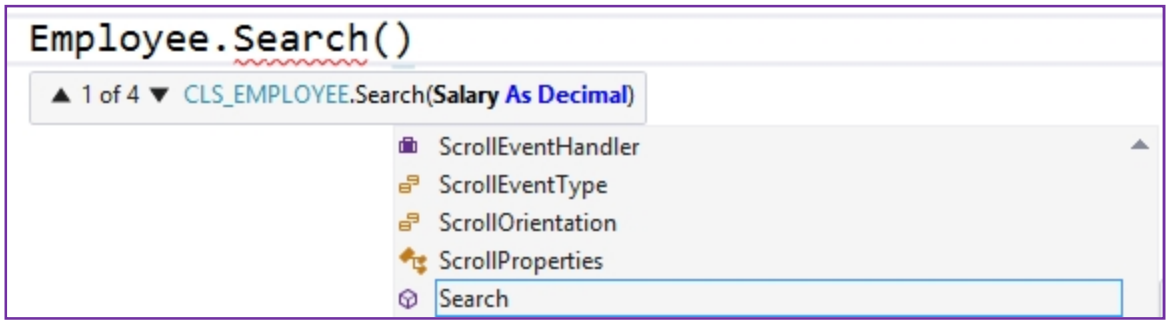
```
    'CLS_EMPLOYEE Class
    Sub Search(Name As String)
        'Serach by name
    End Sub

    Sub Search(Age As Integer)
        'Search by age
    End Sub

    Sub Search(Salary As Decimal)
        'Search by salary
    End Sub

    Sub Search(Name As String, Age As Integer, Salary As
Decimal)
        'Search by all fields
    End Sub
```

كما تلاحظ معي فقد أنشأنا مجموعة من الوظائف كلها تحمل اسم Search() لكن ببنية برامترات مختلفة، الآن لو أتيت وأنشأت كائنا من هذه الفئة وأردت استدعاء الوظيفة Search() سيمنحني أربعة اختيارات كما تعرض الصورة التالية:



زيادة التحميل الذي تعرفنا عليه الآن يشتغل مع الوظائف الخاصة بالفئة، أما إن أردنا أن نطبق هذا المفهوم مع وظائف موجودة في فئة رئيسية مثل الوظيفة Speak() الموجودة في الفئة CLS_Person، لو أردنا زيادة تحميلها داخل الفئة المشتقة CLS_EMPLOYE فعلينا استخدام الكلمة Overloads أمام الوظيفة في الفئة المشتقة كما يعرض الكود التالي:

```
'CLS_PERSON Class
Sub Speak()
    MsgBox("أتكلم أنا")
End Sub

'CLS_EMPLOYEE Class
Overloads Sub Speak(Language As String)

End Sub

Overloads Sub Speak()
End Sub
```

التظليل Shadowing:

كما رأينا فيما تقدم معنا لنتمكن من إعادة تعريف وظيفة ما على مستوى الفئات المشتقة، علينا الإعلان عنها بالكلمة Overridable في الفئة الرئيسية، ثم نعيد تعريفها بواسطة الكلمة Overrides.

أحيانا نريد أن نعيد تعريف بعض الوظائف غير المعلن عنها بالكلمة Overridable، في هذه الحالة إن كان بأيدينا السورس كود الخاص بالفئة الرئيسية، فإنه يستحسن أن نضيف الكلمة Overridable أمام الوظيفة، أما إن لم يكن بمقدورنا عمل ذلك كأن تكون الوظيفة محزمة داخل مكتبة من نوع DLL مثلا، فهنا لا سبيل إلى إعادة تعريفها إلا بواسطة مفهوم التظليل Shadowing، والذي يعني إعادة تعريف وظائف غير معلن عنها Overridable في الفئة الرئيسية.

الكلمة التي نستخدمها في الفئة المشتقة من أجل تظليل وظيفة ما هي Shadows، وهذا مثال عليها:

```
Class Person

    Sub Work()
        'Instructions
    End Sub
End Class

Class Employee
    Inherits Person

    Shadows Sub Work()
        'Instructions
```

```
    End Sub
End Class
```

برمجيا، استخدام إعادة التعريف أفضل من استخدام التظليل، فما دام بإمكانك التعديل على الوظائف الرئيسية فلا تتردد في استخدام مفهوم إعادة التعريف، لأن التظليل لا يحترم تعدد الأشكال، بل ويقوم بإلغاء كافة الوظائف الرئيسية التي تحمل نفس اسم الوظيفة التي تم تظليلها كما توضح قطعة الكود التالية:

```
Dim emp As New Employee
emp.Work() 'Correct
emp.Work("Job Name") 'Error
```

تم قبول الإجراء Work() الأول لأنه النسخة المظللة للوظيفة الرئيسية، وتم رفض الإجراء الثاني علما أن الفئة CLS_EMPLOYEE مشتقة من الفئة CLS_PERSON لأن التظليل يلغي كافة الوظائف التي تحمل نفس الاسم.

ذكرنا كذلك أن التظليل لا يحترم مفهوم تعدد الأشكال، فلو دققت معي في الكود التالي قبل قراءة التعقيب عليه ستلاحظ عن ماذا نتحدث:

```
Dim emp As New Employee
emp.Work()
Dim person As New person
person = emp
person.Work()
```

عند تنفيذ الكود أعلاه، سيتم تنفيذ الإجراء Work() الأول التابع للفئة Employee وهذا عادي جدا، لكن الغريب أننا في السطر الذي أسندنا فيه قيمة الكائن emp

للكائن person المفروض أن الإجراء ()Work الأخير أن يكون التابع للفئة Employee لأن الكائن person أصبح يشير إليها، لكنك ستتفاجأ بأنه سيتم تنفيذ الإجراء ()Work الخاص بالفئة Person دون الاهتمام بمفهوم تعدد الأشكال Polymorphisme وهذا بسبب عملية التظليل Shadowing.

الأحداث Events:

الأحداث هي أفعال تؤدي إلى تنفيذ وظائف معينة، كأن أضغط على زر فيتم تنفيذ إجراء يشغل ملفا صوتيا، أو أقوم بتحديد مربع تكبير الشاشة فيتم استدعاء إجراء يقوم بهذه المهمة.

في لغة الفيجوال بيسك دوت نيت، يمكننا الإعلان عن الأحداث بواسطة الكلمة المحجوزة Events، في المثال التالي سنقوم بإنشاء حدث يتم إطلاقه حينما تتجاوز الكمية المبيعة من منتج ما عدد الكمية المخزنة.

لنفترض أن لدينا الفئة "منتج CLS_PRODUCT" بالشكل الآتي:

```
Class CLS_PRODUCT
    Private Qty_In_Stock As Integer 'الكمية المخزنة
    Private Purchased_Qty As Integer 'الكمية المباعة

    Sub New()
        Qty_In_Stock = 10
        Purchased_Qty = 0
    End Sub
End Class
```

في الفئة أعلاه أنشأنا حقلين، الأول يحدد الكمية المخزنة Qty_In_Stock، والثاني خاص بالكمية المباعة Purchased_Qty، في المشيد نقوم بتصفير الكمية المباعة، وإسناد القيمة 10 للكمية المخزنة.

الآن سننشئ الحدث الذي نريد إصداره حينما تتجاوز الكمية المباعة الكمية المخزنة:

```
Class CLS_PRODUCT
    Private Qty_In_Stock As Integer 'الكمية المخزنة
    Private Purchased_Qty As Integer 'الكمية المباعة

    Public Event OnExceedQty(ByVal Message As String)

    Sub New()
        Qty_In_Stock = 10
        Purchased_Qty = 0
    End Sub
End Class
```

أنشأنا حدثا اسمه OnExceedQty يقوم باستقبال الرسالة التي نود عرضها عند استدعائه.

الآن سنقوم بإنشاء الوظيفة التي تقوم بإطلاق الحدث، وهي وظيفة عادية تنقص الكميه المباعة ب 1 عند كل مبيعة:

```
    Public Sub Purchase()
        Purchased_Qty += 1
        If Purchased_Qty >= Qty_In_Stock Then
            Purchased_Qty = 0
            RaiseEvent OnExceedQty("الكمية المراد بيعها غير متاحة")
        End If
    End Sub
```

عند كل استدعاء لهذه الوظيفة، سيتم زيادة الكمية المباعة ب 1، وفي كل مرة نقوم بمقارنتها مع الكمية المخزنة، فإن ساوتها أو جاوزتها، نقوم بإصدار الحدث OnExceedQty ونزوده بالرسالة المراد استغلالها عند استعمال كائن من هذه الفئة.

سنقوم بإنشاء فورم بسيط لنجرب هذا الحدث، وليكن شكل الفورم كما يلي:

الآن ادخل إلى الكود، وقم بالإعلان عن كائن من الفئة CLS_PRODUCT بحيث تكون طريقة الإعلان مصحوبة بالكلمة WithEvents التي تدل أن هذا الكائن لديه أحداث:

```
    Dim WithEvents product As New CLS_PRODUCT
```

بعد ذلك سنقوم بإنشاء وظيفة تؤشر إلى الحدث OnExceedQty وتقوم بعرض الرسالة الممررة إليه:

```
    Sub CallEvent(ByVal Message As String) Handles
product.OnExceedQty

        MsgBox(Message)

    End Sub
```

الآن بقي لنا فقط أن ندخل إلى الحدث Click الخاص بالزر الذي وضعناه على الفورم، ونقوم باستدعاء الإجراء Purchase() الخاص بعملية البيع الذي يزيد عدد المبيعات إلى أن يتحقق الشرط فيقوم بإصدار الحدث OnExceedQty:

```
    Private Sub btnPurchase_Click(sender As Object, e As
EventArgs) Handles btnPurchase.Click
        product.Purchase()
    End Sub
```

خاتمة

الحمد لله ابتداء وانتهاء على نعمه السوابغ، تم بفضل الله ختم الجزء الثاني من سلسلة الإكليل المتعلق بالبرمجة الكائنية OOP في الفيجوال بيسك.

أسأل الله عز وجل أن يجعله خالصا لوجهه الكريم، وأن ينفع به كل من له به حاجة، يمكنك الحصول على المزيد من الكتب والدروس عبر زيارة مدونتنا الرسمية:

/http://www.mobarmijoun.com

ولا تترددوا في الاشتراك معنا على قناتنا على اليوتيوب:

https://www.youtube.com/ EssaadaniTV

إن بدت لكم أخطاء في الكتاب، أو لديكم أسئلة برمجية أو اقتراحات، فلا تتوانوا في مراسلتنا عبر الايميلات الآتية:

How2progSpace@Gmail.com

Khalid_ESSAADANI@Hotmail.fr

دام لكم البشر والفرح والسلام عليكم ورحمة الله.

خالد السعداني