**الفصل السابع**

**كيف تبني الخلايا الخاصة؟**

**Building Custom Classes**

* [**ما هي الخلية؟:**](file:///C:\Users\Administrator\Desktop\vb.net\c-%20Classes%20الخلايا\07-%20Building%20Classes%20بناء%20الخلايا\07-%20Building%20Classes%20بناء%20الخلايا\0-%20ما%20هي%20الخلية.htm)
* [**إنشاء الخليّة الصغرى Minimal:**](file:///C:\Users\Administrator\Desktop\vb.net\c-%20Classes%20الخلايا\07-%20Building%20Classes%20بناء%20الخلايا\07-%20Building%20Classes%20بناء%20الخلايا\1-%20بناء%20الخليّة%20الصغرى.htm)
* [**تعريف المرقمات Enumerators في الخليّة:**](file:///C:\Users\Administrator\Desktop\vb.net\c-%20Classes%20الخلايا\07-%20Building%20Classes%20بناء%20الخلايا\07-%20Building%20Classes%20بناء%20الخلايا\3-%20تعريف%20المرقمات%20في%20الخليّة.htm)
* [**الأحداث Events:**](file:///C:\Users\Administrator\Desktop\vb.net\c-%20Classes%20الخلايا\07-%20Building%20Classes%20بناء%20الخلايا\07-%20Building%20Classes%20بناء%20الخلايا\4-%20الأحداث%20Events.htm)
* [**الخصائص Properties:**](file:///C:\Users\Administrator\Desktop\vb.net\c-%20Classes%20الخلايا\07-%20Building%20Classes%20بناء%20الخلايا\07-%20Building%20Classes%20بناء%20الخلايا\2-%20الخصائص.htm)
* [**المتغيّرات والخصائص المشتركة Shared Properties:**](file:///C:\Users\Administrator\Desktop\vb.net\c-%20Classes%20الخلايا\07-%20Building%20Classes%20بناء%20الخلايا\07-%20Building%20Classes%20بناء%20الخلايا\5-%20المتغيّرات%20والخصائص%20المشتركة.htm)
* [**إضافة الخليّة كمرجع:**](file:///C:\Users\Administrator\Desktop\vb.net\c-%20Classes%20الخلايا\07-%20Building%20Classes%20بناء%20الخلايا\07-%20Building%20Classes%20بناء%20الخلايا\6-%20إضافة%20الخليّة%20كمرجع.htm)
* [**الوراثة Inheritance:**](file:///C:\Users\Administrator\Desktop\vb.net\c-%20Classes%20الخلايا\07-%20Building%20Classes%20بناء%20الخلايا\07-%20Building%20Classes%20بناء%20الخلايا\7-%20الوراثة%20Inheritance.htm)
* [**تنظيم الخلايا داخل فضاءات الأسماء Namespaces:**](file:///C:\Users\Administrator\Desktop\vb.net\c-%20Classes%20الخلايا\07-%20Building%20Classes%20بناء%20الخلايا\07-%20Building%20Classes%20بناء%20الخلايا\8-%20تنظيم%20الخلايا%20داخل%20فضاءات%20الأسماء.htm)

**ما هي الخلية؟:**

تقع الخلايا Classes في موضع القلب من VS.NET، حتّى لَيمكنُك أن تقول بجرأة: إنّ كل شيء فيها ما هو في النهاية إلا خلية، وإطار العمل .NET Framework نفسه ما هو إلا مجموعة هائلة من الخلايا التي يمكنك استدعاؤها واستخدامها في لغة البرمجة التي تعمل بها.

ليس هذا فحسب، فإنّ النموذج نفسه ما هو إلا خلية، تحتوي على الأدوات والكود الذي يتعامل معها.

وكما قلنا من قبل، إنّ عملية تعريف متغيّر من أيّ نوعٍ، ما هي إلا عمليّة إنشاء نسخة من خلية معيّنة، بحيث تستطيع استخدام كل خصائص ووسائل هذه الخلية عبر هذا المتغيّر.

ولكن... ما رأيك أن نتركنا من كل هذا الكلام، لنتعرف على الخلايا بالتفصيل؟

إنّ الخليّة هي برنامج لا يمكن تنفيذه بمفرده، إذ يجب أن يتمّ استخدامه بواسطة تطبيق آخر أو جزء آخر من نفس التطبيق الذي يحتوي على الخليّة.

ستسألني:

- أتعني أنّنا نستدعي الخليّة كما نستدعي الدوال والإجراءات الفرعيّة، باستخدام تعبير Call مثلا؟

- إطلاقا.. إنّ الخليّة تحتوي أساسا على العديد من الإجراءات والدوال والخصائص الخاصّة بها.

- إذن كيف نستخدم الخليّة؟

- إنّ الخلايا تكاد تكون مماثلة للسجلات Structures، ونحن نتعامل معهما بنفس الطريقة تقريبا.

- تعني أنّنا نعرّف متغيّرا من نفس نوع الخليّة، تماما كما نعرّف متغيرا من نفس نوع السجلّ؟

- بالضبط.. وهذا يعني ببساطة، أنّ الخليّة هي إحدى أنواع المتغيرات التي يعرّفها المستخدم User Defined Types، وإن كانت هي أشملها وأقواها وأهمّها على الإطلاق، كما سنرى بعد قليل.

الآن تدرك بالطبع أنّك أوغلت في استخدام الخلايا، فهي ليست جديدة أو غريبة.. الجديد هو أنّك ستتعلّم هنا كيف تبني الخلايا الخاصّة بك.

ولكن.... لماذا أترجم كلمة Class بكلمة خليّة؟

إنّ كلمة Class في الإنجليزية تعني بالعربيّة "فصل" أو "رتبة" أو "طائفة".. لا أعتقد أنّ أيّا من هذه الكلمات سيكون مستساغا لديك عند قراءته في الترجمة العربيّة.. هنا كان لا بدّ من البحث عن كلمة أخرى مناسبة، تصلح كترجمة غير حرفيّة لهذه الكلمة.

إنّ الغربيّين ـ لا سيّما الأمريكيّين ـ مولعون بنحت المصطلحات بناءً على التشبيهات والكنايات.. والفكرة في اختيار كلمة Class جاءت من وظيفة هذه الوحدة البرمجيّة، فهي تشبه وظيفة الفصائل الحيوانيّة.. فكلّ فصيلة ينتمي إليها آلاف الكائنات التي ترث خصائص هذه الفصيلة، ولكنّ دونَ أن يمنع هذا أن يكون لكلّ كائن خصائصه وسماته المتفرّدة المميّزة له (فلا يتخيّل أن تتماثل القطط مع باقي أعضاء عائلة السنّوريّات كالنمور مثلا، رغم ما بينهما من تشابه.. كما لا يُتخيّل أن تكونَ القطط نفسَها نسخا كربونيّة من بعضها.. لا بدّ أن (مشمش) سيختلف عن (بوسي) في قوّة الشخصيّة والطموح والتذوّق الفنّيّ!).. هذا هو الأمر نفسه الذي يمكنك أن تحاكيه في البرمجة: في البداية يكون لديك خليّة/فصيلة (السنّوريّات مثلا)، قابلة للوراثة Inheritance، حيث يمكن أن ترثها خلايا أخرى (كالنمور والقطط) مع تفرّد كلّ خليّة بسماتها الخاصّة (خصائصها ووسائلها المختلفة، مثل البيئة وطريقة القنص).. هذا مع إمكانيّة تعريف كائنات Objects من خليّة النمور أو خليّة القطط (مشمش وبوسي مثلا)، بحيث يتصرّف كلّ كائن (قط) منها بطريقته الخاصّة، في إطار كونه قطّّا وأحد أفراد عائلة السنوّريّات (نتيجة اختلاف قيم نفس الخصائص التي يمتلكها كلّ منهما، مثل الحجم ولون الفراء).. وسنتناول الوراثة بالتفصيل في نهاية هذا الفصل.

هنا وجدت أن كلمة "خليّة" هي ترجمة مناسبة لمصطلح Class، وذلك لأنّها تحقّق الأمرين التاليين:

1- تصف هذه الوحدة البرمجيّة بدقّة، من حيث كونها عنصرا أساسيّا يدخل في تكوين منظومات أكبر (عن طريق الوراثة أو التراكيب Hierarchies).. ولقد فكّرت في البداية أنّ كلمة "لبِنَة" قد تفي بالغرض، ولكنّي سرعان ما اكتشفت عدم ملاءمتها، لأنّ الخصائص والوسائل والأحداث تعتبر هي اللبنات التي تبني الخلايا، فليست الخليّة هي أصغر وحدة برمجيّة.

2- كلمة خليّة مناسبة لمنظومة الوراثة، فهي لا تخلو من "الحياة"، وتتّسق مع تعبيرات، مثل: الخليّة كذا ترث الخليّة كذا، والكائن كذا نسخة من الخليّة كذا... إلخ.

هذا هو ما اجتهدت فيه، وأتمنّى أن تكون هذه الترجمة مستساغة لك.

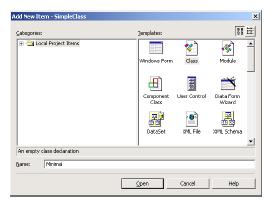
**بناء الخليّةMinimal :**

أوّل خليّة سنبنيها سنسمّيها "الصغرى" Minimal، حيث سنبدأها بأقلّ الإمكانيات، وسنأخذ في زيادة إمكانياتها بالتدريج.

* [**إنشاء مكتبة خلايا Class Liberary:**](file:///C:\Users\Administrator\Desktop\vb.net\c-%20Classes%20الخلايا\07-%20Building%20Classes%20بناء%20الخلايا\1-%20بناء%20الخليّة%20الصغرى\1-%20إنشاء%20مكتبة%20خلايا.htm)
* [**الأعضاء الافتراضيّة Default Members:**](file:///C:\Users\Administrator\Desktop\vb.net\c-%20Classes%20الخلايا\07-%20Building%20Classes%20بناء%20الخلايا\1-%20بناء%20الخليّة%20الصغرى\2-%20الأعضاء%20الافتراضيّة.htm)
* [**إضافة بعض الكود للخليّة:**](file:///C:\Users\Administrator\Desktop\vb.net\c-%20Classes%20الخلايا\07-%20Building%20Classes%20بناء%20الخلايا\1-%20بناء%20الخليّة%20الصغرى\3-%20إضافة%20بعض%20الكود%20لخليتنا.htm)
* [**تخصيص الأعضاء الافتراضيّة Customizing Default Members**](file:///C:\Users\Administrator\Desktop\vb.net\c-%20Classes%20الخلايا\07-%20Building%20Classes%20بناء%20الخلايا\1-%20بناء%20الخليّة%20الصغرى\3-%20تخصيص%20الأعضاء%20الافتراضيّة.htm)

**إنشاء مكتبة خلايا Class Liberary:**

ابدأ مشروعا جديدا وأسمه SimpleClass، وأضف خلية له، بضغط اسم المشروع في نافذة متصفّح المشاريع بزر الفأرة الأيمن، واختيار "إضافة" / "إضافة خلية" Add.Add Class من القائمة الموضعيّة.. في مربع الحوار الذي سيظهر، اختر أيقونة الخليّة Class، واكتب اسمها Minimal.. عند إغلاق مربّع الحوار، سيتمّ إنشاء الملفّ Minimal.vb، الذي سيحتوي على الكود الخاص بهذه الخليّة.



وطبعا لا داعيَ للقول إنّ من حقّك تسمية الخليّة بأيّ اسم تشاء، مع مراعاة أن يكون الاسم واضحا ودالا على وظيفة الخليّة.

ويمكنك أن تضع الخلايا في نفس ملفّ النموذج، أو يمكنك أن تجعل له ملفا مستقلا.. بل إنّ في إمكانك أن تنشئ مشروعا لا يحتوي إلا على خليّة أو أكثر.

ولكن كما ذكرنا من قبل، لا يمكن تشغيل الخليّة بمفردها، وسنحتاج لنموذج لاختبارها بواسطته.

افتح الخليّة، بالنقر عليها في متصفّح المشاريع مرتين بالفأرة.. ستجد مكتوبا بها ما يلي:

**Public Class Minimal**

**End Class**

عُد إلى مصمّم النماذج، وضع زرا على النموذج، واكتب الآتي في حدث ضغطه:

**Dim obj1 As Minimal()**

إنّ هذه الجملة تعرّف الكائن obj1.. ولكن ما هو الكائن؟

إنّه ببساطة نسخة مستقلّة من الخليّة، يتمّ تحميلها في الذاكرة.. ويمكن تعريف أيّ عددٍ من الكائنات من نفس الخليّة، وتغيير خصائص كل منها على حدة، بدون أن تتأثّر باقي الكائنات بهذا التغيير، فكأنّها معزولة تماما عن بعضها.. إلا في حالة واحدة: إذا كانت هناك بعض الخواصّ المشتركة Shared كما سنرى فيما بعد.

**ملحوظة:**

عند تعريف أكثر من كائن من نفس الخليّة، يتمّ تحميل الكود الخاص بالخليّة في الذاكرة مرّة واحدة فقط بحيث يكون عامًّا لكل الكائنات، ثمّ يتم تخصيص مخازن مستقلّة لكل كائن يتم تعريفه، لكتابة قيم خصائصه فيها، بحيث نضمن أن يشغل كل كائن الحدّ الأدنى من المساحة في الذاكرة، في نفس الوقت الذي يحتفظ فيه بخصائصه المستقلّة.

بطريقة أخرى: أيّ كائن ما هو إلا عبارة عن مجموعة من المتغيرات المحلّيّة Local Variables خاصّة به وحده، محمّلة في الذاكرة.

**الأعضاء الافتراضيّة Default Members:**

أضف لحدث ضغط الزرّ ما يلي:

**obj1.**

ستلاحظ بعد كتابة النقطة "." التي تلي اسم المتغيّر ظهور قائمة بأسماء جميع أعضاء الخليّة Minimal، وهي كالتالي:

**- "يساوي" Equals:**

ما لم تقم باستبدالها Overriding، فإنها تكافئ في وظيفتها وظيفة الوسيلة ReferenceEqual.

**- "اقرأ الرمز" GetHashCode:**

تعيد عددا صحيحا، يميّز الكائن عن باقي الكائنات.. وما لم تقم باستبدالها، فإنّها لا تعيد نفس العدد، حتّى للكائنين المتساويين في قيمتيهما.. لذلك عليك أن تقوم باستبدال هذه الوسيلة، بحيث تعيد قيمة أحد الحقول Fields أو الخصائص التي تعرّف الكائن تعريفا متفرّدا Unique ID أو التي تمنحه قيمته، وفي هذه الحالة.. جدير بالذكر أنّ هذه القيمة تفيد في عمليّات البحث في الجداول (مثل الجدول المختلط HashTable) التي تحتوي على عدد ضخم من الكائنات.. وهناك خوارزميات كثيرة لإنتاج هذه الرموز المختلطة بطريقة عشوائيّة Hashing، بحيث تسرّع عمليّات البحث، ولكن لا مجال هنا للحديث عن هذا.

**- "اقرأ النوع" GetType:**

تعيد قيمة من النوع Type، تصف نوع الكائن.. استخدم هذه القيمة كمعامل للكثير من الدوال التي تطلب معاملات من النوع System.Type.

**- "تساوي المرجع" ReferenceEqual:**

(ويكافؤ استخدامها استخدام المعامل Is لفحص تساوي كائنين مرجعيا).

**- "حوّل إلى نصّ" Tostring:**

وهي أشهر من أيّ ممثّل حقير في مصر!!

**ملحوظة:**

لسبب ((مجهول))، قد لا تظهر بعض هذه الأعضاء في القائمة، ولكنّك ما زلت تستطيع استخدامها في الكود.. عامة، يمكنك ضبط خصائص بيئة التطوير، لعرض هذه الخصائص حيث يمكنك أن تعرض كلّ خصائص ووسائل الكائن في قائمة الأعضاء، باتّباع التالي:

افتح القائمة Tools واضغط الأمر Options.. في النافذة التي ستظهر لك، اضغط العنصر Text Editor، ومن عناصره الفرعيّة اختر Basic ومن عناصره الفرعيّة اختر General.. على اليسار ستجد مربّعات اختيار.. أزل العلامة من الاختيار التالي "إخفاء الأعضاء المتقدّمة" Hide Advanced Members.

طبعا ستتساءل: من أين أتت هذه الوسائل، مع أنّنا لم نكتب أيا منها في خليتنا بعد؟

إنّها وسائل عامّة لأي خليّة، نظرا لأنّها ترث تلقائيّا، خليّة الكائن Object Class، وهذه هي أعضاؤها.. ولتجربة هذه الوسائل، اكتب ما يلي في حادثة ضغط الزر:

**Dim obj1 As New Minimal()**

**Console.WriteLine(obj1.Tostring)**

**Console.WriteLine(obj1.GetType)**

**Console.WriteLine(obj1.GetHashCode)**

**Dim obj2 As New Minimal()**

**Console.WriteLine(obj1.Equals(obj2))**

**Console.WriteLine(Minimal.ReferenceEquals(obj1, obj2))**

في شاشة المخرجات Output window ستظهر لك العبارات التالية:

**SimpleClass.Minimal**

**SimpleClass.Minimal**

**18**

**False**

**False**

تلاحظ أنّ الوسيلتين Tostring و GetType تُرجعان نفس النصّ، بينما تصادف أنَّ رمز هذا المتغير هو 18.. ولكن لماذا لم يتساوَ المتغيران obj1 و obj2 رغم أنّه لا يوجد ما يوحي باختلافهما؟

لا تنسَ أبدًا أنّنا نتحدّث عن كائنين مختلفين.. فلكي يكون المتغيران متساويين، يجب أن يُشيرا إلى كائن واحد.. أضف السطر التالي للكود السابق، بعد جملة تعريف المتغير obj2:

**obj2 = obj1**

الآن جرّب البرنامج.. ستجد أنّ الجملة الأخيرة ستطبع كلمة "True" في نافذة المخرجات، وهو شيء متوقّع، فلقد صار المتغيران يشيران لنفس الكائن، وأيّ تغيير في أحدهما سيتأثّر به الآخر.. لقد صارا متساويين.

لاحظ أنّ عمليّة التساوي هنا مرجعيّة، بمعنى أنّ المتغيّر obj2 سيأخذ نفس عنوان الكائن obj1 في الذاكرة، فهما أشبه ما يكون برجل له لقبان (كائن له متغيران يشيران إليه).

**إضافة بعض الكود لخليتنا:**

سنضيف الآن خاصيتين لخليتنا:

- property1: وهي نصّيّة String.

- property2: وهي رقم مزدوج Double.

إنّ أبسط طريقة تعرّف بها خاصيّة للخليّة، هي تعريفها كمتغيّر عام Public، بحيث يمكن استخدامها من خارج الخليّة.. هذه الطريقة ليست أفضل الطرق، وسنعرف بعد قليل كيف نعرّف خصائص حقيقيّة للخليّة.

اكتب ما يلي داخل مقطع الخليّة:

**Public Property1 As String, Property2 As Double**

تعالَ كذلك نعرّف وسيلتين للخليّة:

- الدالة "عكس النصّ" ReverseString: وتقوم بعكس ترتيب الحروف المكتوبة في الخاصيّة property1.

- الدالة "عكس الرقم" NegateNumber: وتقوم بعكس إشارة الرقم الموجود في الخاصيّة property2.

أيضا يجب أن تكون هاتان الوسيلتان عامتين Public، حتّى يمكن استخدامهما من خارج الخليّة.

ها هو ذا تعريف الدالتين:

**Public Class Minimal**

**Public Property1 As String, Property2 As Double**

**Public Function ReverseString() As String**

**Return (StrReverse(Property1))**

**End Function**

**Public Function NegateNumber() As Double**

**Return (-Property2)**

**End Function**

**End Class**

ولاختبار ما فعلناه، أضف زرا جديدا للنموذج، واكتب ما يلي في حدث ضغطه:

**Dim obj As Minimal**

**obj = New Minimal()**

**obj.Property1 = "ABCD"**

**obj.Property2 = 999999**

**Console.WriteLine(obj.ReverseString)**

**Console.WriteLine(obj.NegateNumber)**

**تخصيص الأعضاء الافتراضيّة Customizing Default Members:**

قلنا إنّ هناك أعضاءً افتراضيّة لأيّ خليّة، تمنحها لك VS بدون أن تكتبها أنت.. ولكن في معظم الأحوال، ستجد أنّك تضيق ذرعا بهذه الأعضاء، حينما تجد أنّها لا تؤدّي ما تريده منها بالضبط.

مثلا: الدالة ToString تُرجع اسم الخليّة، والدالة Equals تتأكّد من تساوي خليتين مرجعيا.

فماذا لو أردت تعديل الدالة Tostring، لترجع أيّ نصّ آخر تريده؟

في هذه الحالة يمكنك كتابة دالة جديدة بنفس الاسم، مع استخدام كلمة "تهيمن أو تستبدل" Overrides ليتم استخدامها بدلا من الدالة الأصليّة:

**Public Overrides Function Tostring() As String**

**Return("MINIMAL: " & tBirthDate.Tostring)**

**End Function**

والآن جرّب ما يلي:

**Dim obj1 As New Minimal()**

**Obj1.BirthDate = #1/1/1963#**

**Dim obj2 As New Minimal()**

**Obj2.BirthDate = #12/31/1950#**

**Console.WriteLine(obj1.ToString)**

**Console.WriteLine(obj2.ToString)**

سيظهر لك ما يلي في نافذة المخرجات:

**MINIMAL: 1963-01-01 00:00:00**

**MINIMAL: 1950-12-31 00:00:00**

تلاحظ أنّ الوسيلة ToString تُرجع قيمةً مختلفةً، تبعا للنسخة (الكائن) التي تنتمي إليها.

وأريد هنا أن أحذّرك من شيء خطير:

عند تخصيص الأعضاء الافتراضيّة، التزم بحالات الأحرف.. فمثلا: عند تخصيص دالة التحويل إلى نصّ، لا تكتب اسم الدالة الخاصة بك toString.. التزم بكتابتها ToString.

صحيح أنّ VB غير حسّاس لحالة الأحرف، ولكن لغة ميكروسوفت الوسيطة MSIL التي يترجم إليها كود VB (وكود باقي لغات VS) حسّاسة لحالة الأحرف.. لهذا فقد تنشأ حالات شاذّة لا يعمل فيها كودك بطريقة صحيحة، بسبب اختلاف حالة الأحرف!

إنّ ذلك لن يؤثر في المثال السابق.. ولكن لو جرّبت الكود التالي، فستجد أنّه لا يعمل بالطريقة المتوقّعة.. عرّف الدالة toString مرّتين.. واحدة تمتلك معاملا Overloading، والأخرى تستبدل الدالة الأصليّة Overriding:

**Public Overloads Function toString(ByVal X As Integer) As String**

**Return ("MINIMAL: " & tBirthDate.Tostring)**

**End Function**

**Public Overloads Overrides Function ToString() As String**

**Return ("MINIMAL: " & tBirthDate.Tostring)**

**End Function**

الآن لو جرّبت الكود التالي في حدث تحميل نموذج الاختبار، فستكتشف أنّ الدالة ToString ما زالت تعطي نفس النتيجة القديمة، وكأنّنا لم نخصّصها!

**Dim X As Object = New Minimal()**

**Console.WriteLine(X.ToString())**

ولكن.. لماذا لم يقم محرّر الكود في VB بتغيير الاسم toString تلقائيا ليصبح ToString، كمت تعوّدنا منه؟

بصراحة لا أدري.. ربّما تكون ميكروسوفت قد تلافت ذلك في إصدار VS.Net 2003.. أو ربّما لم تنتبه لهذا بعد، نظرا لأنّ محرّر الكود يقارن الأسماء في نطاق الكود المكتوب فقط، بينما الدالة الأصليّة هنا موجودة ضمن خليّة الكائن Object، فلا مجال ليقارن حالة الأحرف!

عموما، أرح نفسك من هذه المتاهة، والتزم بحالة الأحرف عند تخصيص الأعضاء الافتراضيّة، وعند وراثة أيّ خليّة لا يوجد الكود الخاص بها في مشروعك.. خلاص؟

والآن نريد كتابة دالة التساوي الخاصّة بنا، لنختبر بها تساوي كائنين من عدمه، بدلا من دالة التساوي الافتراضيّة، التي تتحقّق ممّا إذا كان متغيران يشيران إلى نفس الكائن أم لا.

وفي حالتنا هذه، سنعتبر الكائنين متساويين إذا تساوت قيمتا خاصية تاريخ الميلاد فيهما:

**Public Overloads Function Equals(ByVal obj As Object) As Boolean**

**Dim O As Minimal = CType(obj, Minimal)**

**If O.BirthDate = tBirthDate Then**

**Equals = True**

**Else**

**Equals = False**

**End If**

**End Function**

**ملحوظة:**

لعلّك تتساءل عن سبب استخدامنا لكلمة "تعدد التعريفات" Overloads في هذا التعريف، بدلا من كلمة "تهيمن" Overrides!

إنّ هذا يرجع لأنّ للدالة Equals تعريفين، يختلفان في عدد المعاملات، ممّا يعني أنّ بإمكاننا استخدام الصيغتين معا.

وستضطر للّجوء لفعل مشابه، إذا كانت الدالة الخاصّة بك مختلفةً عن الدالّة الأصليّة في عدد المعاملات أو أنواعها.

عامّةً لا تقلق بخصوص هذا الأمر، فبإمكانك أن تستخدم كلمة Overloads، فإذا اعترضَ VB وظهر خطّ متعرّج تحت الجملة، فاستخدم كلمة Overrides!

أو بطريقة أفضل، اكتب جملة في محرّر الكود، تستخدم فيها الدالة، وحلّق فوق اسمها بالفأرة.. ما هي إلا لحظات ويظهر لك تلميح يبيّن لك صيغة تعريفها.. في هذه الصيغة ستجد كلمة Overloads أو كلمة Overrides.. استخدم الكلمة التي تجدها لتعريف الصيغة الجديدة للدالة.

أمّا لو شئت أفضل طرية على الإطلاق، فهي استخدام ملفّات الإرشاد Help Files لتتعرّف على صيغة تعريف الدالة.

عموما، انتظر حتّى تتعرّف على الوراثة في نهاية هذا الفصل، وستتضح لك الصورة أكثر.

والآن استخدم الكود التالي لاختبار دالة التساوي الجديدة:

**Dim O1 As New Minimal()**

**Dim O2 As New Minimal()**

**O1.BirthDate = #3/1/1960#**

**O2.BirthDate = #3/1/1960#**

**O1.Property1 = "object1"**

**O2.Property1 = "OBJECT2"**

**If O1.Equals(O2) Then**

**MsgBox("إنهما متساويان")**

**End If**

**تدريب 2:**

هناك خطأ محتمل في دالة التساوي التي عرّفناها.. هل تستطيع توقعه؟

**إجابة تدريب2:**

نظرا لأنّ معامل هذه الدالة كائن عامّ Object، فمن المحتمل ألا يكون من النوع Minimal، وفي هذه الحالة سيحدث خطأ.. ويمكن معالجة ذلك كالتالي:

**Public Overloads Function Equals(ByVal obj As Object) As Boolean**

**Dim O As New Minimal()**

**Try**

**O = CType(obj, Minimal)**

**Catch typeExc As InvalidCastException**

**Throw typeExc**

**Exit Function**

**End Try**

**If O.BirthDate = tBirthDate Then**

**Equals = True**

**Else**

**Equals = False**

**End If**

**End Function**

لاحظ أنّ "تحويل غير مسموح به" InvalidCastException هو نوع الاستثناء الذي ينتج من دالة التحويل CType عندما لا تتمكّن من التحويل إلى النوع الذي تريده.

وهناك حلّ آخر، فبمنتهى البساطة، يمكنك تعريف معامل الدالة من النوع Minimal مباشرة، وإراحة نفسك!

**Public Overloads Function Equals(ByVal obj As Minimal) As Boolean**

**If obj.BirthDate = tBirthDate Then**

**Equals = True**

**Else**

**Equals = False**

**End If**

**End Function**

**تعريف المرقمات Enumerators في الخليّة:**

عرّف المرقّم التالي في بداية الخليّة:

**Public Enum AgeGroup**

**Baby**

**Child**

**Teenager**

**Adult**

**Senior**

**Overaged**

**End Enum**

والآن سنضيف لخليتنا الوسيلة "اقرأ مجموعة العمر" GetAgeGroup، لمعرفة المرحلة العمريّة التي ينتمي إليها الشخص.. شيءٌ طبيعيٌّ إذن أن يكون نوع هذه الخاصيّة هو المرقّم AgeGroup:

**Public Function GetAgeGroup() As AgeGroup**

**Dim group As AgeGroup**

**Select Case tAge**

**Case Is < 5 : Return (group.Baby)**

**Case Is < 12 : Return (group.Child)**

**Case Is < 21 : Return (group.Teenager)**

**Case Is < 65 : Return (group.Adult)**

**Case Is < 100 : Return (group.Senior)**

**Case Else : Return (group.Overaged)**

**End Select**

**End Function**

ونظرًا لأنّ المرقّم قد تم تعريفه بكلمة Public، فمن الممكن استخدامه من أيّ مكان في التطبيق من خلال الخليّة Minimal.

والآن أضف زرا جديدا للنموذج، وجرّب ما يلي:

**Protected Sub Button3\_Click(ByVal sender As Object, \_**

**ByVal e As System.EventArgs)**

**Dim obj As Minimal**

**obj = New Minimal()**

**obj.BirthDate = #2/9/1932#**

**Console.WriteLine(obj.Age)**

**Dim discount As Single**

**Select Case obj.GetAgeGroup**

**Case Minimal.AgeGroup.Baby, Minimal.AgeGroup.Child**

**discount = 0.4**

**Case Minimal.AgeGroup.Senior**

**discount = 0.5**

**Case Minimal.AgeGroup.Teenager Then**

**discount = 0.25**

**End Select**

**Console.WriteLine(discount)**

**End Sub**

**الأحداث Events**

برغم عدم شيوعه، إلا إنّ بإمكانك إطلاق الأحداث من داخل الخليّة، لتخبر البرنامج بحدوث شيء ما، أو بتغيّر خاصيّة معيّنة.. كما يمكنك إطلاق بعض الأحداث على فترات زمنيّة منتظمة، أو في مواعيد معيّنة.

* [**إطلاق الأحداث Firing Events:**](file:///C:\Users\Administrator\Desktop\vb.net\c-%20Classes%20الخلايا\07-%20Building%20Classes%20بناء%20الخلايا\4-%20الأحداث%20Events\1-%20إطلاق%20الأحداث.htm)
* [**استخدام الأحداث من خارج الخليّة:**](file:///C:\Users\Administrator\Desktop\vb.net\c-%20Classes%20الخلايا\07-%20Building%20Classes%20بناء%20الخلايا\4-%20الأحداث%20Events\2-كيف%20نستخدم%20هذا%20الحدث%20من%20خارج%20الخليّة.htm)
* [**تعريف معاملات للحدث:**](file:///C:\Users\Administrator\Desktop\vb.net\c-%20Classes%20الخلايا\07-%20Building%20Classes%20بناء%20الخلايا\4-%20الأحداث%20Events\3-%20تعريف%20معاملات%20للحدث.htm)
* [**أحداث التقدّم في أداء العمليات Progress Events:**](file:///C:\Users\Administrator\Desktop\vb.net\c-%20Classes%20الخلايا\07-%20Building%20Classes%20بناء%20الخلايا\4-%20الأحداث%20Events\4-%20أحداث%20التقدّم%20في%20أداء%20العمليات.htm)

**إطلاق الأحداث Firing Events:**

تعالَ نضيف حدثا لخليتنا يتمّ إطلاقه في الخامسة عصرا.. في هذه الحالة لا بدّ أن نضيف إليها منبّها Timer control.. استخدم التعبير التالي في بداية الخليّة:

**Dim WithEvents Tmr As System.Timers.Timer**

وطبعا كلمة "بالأحداث" WithEvents ليست جديدة علينا، ونحن نعرف من قبلُ أنّها تستخدم في تعريف أيّ أداةٍ أو خليّة نريد برمجة أحداثها.

وفي الوضع الافتراضيّ، يكون المنبّه مغلقًا، ولحثّه على العمل، يجب تغيير خاصيّة "فعّال" Enabled إلى "صواب" True، هذا بالإضافة لاستخدام خاصيّة "المدى الزمنيّ" Interval.. إنّ للمنبّه حدثا واحدا هو "انقضت المدّة" Elapsed، وهو ينطلقُ تلقائيّا، كلّما مرّت الفترة الزمنيّة المحدّدة في خاصيّة المدى الزمنيّ، وهي تقاس بالميللي ثانية (1/1000 من الثانية).

ولكن أين سنضع هذه القيم الابتدائيّة للمنبّه؟

إنّ أنسبَ مكانٍ لفعل ذلك، هو الحدث "جديد" New()، الذي ينطلق كلّما تم إنشاء نسخة جديدة من الخليّة.

**Public Sub New()**

**Tmr = New Timers.Timer()**

**Tmr.Interval = 10000 ' كل عشر ثواني**

**Tmr.Enabled = True**

**End Sub**

الآن سينطلق الحدث Elapsed كلّ عشر ثوان، وعلينا كتابة كوده.. وكل ما سنفعله، هو أن نتأكّد إذا ما كانتِ الساعة الخامسة عصرا، فلو كانت كذلك يمكننا إطلاق حدث ما، وليكن هذا الحدث هو "وقت احتساء الشاي" TeaTime.. فلنعرّف هذا الحدث أولا.. أضف هذه الجملة لبداية الخليّة:

**Public Event TeaTime(ByVal sender As Object)**

تلاحظ أنّنا جعلنا للحدث معاملا واحدا، هو المعامل Sender، حيث سنستخدمه لتمرير مرجعا يشير لنسخة الخليّة Minimal التي أطلقت هذا الحدث.. وبإمكانك أن تعرّف أيّ عدد من المعاملات الإضافيّة لأحداثك من أيّ نوع من أنواع البيانات.

والآن تعالَ نكتب الكود الذي يُطلقُ هذا الحدث، في أيّ وقت ما بين الخامسة والخامسة ودقيقتين:

**Private Sub Tmr\_Elapsed(sender As Object, e As \_**

**System.Timers.ElapsedEventArgs) Handles Tmr.Elapsed**

**Dim T As Date**

**' الجملة التالية تحسب تاريخ اليوم في الساعة الخامسة عصرا (الساعة 17)**

**T = DateAdd( DateInterval.Hour, 17, System.DateTime.Today)**

**If DateDiff(DateInterval.Second, Now(), T) < 120 Then**

**Tmr.Enabled = False**

**RaiseEvent TeaTime(Me)**

**End If**

**End Sub**

**تدريب3:**

قبل إطلاق الحدث، أغلقنا المنبّه.. هل تستطيع تخيّل لماذا؟

**ولكن كيف نستخدم هذا الحدث من خارج الخليّة؟**

لا جديد.. استخدم كلمة "بالأحداث" في تعريف متغير من هذه الخليّة:

**Dim WithEvents TimerObj As New Minimal**

وستجد أنّ الاسم TimerObj سيظهر لك في القائمة المنسدلة اليسرى في نافذة محرر الكود.. اضغط هذا الاسم، وستجد أن الحدث TeaTime سيظهر في القائمة المنسدلة اليمنى.

جرّب ما يلي في حدث الشاي:

**Private Sub Timerobj\_TeaTime(sender As Object) \_**

**Handles Timerobj.TeaTime**

**MsgBox("حان وقت الشاي!")**

**End Sub**

**تعريف معاملات للحدث:**

ليس هناك جديد أيضا، فتعريف معاملات الحدث يتمّ مثلما يتم تعريف معاملات الإجراء:

**Public Event Ev(ByVal A As String, ByVal B As Integer, ByRef C \_**

**As Single)**

ولإطلاق هذا الحدث استخدم كودا كالتالي:

**Dim X As Single = 1**

**RaiseEvent Ev("تجربة", "المعاملات", X)**

لاحظ أن المعامل الثالث تم تعريفه مرجعيا ByRef.. معنى هذا أن الإجراء الذي يستجيب للحدث يمكن أن يغيّره، بحيث يمكنك أن ترى هذا التغيير.. لقد أرسلنا المتغير X وقيمته 1، وربما يتغيّر.. لهذا يمكن أن نكتب الكود التالي بعد السطر الذي يطلق الحدث:

**If X <> 1 Then**

**' أي كود مناسب لك**

**End If**

**أحداث التقدّم في أداء العمليات Progress Events:**

عندما تقوم الخليّة بأداء مهمّة طويلة، يجب إطلاق حدث يدل على مدى التقدّم في أداء هذه العمليّة، بحيث يشعر المستخدم بالاطمئنان أنّ العمليّة ما زالت مستمرّة (فقد يظنّ أنّ البرنامج عالق أو متوقّف عن الاستجابة)، كما يفضّل أن تمنح في هذا الحدث معاملا مرجعيًّا، يمكن بواسطته إلغاء العمليّة.

ويمكنك إطلاق هذا الحدث بإحدى طريقتين:

- إذا كنت تعرف الوقت الذي ستستغرقه العمليّة، يمكنك إطلاق هذا الحدث كلما مضى 5% أو 10% من الوقت.

- إذا كانت العمليّة عبارة عن جملة تكرارية طويلة، يمكنك إطلاق الحدث بعد كل بضع لفات من لفات الجملة التكراريّة.

ابدأ مشروعا جديدا، وأضف له خليّةً أسمها Progress.. في هذه الخليّة سنعرّف الإجراء Process، وفيه سنقوم بكتابة جملة تكرارية طويلة لا هدف لها سوى إيضاح الفكرة، ومنها سنطلق الحدث Action، ليخبرنا معامله الأوّل بالنسبة المئويّة للتنفيذ، وسيكون معامله الثاني مرجعيًّا، ليمكننا من إلغاء العمليّة لو أردنا:

**Public Class Progress**

**Public Event Action(ByVal Percentage As Integer, ByRef \_**

**Cancel As Boolean)**

**Public Sub Process()**

**Dim Cancel As Boolean, I As Integer**

**For I = 0 To CInt(10 ^ 8)**

**If I Mod 10 ^ 6 = 0 Then**

**RaiseEvent Action(CInt(I/10^6), Cancel)**

**If Cancel = True Then Exit For**

**End If**

**Next**

**End Sub**

**End Class**

ولاختبار هذه الخليّة، عرّف ما يلي في بداية النموذج:

**Dim WithEvents Test As New Progress()**

**Dim CancelAction As Boolean ' سنجعل قيمة هذا المتغير صواب عند ضغط زر الإيقاف**

أضف زرين على النموذج اكتب على أحدهما "ابدأ" وعلى الآخر "توقّف".. في زر البدء اكتب ما يلي:

**CancelAction = False**

**Test.Process()**

وفي زر الإيقاف اكتب ما يلي:

**CancelAction = True**

ويجب أن يبدو الحدث Action الخاص بالمتغيّر Test كما يلي:

**Private Sub Test\_Action(ByVal Percentage As Integer, ByRef \_**

**Cancel As Boolean) Handles Test.Action**

**Text = Str(Percentage) ' تغيير عنوان النموذج ليعرض النسبة المئويّة للعمليّة**

**' لن تستطيع ضغط زر الإيقاف بدون الجملة التالية**

**Application.DoEvents()**

**' فهي تخبر البرنامج بأن يترك لوهلة العمليّة التي تشغله،**

**' ليستجيب لباقي أحداث المستخدم، قبل أن يعود لمواصلة ما كان فيه**

**Cancel = CancelAction ' تعتمد قيمة هذا المتغير على ضغط زر الإيقاف**

**End Sub**

**إجابة** [**تدريب3**](#تدريب3)**:**

نظرًا لأنّنا كتبنا البرنامج ليطلق الحدث ما بين الخامسة والخامسة ودقيقتين، والمنبّه ينطلق كل عشر ثوانٍ، لهذا فمن الممكن أن يتمّ إطلاق الحدث 12 مرّة، ونحن لا نريد سوى إطلاقه مرّة واحدة.. لهذا يجب إغلاق المنبّه عند إطلاق الحدث للمرّة الأولى.. طبعا ستتبقّى مشكلة: كيف سنشغّل المنبّه مرّة أخرى؟

عامّة ليس من المتوقّع أن يترك المستخدم أي برنامج عاملا طوال 24 ساعة كاملة.. لا بدّ أنّه سيغلق الجهاز، ويعيد فتحه.. وعندما يشغّل البرنامج الذي يستخدم هذه الخليّة، سيتمّ تعريف نسخة جديدة منها، وبهذا يتمّ تشغيل المنبّه.

طبعا لا بدّ أن تبحث عن حلّ حقيقيّ لو كان الحدث يُطلقُ في فترات زمنيّة أكثر تقاربا (كل ساعة مثلا)، وفي هذه الحالة لديك حلان:

1- أن تجعل فترة السماح لإطلاق الحدث أقلّ من 10 ثوانٍ (بدلا من دقيقتين)، مع ملاحظة أنّ انشغال الجهاز في أيّ عمليّة طويلة يمكن أن يعطّل المنبّه، وبذلك يمكن ألا ينطلق الحدث.

2- أن تضع منبها آخر Tmr2 مداه الزمني أكبر من 120 ثانية (وليكن 121 ثانية)، يتمّ تشغيله عند إيقاف المنبّه الأوّل:

**Tmr.Enabled = False**

**Tmr2.Enabled = True**

وفي حدث المنبّه الثاني تكتب ما يلي:

**Tmr.Enabled = True**

**Tmr2.Enabled = False**

**الخصائص Properties**

سنتعرّف الآن على مقطع جديد من المقاطع البرمجيّة، يعتبر بمثابة نوع من أنواع الإجراءات Procedures مثله مثل الإجراءات الفرعيّة Sub Routines والدوال Functions.. هذا المقطع هو الخاصيّة Property:

* [**الخاصيّة والمتغيّر:**](file:///C:\Users\Administrator\Desktop\vb.net\c-%20Classes%20الخلايا\07-%20Building%20Classes%20بناء%20الخلايا\2-%20الخصائص\1-%20الخاصيّة%20والمتغيّر.htm)
* [**تعريف الخصائص:**](file:///C:\Users\Administrator\Desktop\vb.net\c-%20Classes%20الخلايا\07-%20Building%20Classes%20بناء%20الخلايا\2-%20الخصائص\2-%20تعريف%20الخصائص.htm)
* [**إطلاق الاستثناءات Raising Exceptions من داخل الخصائص:**](file:///C:\Users\Administrator\Desktop\vb.net\c-%20Classes%20الخلايا\07-%20Building%20Classes%20بناء%20الخلايا\2-%20الخصائص\3-%20إطلاق%20الاستثناءات.htm)
* [**الخواص التي يمكن قراءتها فقط Read-Only Properties:**](file:///C:\Users\Administrator\Desktop\vb.net\c-%20Classes%20الخلايا\07-%20Building%20Classes%20بناء%20الخلايا\2-%20الخصائص\4-%20الخواص%20التي%20يمكن%20قراءتها%20فقط%20Read.htm)

**الخاصيّة والمتغيّر:**

قلنا إنّ تعريف خاصيّة كمتغير عامّ Public ليس الحلّ الأمثل، وذلك للآتي:

- هذه الطريقة لا تسمح لك بالتأكّد من القيمة الموضوعة في الخاصيّة.. افترض أنّك تريد أن تجعل الخاصيّة لا تقبلُ إلا الأرقام الأكبر من 100 مثلا.. لن تستطيع فعل هذا عن طريق المتغير العامّ.

- هذه الطريقة لا تسمح لك باتخاذ أي رد فعل يناسب قيمة الخاصيّة، فمثلا: لو كانت لديك خاصيّة تحدّد لون خلفية البرنامج، فإنّ من المفروض أن يتغيّر لون خلفيّة البرنامج عند تغيير هذه الخاصيّة.. لن يمكنَك فعل هذا عن طريق المتغيّر العامّ.

**ملحوظة لمبرمج VB6:**

صار بإمكانك تعريف متغيّر للقراءة فقط، كالتالي:

**Public ReadOnly X As Integer**

وبذلك لم تعُد هذه السمة ممّا يميّز الخصائص عن المتغيّرات.

وطبعا ستتساءل في استغراب:

- ولماذا أضع في الخاصيّة قيمة خاطئة؟.. ولماذا أحاول أن أغير قيمة خاصيّة من المفروض عليّ أن أقرأها فقط؟.. ولماذا لا أعرّف دالة تُغيّر لون الخلفيّة وأرسل لها اللون كمعامل، بدلا من الاعتماد على الخصائص؟

معك حقّ.. هذا بافتراض أنّك تكتب برنامجك بنفسك ولنفسك، ففي هذه الحالة أنت تفهم كل مشاكلك وكيف تحلّها.. ولكن لو كنت تعمل في فريق، والخلايا التي تكتبها ستستخدم في عشرات البرامجِ لسنوات طويلة، في هذه الحالة يجبُ أن تكتبُ كودك بطريقة قياسيّةٍ منظّمة، بحيث يمكن لأيّ فردٍ من أفراد الفريق استخدامه في أيّ وقت، بدون البحث عنك لكي تشرح له الخدع المعقّدة التي يحتاج إليها لاستخدامه!.. كما أنّك أنت نفسك ستجد صعوبةً في تذكّر هذه الخدع عند استخدامك لهذه الخلايا بعد سنواتٍ!

إذن ما هو الحلّ؟

إنّ VB يمنحك طريقةً سهلةً لتعريف الخصائص، تسمح لك بمعرفة متى يتمّ تغيير قيمة الخاصيّة، ومتى تتمّ قراءة قيمتها، بحيث يمكنك كتابة الكود الذي تريده في هذين "الحدثين".

**تعريف الخصائص:**

للخاصيّة إجراء يختلف قليلا عن الإجراء الفرعيّ والدوال، حيث يتكوّن من مقطعين:

- Get: يتم استدعاؤه عندما تتم قراءة قيمة الخاصيّة في أي موضع من التطبيق.

- Set: يتم استدعاؤه عندما يتم تغيير قيمة الخاصيّة في أي موضع من التطبيق.

وبهذه الطريقة يمكنك التحكّم تماما في وظيفة الخصائص وقيمها، فمثلا عند تغيير قيمة الخاصيّة، يمكنك التحقّق من صلاحية القيمة الجديدة، فإن كانت صالحةً، تقوم بتخزين قيمتها في متغيّر خاصّ بالخليّة Private ـ وذلك لأنّ إجراء الخاصيّة في حدّ ذاته لا يحتفظ بأيّ قيم ـ كما يمكنك تنفيذ أي كود مرتبط بهذه القيمة، كتغيير لون الخلفيّة مثلا.

وعند قراءة قيمة الخاصيّة، يمكنك أن تمرّر للخاصيّة قيمة المتغيّر الخاصّ الذي تحفظ فيه قيمتها.

وفي المثال التالي، ترى كيف يمكن تعريف خاصيّة "العمر" Age.. اكتب ما يلي في الخليّة Minimal:

**Private tAge As Integer**

**Property Age() As Integer**

**Get**

**Age = tAge ' إرجاع قيمة الخاصيّة للموضع الذي تتم قراءتها فيه**

**End Get**

**Set (ByVal Value As Integer)**

**' التحقّق من صلاحية قيمة الخاصيّة**

**If Value < 0 Or Value >= 125 Then**

**MsgBox("يجب أن يكون العمر موجبا وأصغر من 126")**

**Else**

**tAge = Value ' حفظ قيمة الخاصيّة في المتغيّر الخاصّ**

**End If**

**End Set**

**End Property**

واضح هنّا أنّنا استخدمنا المتغيّر الخاصّ tAge لحفظ قيمة الخاصيّة، وطبعا هو خاصّ، لأنّنا لا نريد لأي كود من خارج الخليّة أن يراه أو يغيّره.

واضح كذلك أنّ المعامل Value هو الذي يحمل القيمة التي يحاول المبرمج وضعها في الخاصيّة.

والآن عندما تكتب في تطبيقك سطرا كالتالي:

**obj.Age = 39**

في هذه الحالة يتمّ استدعاء المقطع Set للخاصيّة Age، حيث تكون قيمة المعامل Value 39.. وبما أنّ هذه القيمة مسموح بها، فسيتم تخزينها في المتغيّر الخاصّ.

وعندما يتم تنفيذ سطر كالتالي:

**Console.WriteLine(obj.Age)**

يتمّ استدعاء المقطع Get للخاصيّة Age ، حيث يتمّ إرجاع القيمة 39 لموضع الاستدعاء (وهو هنا الدالة Console.WriteLine).

والآن عد إلى النموذج، وأضف الكود التالي على الكود الموجود في حدث ضغط الزر:

**obj.Age = 39**

**Console.WriteLine("بعد جعل العمر 39 صارت قيمة الخاصيّة هي " & obj.Age.Tostring)**

**obj.Age = 199**

**Console.WriteLine("بعد جعل العمر 199 صارت قيمة الخاصيّة هي " & obj.Age.Tostring)**

ستلاحظ أن القيمة 39 ستظهر لك في نافذة المخرجات، ممّا يعني أنّها قد قُبلت، ثم ستظهر لك رسالة الخطإ "يجب أن يكون العمر موجبا وأصغر من 126"، ثم ستظهر القيمة 39 مرّة أخرى في شاشة المخرجات، ممّا يعني أنّ الخاصيّة لم تقبل القيمة 199.

**إطلاق الاستثناءات Raising Exceptions:**

في الخاصيّة السابقة، تلاحظ أنّنا تصيّدنا الخطأين المحتملين بطريقة مقبولة.. ولكن....!

لو حاول أي فرد في فريقك استخدام هذه الخليّة، فإنّه لن يتمكّن من معالجة الخطإ، كما أنّه ليس من الصواب أن تعرض رسائل خطإ من داخل أيّ خليّة، لأنّها ستظهر للمستخدم النهائيّ في هذه الحالة، رغمّ أنّ الخطأ ناتج عن المبرمج وليس المستخدم.

الأفضل إذن أن تطلق استثناءً من النوع "استثناء المعاملات" ArgumentException، بحيث يمكن للمبرمج أن يعالجه من الكود.. تعال نعدّل الخاصيّة كالتالي:

**Private tAge As Integer**

**Property Age() As Integer**

**Get**

**Age = tAge**

**End Get**

**Set (ByVal Value As Integer)**

**If Value < 0 Or Value >= 125 Then**

**Dim AgeException As New ArgumentException()**

**Throw AgeException ' إطلاق الاستثناء**

**Else**

**tAge = Value**

**End If**

**End Set**

**End Property**

ولاختبار هذا التعديل، أضف زرا جديدا للنموذج، واكتب ما يلي في حدث ضغطه:

**Dim obj As New Minimal()**

**Dim userAge As Integer**

**UserAge = InputBox("أدخل العمر")**

**Try**

**obj.Age = userAge**

**Catch exc As ArgumentException**

**MsgBox("لا يمكن قبول هذه القيمة")**

**End Try**

**الخواص التي يمكن قراءتها فقط Read-Only Properties:**

قد يكون أكثر منطقيّة أن نعرّف خاصيّة تسمّى "يوم الميلاد"، ومنها نستنتج العمر.. في هذه الحالة سنجعل خاصيّة العمر للقراءة فقط:

**Private tBirthDate As Date**

**Property BirthDate() As Date**

**Get**

**BirthDate = tBirthDate**

**End Get**

**Set**

**If Not IsDate(Value) Then**

**MsgBox("تاريخ غير صالح")**

**Exit Property**

**End If**

**If Value > Now() Or DateDiff(DateInterval.Year, Now(), Value) >= 125 Then**

**MsgBox("يجب أن يكون العمر موجبا وأصغر من 126")**

**Else**

**tBirthDate = Value**

**End If**

**End Set**

**End Property**

**ReadOnly Property Age() As Integer**

**Get**

**Age = DateDiff(DateInterval.Year, Now(),tBirthDate)**

**End Get**

**End Property**

لاحظّ أنّ كلّ ما فعلناه هو استخدام الكلمة "للقراءة فقط" ReadOnly عند تعريف الخاصيّة.. بالمثل يمكنك استخدام الكلمة "للكتابة فقط" WriteOnly، وإن كان من الواضح أنّك لن تستخدمها أبدا، فلا يوجد دافع يجعلك تسمح لشخص بتغيير قيمة الخاصيّة وتحرمه من إعادة قراءة القيمة التي أدخلها بنفسه!!

**تدريب 1:**

هل تتوقّع حدوث أيّة أخطاء عند قراءة العمر؟

**ملاحظة:**

لا يوجد ما يمنعك من تعريف الخصائص داخل النموذج Form أو قالب الكود Module أو السجلّ Structure.

**إجابة** [**تدريب1**](#تدريب1)**:**

- قد ينتج العمر سالبا، إذا عبث المستخدم في تاريخ جهاز الكمبيوتر، الذي تُرجعه دالة Now.

- كما يجب عليك أن تضع قيمة ابتدائيّة لتاريخ الميلاد، فقد يحاول المبرمج حساب العمر قبل أن يضع قيمة تاريخ الميلاد.

**المتغيّرات والخصائص المشتركة Shared Properties:**

أحيانا تحتاج لأن ترى كل الكائنات المنسوخة من نفس الخليّة، قيمة خاصيّة ما، بحيث تكون قيمتها ثابتة لجميع النسخ، وأيّ تغيير في قيمة الخاصيّة يكون ظاهرا لجميع الكائنات.. افترض مثلا أنّك تريد كتابة خاصيّة تحتوي على عدد النسخ (عدد الكائنات) المعرّفة من نفس الخليّة، بحيث يتم زيادتها بواحد عند تحميل أي نسخة جديدة، وإنقاصها بواحد عند إنهاء أيّ نسخة.. في هذه الحالة لو كانت هذه الخاصيّة خاصّة بكل كائن على حدة، فلن تزيد قيمتها أبدا عن 1.. أمّا لو جعلناها خاصيّة مشتركة لجميع الكائنات، فستحمل بالفعل عدد الكائنات المعرّفة من الخليّة.

تعالَ نرى مثالا:

سنعرّف خاصيّة مشتركة Sharedاسمها "المستخدمون المتصلون" LoggedUsers، وسنضيفها للخلية Minimal.. وسنعرّف كذلك وسيلتين: "اتصال" Connect وتعمل على زيادة عدد المستخدمين، و"انفصال" Disconnect، وتعمل على إنقاص عدد المستخدمين.

**Shared LoggedUsers As Integer**

**ReadOnly Property Users() As Integer**

**Get**

**Users = LoggedUsers**

**End Get**

**End Property**

**Public Function Connect() As Integer**

**LoggedUsers += 1**

**End Function**

**Public Function Disconnect() As Integer**

**If LoggedUsers > 1 Then**

**LoggedUsers -= 1**

**End If**

**End Function**

ولاختبار ذلك، أضف زرا جديدا للنموذج، واكتب فيه ما يلي:

**Dim obj1 As New Minimal()**

**obj1.Connect()**

**Console.WriteLine(obj1.Users) ' 1**

**obj1.Connect()**

**Console.WriteLine(obj1.Users) ' 2**

**Dim obj2 As New Minimal()**

**obj2.Connect()**

**Console.WriteLine(obj1.Users) ' 3**

**Console.WriteLine(obj2.Users) ' 3**

**Obj2.Disconnect()**

**Console.WriteLine(obj1.Users) '2**

إنّ جعل بعض الخصائص والوسائل مشتركة، يمكّنك من استدعاؤها مباشرة من اسم الخليّة، وذلك بخلاف الخصائص والوسائل الخاصّة بكل كائن، والتي لا يمكن استدعاؤها إلا من اسم الكائن فقط.. انظر للخليّة التالية، حيث سنقوم فيها بتعريف وسيلتين كلتاهما تقوم بعكس ترتيب حروف النصّ، ولكنّ إحداهما ستكون خاصّة بكلّ كائن على حدة Instant Methode، لهذا سيتم تطبيقها على النصّ الخاصّ به، والثانية ستكون مشتركة Shared، لهذا سنرسل لها النصّ كمعامل:

**Public Class SharedMembersClass**

**Public strProperty As String**

**Public Function IReverseString() As String**

**Return (StrReverse(strProperty))**

**End Function**

**Public Shared Function SReverseString(ByVal str As String) As String**

**Return (StrReverse(str))**

**End Function**

**End Class**

ولتجربة هذه الخليّة، ضع زرّا على نموذج واكتب فيه ما يلي:

**Dim X As New SharedMembersClass**

**' استخدام وسيلة الكائن**

**X.StrProperty = "كلام"**

**MsgBox (X. IReverseString)**

**' استخدام الوسيلة المشتركة**

**MsgBox (X. SReverseString("كلام"))**

**' استخدام الوسيلة المشتركة عبر اسم الخليّة مباشرةً**

**MsgBox (SharedMembersClass.SReverseString("كلام"))**

نقطة أخيرة يجب أن تضعها في اعتبارك:

عند كتابة كود الوسائل المشتركة، لا يمكنك أن تستخدم أيّ متغيّر أو وسيلة يخصّان الخليّة، إلا إذا كانت مشتركين Shared بدورهما.. وإلا فإنّ عليك أن تعرّف نسخة من الكائن نفسه داخل الوسيلة المشتركة، ومن خلاله تستخدم المتغيّرات والوسائل غير المشتركة.. هذا هو السبب الذي منعنا من استخدام المتغيّر من داخل الوسيلة المشتركة SReverseString، واستخدام معامل لتلقّي النصّ بدلا منه، في حين قمنا باستخدام هذا المتغيّر strProperty من خلال الوسيلة IReverseString دون احتياج لمعامل.

وطبعا لا توجد قيود على استخدام المتغيّرات والوسائل المشتركة، من أيّ وسيلة، سواء أكانت مشتركة أم لا.

**ملحوظة:**

قالب الكود Module هو خليّة كلّ وسائلها ومتغيّراتها مشتركة من تلقاء نفسها.. وليس بإمكانك تعريف نسخة من هذا القالب أو وراثته.. إنّه مجرّد طريقة لتنظيم الكود في مكتبات ذات وظيفة محددة.

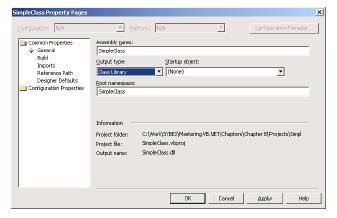
**إضافة الخليّة كمرجع**

* [**استخدام الخليّة من مشروع آخر:**](file:///C:\Users\Administrator\Desktop\vb.net\c-%20Classes%20الخلايا\07-%20Building%20Classes%20بناء%20الخلايا\6-%20إضافة%20الخليّة%20كمرجع\1-%20استخدام%20الخليّة%20من%20مشروع%20آخر.htm)
* [**التعامل مع الخلايا من لغات برمجة مختلفة:**](file:///C:\Users\Administrator\Desktop\vb.net\c-%20Classes%20الخلايا\07-%20Building%20Classes%20بناء%20الخلايا\6-%20إضافة%20الخليّة%20كمرجع\2-%20التعامل%20مع%20الخلايا%20من%20لغات%20برمجة%20مختلفة.htm)

**استخدام الخليّة من مشروع آخر:**

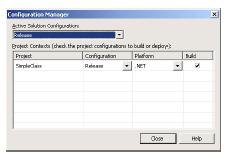
إذا أردت استخدام خليّتك من مشروع آخر، فعليك أولا تغيير نوع المشروع الحاليّ، لأنّ مشاريع الويندوز لا يمكن استخدامها كمكونات في مشاريع أخرى.

اضغط بزر الفأرة الأيمن على اسم المشروع في متصفّح المشاريع، ومن القائمة اليمنى اضغط "خصائص" Properties.. في مربع حوار خصائص المشروع، اضغط القائمة المنسدلة "مخرجات" Output، وغير نوع المشروع من تطبيق ويندوز إلى "مكتبة خلايا" Class Library.. أغلق النافذة، وعد إلى المشروع.



الآن لم نعد بحاجة لنموذج الاختبار.. اضغط اسم النموذج في متصفّح المشاريع بزر الفأرة الأيمن، ومن القائمة الموضعيّة اضغط "استبعاد من المشروع" Exclude From Project.

افتح القائمة الرئيسيّة "بناء" Build، واضغط "مدير الإعدادات" Configuration Manager.. غيّر الإعداد الحاليّ من "تصحيح" Debug إلى "إصدار" Release.



من قائمة "بناء" أيضا، اضغط Build SimpleClass.. هذا الأمرُ سيترجم كود الخليّة، ليبنيَ ملفّا من النوع DLL (مكتبة الربط الديناميكية Dynamic Link Library).. هذا هو الملفّ الذي يحتوي على الخلايا، والذي يمكن ربطه بأيّ مشروع لاستخدام هذه الخلايا فيه.. ستجد هذا الملفّ في المسار \Obj\Release\ SimpleClass.dll في مجلّد المشروع.

تعالَ نستخدم هذا الملفّ:

ابدأ مشروعا جديدا، ومن القائمة الرئيسيّة "مشروع" Project اضغط الأمر "إضافة مرجع" Add Reference.. في مربّع الحوار الذي سيظهر، اضغط "شريط المشاريع" Projects Tab، ثمّ اضغط الزرّ "تصفّح" Browse.. سيظهر لك مربع حوار فتح ملفّ، استخدمه للوصول إلى الملفّ SimpleClass.dll، ثم اضغط موافق.

أغلق مربع حوار "إضافة مرجع" وعد إلى المشروع.

الآن صار بإمكانك تعريف متغيّر من النوع Minimal في مشروعك، حيث تستطيع استخدام كل خصائصه ووسائله:

**Dim obj As New SimpleClass.Minimal()**

**' لاحظ استخدام اسم المشروع قبل اسم الخليّة**

**obj.Age = 45**

**obj.Property2 = 5544**

**MsgBox(obj.Negate())**

ولا يشترط أن تستخدم الملفّ SimpleClass.dll كمرجع في برامج VB فقط، فبإمكانك أن تستخدمه أيضا في برامج لغة C# أو VC++.

**التعامل مع الخلايا من لغات برمجة مختلفة:**

من الأمور الجديدة والطريفة في VS.Net قدرتك على تكوين مجموعة مشاريع Solution تحتوي على مشاريع من لغات مختلفة، مع قدرتك على الربط بينها.. تعالَ نرى ذلك عمليّا:

- ابدأ مشروع C# جديدا من النوع "تطبيق ويندوز" WindowsApplication.. سمّ المشروع TwoLang.. لأوّل وهلة ستشعر أنّك في VB، فالنموذج الذي سيظهر لك هو نفسه الذي تتعامل معه في VB.

- من قائمة "ملفّ" File اختر "إضافة مشروع" Add Project، واضغط الأمر "مشروع جديد" New Project.. اختر مشاريع VB، وحدّد النوع "مكتبة خلايا" Class Library.. سمّ المشروع Test.

- في محرّر الكود غيّر اسم الخليّة من Class1 إلى Hello، واكتب فيها ما يلي:

**Public Class Hello**

**Public Shared Sub Msg()**

**MsgBox("أهلا بك من فيجيوال بيزيك")**

**End Sub**

**End Class**

- في متصفّح المشاريع انتقل إلى المشروع TwoLang، واضغط كلمة المراجع References بزرّ الفأرة الأيمن، ومن القائمة الموضعيّة اختر "إضافة مرجع" Add Reference.

- في مربّع الحوار الذي سيظهر، اضغط شريط "المشاريع" Projects.. ستجد أنّ القائمة تحتوي على المشروع Test.. انقره مرّتين بالفأرة ليظهر اسمه في القائمة السفليّة، ثمّ اضغط زرّ OK.

- اعرض النموذج Form1.cs بنقر اسمه مرّتين بالفأرة في متصفّح المشاريع.. أضف للنموذج زرّا من صندوق الأدوات ToolBox بنفس الطريقة التي اعتدتها في VB.

- انقر الزرّ مرّتين بالفأرة.. سيعرض لك محرّر الكود الإجراء المستجيب لضغط الزرّ.. اكتب فيه السطر التالي (مع مراعاة حالة الأحرف):

**Private void button1\_Click(Object sender, System.EventArgs e)**

**{**

**Test.Hello.Msg();**

**}**

ستلاحظ أنّ قائمة الأعضاء لن تظهر بعد كتابة النقطة ".".. إنّ ذلك يعود إلى أنّ المشروع Test لم يتمّ بناؤه بعد، ولو كنت بنيته (أنشأت الملفّ Test.Dll) لكانت قائمة العناصر قد ظهرت كما اعتدت.

- اضغط F5.. إنّ هذا سيؤدّي لبناء المشروعين وتشغيل المشروع TwoLang، حيث سيظهر لك النموذج Form1.

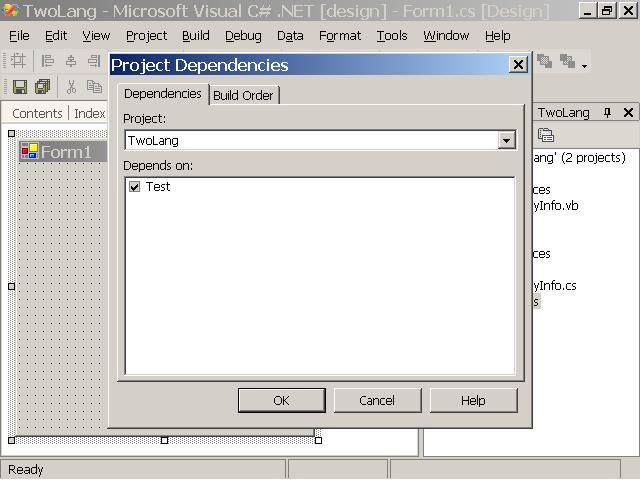
- اضغط الزرّ.. ستظهر لك الرسالة التي كتبتها في VB!

- أغلق التنفيذ، وعد إلى الخليّة Hello.. غيّر نصّ الرسالة، ثم اضغط F5.. ستجد أنّ الرسالة الجديدة قد ظهرت لك!

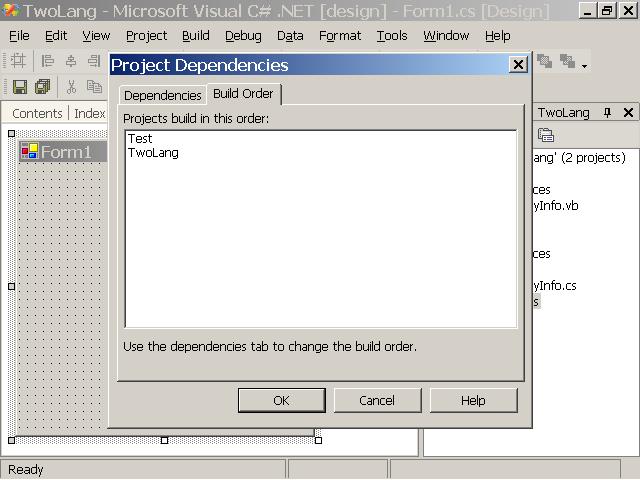
وبطريقة مشابهة لهذه الطريقة يمكنك استخدام الخلايا المكتوبة بلغة C# في VB.

إنّ هذه الميزة في غاية الأهمّيّة، فهي تسمح بكتابة تطبيقات يشترك فيها أكثر من مبرمج بأكثر من لغة، بحيث يتمّ عرض كلّ مكوّنات التطبيق معا في وقت التصميم، لتعديل وتصحيح أيّ كود في أيّ لغة في أيّ وقت!!

ويمكنك استخدام الأمر "تبعيّة البناء" Build Dependency من قائمة "مشروع" Project، لتحدّد المشاريع التي يعتمد عليها المشروع المحدّد حاليّا في متصفّح المشاريع، وذلك ليتمّ بناء هذه المشاريع أولا قبل بنائه.. وستجد أنّ المشروع TwoLang يعتمد على المشروع Test:



كما يمكنك أن تضغط شريط "ترتيب البناء" Build Order، لتظهر لك قائمة بأسماء المشاريع، حيث يمكنك سحب كلّ منها بالفأرة لأعلى أو لأسفل لتغيير ترتيبه.



**الوراثة Inheritance**

* [**مفهومهاوأهمّيتها:**](file:///C:\Users\Administrator\Desktop\vb.net\c-%20Classes%20الخلايا\07-%20Building%20Classes%20بناء%20الخلايا\7-%20الوراثة%20Inheritance\0-%20مفهومهاوأهمّيتها.htm)
* [**وراثة خلايا إطار العمل Framework:**](file:///C:\Users\Administrator\Desktop\vb.net\c-%20Classes%20الخلايا\07-%20Building%20Classes%20بناء%20الخلايا\7-%20الوراثة%20Inheritance\1-%20وراثة%20خلايا%20إطار%20العمل.htm)
* [**تعدد الأشكال Polymorphism:**](file:///C:\Users\Administrator\Desktop\vb.net\c-%20Classes%20الخلايا\07-%20Building%20Classes%20بناء%20الخلايا\7-%20الوراثة%20Inheritance\2-%20تعدد%20الأشكال.htm)
* [**خليّة الشكلShape :**](file:///C:\Users\Administrator\Desktop\vb.net\c-%20Classes%20الخلايا\07-%20Building%20Classes%20بناء%20الخلايا\7-%20الوراثة%20Inheritance\3-%20خليّة%20الشكل.htm)
* [**منشئ الكائن Constructor وهادمه Destructor:**](file:///C:\Users\Administrator\Desktop\vb.net\c-%20Classes%20الخلايا\07-%20Building%20Classes%20بناء%20الخلايا\7-%20الوراثة%20Inheritance\4-%20منشئ%20الكائن%20وهادمه.htm)
* [**من يرثُ ماذا؟**](file:///C:\Users\Administrator\Desktop\vb.net\c-%20Classes%20الخلايا\07-%20Building%20Classes%20بناء%20الخلايا\7-%20الوراثة%20Inheritance\5-%20من%20يرثُ%20ماذا.htm)
* [**كلمات وراثيّة هامّة:**](file:///C:\Users\Administrator\Desktop\vb.net\c-%20Classes%20الخلايا\07-%20Building%20Classes%20بناء%20الخلايا\7-%20الوراثة%20Inheritance\6-كلمات%20وراثية%20هامة.htm)

**مفهومهاوأهمّيتها:**

أهمّ نقطة تهمّ المبرمج، هي قدرته على إعادة استخدام الكود الذي كتبه من قبل، بحيث لا يُضطر لكتابة الكود من جديد كلما تغيرت جزئيّة صغيرة في المشكلة التي يتعامل معها.

هنا يبرز مفهوم الوراثة: كيف نبني خلية جديدة (تسمّى الخلية الفرعيّة Subclass أو الخليّة المشتقّة Derived Class) من خلية قديمة (تسمّى الخليّة الأمّ Parent Class أو الخليّة الأساسيّة Base Class)، بحيث ترث كل خصائص ووسائل الخلية القديمة، مع إمكانيّة احتوائها على خصائص جديدة، أو تعديل عمل بعض الخصائص القديمة باستبدالها بأخرى معدّلة Overriding.

طبعا ما زلت لا تشعر بأهمّيّة الوراثة، وتتساءل في لماضة:

- وما الذي يدفعني لأيّ مفهوم غريب مثل هذه الوراثة؟.. لماذا لا أعدّل الخليّة التي لديّ مباشرة، فأضيف إليها أو أحذف منها كما أشاء؟

قد يبدو أنّ معك حقا، ولكنّ المشكلة التي ستواجهك هي: ماذا تفعل لو كانت الخليّة التي لديك مغلقة (موجود في ملف DLL اشتريته من شخص آخر)، وكانت بها بعض الخصائص والوسائل التي تحتاج لتعديل كي تناسبك؟

أرأيت؟.. هنا تبزغ أهمّيّة الوراثة: إنّ بإمكانك أن ترثَ كل خبرات الآخرين، دونَ أن ترهقَ نفسك بمعرفة كيف كتبوا الكود، في نفس الوقتِ الذي يمكنك أن تضيف خبرتك الخاصّة وإجراء التعديلات التي تناسبك.

**وراثة خلايا إطار العمل Framework:**

تعال نبدأ بوراثة وتطوير واحدة من الخلايا البالغ عددها 3500 خليّة، الموجودة في إطار العمل Framework.. هذه الخليّة هي خليّة "المصفوفة القائمة" ArrayList، التي هي عبارة عن مصفوفة مرنة Dynamic Array، خاناتها عبارة عن عناصر من النوع كائن Object، وتمتلك الخصائص والوسائل المميّزة للقوائم Lists، مثل إضافة العناصر لأيّ موضع، وحذفها من أيّ موضع، وترتيبها، ... إلخ.. (لمزيد من التفاصيل، انظر الفصل التاسع).

والآنَ سننشئ المشروع CustomArrayList، وفيه سنرثُ كلّ مواصفات المصفوفة القائمة، وسنضيف إليها الوسيلة "امنع التكرار" EliminateDuplicates.

دعنا نطلق على الخليّة الجديدة اسم myArrayList.

إنّ أول سطر في الخليّة الجديدة يجب أن يكون جملة الوراثة Inherits Statement، وهي في منتهى السهولة:

**Class myArrayList**

**Inherits ArrayList**

**End Class**

إذا لم تُضف أيّ سطر زائد، فستجد أنّ الخليّة الجديدة تعمل بالضبط كالخليّة الأساسيّة، بدون أيّ اختلاف.. فإذا أضفت دالة إلى الخليّة الجديدة، تكون قد أضفت لنفسك إمكانيّة جديدة بالإضافة إلى قدرات الخليّة الأساسيّة.. أضف هذا الإجراء للخليّة المشتقّة:

**Sub EliminateDuplicates()**

**Dim I As Integer = 0**

**Dim delEntries As ArrayList**

**While I <= MyBase.Count - 2**

**Dim J As Integer = I + 1**

**While J <= MyBase.count - 1**

**If MyBase.Item(I).Tostring =\_**

**MyBase.item(J).Tostring Then**

**MyBase.RemoveAt(J) ' احذف العنصر المكرّر**

**End If**

**J += 1**

**End While**

**I += 1**

**End While**

**End Sub**

قد يبدو لك الكود صعبا، ولكنّه يقوم ببساطة بمقارنة العنصر الحالي (I) بالعناصر التالية له (بدءا من I + 1 إلى النهاية)، فإذا تساوي مع أيٍ منهم، يتم حذف العنصر المكرّر.

لاحظ أنّ الخليّة الجديدة لا تمتلك خاصيّة تدعى Count، وإنّما هي قد ورثتها من الخليّة الأمّ، لهذا استخدمنا التعبير MyBase.Count عند التعامل مع هذه الخاصيّة، حيث إنّ كلمة "خليّتي الأساسيّة" MyBase تشير إلى الخليّة الأمّ.

تعال نضيف وسيلة أخرى، لتحويل كل عناصر القائمة إلى نصّ واحد تفصل بين عناصره علامة فصل الأسطر (كلّ عنصر في سطر مستقلّ):

**Function GetString() As String**

**Dim I As Integer**

**Dim strValue As String**

**strValue = MyBase.Item(0).Tostring**

**For I = 1 To MyBase.Count - 1**

**strValue &= vbCrLf & MyBase.Item(I).Tostring**

**Next**

**GetString = strValue**

**End Function**

الآن تعال نختبر خليّتنا الجديدة:

أضف زرا لنموذج الاختبار، واكتب فيه الكود التالي:

**Private Sub Button2\_Click(ByVal sender As System.Object, \_**

**ByVal e As System.EventArgs) Handles Button2.Click**

**Dim mlist As New myArrayList()**

**mlist.Add(" 10")**

**mlist.Add("A")**

**mlist.Add("20")**

**mlist.Add("087")**

**mlist.Add("c")**

**mlist.Add("A")**

**mlist.Add("b")**

**mlist.Add("a")**

**mlist.Add("A")**

**mlist.Add("87")**

**mlist.Add(10)**

**mlist.Add(100)**

**mlist.Add(110)**

**mlist.Add("1001")**

**Console.WriteLine(mlist.GetString())**

**mlist.EliminateDuplicates()**

**Console.WriteLine(mlist.GetString())**

**End Sub**

والآنَ سنضيف وسيلتين جديدتين:

- Min: تُرجع العنصر الأسبق في الترتيب الأبجديّ.

- NumMin: ترجع أصغر رقم في عناصر المصفوفة.

ها هما تان:

**Function Min() As String**

**Dim I As Integer**

**Dim minValue As String**

**minValue = MyBase.Item(0).Tostring**

**For I = 1 To MyBase.Count - 1**

**If MyBase.Item(I).Tostring < minValue Then**

**minValue = MyBase.Item(I).Tostring**

**End If**

**Next**

**Min = minValue**

**End Function**

**Function NumMin() As Double**

**Dim I As Integer**

**Dim minValue As Double**

**minValue = 1E+230 ' رقم كبير جدا، ليكون أيّ رقم في المصفوفة أصغر منه**

**For I = 1 To MyBase.Count - 1**

**If IsNumeric(MyBase.item(I)) Then**

**If Val(MyBase.Item(I) Tostring) < minValue Then**

**minValue = val(MyBase.Item(I).Tostring)**

**End If**

**End If**

**Next**

**NumMin = minValue**

**End Function**

وبعد: تلاحظ أنّنا طورنا الخليّة الموجودة لتناسب أغراضنا، دونَ أن تتأثّر الخليّة الأصليّة بتعديلاتنا، ممّا يعني أنّ كل البرامج التي تستخدمها ستستمرّ في العمل بدون قلق.. أليس شيئا رائعا؟

الأدهى أنّ خليّتك الجديدة نفسها قد يرثها شخص آخر ويستخدمها لإنشاء خليّة ثالثة، ... وهكذا!

**تعدد الأشكال Polymorphism:**

ذكرنا من قبلُ أنّ تعدّد الأشكال، يعني أن تمتلك مجموعة من الخلايا خاصيّة أو وسيلة مشتركة في الاسم، ولكنها تؤدّي مهامّ مختلفةً، تبعا للخليّة التي تنتمي إليها.. مثلا: لو لدينا خليّة اسمها شكل Shape، ترثها مجموعة من الخلايا: مثلث، مربع، مستطيل، دائرة، فإنّ كلا من هذه الخلايا سيمتلك وسيلة لحساب مساحة سطحه، حيث ستختلف الصيغة التي تحسب هذه المساحة تبعا للشكل، ولكن رغم هذا ستحمل هذه الوسيلة في كل هذه العناصر الاسم Area.

تعالَ نطبّق ذلك عمليّا.

**خليّة الشكلShape :**

هذه هي الخليّة الأساسيّة، وهي تمنحنا وسيلتين: المساحة Area، والمحيط Perimeter.. والحقيقة أنّهما لن يقوما بأيّ وظيفة في الخليّة الأساسيّة، اللهمّ إلا منحنا الاسمين اللذين سنستخدمهما في تعدد الأشكال.

ابدأ مشروعا جديدا، وأضف إليه خليّةً أسمِها Shape، واكتب بها ما يلي:

**Class Shape**

**Overridable Function Area() As Double**

**End Function**

**Overridable Function Perimeter() As Double**

**End Function**

**End Class**

تلاحظ استخدامنا لكلمة "قابلة للاستبدال" Overridable، وأعتقد أنّك تفهم وظيفتها، فهي تعني أنّ باستطاعة أي خليّة مشتقّة من هذه الخليّة، أن تستبدل الدوال والخصائص المسبوقة بهذه الكلمة.

لاحظ كذلك أنّ أيّ وسيلة أو خاصيّة عامّة لجميع الأشكال الهندسيّة، يمكنك أن تضيفها في هذه الخليّة.

تعالَ الآن ننشئِ الخلايا المشتقّة.. أضف خليّةً أخرى للمشروع أسمها Shapes، حيث سنضع فيها كل الخلايا المشتقّة المعبّرة عن الأشكال.. في هذه الخليّة اكتب ما يلي:

**Public Class Square**

**Inherits Shape**

**Private sSide As Double**

**Public Property Side() As Double**

**Get**

**Side = sSide**

**End Get**

**Set**

**sSide = Value**

**End Set**

**End Property**

**Public Overrides Function Area() As Double**

**Area = sSide ^ 2**

**End Function**

**Public Overrides Function Perimeter() As Double**

**Return (4 \* sSide)**

**End Function**

**End Class**

**Public Class Triangle**

**Inherits Shape**

**Private side1, side2, side3 As Double**

**Property SideA() As Double**

**Get**

**SideA = side1**

**End Get**

**Set**

**side1 = Value**

**End Set**

**End Property**

**Property SideB() As Double**

**Get**

**SideB = side2**

**End Get**

**Set**

**side2 = Value**

**End Set**

**End Property**

**Public Property SideC() As Double**

**Get**

**SideC = side3**

**End Get**

**Set**

**side3 = Value**

**End Set**

**End Property**

**Public Overrides Function Area() As Double**

**Dim perim As Double**

**perim = Perimeter()**

**Return (Math.Sqrt(perim \* (perim - side1) \* \_**

**(perim - side2) \* (perim - side3)))**

**End Function**

**Public Overrides Function Perimeter() As Double**

**Return (side1 + side2 + side3)**

**End Function**

**End Class**

**Public Class Circle**

**Inherits Shape**

**Private cRadius As Double**

**Public Property Radius() As Double**

**Get**

**Radius = cRadius**

**End Get**

**Set**

**cRadius = Value**

**End Set**

**End Property**

**Public Overrides Function Area() As Double**

**Return (Math.Pi \* cRadius ^ 2)**

**End Function**

**Public Overrides Function Perimeter() As Double**

**Return (2 \* Math.Pi \* cRadius)**

**End Function**

**End Class**

كما ترى، فإنّ لدينا ثلاث خلايا: المربّع والمثلّث والدائرة، لكلّ منها خصائصه، مثل أضلاع المثلّث SideA، SideB، SideC، ونصف قطر الدائرة Radius، كما أنّ لكلٍّ منها الوسيلتين Area، Perimeter، لحساب المساحة والمحيط.

تعالَ نختبر ما فعلنا:

**Dim shape1 As New Triangle()**

**Dim shape2 As New Circle()**

**Dim shape3 As New Square()**

**' توصيف المثلّث**

**shape1.SideA = 3**

**shape1.SideB = 3.2**

**shape1.SideC = 0.94**

**Console.WriteLine("مساحة المثلّث هي " & shape1.Area.Tostring)**

**' توصيف الدائرة**

**shape2.Radius = 4**

**Console.WriteLine("مساحة الدائرة هي " & shape2.Area.Tostring)**

**' توصيف المربّع**

**shape3.Side = 10.01**

**Console.WriteLine("مساحة المربّع هي " & shape3.Area.Tostring)**

**Dim shapes() As Shape**

**shapes(0) = shape1**

**shapes(1) = shape2**

**shapes(2) = shape3**

**Dim shapeEnum As IEnumeratOr = shapes.GetEnumerator**

**Dim totalArea As Double**

**While shapeEnum.MoveNext**

**totalArea += CType(shapeEnum.Current, shape).Area**

**End While**

**Console.WriteLine("المساحة الكلّيّة هي " & totalArea.Tostring)**

طبعا تتساءل عن معنى العداد IEnumerator؟

هذه طريقة أخرى من طرق الجمل التكراريّة، حيث يتم تحميل المصفوفة في عدّاد عن طريق الوسيلة GetEnumerator، للمرور عبر كل العناصر باستخدام الوسيلتين:

- Current: لقراءة العنصر الحاليّ في العداد.

- MoveNext: للانتقال للعنصر التالي في العدّاد.. وإذا كان العنصر الحالي هو العنصر الأخير، تُرجع هذه الوسيلة False.

ولا داعيَ لأن تشغلَ ذهنكَ كثيرًا بهذه الطريقة، فإنّ بإمكانك أداء نفس الوظيفة باستخدام الجملة التكراريّة التقليديّة "من إلى" For.. Next مع عناصر المصفوفة.

ما يجب أن تهتمّ به هنا، هو الجملة:

**totalArea += CType(shapeEnum.Current, shape).Area**

حيث تقوم بتحويل العنصر الحاليّ في المصفوفة (في العدّاد) إلى النوع shape، واستخدام الوسيلة Area لحساب مساحة سطحه.. طبعا تعرفُ أنّ هذا التحويل ضروريٌّ فقط عندما يكون اختيار التحويل الدقيق فعّالا Option Strict On.. ولو لم يكن، فيمكنك أن تكتب هذه الجملةَ مباشرةً كالتالي:

**totalArea += shapeEnum.Current.Area**

وطبعا لاحظت أنّنا حوّلنا كلّ الأنواع للخليّة الأساسيّة shape حتّى نكتب جملة واحدة مختصرة، بدلا من كتابة جملة شرطيّة للتفريق بين أنواع الأشكال الثلاثة كالتالي:

**Select Case shapeEnum.Current.GetType.ToString**

**Case GetType(Circle).ToString**

**totalArea += CType(shapeEnum.Current, Circle).Area**

**Case GetType(Triangle).ToString**

**totalArea += CType(shapeEnum.Current, Triangle).Area**

**Case GetType(Square).ToString**

**totalArea += CType(shapeEnum.Current, Square).Area**

**End Select**

**منشئ الكائن Constructor وهادمه Destructor:**

يتمّ إنشاء الكائن في السطر الذي تنفّذ فيه كلمة New، كما في الأمثلة التالية:

* **Dim shape1 As Triangle = New Triangle()**
* **Dim shape1 As New Triangle()**
* **Dim shape1 As Triangle() ' لم يُنشئ الكائن بعد**

**shape1 = New Triangle() ' الآن تمّ إنشاؤه**

ويعيش الكائنُ ما دام المتغيّر الذي يحمله فعالا، فإذا انتهى الإجراء الذي تم تعريف المتغيّر فيه، يتمّ تدمير الكائن وتحريرُ الذاكرة منه.. كما يمكنك أن تقتل الكائن بربط المتغيّر الذي يحمله بلا شيء Nothing:

**shape1 = Nothing**

ومن الممكن أن تضع القيم الابتدائيّة لبعض خصائص الكائن، في نفس سطر إنشائه، حيث يتم استدعاء منشئ الكائن Object's Constructor.. انظر للمثال التالي:

**Dim rect1 As Rectangle = New Rectangle(10, 10, 50, 90)**

ولكن ماذا عن الخلايا التي أنشأناها؟.. هل نستطيع وضع القيم الابتدائيّة لها بمثل هذه الطريقة؟

حتّى الآن لا، فنحن لم نكتب أيّ كود لمُنشئات هذه الخلايا.. وإن لم يكن هناك ما يمنعنا!

كلّ ما سنفعله، هو كتابة تعريفات مختلفة للحدث New الخاص بكلّ خليّة، يأخذ كل تعريف منها العدد الذي نريده من المعاملات.

فلنبدأ بخليّة المثلّث.. في هذه الحالة نريد منحه أطوال الأضلاع كقيم ابتدائيّة.. هذه هي الطريقة:

**Sub New(sideA As Double, sideB As Double, sideC As Double)**

**MyBase.New()**

**' تأكّد من وجود الجملة السابقة دائمًا في كلّ منشئ**

**' حتّى تتأكد من تنفيذ الكود الذي يضع القيم الابتدائيّة للخليّة الأصليّة**

**side1 = sideA**

**side2 = sideB**

**side3 = sideC**

**End Sub**

جرّب الآن أن تكتب الجملة التالية في كود النموذج:

**Dim shape1 As New Triangle(**

ستجد أن تلميحا قد ظهر على الشاشة بمجرّد فتح القوس، يعرض لك أسماء المعاملات وأنواعها.

وإذا أردت أن تعرّف كائنا بدون إرسال قيم ابتدائيّة، فلا بدّ أن تعرّف أكثر من صيغة للمنشئ:

**Sub New()**

**MyBase.New()**

**End Sub**

ويمكنك تعريف أيّ عدد تريده من صيغ المنشئات.. انظر مثلا كيف سنعرّف ثلاث صيغ لمنشئ الدائرة:

**' منشئ بدون معاملات**

**Sub New()**

**MyBase.New()**

**End Sub**

**' منشئ يتلقّى نصف القطر**

**Sub New(ByVal radius As Double)**

**MyBase.New()**

**cRadius = radius**

**End Sub**

**' منشئ يتلقّى المربّع الذي يحتوي الدائرة**

**Sub New(ByVal rect As Rectangle)**

**MyBase.New()**

**cRadius = rect.Width / 2**

**End Sub**

**ملحوظة:**

يمكن تعريف منشئ Constructor للسجلّ Structure،.. كلّ ما عليك هو إضافة الحدث New لمقطع السجلّ بنفس الطريقة التي تعلّمناها هنا.. عندئذ يمكنك استخدام كلمة New في تعريف المتغيّرات من نوع هذا السجلّ، مع منحها المعاملات المطلوبة.

وعموما: إنّ معظم سمات السجلّ والخليّة مشتركة، والفروق بينهما تنحصر في أنّ السجلّ لا يستطيع استخدام إمكانيّات الوراثة كما تفعل الخلايا، بالإضافة لأنّ أي متغيّر يتمّ تعريفه في السجلّ هو Public ما لم يُذكر غير ذلك، بينما أيّ متغيّر يتمّ تعريفه في الخليّة هو Private ما لم يُذكر غير ذلك.

هذا هو المنشئ، فما بالنا بالهادم؟

عندما ينتهي الكائن لا تتحرّر الذاكرة منه مباشرة، وذلك لتسريع تنفيذ البرامج.. وهناك ما يُدعى "جامع القمامة" Garbage collector (GC)، وهو يقوم على فترات بفحص الكائنات التي لم تعد قيدَ الاستخدام ليمحوها من الذاكرة.. وعندما يفعل ذلك، يقوم باستدعاء هادم الكائن (وهو الحدث Finalize)، لتنفيذ أيّ مهامّ إضافيّة تريدها أنت.. أمّا لو أردت تدمير الكائن وتحرير الذاكرة منه في الحال، فيمكنك أن تستخدم الوسيلة "التخلّص" Dispose الخاصّة بهذا الكائن.

وهنا يجب عليك أن تراعي ما يلي:

إذا كانت الخلّيّة التي تنشئها تقوم بفتح ملفّ أو ما شابه، وتريد أن تتأكّد من إغلاق هذا الملفّ عند انتهاء عمل هذه الخليّة، فلا تستخدم الحدث Finalize، لأنّ استخدامه لن يضمن لك متّى ستتمّ هذه العمليّة (نتيجة لطريقة عمل جامع القمامة).

لهذا فإنّ عليك أن تنشئ في خلّيّتك الوسيلة Dispose أو Close (ويمكن تسميتها بأي اسم آخر، وإن كان هذان الاسمان أكثر شيوعا).. وعلى مستخدم الخليّة أن يستدعي هذه الوسيلة عندما يريد أن يتأكّد من إغلاق الملفّ (أو أيّ عمليّة أخرى يجب أداؤها).

وفي هذه الحالة، لا تنس أن تكتب في الوسيلة Dispose أو Close الجملة التالية:

**GC.SuppressFinalize(Me)**

حيث تعمل الوسيلة "إلغاء حدث الانتهاء" SuppressFinalize على منع استدعاء الحدث Finalize، فلا داعي من تنفيذه في هذه الحالة.

ولو كنت تخشى من عدم استدعاء المبرمج للوسيلة Dispose، فلا مانع من أن تستدعيها أنت في الحدث Finalize.

**من يرثُ ماذا؟**

ماذا سيحدث لو أنشأنا خليّةً مشتقّةً من خليّة الشكل Shape، دونَ أن نكتب في الخليّة الجديدة دالتي المساحة والمحيط؟

في هذه الحالة لن تعمل هاتان الدالتان، فهما فارغتان بدون أيّ وظيفة في الخليّة الأمّ.. معنى هذا أنّ هاتين الدالتين يجب أن يُستبدلا MustOverride.. ليس هذا فحسب، بل إنّ الخليّة الأمّ يجب أن تورّث MustInherit، حيث إنّها لن تعمل بمفردها بصورة جيّدة.

**تعريف:**

الخليّة التي لا تعمل بمفردها، ويجب أن ترثها خليّة أخرى على الأقلّ، تسمّى خليّة أساسيّة مجرّدة Abstract Base Class، أو خليّة افتراضيّة Virtual Class، حيثُ لا يمكنك أن تعرّف متغيّرا من نوع هذه الخليّة.. وأيّ خليّة من هذا النوع، يجب أن تظهر في تعريفها كلمة MustInherit، كما يجب أن تظهر في تعريف واحدة على الأقل من وسائلها كلمة MustOverride.

تعال نعدّل من تعريف خليتنا الأمّ:

**Public MustInherit Class Shape**

**Public MustOverride Function Area() As Double**

**Public Function Perimeter() As Double**

**End Class**

تلاحظُ طبعا أنّ الدالةَ التي يجب أن تُستبدل MustOverride، ليس لها مقطع (ليس لها سطر نهاية End Function)، وذلك لأنّنا لن نكتب بها أيّ كود.

بهذا التعديل أصبح لزاما على الخلايا المشتقّة أن تكتب دالتين لحساب المساحة والمحيط، وهذا ما فعلناه في المربع والدائرة والمثلّث.

ولكن ماذا سيحدث مثلا، لو لم نعرّف هاتين الدالتين في خليّة المثلّث؟

في هذه الحالة يجب أن نعرّف المثلّث بكلمة MustInherit، حيث يمكن أن نشتقّ منه خلايا جديدة (كالمثلث القائم، والمثلّث الحادّ، والمثلّث متساوي الساقين، والمثلّث متساوي الأضلاع) حيث يجب على كلّ خليّة منها أن تعرّف دالتي المساحة والمحيط.

ويجرّنا هذا الحديث إلى بقيّة المعدِّلات Modifiers التي يمكن استخدامها في تعريف الخليّة وأعضائها:

**- "غير قابلة للتوريث" NotInheritable:**

استخدم هذه الكلمة في تعريف الخليّة التي لا تريد أن يرثها أحد.

وكمثال: لو حاولت وراثة خليّة المصفوفة Array class، فسيعترض VB، لأنّ هذه الخليّة غير قابلة للوراثة.

**- "غير قابلة للاستبدال" NotOverridable:**

واضح من الاسم أنّ الوسيلة التي تدخل هذه الكلمة في تعريفها لا يمكن استبدالها من الخلايا المشتقّة.. ولكن.. أليست كلّ وسيلة غير قابلة للاستبدال في الوضع التلقائيّ؟

هذا صحيح.. لهذا فإنّ هذه الكلمة لا تستخدم إلا مع وسيلة في خليّة مشتقّة من خليّة أخرى، بحيث تكون هذه الوسيلة استبدالا لإحدى وسائل الخليّة الأصليّة، ولا تريد السماح بالمزيد من عمليات الاستبدال، في خلايا أخرى ترث هذه الخليّة المشتقّة.. معنى هذا أنّ كلمة NotOverridable دائما وأبدا يجب أن يتبعها كلمة Overdides:

**Class A**

**' أيّ خليّة ترث هذه الخليّة، سيكون بمقدورها استبدال هذه الوسيلة**

**Public Overridable Sub DoSomething()**

**' …………………..**

**End Sub**

**End Class**

**Class B**

**Inherits A**

**' أيّ خليّة ترث هذه الخليّة، لن يكون بمقدورها استبدال هذه الوسيلة**

**' Sealed Methods وتسمّى هذه الوسيلة في هذه الحالة: وسيلة مختومة (مغلقة)**

**Public NotOverridable Overrides Sub DoSomething()**

**' …………………..**

**End Sub**

**End Class**

**- محميّة Protected:**

رأينا استخدام كلمتي Public و Private في تعريف الخصائص والإجراءات.. ولدينا هنا كلمة تقع في الوسط بينهما، هي كلمة محميّة Protected.. إنّ الخصائص والدوال التي تعرّف بهذه الكلمة تكونُ خاصّة بالخليّة التي تمّ تعريفها فيها، ويمكن استخدامها من أيّ خليّة مشتقّة منها، ولكن ليس من خارج هاتين المنطقتين.

**- "محميّة صديقة" Protected Friend:**

باستخدام هذه الكلمة، تصبح الخاصيّة أو الإجراء متاحين للاستخدام من داخل الخليّة ومن باقي المشروع ومن الخلايا المشتقّة منها بغض النظر عن فضاء الاسم Name Space الذي تنتمي إليه، ولكن ليس من أيّ مشروع آخر يستخدم هذه الخلايا.

**- "تحجب" Shadows:**

هذه الكلمة مشابهة لكلمة "تستبدل" Overrides، حيث يمكنك أن تستخدمها في الخلايا المشتقّة لتعريف عنصر جديد، بحيث تستطيع أن تجعل له نفس اسم عنصر موجود سابقا في الخليّة الأمّ.. وإن كانت تمتاز عنها بما يلي:

- يمكنك أن تستخدم هذه الكلمة في تعريف الخلايا أو الإجراءات أو الدوال أو الخصائص أو المتغيّرات.

- إذا استخدت هذه الكلمة لتعريف صيغة جديدة من دالّة سابقة، فبإمكانك تغيير أسماء المعاملات أو أنواعها أو عددها، بل وتغيير نوع القيمة المعادة Return Value.. بل يمكنك أن تجعل إجراءً يحجب Shadows دالّة، أو دالة تحجب إجراءً.

- بل يمكنك أن تجعل متغيّرا يحجب دالّة بنفس الاسم ، أو خاصيّة تحجب خليّة بنفس الاسم ... باختصار: أيّ نوع من العناصر يستطيع أن يحجب أيّ نوع آخر.

افترض أنّك تريد تعريف متغيّر في الخليّة المشتقّة اسمه Name، وتصادف أنّ هناك خاصيّة في الخليّة الأصليّة اسمها Name.. إذن عرّف المتغيّر كالتالي:

**Public Shadows Name As String**

في هذه الحالة سيحجب المتغيّر الجديد الخاصيّة التي تحمل نفس الاسم، بحيث يحتلّ مكانها.. فلو استخدمت الاسم Name في أيّ موضع في الخليّة المشتقّة فسيشير إلى المتغيّر وليس إلى الخاصيّة.

ولكن ماذا لو أردت أن تستخدم الخاصيّة في الخليّة المشتقّة؟.. عندئذ عليك باستخدام الكلمة MyBase، كما سنرى في الفقرة القادمة.

**ملحوظة:**

لو لم تحدّد مجال الخليّة، فسيكون مجالها الافتراضي Friend.. ولو لم تحدّد مجال الوسائل والخصائص، فسيكون مجالها الافتراضيّ Public.

والآن لديك التطبيق InheritanceKeywords، لتتدرّب فيه على استخدام هذه الكلمات، وهو موجود في مجلّد برامج هذا الفصل.. افحصه جيّدا.

**الكلماتMyBase و MyClassو Me:**

مررنا من قبل بكلمة Me وقلنا إنّها تُشير إلى الكائن أو الخليّة الحاليّة، وكلمة MyBase وقلنا إنّها تشير إلى الخليّة التي ترثها الخليّة الحاليّة.. فما هي إذن فائدة كلمة MyClass؟

إنّ هذه الكلمة موجودة لحلّ المشاكل التي تنشأ من استبدال الدوال بين الخليّة الأمّ والخليّة المشتقّة.. انظر لهذا المثال:

**Class BaseClass**

**Public Overridable Sub MyMethod()**

**MsgBox("الخليّة الأمّ")**

**End Sub**

**Public Sub UseMe()**

**Me.MyMethod()**

**' سيتم تنفيذ هذه الدالة من الخليّة المشتقّة، نظرا لأنّها قد تمّ استبدالها**

**' معنى هذا أنّ الرسالة التي ستظهر لك هي "الخليّة المشتقّة"**

**End Sub**

**Public Sub UseMyClass()**

**MyClass.MyMethod()**

**' سيتم تنفيذ هذه الدالة من الخليّة الأمّ، حتّى لو كانت قد تمّ استبدالها**

**' معنى هذا أنّ الرسالة التي ستظهر لك هي "الخليّة الأمّ"**

**End Sub**

**End Class**

**Class DerivedClass : Inherits BaseClass**

**Public Overrides Sub MyMethod()**

**MsgBox("الخليّة المشتقّة")**

**End Sub**

**End Class**

والآن جرّب ما يلي في أي زر:

**Dim TestObj As DerivedClass = New DerivedClass()**

**TestObj.UseMe() ' ستظهر رسالة "الخليّة المشتقّة"**

**TestObj.UseMyClass() ' ستظهر رسالة "الخليّة الأمّ"**

أعتقد أنّ الفارق قد اتّضح: إنّ كلمة MyClass عمياء، لا ترى خارج حدود خليّتها، ولكنّ كلمة Me أكثر ذكاءً، فلو وجدت أنّ الوسيلة التي تستدعيها لها نسخة أحدث في خليّة مشتقّة، فإنّها تقوم باستخدام هذه النسخة البديلة، ولو كانت الخليّة المشتقّة تعمل كخليّة أمّ لخلايا أخرى، وكانت الوسيلة المعنيّة مستبدلة أيضا في هذه الخلايا، فإنّ كلمة Me تستدعي آخر نسخة من الدالة في هذه السلسة!

تذكّر أنّ كلمة Me افتراضيّة، وبإمكانك عدم كتابتها، بمعنى أنّ :

**Me.MyMethod()**

في المثال السابق مكافئة لـ:

**MyMethod()**