**الفصل التاسع**

**المجموعات Collections**

* [**الموضوعات المتقدّمة في المصفوفات:**](file:///C:\Users\Administrator\Desktop\vb.net\c-%20Classes%20الخلايا\09-%20Collections%20خلايا%20المجموعات\09-%20Collections%20خلايا%20المجموعات\0-%20الموضوعات%20المتقدّمة%20في%20المصفوفات.htm)

أسهل طريقة تعلمناها للتعامل مع عدد ضخم من البيانات المتشابهة، هي وضعها في مصفوفة Array، وبذلك يمكن التعامل معها في جمل تكراريّة Loops باستخدام رقم الخانة Index.

لهذا قد نقول: إنّ المصفوفة هي نوع من المجموعات Collections، التي تُستخدم لحفظ البيانات المتشابهة في النوع.

لهذا سنبدأ بشرح القدرات المتقدّمة للمصفوفات، التي تمنحها لك الخليّة System.Array، قبلَ أن نلجَ في موضوعنا.

* [**المجموعات:**](file:///C:\Users\Administrator\Desktop\vb.net\c-%20Classes%20الخلايا\09-%20Collections%20خلايا%20المجموعات\09-%20Collections%20خلايا%20المجموعات\1-%20المجموعات.htm)

لا تمنحنا المصفوفات كلّ ما نريد من الإمكانيات.. لهذا فإنّ لدينا العديد من أنواع المجموعات الأخرى، موجودة تحت العنوان System.Collections، الذي يمنحنا العديد من الخلايا Classes التي تمثّل أنواعا مختلفة من المجموعات، مثل: "المصفوفة القائمة" ArrayList، و "الجدول المختلط" HashTable و"القائمة المرتّبة" SortedList وغيرها.

* [**واجهة العداد IEnumerator وواجهة المقارنة IComparer :**](file:///C:\Users\Administrator\Desktop\vb.net\c-%20Classes%20الخلايا\09-%20Collections%20خلايا%20المجموعات\09-%20Collections%20خلايا%20المجموعات\2-%20واجهة%20العداد%20وواجهة%20المقارنة.htm)

تمكّنك واجهة العداد من المرور عبر كلّ عناصر المجموعة، بينما تمكّنك واجهة المقارنة من تحديد الطريقة التي تريد بها مقارنة كائنين.

* [**إنشاء مجموعاتك الخاصّة:**](file:///C:\Users\Administrator\Desktop\vb.net\c-%20Classes%20الخلايا\09-%20Collections%20خلايا%20المجموعات\09-%20Collections%20خلايا%20المجموعات\3-%20إنشاء%20المجموعات%20الخاصّة.htm)

حتّى لا يحرمك من شيء، يمنحك VB القدرة على إنشاء مجموعات خاصّة بك، حتّى تستطيع ضبط أدائها لتتناسب مع الخلايا التي تنشئها وتريد تخزينها بها.. كما أنّ هذا أيضا يمكّنك من إنشاء تراكيب Hierarchies عبارة عن خلايا متداخلة Nested Classes.

**الموضوعات المتقدّمة في المصفوفات**

* + - [**ترتيب المصفوفات:**](file:///C:\Users\Administrator\Desktop\vb.net\c-%20Classes%20الخلايا\09-%20Collections%20خلايا%20المجموعات\0-%20الموضوعات%20المتقدّمة%20في%20المصفوفات\1-%20ترتيب%20المصفوفات.htm)
    - [**البحث في المصفوفة:**](file:///C:\Users\Administrator\Desktop\vb.net\c-%20Classes%20الخلايا\09-%20Collections%20خلايا%20المجموعات\0-%20الموضوعات%20المتقدّمة%20في%20المصفوفات\2-%20البحث%20في%20المصفوفة.htm)
    - [**عمليات أخرى على المصفوفة:**](file:///C:\Users\Administrator\Desktop\vb.net\c-%20Classes%20الخلايا\09-%20Collections%20خلايا%20المجموعات\0-%20الموضوعات%20المتقدّمة%20في%20المصفوفات\3-%20عمليات%20أخرى%20على%20المصفوفة.htm)

**ترتيب المصفوفات:**

في الفصلين الثالث والرابع رأينا كيف نكتب كودا يرتّب المصفوفة.. وقلنا حينذاك إنّ ذلك لمجرّد التدريب، إذ يمكنك ترتيب المصفوفة مباشرةً، بمجرّد استدعاء الوسيلة "ترتيب" Sort الخاصّة بخليّة المصفوفات.

وأسهل شكل لهذه الوسيلة، يقبل معاملا واحدا، ذلك هو اسم المصفوفة التي سيتم ترتيبها:

**System.Array.Sort(اسم المصفوفة)**

ويتم ترتيب العناصر تبعا لنوعها: نصّيّة أم رقميّة أم تواريخ.. وخلاف ذلك ـ مثل احتواء المصفوفة على كائنات أو سجلات أو غيرها ـ سيحتاج ترتيب عناصر المصفوفة لبعض العمل، كما سنرى فيما بعد.. فإذا لم تكن واثقا من اتساق أنواع عناصر المصفوفة، فاستخدم تعبير Try…Catch، حتّى تتوقّى حدوث أيّ خطإ.

أمّا لو أحببت أن ترتّب جزءا فقط من المصفوفة، فاستخدم الصيغة التالية من وسيلة الترتيب:

**Array.Sort(اسم المصفوفة, خانة البداية, عدد الخانات)**

الأروع من هذا، هو قدرتك على ترتيب مصفوفة تبعا لقيم مصفوفة أخرى!

افترض أنّ لديك مصفوفةً لأسماء الطلبة، وأخرى لدرجاتهم في الامتحان، وأردت أن ترتب أسماء الطلبة تبعا لدرجاتهم في الامتحان.. في الماضي كانت هذه عمليّةً مملّة وتحتاج لكود مرهق، ولكن الآن يتمّ تنفيذها بخطوة واحدة، حيث سيكون لوسيلة الترتيب الصيغة التالية:

**Array.Sort(المصفوفة القائدة للترتيب, المصفوفة المراد ترتيبها)**

شيء رائع بالطبع، وإن كان شرطا أن تكون المصفوفتان أحاديتي البعد One Dimensional، ولهما نفس عدد الخانات.. منطقيّ طبعا.

وهناك صيغة رابعة، تمنحك القدرة على ترتيب جزء من مصفوفة بناءً على جزء من مصفوفة أخرى، كالتالي:

**Array.Sort(المصفوفة القائدة, المصفوفة المراد ترتيبها, البداية, عدد الخانات)**

ويمكنك أن تجرّب هذه الإمكانيات في تطبيق SortArrayByLength، والذي ستجده في مجلّد برامج هذا الفصل، والذي يقوم بترتيب المصفوفة على حسب طول نصوصها!

طبعا استنتجت كيف نفعل ذلك: في البداية نضع أطوال عناصر المصفوفة في مصفوفة أخرى، ثم نرتب على أساسها مصفوفة النصوص.

لاحظ أنّك لن تحتاج حتّى لترتيب مصفوفة الأطوال في البداية، فدالة الترتيب ستقوم بذلك بمفردها.. إن اللغة تبدو في منتهى الذكاء!

هذه فقط هي الجملة التي سترتب المصفوفتين:

**Array.Sort(MyStringsLen, MyStrings)**

**البحث في المصفوفة:**

الطريقة العامّة للبحث في المصفوفة، سواءٌ أكانت مرتّبةً أم لا، هي استخدام الوسيلة "رقم الخانة" IndexOf للبحث عن أوّل خانة يوجد فيها عنصر معيّن، أو الوسيلة "رقم آخر خانة" LastIndexOf للبحث عن آخر خانة يوجد فيها عنصر معيّن (وهي تبدأ البحث من نهاية المصفوفة بالطبع).. وفي حالة عدم العثور على عنصر البحث، تُرجع هاتان الوسيلتان -1.

وللدالتين الصيغة التالية:

**رقم العنصر = Array.IndexOf(المصفوفة, عنصر البحث)**

أو الصيغة التالية:

**رقم العنصر = System.Array.IndexOf(المصفوفة, عنصر البحث, خانة بداية البحث)**

وطبعا في حالة دالة البحث عن آخر عنصر، يبدأ البحث من رقم الخانة الذي تحدّده ويستمرّ فيما قبله من خانات.

كما أنّ هناك صيغة ثالثة، تحدّد فيها خانة البداية، وعدد الخانات التي سيتمم البحث فيها:

**رقم العنصر = System.Array.IndexOf(المصفوفة, عنصر البحث, خانة البداية, عدد الخانات)**

وهناك طريقة خاصّة فقط بالبحث في المصفوفات المرتّبة، ولك أن تتخيّل طبعا أّنها أسرع بكثير، لذا فهي أنسب للمصفوفات الضخمة.. هذه الطريقة هي "البحث الثنائي" BinarySearch، وقد تعرّفنا علي كود هذا البحث بالتفصيل ـ كنوعٍ من التدريب ـ في الفصل الرابع.

ولهذه الوسيلة الصيغة التالية:

**رقم الخانة = System.Array.BinarySearch(المصفوفة, عنصر البحث)**

أو الصيغة:

**رقم الخانة = System.Array.BinarySearch(المصفوفة, عنصر البحث, خانة البداية, عدد الخانات)**

وفي حالة عدم العثور على عنصر البحث، تعثر الدالة على أكبر عنصر يليه في الترتيب في المصفوفة، وتُعيد رقم خانته، لكن بإشارة سالبة، مطروحا منه 1، حتّى يمكنك أن تعرف أنّها لم تعثر على العنصر المطلوب.. فمثلا لو وجدت العنصر الأكبر في الخانة 10، فإنّها تُرجع لك الرقم -11.

**ملحوظة:**

كل هذه الطرق الثلاث للبحث حسّاسة لحالة الأحرف Case Sensetive.

ويمكنك أن تختبر هذه الوسيلة في تطبيق ArraySearch المرفق مع هذا الفصل.

**عمليات أخرى على المصفوفة:**

**الوسيلة "عكس" Reverse:**

لإرجاع مصفوفة جديدة ترتيب عناصرها عكس ترتيب المصفوفة الأصليّة.

مثال:

**Dim Names(99) As String**

**' الكود المناسب لملء مصفوفة الأسماء**

**Dim ReverseNames() As String**

**ReverseNames = Array.Reverse(Names)**

**الوسيلة "نسخ" Copy:**

لنسخ جزء من المصفوفة إلى مصفوفة أخرى، ولها الصيغة التالية:

**System.Array.Copy(مصفوفة المصدر, المصفوفة المستهدفة, عدد الخانات المنسوخة)**

أو الصيغة التالية:

**System.Array.Copy(مصفوفة المصدر, خانة بداية النسخ في مصفوفة المصدر,**

**المصفوفة المستهدفة, خانة بداية النسخ في المصفوفة المستهدفة, عدد الخانات)**

ويمكنك استخدام هذه الوسيلة لكتابة دالة ذكيّة لحذف أو إضافة أيّ عنصر أو مجموعة من العناصر من المصفوفة، وذلك كالتالي:

1. انسخ جزء المصفوفة السابق لموضع الحذف أو الإضافة إلى مصفوفة جديدة.
2. في حالة الإضافة، أضف العنصر أو العناصر التي تريدها للمصفوفة الجديدة.
3. أضف جزء المصفوفة الأصليّة التالي لموضع الحذف أو الإضافة إلى المصفوفة الجديدة.

وبهذا تكون قد تخلّصت من الطريقة التقليديّة، التي تُضطرّ فيها لزحزحة عناصر المصفوفة للأمام في حالة الإضافة، أو للخلف في حالة الحذف، عن طريق جملة تكراريّة عقيمة.. وكتدريب، حاول أن تكتب دالتين، إحداهما للحذف والأخرى للإضافة، بناءً على الخوارزميّة الموضّحة هنا، وأضفهما لقالب الكود Module الذي أسميناه Utilities في الفصل السادس.

**الوسيلة "نسخ إلى" CopyTo:**

وهي تنسخ مصفوفة بكاملها إلى مصفوفة أخرى، ولها الصيغة التالية:

**System.Array.CopyTo(مصفوفة المصدر, المصفوفة المستهدفة)**

**المجموعات**

* + [**مجموعة المصفوفة القائمة ArrayList Collection:**](file:///C:\Users\Administrator\Desktop\vb.net\c-%20Classes%20الخلايا\09-%20Collections%20خلايا%20المجموعات\1-%20المجموعات\1-%20مجموعة%20المصفوفة%20القائمة.htm)
  + [**مجموعة الجدول المختلط HashTable Collection:**](file:///C:\Users\Administrator\Desktop\vb.net\c-%20Classes%20الخلايا\09-%20Collections%20خلايا%20المجموعات\1-%20المجموعات\2-%20مجموعة%20الجدول%20المختلط.htm)
  + [**خليّة القائمة المرتّبة SortedList Class:**](file:///C:\Users\Administrator\Desktop\vb.net\c-%20Classes%20الخلايا\09-%20Collections%20خلايا%20المجموعات\1-%20المجموعات\3-%20خليّة%20القائمة%20المرتّبة.htm)
  + [**خليتا الرصّة Stack والطابور Queue:**](file:///C:\Users\Administrator\Desktop\vb.net\c-%20Classes%20الخلايا\09-%20Collections%20خلايا%20المجموعات\1-%20المجموعات\4-%20خليتا%20الرصّة%20والطابور.htm)
  + [**خليّة مصفوفة الخانات الثنائيّة BitArray Class:**](file:///C:\Users\Administrator\Desktop\vb.net\c-%20Classes%20الخلايا\09-%20Collections%20خلايا%20المجموعات\1-%20المجموعات\5-%20خليّة%20مصفوفة%20الخانات%20الثنائيّة.htm)
  + [**سجلّ المتّجه الثنائي Structure** **BitVector32 وسجلّ مقطع المتّجه الثنائي Structure** **BitVector32.Section:**](file:///C:\Users\Administrator\Desktop\vb.net\c-%20Classes%20الخلايا\09-%20Collections%20خلايا%20المجموعات\1-%20المجموعات\6-%20سجلّ%20المتّجه%20الثنائي%20وسجلّ%20مقطع%20المتّجه%20الثنائي.htm)

**مجموعة المصفوفة القائمة ArrayList Collection**

* + - [**وظيفتها وكيفيّة تعريفها:**](file:///C:\Users\Administrator\Desktop\vb.net\c-%20Classes%20الخلايا\09-%20Collections%20خلايا%20المجموعات\1-%20مجموعة%20المصفوفة%20القائمة\0-%20وظيفتها%20وكيفيّة%20تعريفها.htm)
    - [**إضافة وإزالة العناصر:**](file:///C:\Users\Administrator\Desktop\vb.net\c-%20Classes%20الخلايا\09-%20Collections%20خلايا%20المجموعات\1-%20مجموعة%20المصفوفة%20القائمة\1-%20إضافة%20وإزالة%20العناصر.htm)
    - [**نسخ العناصر:**](file:///C:\Users\Administrator\Desktop\vb.net\c-%20Classes%20الخلايا\09-%20Collections%20خلايا%20المجموعات\1-%20مجموعة%20المصفوفة%20القائمة\2-%20نسخ%20العناصر.htm)
    - [**ترتيب المصفوفة القائمة:**](file:///C:\Users\Administrator\Desktop\vb.net\c-%20Classes%20الخلايا\09-%20Collections%20خلايا%20المجموعات\1-%20مجموعة%20المصفوفة%20القائمة\3-%20ترتيب%20المصفوفة%20القائمة.htm)
    - [**البحث في المصفوفة القائمة:**](file:///C:\Users\Administrator\Desktop\vb.net\c-%20Classes%20الخلايا\09-%20Collections%20خلايا%20المجموعات\1-%20مجموعة%20المصفوفة%20القائمة\4-%20البحث%20في%20المصفوفة%20القائمة.htm)
    - [**المرور عبر عناصر المصفوفة القائمة:**](file:///C:\Users\Administrator\Desktop\vb.net\c-%20Classes%20الخلايا\09-%20Collections%20خلايا%20المجموعات\1-%20مجموعة%20المصفوفة%20القائمة\5-%20المرور%20عبر%20عناصر%20المصفوفة%20القائمة.htm)

**وظيفتها وكيفيّة تعريفها:**

تأخذ هذه الخليّة من المصفوفات قدرتها على تخزين عدد كبير من العناصر المتشابهة، حيث تمكّنك من ترتيبها والبحث فيها.. وتأخذ من القوائم Lists قدرتها على إضافة العناصر في أيّ موضع وحذفها من أيّ موضع.

ولتعريف متغير من هذه الخليّة، استخدم تعبيرا كالتالي:

**Dim aList As New ArrayList**

لاحظّ أنّك لا تحتاجُ لتحديد عدد الخانات، فهي 16 خانة مبدئيّا.. ولو كتبت تعبيرا كالتالي:

**Dim aList(2) As New ArrayList**

فهذا معناه أنّك تعرّف مصفوفة من خانتين، كل خانة منها تحتوي على مصفوفة قائمة!

ولكن ماذا سنفعل بـ 16 خانة فقط؟

إنّ للمصفوفة القائمة خاصيّة تسمّى السعة Capacity، يمكنك استخدامها لتحديد عدد الخانات التي تريد:

**aList.Capacity = 1000**

ويمكنك أن تعرف عدد العناصر الموجودة في المصفوفة القائمة، باستخدام خاصيّة "العدد" Count، التي دائما ما تكون أصغر من أو تساوي السعة.

وليست بك حاجة ماسّة لتحديد السعة، ففي كلّ مرّة تتجاوز فيها سعة المصفوفة القائمة، يتم مضاعفة هذه السعة.. ولكن عندما تحذف العناصر، لا يتم تقليص السعة.. وإن كان من الممكن تحديد السعة في المواقف التي تخشى فيها ضياع مساحة ضخمة من الذاكرة، خاصّةً عندما تتعامل مع عدد ضخم من العناصر.

وحينما تثقُ أنّك لن تضيف أيّة عناصر للمصفوفة القائمة، استخدم الوسيلة "تقليص الحجم" TrimToSize، وذلك لتجعل السعة مساوية لعدد العناصر بالضبط.

**إضافة وإزالة العناصر:**

تمنحك المصفوفة القائمة عددا من الوسائل لهذا الغرض:

**"إضافة" Add:**

لإضافة عنصر جديد استخدم الوسيلة "إضافة" Add، بالصيغة التالية:

**رقم الخانة = aList.Add(عنصر)**

وترجع هذه الوسيلة رقم الخانة التي تمّت إضافة العنصر فيها.. طبعا لو كانت المصفوفة القائمة غير مرتّبة فستتم الإضافة في نهايتها، ولو كانت مرتّبة، فستتم الإضافة في الموضع المناسب للترتيب.

تعال نعرّف سجلا كالتالي:

**Structure Person**

**Dim LastName As String**

**Dim FirstName As String**

**Dim Phone As String**

**Dim EMail As String**

**End Structure**

والآن تعال نرى كيف نضيف المتغيرات من نوع هذا السجلّ للمصفوفة القائمة:

**Dim Persons As New ArrayList**

**Dim p As New Person**

**p.LastName = "Last Name"**

**p.FirstName = "First Name"**

**p.Phone = "Phone"**

**p.EMail = "name@server.com"**

**Persons.Add(p)**

**p.LastName = "another name"**

**Persons.Add(p)**

وللتحقّق من وجود عنصر بعينه في المصفوفة القائمة قبل أن تحاول إضافته (لمنع التكرار)، استخدم الوسيلة "تحتوي" Contains.. انظر للمثال التالي:

**If Persons.Contains(p) Then**

**MsgBox("غير مسموح بالتكرار")**

**Else**

**Persons.Add(p)**

**End If**

لعلك لاحظت التشابه في الوسائل مع وسائل القائمة List Control.. سيتضح لك هذا التشابه أكثر، عندما أخبركَ أنّ لديك كذلك الوسائل التالية:

**"إدراج" Insert:**

تُستخدم لإضافة العنصر في الموضع الذي تريده.. ولها نفس الاسم والصيغة والوظيفة كما في القوائم ListBoxes:

**aList.Insert(رقم الخانة, عنصر)**

**"إضافة مجموعة عناصر" AddRange:**

الجملة التالية تضيف أربع عناصر لنهاية لمصفوفة القائمة:

**aList.AddRange(New String() {"محمد", "حمدي", "يرحب", "بكم"})**

لاحظ مدى الاختصار الذي قدّمته لنا هذه الجملة.. ففي سطر واحد عرّفنا مصفوفة نصّيّة (لم نعطها اسما لأنّنا لن نستخدمها فيما بعد) ووضعنا بها أربع قيم ابتدائيّة، ثمّ حمّلناها في المصفوفة القائمة.. وهو ما يكافئ:

**Dim S As String() = {"محمد", "حمدي", "يرحب", "بكم"}**

**aList.AddRange(S)**

مع ملاحظة أنّ الوسيلة AddRange لا تقبل المصفوفات فقط كمعامل.. بل أيّ خليّة تطبّق Implement الواجهة ICollection.. مثل المصفوفة القائمة وباقي أنواع المجموعات التي سنتعرّف عليها لاحقا.

**"إدراج مجموعة عناصر" InsertRange:**

مشابهة للوسيلة السابقة، ولكنّها تسمح لك بإدراج مجموعة العناصر بدءا من الموضع الذي تريده.. ومشابهة لتلك الخاصة بالقوائم.

**"استبدال مجموعة عناصر" SetRange:**

استخدم هذه الوسيلة لو أردت أن تستبدل مجموعة عناصر بغيرها.. انظر للمثال التالي، لتعرف كيف يمكن تغيير العناصر من العنصر الخامس إلى التاسع:

**Dim words() As String = {"Just", "a", "few", "more", "words"}**

**aList.SetRange(5, words)**

وهو ما يمكن اختصاره كالتالي:

**aList.SetRange(5, New String(){"Just", "a", "few", "more", "words"})**

**"حذف" Remove:**

تحذف عنصرا من المصفوفة القائمة.. ولها الصيغة التالية:

**aList.Remove(عنصر)**

مثال:

**Dim R1 As New Rectangle(10, 10, 100, 100)**

**Dim R2 As New Rectangle(5, 7, 30, 40)**

**aList.Add(R1)**

**aList.Add(R2)**

**Dim R As New Rectangle(10, 10, 100, 100)**

**aList.Remove(R) ' سيتم حذف المستطيل الأوّل في المصفوفة القائمة**

**حذف من موضع "RemoveAt":**

تحذف العنصر الذي قمت بتحديد رقم خانته.. ولها الصيغة التالية:

**aList.RemoveAt(رقم الخانة)**

**"حذف مجموعة عناصر" RemoveRange:**

يمكنك حذف مجموعة متتالية من العناصر باستخدام هذه الوسيلة.. ولها الصيغة التالية:

**aList.RemoveRange(رقم خانة البداية, عدد الخانات)**

**نسخ العناصر:**

لديك عدد من الوسائل لنسخ العناصر من مصفوفة قائمة إلى أخرى، أو تعديل عدد من العناصر في مصفوفة قائمة:

**"اقرأ مجموعة عناصر" GetRange:**

تنسخ مجموعة من العناصر المتتالية من مصفوفة قائمة إلى أخرى.. ولها الصيغة التالية:

**newList = ArrayList.GetRange(رقم بداية الخانات, عدد الخانات)**

والجملة التالية تنسخ العناصر من الثالث إلى الخامس من مصفوفة قائمة، وتضعها في الخانات الثلاثة الأولى من أخرى:

**bList.InsertRange(0, aList.GetRange(3, 3))**

**"استنساخ" Clone:**

تصنع نسخة مطابقة من المصفوفة القائمة:

**bList = aList.Clone**

**"تكرار" Repeat:**

تملأ مصفوفة قائمة بنفس العنصر:

**newList = aList.Repeat(العنصر, العدد)**

**"عكس" Reverse:**

تعكس ترتيب العناصر في المصفوفة القائمة.. ولها الصيغة:

**aList.Reverse()**

أو الصيغة:

**aList.Reverse(رقم بداية الخانات, عدد الخانات)**

ولديك وسيلتان لنسخ العناصر من مصفوفة قائمة إلى مصفوفة عاديّة Array، وهما:

**"إلى مصفوفة" ToArray:**

انظر للمثال التالي:

**Dim aList As New ArrayList()**

**aList.AddRange(New String() {"محمد", "حمدي", "يرحب", "بكم"})**

**Dim Y As Array = aList.ToArray()**

**MsgBox(Y(0)) ' تعرض الرسالة "محمد"**

**"نسخ إلى" CopyTo:**

تمتاز الوسيلة السابقة، بأنّها لا تتطلّب تحديد حجم المصفوفة أولا.. وهو ما يجب أن تراعيه عند استخدام الوسيلة CopyTo، بحيث تحتوي مبدئيّا على عدد من الخانات أكبر من أو يساوي عدد العناصر التي ستنسخها إليها.. وإلا فسيحدث خطأ برمجيّ.

تعال تستخدم هذه الوسيلة لأداء نفس ما يفعله المثال السابق:

**Dim aList As New ArrayList()**

**aList.AddRange(New String() {"محمد", "حمدي", "يرحب", "بكم"})**

**Dim Y(aList.Count - 1) As String ' حدّدنا حجم المصفوفة أولا**

**aList.CopyTo(Y)**

**MsgBox(Y(0)) ' تعرض الرسالة "محمد"**

و تمتاز CopyTo بامتلاكها صيغتين أخريين، تمكّنانك من تحديد موضع بداية النسخ في المصفوفة المستهدفة، وعدد العناصر المنسوخة.

**ترتيب المصفوفة القائمة:**

لا جديد.. استخدم الوسيلة Sort.. وهي تنطبق على النسخة الحاليّة من المصفوفة القائمة، فلا حاجة لإرسال اسم المصفوفة القائمة إليها:

**aList.Sort()**

ورغم تشابهها مع المصفوفات العاديّة، إلا إنّ وسيلة الترتيب الخاصّة بالمصفوفة القائمة لا تمنحك القدرة على ترتيبها تبعا لمصفوفة قائمة أخرى.

ويجب أن تنتبه إلى احتمال فشل ترتيب المصفوفة القائمة، نتيجة لاختلاف أنواع عناصرها، لهذا يجب عليك استخدام كود كالتالي:

**Dim Sorted As Boolean = True**

**Try**

**aList.Sort()**

**Catch SortException As Exception**

**MsgBox(SortException.Message)**

**Sorted = False**

**End Try**

**If Sorted Then**

**' تعامل مع المصفوفة القائمة المرتّبة**

**Else**

**' تعامل مع المصفوفة القائمة غير المرتّبة**

**End If**

**البحث في المصفوفة القائمة:**

لا جديد.. بالإضافة إلى الوسيلة **Contains**، لديك نفس وسائل البحث الثلاث: **IndexOf** و **LastIndexOf** و **BinarySearch**،.. وهي لا تختلف في شيء عن تلك الخاصّة [بالمصفوفات](file:///C:\Users\Administrator\Desktop\vb.net\0-%20الموضوعات%20المتقدّمة%20في%20المصفوفات\2-%20البحث%20في%20المصفوفة.htm).. هل تحتاج لمزيد من الشرح؟!

**المرور عبر عناصر المصفوفة القائمة:**

لديك الطريقة التكراريّة التقليديّة كالتالي:

**For i = 0 To ArrayList.Count – 1**

**ArrayList(i).Item = ""**

**Next**

كما يمكنك استخدام تعبير For Each…Next كالتالي:

**Dim itm As Object**

**For Each itm In ArrayList**

**ArrayList(i) = ""**

**Next**

لاحظّ أنّ خاصيّة العنصر Item هي خاصيّة افتراضيّة Defult ويمكن عدم كتابة اسمها كما في المثال السابق.

لاحظّ أيضا أنّنا استخدمنا متغيرا من النوع Object للمرور عبر عناصر المصفوفة القائمة، ولكن لو كانت كل هذه العناصر من نفس النوع (أعدادا صحيحة مثلا)، فيمكن استخدام متغير من هذا النوع.

وهناك وسيلة ثالثة صالحة لكلّ أنواع المجموعات Collections، وهي إنشاء عدّاد Enumerator للمجموعة، واستخدامه للمرور عبر كل العناصر.. وسنناقش هذا في نهاية الفصل بإذن الله.

**مجموعة الجدول المختلط HashTable Collection:**

الميزة الرئيسيّة في الجدول المختلط، هي قدرته على التعامل مع العنصر بالرمز الدالّ عليه Key، بدلا من رقم الخانة.. معنى هذا أنّك تخزّن معلومتين لكلّ عنصر في الجدول: القيمة Value والرمز Key.

ونظرا للتشابه الجليّ بين الجدول المختلط والمصفوفة القائمة، فسنركّز هنا على الاختلافات، وعليك أنت باستطلاع باقي الخصائص والوسائل، والتي لن تكون جديدة عليك.

ولاستخدام هذه الخليّة، ابدأ بتعريف متغيّر من نوعها، كالتالي:

**Dim hTable As New HashTable**

ولإضافة العناصر، استخدم الصيغة التالية:

**hTable.Add(رمز, قيمة)**

حيث الرمز هو أي نصّ مسموح به، والقيمة من أي نوع (يمكن أن تكون كائنا).

انظر للمثال التالي، لتخزين درجات حرارة مدينتين:

**Dim Temperatures As New HashTable**

**Temperatures.Add("القاهرة", 30)**

**Temperatures.Add("دمياط", 25)**

أهمّ ما يجب أن تراعيه، هو أنّ الرمز يجب أن يكون فريدا Unique لكلّ قيمة، حيثُ لن يُسمح لك بتكرار الرموز.

**ملحوظة:**

يمكن أن يكون الرمز Key أيّ نوع من الكائنات Objects وليس نصوصا فقط.. في هذه الحالة يستخدم الجدول المختلط الوسيلتين GetHashCode و Equals، اللتين تخصّان كلّ كائن، ليعرف على أيّ أساس سيقارن الرموز ويقوم بترتيبها.. وبإمكانك أن تعيد تعريف هاتين الوسيلتين، لتحدّد الكيفيّة التي على أساسها تتمّ مقارنة كلّ كائنين معا.. (راجع موضع تخصيص الأعضاء الافتراضيّة في الفصل 7، إنشاء الخليّة الصغرى).

ولتغيير درجة حرارة مدينة دمياط بعد ذلك، استخدم جملة كالتالية:

**Temperatures.Item("دمياط") = 20**

أو باختصار:

**Temperatures("دمياط") = 20**

ولحذف أيّ عنصر من الجدول، يمكنك حذفه عن طريق رمزه، ولا توجد وسيلة غير هذه:

**hTable.Remove(رمز العنصر)**

وللتأكّد ممّا إذا كان أحد الرموز مستخدما في الجدول، استخدم الوسيلة المنطقيّة Boolean "يحتوي على الرمز" ContainsKey، حيث تعطيك القيمة True إذا كان الرمز موجودا في الجدول:

**hTable.ContainsKey(رمز)**

ولسبب ما، توجد وسلة أخرى مماثلة لنفس هذه الوسيلة تماما في كلّ شيء إلا في الاسم، وهي Contains:

**hTable.ContainsKey(رمز)**

مثال:

**Dim value As New Rectangle(100, 100, 50, 50)**

**Dim key As String = "object1"**

**If Not hTable.ContainsKey(key) Then**

**hTable.Add(key, value)**

**End If**

وبالمثل لديك الوسيلة ContainsValue للتأكّد من وجود قيمة ما في الجدول:

**hTable.ContainsValue(قيمة)**

وللتعامل مع رموز الجدول، يمكنك استخدام مجموعة الرموز Keys Collection، وللتعامل مع القيم، استخدم مجموعة القيم Values Collection، ولهما نفس الأعضاء التقليديّة للمجموعات.. انظر للمثال التالي الذي يريك كيف تمرّ على كلّ قيم الجدول:

**Dim itm As Object**

**For Each itm In hTable.Values**

**Console.WriteLine(itm)**

**Next**

والدالة التالية تريك كيف يمكن المرور عبر كل عناصر الجدول المختلط:

**Private Function ShowHashTableContents(ByVal table As Hashtable) As String**

**Dim msg As String**

**Dim element, key As Object**

**msg = "الجدول يحتوي على " & table.Count.tostring & " من العناصر"**

**For Each key In table.keys**

**element = table.Item(key)**

**msg = msg & vbCrLf**

**msg += " نوع العنصر = " & element.GetType.ToString & vbCrLf**

**msg = msg & " رمز العنصر= " & Key.ToString**

**msg += " قيمة العنصر= " & element.ToString & vbCrLf**

**Next**

**Return(msg)**

**End Function**

ولتجربة هذه الدالّة استخدم الكود التالي:

**Dim HT As New HashTable**

**' الكود المناسب لملء الجدول المختلط بالعناصر**

**MsgBox (ShowHashTableContents(HT))**

وهناك طريقة أخرى للمرور عبر عناصر الجدول.. إنّ كلّ عنصر من عناصره يتكوّن، من رمز وقيمة.. لهذا فإنّ هذا العنصر هو سجلّ من النوع DictionaryEntry.. لهذا يمكنك استخدام الكود التالي للمرور عبر كلّ عناصر الجدول:

**Dim HEntry As DictionaryEntry**

**For Each HEntry In hTable**

**Console.WriteLine(HEntry.Key)**

**Console.WriteLine(HEntry.Value)**

**Next**

ولمزيد من التدريب على استخدام الجدول المختلط، افحص مشروع WordFrequencies في مجلّد برامج هذا الفصل.

**خليّة القائمة المرتّبة SortedList Class:**

القائمة المرتّبة هي خليط من المصفوفة والجدول المختلط، حيث يمكنك التعامل مع عناصرها برقم الخانة (كالمصفوفة) أو بالرمز (كالجدول المختلط).. هذا بالإضافة إلى أنّ عناصرها مرتّبة دائما على حسب رموزها، وحتّى لو أدخلت الرموز غير مرتّبة، فسيتمّ ترتيبها آليّا.

ويمكنك تعريف متغيّر منها كالتالي:

**Dim sList As New SortedList**

ونظرا لأنّ معظم خصائص هذه الخليّة ووسائلها ستكون مألوفة لك، فسنشرح هنا فقط ما هو جديد منها.

فمثلا، إذا أردت تغيير قيمة خانة ما عن طريق رقمها، استخدم الطريقة التالية:

**sList.SetByIndex(رقم الخانة, عنصر)**

ولا توجد وسيلة مماثلة لتغيير رمز العنصر، فإذا أردت أن تفعل ذلك، فعليك أن تحذف العنصر أولا، ثم تعيد إدخاله برمز جديد.

ويمكنك إفراغ المصفوفة المرتّبة من كلّ محتوياتها باستخدام الوسيلة Clear.. ولا تنس بعد أن تفعل ذلك أن تستخدم الوسيلة "تقليص الحجم" TrimToSize، لتعود سعة المصفوفة المرتّبة إلى 16 خانة.

والمثال التالي يريك كيف نبني قائمة مرتّبة ونملؤها بعشرة عناصر ثمّ نطبع كلّ عناصرها.. لاحظّ أنّنا سنجعل الرموز في حالتنا هذه أرقاما.

**Public Sub Button1\_Click(ByVal sender As System.Object, \_**

**ByVal e As System.EventArgs) Handles Button1.Click**

**Dim sList As New System.Collections.SortedList()**

**' ملء القائمة المرتّبة**

**sList.Add(16, "عنصر3 ")**

**sList.Add(10, "عنصر9")**

**sList.Add(15, "عنصر 4")**

**sList.Add(17, "عنصر2")**

**sList.Add(11, "عنصر8")**

**sList.Add(14, "عنصر5")**

**sList.Add(18, "عنصر1")**

**sList.Add(12, "عنصر7")**

**sList.Add(19, "عنصر0")**

**sList.Add(13, "عنصر6")**

**Dim SLEnum As IDictionaryEnumerator = sList.GetEnumerator()**

**‘ اطبع أزواج الرموز والقيم**

**While SLEnum.MoveNext**

**Console.WriteLine("الرمز = " & SLEnum.Key.Tostring & ", القيمة = " & \_**

**SLEnum.Value.ToString**

**End While**

**End Sub**

وللبحث في المصفوفة القائمة عن قيمة معيّنة، استخدم الوسيلة IndexOfValue، حيث ستعطيك رقم الخانة التي توجد فيها هذه القيمة، فإذا لم تجدها، فإنّها ترجع -1.. مثال:

**Console.WriteLine(sList.IndexOfValue("عنصر 7"))**

وللبحث عن رمز معيّن، استخدم الوسيلة IndexOfKey:

**Console.WriteLine(sList.IndexOfKey(17))**

أمّا لو كنت تعرف رقم الخانة، فيمكنك أن تقرأ قيمة هذه الخانة بالوسيلة GetByIndex، ويمكنك أن تقرأ رمزها عن طريق الوسيلة GetKey.

فمثلا، إذا أردت أن تعرف الرمز المناظر لقيمة معيّنة، فاحصل أولا على رقم الخانة، ثم احصل عن طريقها على الرمز، كما يلي:

**MsgBox (sList.GetKey(sList.IndexOfValue("عنصر 7")))**

ويمكنك أن تستخدم الوسيلة GetKeyList لتحصل على واجهة قائمة List Interface (IList) بكل رموز القائمة المرتّبة.. انظر للمثال التالي:

**Dim keys As IList**

**keys = slist.GetKeyList()**

**Dim key As Integer**

**For Each key In Keys**

**Console.WriteLine(key)**

**Next**

بالمثل، فإنّ لديك الوسيلة GetValueList، للحصول على قائمة بكل قيم القائمة المرتّبة.. وفي المثال التالي نستخدم هذه الوسيلة لنسخ قيم ورموز القائمة المرتّبة إلى قائمة مرتّبة جديدة:

**Dim AllKeys As New ArrayList()**

**AllKeys.InsertRange(0, sList.GetValueList)**

ولمزيد من التدريب، افحص مشروع SortedList في مجلّد برامج هذا الفصل.

**خليتا الرصّة Stack والطابور Queue:**

من الخلايا الأخرى التي يمنحها لك الفضاء System.Collections، الخليتان الشهيرتان: "الرصّة" Stack و"الطابور أو الصفّ" Queue، وأهمّ ما يميّزهما عن المصفوفات والقوائم، هو ترتيب دخول وخروج العناصر، فهما على خلاف باقي المجموعات، لا يسمحان لك باختيار المكان الذي تضع فيه العناصر.. كل ما عليك أن تفعله، هو أن تُدخل العناصر، فإذا أردت أن تقرأ أو تحذف أحدها، فلن يكون أمامك سوى قراءة أو حذف العنصر الموجود في قمّة الرصّة أو في قمّة الطابور.

خذ الرصّة مثلا: إنّها تعمل طبقا لمبدإ "آخر من يدخل هو أوّل من يخرج" Last In First Out (LIFO)، بمعنى أنّك لو أدخلت مجموعة من العناصر في الرّصّة واحدا بعد الآخر باستخدام الوسيلة "ادفع" Push، فإنّ بإمكانك استخدام الوسيلة "اسحب" Pop لإخراج آخر عنصر قد أدخلته، حيث تتمّ إزالته من الرصّة، ولو استخدمتها مرّةً ثانيةً فستُخرجُ لك العنصر الذي سبقه في الدخول... وهكذا، حتّى تفرغ الرصّة تماما.. حاول أن تتخيّل هذا الأمر كرصّة من الأطباق: آخر طبق تضعه على الرصّة هو أوّل طبق تأخذه منها.

أمّا الطابور فهو يتصرّف بطريقة عكسيّة، فأوّل من يدخل الطابور هو أوّل من يخرج منه: First In First Out (FIFO).. ويمكنك أن تضيف العناصر في نهاية الطابور باستخدام الوسيلة Enqueue، ويمكنك إخراج العناصر من بداية الطابور بواسطة الوسيلة Dequeue.

حاول أن تتخيّل هذا الأمر كطابور القوى العاملة: أوّل من يتقدّم لهذا الطابور هو أوّل من يحصل على الوظيفة.. ولحسن الحظّ أنّ الوساطة لا تتدخّل في عمل الكمبيوتر، وإلا لكانت مهزلة!!

ويمكنك أن تقرأ العنصر الموجود في قمّة الرصّة أو الطابور (دون أن تخرجه منهما) باستخدام الوسيلة "القمّة" Peek.

ويمنك أن تنسخ محتويات هاتين الخليتين لمصفوفة عن طريق الوسيلة ToArray، كما يمكنك أن تنسخها لعدّاد عن طرق الوسيلة GetEnumerator، وبذلك يمكنك التعامل معها بالطرق التقليديّة.

ولمزيد من التدريب على استخدام الرصّة والطابور، افحص مشروع StackAndQueue في مجلّد برامج هذا الفصل.

**خليّة مصفوفة الخانات الثنائيّة BitArray Class:**

هذه الخليّة تحلّ لمبرمج VB مشاكل عديدة، فهي تسمح له بالتعامل مع الخانات الثنائيّة Bits، وهي عمليّة هامّة جدا، خاصّة عند ضغط وفكّ الملفّات، وغير ذلك.

إنّ كلّ خانة في المصفوفة الثنائيّة BitArray تمثّل خانة ثنائيّة Bit، وأنت تعرف أنّ لها قيمتين فقط: إمّا صفر (ويمثّل في المصفوفة بالقيمة False) وإمّا 1 (ويمثّل في المصفوفة بالقيمة True).

ولتعريف المصفوفة الثنائيّة 6 صيغ:

الأولى تعرّف عدد الخانات فقط:

**Dim myBA1 As New BitArray(5)**

والثانية تعرّف عدد الخانات والقيمة الابتدائيّة لكلّ خانة:

**Dim myBA2 As New BitArray(5, False)**

والثالثة تملأ خانات المصفوفة الثنائيّة من مصفوفة وحدات Bytes، وأنت تعرف بالطبع أنّ كل وحدة Byte = 8 خانات Bits:

**Dim myBytes() As Byte = {1, 2, 3, 4, 5}**

**Dim myBA3 As New BitArray(myBytes)**

لاحظ أنّ عدد خانات المصفوفة الثنائيّة myBA3 = 5 × 8 = 40 خانة.

والصيغة الرابعة تملأ خانات المصفوفة الثنائيّة من مصفوفة متغيرات منطقيّة Boolean:

**Dim myBools() As Boolean = {True, False, True, True, False}**

**Dim myBA4 As New BitArray(myBools)**

والصيغة الخامسة تملأ خانات المصفوفة الثنائيّة من مصفوفة أعداد صحيحة، وأنت تعرف أنّ العدد الصحيح Integer يتكوّن من 32 خانة ثنائيّة.

**Dim myInts() As Integer = {6, 7, 8, 9, 10}**

**Dim myBA5 As New BitArray(myInts)**

أمّا الصيغة الأخيرة فهي تملأ خانات المصفوفة الثنائيّة من مصفوفة ثنائيّة أخرى:

**Dim myBA6 As New BitArray(myBA5)**

**ومن خصائص هذه الخليّة:**

عدد الخانات **Count** ـ العنصر **Item** ـ الطول **Length** (وهي مكافئة لعدد الخانات، ما عدا أنّك تستطيع تغيرها زيادةً ونقصانا).

**وتمنحك هذه الخليّة بعض الوسائل الهامّة، ومنها:**

**And و Or و Xor:**

أرسل لأيّ من هذه الوسائل مصفوفة ثنائيّة كمعامل، لتعيد لك مصفوفة ثنائيّة جديدة، عبارة عن ناتج إجراء عمليّة And أو Or أو Xor بين كل خانة فيها وما يناظرها في المصفوفة الثنائيّة الحاليّة:

**MyBA1 = myBA2.And(myBA3)**

**MyBA1 = myBA2.Or(myBA3)**

**MyBA1 = myBA2.Xor(myBA3)**

**Not:**

تعيد لك مصفوفة ثنائيّة، كلّ خانة من خاناتها هي عكس الخانة المناظرة لها في المصفوفة الحاليّة (الصفر يصبح 1، والواحد يصبح صفرا):

**MyBA1 = myBA2.Not()**

**نسخ إلى CopyTo:**

استخدم هذه الوسيلة لنسخ كل محتويات المصفوفة الثنائيّة إلى مصفوفة عاديّة.. ولك أن تتوقّع أن نوع هذه المصفوفة لا بدّ أن يكون إمّا منطقيّا Boolean أو وحدة ذاكرة Byte أو عددا صحيحا Integer، حتّى يكون متوافقا مع المصفوفة الثنائيّة:

**MyBA1.CopyTo(مصفوفة, خانة البداية النسخ في المصفوفة)**

**اقرأ Get:**

تقرأ قيمة خانة معيّنة في المصفوفة الثنائيّة:

**Dim B As Boolean = MyBA1.Get(3)**

وهي مكافئة لاستخدام خاصيّة Item:

**Dim B As Boolean = MyBA1.Item(3)**

**ضع Set:**

تضع قيمة معيّنة في الخانة المحدّدة في المصفوفة الثنائيّة:

**MyBA1.Set(3,True)**

وهي مكافئة لاستخدام خاصيّة Item:

**MyBA1.Item(3) = True**

**غيّر كلّ الخانات SetAll:**

تضع قيمة معيّنة في كلّ خانات المصفوفة الثنائيّة:

**MyBA1.SetAll(True)**

**سجلّ المتّجه الثنائي Structure** **BitVector32 وسجلّ مقطع المتّجه الثنائي Structure** **BitVector32.Section:**

هذان السجلان لا ينتميان للفضاء System.Collections، ولكنّهما ينتميان لفضاء آخر ضمن هذا الفضاء، هو System.Collections.Specialized، الذي يحتوى على بعض المجموعات الخاصّة الهامّة.

ويمكّنك سجلّ المتّجه الثنائي Structure BitVector32 من حفظ الخانات الثنائيّة، ولكن بحدّ أقصى 32 خانة فقط، وهو بذلك أكثر كفاءةً وسرعةً من خليّة المصفوفة الثنائيّة BitArray عند التعامل مع عدد خانات قليل (مثل خانات العدد الصحيح Integer).

ولتعريف متغيّر من هذا السجلّ، استخدم الصيغة التالية:

**Dim BV1 As New System.Collections.Specialized.BitVector32(عدد صحيح)**

هذه الصيغة تمكّنك من تحميل خانات عدد صحيح في سجلّ المتجّه الثنائي.

وهناك صيغة أخرى تمكّنك من تحميل الخانات من متّجه ثنائي آخر:

**Dim BV2 As New System.Collections.Specialized.BitVector32(BV2)**

وفي الغالب يتمّ استخدام هذا السجلّ مع سجلّ آخر، يمثّل مقطعا من المتّجه الثنائي.. هذا السجلّ هو BitVector32.Section، والذي تستطيع تعريفه كالتالي:

**Dim BVS As System.Collections.Specialized.BitVector32.Section**

ويمكن تحديد هذا المقطع باستخدام الوسيلة إنشاء مقطع CreateSection.. حيثُ تمكّنك أوّل صيغة لهذه الوسيلة، من إنشاء مقطع من الخانات، يبدأ من أوّل خانة في المتّجه الثنائيّ بالطول الذي يمكن أن يحتوي على أكبر عدد تريده، كالتالي:

**BVS = Bv1.CreateSection(أقصى عدد يمكن وضعه في المقطع)**

وكمثال:

**BVS1 = Bv1.CreateSection(7)**

وبما أنّ الرقم 7 يمكن احتواؤه في 3 خانات ثنائيّة (111)2، فإنّ طول المقطع BVS1 سيكون 3 خانات.. ولا تقبل هذه الصيغة إلا متغيّرا من النوع عدد قصير Short، ممّا يعني أنّ أطول مقطع تكوّنه لن يزيد عن 16 خانة.

أمّا الصيغة الثانية، فتمكّنك من تعريف مقطع تالٍ لمقطع يسبقه في المتّجه الثنائيّ، وذلك كالتالي:

**BVS2 = Bv1.CreateSection(3, BVS1)**

حيث في هذه الحالة، سيبدأ المقطع BVS2 من الخانة رقم 4، وسيكون طوله خانتين.

**أهمّ خصائص المتّجه الثنائي:**

**العنصر Item:**

يمكنك بهذه الخاصيّة أن تتعامل مع خانات المتّجه.. فمثلا لو أردت أن تجعل قيمة الخانة الثالثة في المتجه صفرا، فاستخدم الجملة التالية:

**BV1.Item(2) = False**

ونظرا لأنّ هذه الخاصيّة افتراضيّة، فإنّ بإمكانك أن تختصر الجملة السابقة كالتالي:

**BV1(2) = False**

وهناك صيغة أخرى من هذه الدالة، تمكّنك من تغيير قيمة مقطع من المتّجه دفعة واحدة، وذلك كالتالي:

**BV1.Item(BVS1) = 5**

أو باختصار:

**BV1(BVS) = 5**

حيث BVS هو متغيّر من النوع BitVector32.Section.

**البيانات Data:**

استخدم هذه الخاصيّة لتقرأ القيمة العشريّة المخزّنة في الخانات الثنائيّة، حيث تعيد لك هذه الخاصيّة العدد الصحيح الذي تمثّله خانات المتّجه الثنائيّ.

**واجهة العداد IEnumerator وواجهة المقارنة IComparer**

* + [**ما هي الواجهة Interface:**](file:///C:\Users\Administrator\Desktop\vb.net\c-%20Classes%20الخلايا\09-%20Collections%20خلايا%20المجموعات\2-%20واجهة%20العداد%20وواجهة%20المقارنة\1-%20ما%20هي%20الواجهة.htm)
  + [**تعديد المجموعات Enumerating Collections:**](file:///C:\Users\Administrator\Desktop\vb.net\c-%20Classes%20الخلايا\09-%20Collections%20خلايا%20المجموعات\2-%20واجهة%20العداد%20وواجهة%20المقارنة\2-%20تعديد%20المجموعات.htm)
  + [**الترتيب المخصّص Custom Sorting:**](file:///C:\Users\Administrator\Desktop\vb.net\c-%20Classes%20الخلايا\09-%20Collections%20خلايا%20المجموعات\2-%20واجهة%20العداد%20وواجهة%20المقارنة\3-%20الترتيب%20المخصّص.htm)

**ما هي الواجهة Interface:**

يمكن اعتبار الواجهة Interface نوعا من الكائنات، التي تتعامل مع الخليّة بطريقة خاصّة.. فمثلا واجهة العدّ IEnumerator، هي كائن يستقبل قائمةً من المؤشّرات Pointers لكلّ عناصر المجموعة Collection، بحيث يمكننا التعامل مع عناصر المجموعة من خلال هذه الواجهة.

وكلّ مجموعة من التي عرفناها لها عدّاد داخليّ ـ بما في ذلك المصفوفة ـ يمكن الحصول عليه باستخدام الوسيلة GetEnumerator.

كذلك فإنّ لدينا واجهة المقارنة، التي يمكن استخدامها مع كلّ المجموعات سالفة الذكر، لتحديد كيفيّة إجراء المقارنة بين عناصرها.. ويمكن استخدام واجهة المقارنة كذلك في دالة البحث الثنائي BinarySearch، وفي ترتيب رموز القائمة المرتّبة عندما تكون هذه الرموز كائنات.

ولكن ما هي الواجهة؟

الواجهة هي تقنيّة هامّة جدّا من تقنيات البرمجة بالكائنات Object-Oriented Programming، وهي تفيدك عندما تريد أن تعرف في خليّتك وسيلةً، ولكنّك غير واثق من كيفية كتابة كودها، نظرا لأنّه يختلف من برنامج لآخر.. في هذه الحالة تعرّف الدالة بدون كتابة كودها، وتترك للمبرمج الذي سيستخدم الخليّة أن يُعرّف الواجهة التي تكتب كود هذه الدالّة.

هذا هو ما سنفعله في حالة واجهة المقارنة، حيث إنّ VB لا يعرف كيف يقارن كائنين ببعضهما نظرا لأنّ الكائن يحتوى على العديد من الخصائص.. فمثلا: لا يمكن أن يعرف VB كيف يقارن لونين ببعضهما، لأنّ المقارنة يمكن أن تكون بدرجة تشبّع اللون أو إضاءته أو نسبة الأحمر فيه... إلخ.. لهذا فإنّ كل المجموعات تمنحك واجهة المقارنة، حيث يمكنك أن تعرّف دالة خاصّة بك، تقوم بعمليّة المقارنة بين كائنين من النوع الذي يعنيك بالطريقة التي تناسبك، حتّى يتمّ استخدامها في ترتيب عناصر المجموعة.

**تعديد المجموعات Enumerating Collections:**

لقد تعرّضنا لهذه الواجهة من قبل بالفعل، وكما رأينا فإنّ كل وظيفتها هي أن تسمح لنا بالمرور عبر عناصر المجموعة بدون أن نشغل ذهننا بعدد هذه العناصر.

ويمكن الحصول على عدّاد المجموعة باستخدام الوسيلة GetEnumerator، كالتالي:

**Dim ALEnum As IEnumerator**

**ALEnum = AList.GetEnumerator**

وللتحرّك عبر عناصر العدّاد، لدينا الوسيلة MoveNext للانتقال للعنصر التالي، وهي تُرجع True لو نجح هذا الانتقال، و False لو لم يكن هناك أيّ عنصر تالي..لدينا كذلك الوسيلة Reset للعودة إلى أوّل عنصر في العدّاد.

**ملحوظة:**

عند إنشاء العدّاد لا يكون هناك عنصر حاليّ، حيث يكون العداد متوقفا قبل العنصر الأوّل، لهذا يجب استدعاء الوسيلة MoveNext مرّة لكي يصبح العنصر الأوّل هو العنصر الحاليّ.

وللتعامل مع العنصر الحاليّ في العدّاد، استخدم الوسيلة Current، مع ملاحظة أنّها للقراءة فقط، ولا يمكن تغيير قيمتها.

انظر كيف نستخدم العدّاد في المرور على عناصر المصفوفة القائمة:

**Dim aList As New ArrayList()**

**Dim R1 As New Rectangle(1, 1, 10, 10)**

**aList.Add(R1)**

**R1 = New Rectangle(2, 2, 20, 20)**

**aList.Add(R1)**

**aList.add(New Rectangle(3, 3, 2, 2))**

**' العداد**

**Dim REnum As IEnumerator**

**REnum = aList.GetEnumerator**

**Dim R As New Rectangle()**

**While REnum.MoveNext**

**R = CType(REnum.Current, Rectangle)**

**MsgBox(R.Width \* R.Height)**

**End While**

ولدينا نوع آخر من واجهة العدّاد، هو "واجهة عدّ القاموس" IDictionaryEnumerator، وهي تستخدم مع الجدول المختلط HashTable، حيث تمنحك بعض الخصائص الإضافيّة، مثل "القيمة" Value و"الرمز" Key و"القيمة والرمز" Entery، وذلك حتّى يمكنك أن تتعامل مع القيمة الحاليّة بالصيغة DEnum.Value أو الصيغة DEnum.Entry.Value، ومع الرمز الحالي بالصيغة DEnum.Key أو الصيغة DEnum.Entry.Key.

**تحذير:**

لو تغيّرت قيمة أحد عناصر المجموعة أو المصفوفة التي يتعامل معها العدّاد أثناء استخدامه، فإنّ استثناءً من النوع InvalidOperationException ينطلق، وعليك أن تعالجه.. ولو كنت مضطرا لتعديل أحد العناصر أثناء استخدام العداد، فعليك أن تستخدم نسخة احتياطيّة من المجموعة وتجري التعديلات عليها، ثمّ تحفظ التعديلات إلى المجموعة الأصليّة فيما بعد.

ويمكنك أن تتدرّب على استخدام العدّدادات في مشروع Enumerations في مجلّد برامج هذا الفصل.

**الترتيب المخصّص Custom Sorting:**

افترض أن لدينا سجلا للشخص تعريفه كالتالي:

**Structure Person**

**Dim Name As String**

**Dim BDate As Date**

**Dim EMail As String**

**End Structure**

سنضيف الآن شخصين للمصفوفة القائمة كالتالي:

**Dim p As New Person**

**Dim aList As New ArrayList**

**p.Name = "Adams, George"**

**p.Bdate = #4/17/1957#**

**p.EMail = "gadams@example.com"**

**aList.Add(p)**

**p = Nothing**

**p.Name = "New Name"**

**p.BDate = #1/1/1950#**

**' حقل العنوان البريدي فارغ**

**aList.Add(p)**

والآن نريد ترتيب هذين الشخصين.. ولكن على أيّ أساس؟.. الاسم أم تاريخ الميلاد أم عنوان البريد الالكترونيّ؟

هذا إذن يستوجب كتابة دالّة تقوم بعمليّة المقارنة تبعا للأساس الذي نريده.. ولكنّنا لا نستطيع إرسال اسم هذه الدالة إلى وسيلة الترتيب.. فما هو الحل؟

في هذه الحالة سنقوم بإنشاء خليّة جديدة تكتب كود واجهة المقارنة، حيث ستبدو هذه الخليّة كالتالي:

**Class customComparer**

**Implements IComparer**

**Public Function Compare(ByVal obj1 As Object, ByVal obj2 As Object) As Integer Implements IComparer.Compare**

**' كود دالة المقارنة**

**End Function**

**End Class**

ويمكن أن يكون اسم الخليّة أيّ شيء، وطبعا يُفضّل أن يكون دالا على وظيفتها.. ولكن دائما يجب أن تكون الجملة "ينفّذ واجهة المقارنة" Implements IComparer هي أوّل سطر في الخليّة.

أمّا دالّة المقارنة، فيجب أن يكون اسمها دائما Compare، ويجب أن يلحق تعريفها المقطع Implements IComparer.Compare.

الخطوة التالية هي تعريف متغيّر من خليّة المقارنة هذه، وإرساله كمعامل لدالة الترتيب، لتقوم باستخدام دالة المقارنة من هذه الخليّة:

**Dim CompareThem As New customComparer**

**aList.Sort(CompareThem)**

أو باختصار:

**aList.Sort(New customComparer())**

ويمكن إرسال هذه الخليّة أيضا لدالة البحث الثنائيّ كالتالي:

**BinarySearch(عنصر, New customComparer())**

ويمكن استخدام هذه الخليّة في ترتيب رموز القائمة المرتّبة SortedList بإرسالها كقيمة مبدئيّة عند تعريف القائمة المرتّبة كالتالي:

**Dim sList As New SortedList(New CustomComparer())**

وهذه الخطوات ثابتة دائما، ما عدا اختلاف اسم الخليّة من مرّة لأخرى.

ولكن كيف نكتب دالة المقارنة؟

إنّ لك مطلق الحرّيّة في ذلك، ولكن يجب عليك فقط أن تجعل القيمة المعادة من الدالة -1 إذا كان الكائن الأوّل أصغر من الثاني، و 0 إذا كان الكائنان متساويين، و 1 إذا كان الكائن الأوّل أكبر من الثاني.

وفي المثال التالي سنكتب الخليّة التي تقارن سجلي شخصين تبعا لخانة العمر:

**Class PersonAgeComparer : Implements IComparer**

**Public Function Compare(ByVal Obj1 As Object, ByVal Obj2 As Object) As Integer Implements IComparer.Compare**

**Dim person1, person2 As Person**

**Try**

**person1 = CType(Obj1, Person)**

**person2 = CType(Obj2, Person)**

**Catch compareException As System.Exception**

**Throw (compareException)**

**Exit Function**

**End Try**

**If person1.BDate < person2.BDate Then**

**Return -1**

**ElseIf person1.BDate > person2.BDate Then**

**Return 1**

**Else ' العمران متساويان**

**Return 0**

**End If**

**End Function**

**End Class**

ولا يوجد ما يمنعك من كتابة أيّ عدد آخر من الخلايا، لترتيب سجلات الأشخاص باعتبارات أخرى، مثل الاسم أو الرقم أو العنوان... إلخ.

بعد ذلك يمكنك استخدام دالة الترتيب كالتالي:

**AList.Sort(New PersonAgeComparer())**

وللتدريب على هذه العمليّة، افحص المشروع CustomComparer، في مجلّد برامج هذا الفصل.

**ملحوظة1:**

كلّما قمت بكتابة خليّة تنفّذ واجهة المقارنة، قم باستبدال الوسيلة الافتراضيّة Equels لهذه الخليّة، لتعطي نفس نتيجة المقارنة.. راجع الفصل السابع، الجزء الخاصّ بإنشاء الخليّة الصغرى، المقطعين الخاصّين بالأعضاء الافتراضيّة وتخصيصها.

**ملحوظة2:**

تحتوي المجموعات على خليّة تسمّى "المقارنة بدون مراعاة حالة الأحرف" CaseInsensitiveComparer، هذه الخليّة تكتب كود واجهة المقارنة ICpmparer Interface، ويمكنك استخدامها إذا أردت إجراء بحث أو ترتيب نصّي لا يهتمّ بحالة الأحرف.. أمّا لو أردت مراعاة حالة الأحرف، فاستخدم الخليّة Comparer، وهو الوضع الافتراضيّ.

**إنشاء مجموعاتك الخاصّة**

* + [**تعريف مجموعة خاصّة:**](file:///C:\Users\Administrator\Desktop\vb.net\c-%20Classes%20الخلايا\09-%20Collections%20خلايا%20المجموعات\3-%20إنشاء%20المجموعات%20الخاصّة\0-%20تعريف%20مجموعة%20خاصّة.htm)
  + [**استخدام المجموعات Collections في الخلايا Classes:**](file:///C:\Users\Administrator\Desktop\vb.net\c-%20Classes%20الخلايا\09-%20Collections%20خلايا%20المجموعات\3-%20إنشاء%20المجموعات%20الخاصّة\1-%20استخدام%20المجموعات%20في%20الخلايا.htm)
  + [**إنشاء مجموعة ذات نوع محدّد:**](file:///C:\Users\Administrator\Desktop\vb.net\c-%20Classes%20الخلايا\09-%20Collections%20خلايا%20المجموعات\3-%20إنشاء%20المجموعات%20الخاصّة\2-%20إنشاء%20مجموعة%20ذات%20نوع%20محدّد.htm)
  + [**التراكيب Hierarchies:**](file:///C:\Users\Administrator\Desktop\vb.net\c-%20Classes%20الخلايا\09-%20Collections%20خلايا%20المجموعات\3-%20إنشاء%20المجموعات%20الخاصّة\3-%20إنشاء%20التراكيب.htm)

**تعريف مجموعة خاصّة:**

يمكنك تعريف مجموعةٍ خاصّةٍِ بك Custom Collection بجملة كالتالية:

**Public MyCollection As New Collection**

وبهذا أصبح للمتغيّر MyCollection الخصائص والوسائل التقليديّة المعروفة للمجموعات:

**إضافة Add ـ عدد العناصر Count ـ العنصر Item ـ حذف Remove.**

وبإمكانك أن تضع في هذه المجموعة أيّ نوع من الكائنات والبيانات، وذلك لأنّ عناصر المجموعة من النوع Object .. افترض أنّ لديك خليّة اسمها MyOwnClass.. يمكنك أن تعرّف كائنا منها وتضيفه لمجموعتك كالتالي:

**Dim C As New MyOwnClass()**

**MyCollection.Add(C)**

كما يمكنك أن تضيف نصّا لنفس المجموعة كالتالي:

**Dim S As String = "هذا نص للتجريب"**

**MyCollection.Add(S)**

ويجب أن ألفت انتباهك هنا إلى أنّ المجموعة الخاصّة تختلف عن باقي المجموعات التي تعاملنا معها من قبل، فهي على نقيضها، تبدأ ترقيم عناصرها بالرقم 1 وليس صفر.. إنّ هذا يرجع إلى أنّ هذه المجموعة خاصّة بـ VB ولا يمنحها لك إطار العمل.. معنى هذا أنّ الكائن C تمّ وضعه في الخانة رقم 1، والنصّ S تمّ وضعه في الخانة 2.. لهذا لو أردت أن تعرض النصّ في رسالة، فاستخدم الجملة التالية:

**MsgBox (CType(MyCollection.Item(2), String)**

وطبعا لو كانت جملة التحويل الدقيق مغلقة Option Strict Off، فإنّ بإمكانك كتابة الجملة السابقة كالتالي:

**MsgBox (MyCollection.Item(2))**

ولحذف أيّ عنصر من المجموعة استخدم جملة كالتالية:

**MyCollection.Remove(1)**

وللمرور عبر كلّ عناصر المجموعة، لديك الوسيلتان المعروفتان:

1- استخدام الجملة التكراريّة، اعتمادا على خاصيّة Count لمعرفة عدد عناصر المجموعة:

**Dim I As Integer**

**For I = 1 To MyCollection.Count**

**MsgBox(MyCollection.Item(I))**

**Next**

2- استخدام جملة For Each...Next:

**Dim Itm As Object**

**For Each Itm In MyCollection**

**MsgBox (Itm)**

**Next**

**استخدام المجموعات Collections في الخلايا Classes:**

يمكنك إضافة المجموعات للخلايا التي تنشئها، لتستخدمها في ترتيب الكائنات الموجودة فيها.. وأسهل طريقة لفعل ذلك، هي تعريف متغيّر عامّ Public من النوع مجموعة Collection.. افترض أنّ لديك خليّةً اسمها MyOwnClass.. استخدم الكود التالي لتعرّف داخلها المجموعة MyCollection:

**Public Class MyOwnClass**

**Public MyCollection As New Collection()**

**' باقي كود الخليّة**

**End Class**

الآن يمكنك أن تكتب كودا كالتالي:

**Dim C As New MyOwnClass**

**Dim S As String = "هذا النصّ للتجريب"**

**C.MyCollection.Add(S)**

إنّ هناك مشكلة في هذه الطريقة، وهي واضحة في المثال السابق: ليس باستطاعتك تحديد نوع العناصر التي تضيفها للمجموعة، فهي ستقبل كلّ الأنواع، وهو ليس مرغوبا فيه في البرمجة، لأنّه يفتح المجال لأخطاء كثيرة.. لهذا لا بدّ من البحث عن حلّ ما.

**إنشاء مجموعة ذات نوع محدّد:**

يمنحك الفضاء System.Collections.Specialized بعض المجموعات الخاصّة ذات الأنواع المحدّدة، مثل مجموعة النصوص StringCollection التي تستطيع تخزين النصوص فيها، ومثل المجموعة NameValueCollection التي تماثل مجموعة النصوص، إلا إنّها مرتّبة، ويمكن التعامل معها برقم العنصر في المجموعة، أو برمز نصّي String Key مناظر له (تشبه الجدول المختلط).. أنصحك بدراسة المجموعات الموجودة في هذا الفضاء، ممّا لم نتطرّق إليها في هذا الفصل.

ولكن رغمَ هذا، ما تزال تحتاج لإنشاء مجموعاتٍ خاصّة بك، تحدد لها النوع الذي تريده من البيانات، حتّى لو كان هذا النوع خليّة من تعريفك أنت.

إنّ ذلك هامّ جدّا في الحالات التي تحتاج فيها لتسريع البرنامج، وذلك بالتخلّص من عمليّات التحويل بين الأنواع، وعمليّات التعبئة Boxing والتفريغ UnBoxing التي يقوم بها VB عند التعامل مع كلّ عنصر في المجموعات الجاهزة.

كما أنّ إنشاء مجموعتك الخاصّة، يضمن لك التأكّد من صحّة البيانات المخزّنة فيها، وموافقتها لشروطك ونوع البيانات المطلوب.

ويمكنك أن تفعل ذلك بتعريف خليّة جديدة ترث الخليّة "أساس المجموعة" CollectionBase.. هذه الخليّة تحتوي على كود الخاصيّتين Count و Clear، وكود الوسيلتين RemoveAt و GetEnumerator فقط.. أمّا خاصيّة العنصر Item، والوسيلتان Add و Remove، فهي متروكة لك لتكتبها، وذلك حتّى تحدد أنت نوع العناصر التي تريد التعامل معها في المجموعة.

وتحتوي خليّة أساس المجموعة CollectioBase على خاصيّةٍ تسمّى List، هي التي يمكنك استخدامها لتخزين العناصر في المجموعة.. وهذه الخاصيّة محميّة Protected، حتّى لا يمكن استخدامها من خارج الخليّة.

والآن اتبع هذه الخطوات لتعريف مجموعة خاصّة بك، لا تقبل إلا نوعا واحدا من العناصر.

في البداية سنبدأ بتعريف خليّة اسمها MyOwnClass، كالتالي:

**Public Class MyOwnClass**

**Public Name As String**

**End Class**

وهي كما ترى خليّة بسيطة، لا تحتوي إلا على متغيّر عامّ واحد.

والآن سنحاول إنشاء مجموعة لا تقبل إلا العناصر من النوع MyOwnClass.. ابدأ بتعريف خليّة هذه المجموعة كالتالي:

**Public Class MyOwnClassCollection**

**Inherits System.Collections.CollectionBase**

**End Class**

وكما ذكرنا، فإنّ مجموعتك الآن صارت تمتلك خاصيّة العدد Count، والوسيلة Clear.. ما نريده الآن هو أن نعرّف وسيلة لإضافة العناصر Add ووسيلة لحذفها Remove ووسيلة للتعامل معها Item، بحيث لا تقبل هذه الوسائل إلا عناصر من النوع MyOwnClass.. تعال نبدأ بوسيلة الإضافة.. اكتب هذا الكود في الخليّة MyOwnClassCollection:

**Public Sub Add(ByVal aMyOwnClass As MyOwnClass)**

**List.Add(aMyOwnClass)**

**End Sub**

منتهى البساطة.. استخدمنا القائمة List التي ورثناها من خليّة أساس المجموعة CollectionBase لنحفظ فيها العنصر.

في الحقيقة لا تتركّز الأهمّيّة في هذا السطر البسيط، وإنّما الأهميّة كلّها تنحصر في نوع معامل هذه الوسيلة، الذي جعلناه من النوع MyOwnClass، بحيث نجبر من يستخدم مجموعتنا على إضافة هذا النوع فقط.. أمّا المتغيّر List نفسه، فهو يقبل أيّ نوع من الكائنات.

أمّا وسيلة حذف عنصر فهي كالتالي:

**Public Sub Remove(ByVal Item As MyOwnClass)**

**List.Remove (Item)**

**End Sub**

لم يبق إلا أن نكتب خاصيّة العنصر Item، لتقرأ أو تعيد كائنا من النوع MyOwnClass.. ولا بدّ هنا من أن نتحقّق أولا من أنّ رقم العنصر مقبول:

**Public Property Item(ByVal index As Integer) As MyOwnClass**

**Get**

**If index > Count - 1 Or index < 0 Then**

**' تعريف استثناء**

**Dim Exc As New Exception("لا يوجد عنصر بهذا الرقم.")**

**' إطلاق الاستثناء**

**Throw Exc**

**Else**

**Return CType(List.Item(index), MyOwnClass)**

**End If**

**End Get**

**Set (ByVal Value As MyOwnClass)**

**If index > Count - 1 Or index < 0 Then**

**Dim Exc As New Exception("لا يوجد عنصر بهذا الرقم.")**

**Throw Exc**

**Else**

**List.Item(index) = Value**

**End If**

**End Set**

**End Property**

وطبعا يمكنك إضافة المزيد من الخصائص والوسائل الجديدة لهذه المجموعة، على حسب ما يناسبك، مثل وسيلتي الترتيب والبحث وخلافهما.

الآن انتهينا من كتابة مجموعتنا الخاصّة.. إنّ نفس هذه الخطوات ستتكرّر في كلّ مرّة مع أيّ مجموعة جديدة.. ما سيختلف فقط هو نوع عناصر المجموعة.. بمعنى أنّ كلّ ما عليك هو تغيير النوع MyOwnClass في كلّ موضع تمّ استخدامه فيه، إلى النوع الجديد الذي تريده.

تعال الآن نختبر مجموعتنا الجديدة.. ابدأ بتعريف متغيّر من هذه المجموعة كالتالي:

**Dim Coll As New MyOwnClassCollection()**

الآن يمكنك أن تضيف لهذه المجموعة أيّ عدد من العناصر من النوع MyOwnClass:

**Dim Class1 As New MyOwnClass()**

**Class1.Name = "خلية 1"**

**Coll.Add(Class1)**

ويمكنك عرض نصّ أوّل عنصر في المجموعة كالتالي:

**MsgBox(Coll.Item(0))**

طبعا لاحظت أنّ العنصر الأوّل في هذه المجموعة رقمه صفر.. هذه هي القاعدة العامّة في أيّ مجموعة، ما عدا تلك التي تنتمي للنوع Collection، الذي رأيناه من قبل.

ولمزيد من التدريب، افحص مشروع MyCollection في مجلّد برامج هذا الفصل.

وأودّ أن ألفت نظرك إلى أنّك أحيانا ما تحتاج لمجموعة يمكن قراءة عناصرها دون القدرة على تغييرها (مجموعة للقراءة فقط).. في هذه الحالة استخدم الخليّة ReadOnlyCollectionBase، بدلا من الخليّة CollectionBase، ولا تضف لمجموعتك وسيلتي الحذف أو الإضافة.. عامّة حتّى لو حاولت فلن تستطيع، فالقائمة التي تضع فيها العناصر في الخليّة ReadOnlyCollectionBase هي للقراءة فقط (وعلى فكرة، اسمها InnerList).

**التراكيب Hierarchies**

* + - [**إنشاء التراكيب Hierarchies:**](file:///C:\Users\Administrator\Desktop\vb.net\c-%20Classes%20الخلايا\09-%20Collections%20خلايا%20المجموعات\3-%20إنشاء%20التراكيب\1-%20إنشاء%20التراكيب.htm)
    - [**نطاقات الخلايا المتداخلة Domains Of Nested Classes:**](file:///C:\Users\Administrator\Desktop\vb.net\c-%20Classes%20الخلايا\09-%20Collections%20خلايا%20المجموعات\3-%20إنشاء%20التراكيب\2-%20نطاقات%20الخلايا%20المتداخلة.htm)

**إنشاء التراكيب Hierarchies:**

يمكن أن تضع تعريف خليّة المجموعة Collection Class وأيّ متغيّرات يتمّ تعريفها من هذه المجموعة داخل أيّ خليّة تُعرّفها أنت.. إنّ هذا يفيدك في تعريف تراكيب متداخلة من الخلايا Hierarchies.

فمثلا يمكن أن تُعرّف خليّة اسمها Class1 وبداخلها تعرّف الخليّة Class2 وكذلك مجموعة لعناصر هذه الخليّة Class2Collection.. هذا بالإضافة إلى إمكانيّة تعريف متغيّر من النوع Class2Collection:

**Public Class Class1**

**' خليّة داخل الخليّة**

**Public Class Class2**

**Public ID As Integer**

**End Class**

**' مجموعة داخل الخليّة**

**Private Class Class2Collection**

**Inherits CollectionBase**

**' كود المجموعة**

**End Class**

**' متغيّر من نوع المجموعة**

**Public Class2s As New Class2Collection()**

**End Class**

ولاستخدام هذه الخليّة انظر للجمل التالية:

**Dim X As New Class1**

**Dim Y As New Class1.Class2()**

**X.Class2s.Add(Y)**

أو باختصار:

**Dim X As New Class1**

**X.Class2s.Add(New Class1.Class2())**

ويمكن استخدام النسخة المعرّفة من Class2 كالتالي:

**Y.ID = 1**

أو بطريقة أخرى:

**X.Class2s.Item(0).ID = 1**

ولو أردت، يمكنك أن تمنع المستخدم من تعريف نسخ من الخليّة Class2، وذلك بتعريف حدث الإنشاء New بكلمة Friend بدلا من Public.. وبهذا يمكن للمستخدم أن ينشئ متغيّرات من هذه الخليّة، ولكنّه لا يستطيع استخدام كلمة New لربط هذه المتغيّرات بنسخة جديدة New Instance من الخليّة.. (إنّ هذا التأثير مماثل لوضع خاصيّة Instancing بالقيمة PublicNotCreatable في VB6).

ولكن.. لو فعلنا ذلك، فكيف يمكن تعريف نسخة جديدة من هذه الخليّة؟

في هذه الحالة يجب تعديل وسيلة الإضافة Add للمجموعة Class2Collection، بحيث تقوم هي بتعريف نسخة جديدة من المتغيّر Class2، كالتالي:

**Public Sub Add()**

**List.Add(New Class2())**

**End Sub**

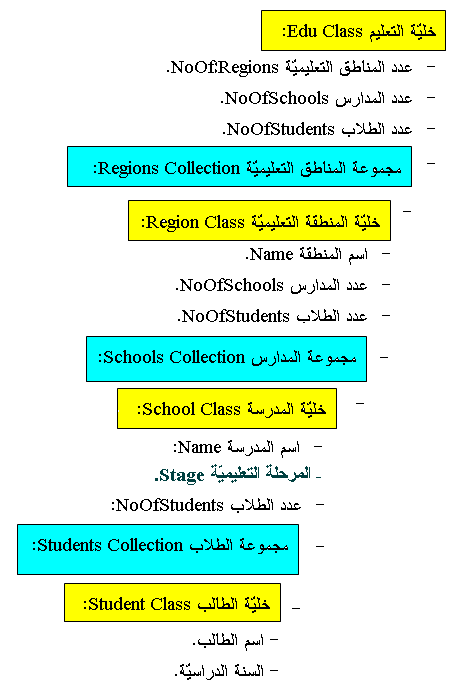
وبذلك لن يستطيع المبرمج التعامل مع الخليّة Class2 إلا من خلال المجموعة Class2s.. لا تنسَ كذلك أن تمنع المستخدم من تعريف نسخ من المجموعة Class2Collection، عن طريق جعل حدث الإنشاء New لها هي أيضا Friend.

ولا يتوقّف الأمر عند تعريف خليّتين متداخلتين، فبالإمكان تعريف خليّة ثالثة Class3 داخل الخليّة Class2 مع مجموعة لها، وخليّة رابعة داخل الخليّة الثالثة، وهكذا... .

أعرف أنّك ستساءل في لماضة: ولماذا كلّ هذا التعقيد؟

إنّ هذا يفيدك في التعامل مع التراكيب المعقّدة.. تعال نرى المثال التالي وسيتّضح لك الأمر:

نريد أن ننشئ خليّة للتعليم المصريّ تخطيطها كالتالي:



أعتقد أنّك تستطيع كتابة الكود الذي يفعل ذلك.. اعتبره تدريبا، واسترشد بما يلي:

- اجعل كلّ الخصائص التي تحسب عدد المناطق وعدد المدارس وعدد الطلبة للقراءة فقط ReadOnly، فهذه الأعداد هي أعداد العناصر الموجودة في المجموعات Collections، ولا معنى لأن يتمّ تغييرها.. إنّها تتغيّر فقط عن طريق إضافة العناصر للمجموعة وحذفها منها.. أمّا خصائص اسم المنطقة واسم المدرسة والمرحلة التعليميّة واسم الطالب، فيحقّ للمبرمج تغييرها كيفما شاء.

- اجعل كذلك كلّ المتغيّرات المُعرّفة من المجموعات Collections، للقراءة فقط ReadOnly، كالتالي:

**Public ReadOnly Regions As New RegionCollection**

إنّ هذا لا يعني عدم القدرة على إضافة وحذف العناصر منها، ولكنّه يعني أنّ المبرمج لا يستطيع جعل قيمة متغيّر المجموعة بـ Nothing مثلا، أو أيّ قيمة أخرى غير صالحة.. مثلا الجملة التالية غير صالحة:

**Dim X As New Edu**

**X.Regions = Nothing ' لا تسمح للمبرمج بهذه الجملة**

بينما الجملة التالية متاحة:

**X.Regions.Add("دمياط")**

- رغم أنّنا وضعنا الخلايا الفرعيّة تحت المجموعات Collections في المخطّط، إلا إنّ عليك أن تعرّفها في نفس مستوى المجموعة وليس داخل المجموعة.. إنّ ذلك سيختصر كتابة الكود على المبرمج عند تعريف متغيّرات من هذه الخلايا.

- اجعل لوسيلة الإضافة في كلّ مجموعة معاملات مناسبة، بحيث تسمح للمبرمج بإرسال المعلومات الأساسيّة للخليّة (مثل اسم المنطقة أو المدرسة أو الطالب) مباشرةً.. وكذلك اجعل وسيلة الإضافة تعيد النسخة المضافة من الخليّة، حتّى يستقبلها المبرمج في متغيّر لو أراد، حتّى يستخدمه في الكود مباشرةً، وبذلك يتمّ اختصار الجمل الطويلة .. تعال نرى مثالا للوسيلة Add في ثوبها الجديد.. هذا هو نصّها في مجموعة الطلاب:

**Public Function Add(Name As String, Year As String) As Student**

**Dim S As New Student()**

**S.Name = Name**

**S.Year = Year**

**List.Add (S)**

**Return S**

**End Function**

ولكي تدرك أهمّية هذا التطوير، جرّب الطريقة القديمة، وانظر لمدى المعاناة التي ستعانيها لإضافة منطقة تعليميّة ومدرسة وطالبين للتعليم المصريّ!!:

**Dim X As New Edu()**

**X.Regions.Add("دمياط")**

**X.Regions.Item(0).Schools.Add(1, "محمد عبده")**

**X.Regions.Item(0).Schools.Item(0).Students.Add("عثمان", 2)**

**X.Regions.Item(0).Schools.Item(0).Students.Add("أحمد", 3)**

**Dim Y As Integer**

**Y = X.Regions.Item(0).Schools.Item(0).Students.Item(0).Year \_**

**+ X.Regions.Item(0).Schools.Item(0).Students.Item(1).Year**

إنّ بالإمكان تخفيفَ كلّ هذه المعاناة، وذلك لو استخدمت نسخة الخليّة التي تعيدها الوسيلة Add كالتالي:

**Dim X As New Edu()**

**Dim R As Edu.Region = X.Regions.Add("دمياط")**

**Dim Sch As Edu.Region.School = R.Schools.Add(1, "محمد عبده")**

**Dim S1 , S2 As Edu.Region.School.Student**

**S1 = Sch.Students.Add("عثمان", 2)**

**S2 = Sch.Students.Add("أحمد", 3)**

**Dim Y As Integer = S1.Year + S2.Year**

أعتقد أنّ الفارق في حجم الجمل واضح.. وطبعا لا داعيَ للّماضة والتساؤل: لماذا نجمع السنة الدراسيّة لطالبين؟.. إنّ ذلك مجرّد مثال للإيضاح فقط يا لمض، ولا يعني أيّ شيء.

بالمناسبة: يمكنك استخدام مقطع With… End With لمزيد من اختصار الكود.. هذا بالإضافة لقدرتك على استخدام جملة الاستيراد Imports.. اكتب ما يلي في أقصى بداية ملفّ النموذج:

**Imports Education.Edu**

**Imports Education.Edu.Region**

**Imports Education.Edu.Region.School**

الآن يمكنك كتابة المثال السابق كالتالي:

**Dim X As New Edu()**

**Dim R As Region = X.Regions.Add("دمياط")**

**Dim Sch As School = R.Schools.Add(1, "محمد عبده")**

**Dim S1 , S2 As Student**

**S1 = Sch.Students.Add("عثمان", 2)**

**S2 = Sch.Students.Add("أحمد", 3)**

**Dim Y As Integer = S1.Year + S2.Year**

منتهى الاختصار.. أليس كذلك؟!

- يمكنك أن تجعل لحدث الإنشاء New بعض المعاملات المناسبة.. فمثلا يمكنك أن تجعل لحدث إنشاء خليّة الطالب معاملين هما اسم الطالب والفصل كالتالي:

**Friend Sub New(Name As String, Year As String)**

**MyBase.New**

**Me.Name = Name**

**Me.Class = Year**

**End Sub**

وبذلك يمكن إعادة كتابة وسيلة الإضافة في مجموعة الطلاب كالتالي:

**Public Function Add(Name As String, Year As String) As Student**

**Dim S As New Student(Name, Year)**

**List.Add (S)**

**Return S**

**End Function**

وأنت بالخيار.

- عرّف لكلّ المجموعات والخلايا (ما عدا الخليّة الرئيسيّة Edu بالطبع) حدث الإنشاء New بكلمة Friend كالتالي:

**Friend Sub New()**

**MyBase.New**

**End Sub**

مع مراعاة أنّ بعض أحداث الإنشاء قد يكون لها معاملات وبعض الكود الإضافيّ.

إنّ ذلك سيسمح للمبرمج بتعريف متغيّرات من هذه الأنواع، ولكن دون القدرة على ربطها بنسخ جديدة New Instances.. وبذلك يستطيع المبرمج كتابة جمل كالتالية:

**Dim Y As New Edu**

**Dim X As Edu.Region**

**X = Y.Regions.Add ("دمياط")**

**MsgBox (X.Name)**

ولكنّه لا يستطيع كتابة جملة كالتالية:

**Dim X As New Edu.Region**

فهو مجبر على استخدام الخلايا من داخل خليّة التعليم Edu فقط، ولا يستطيع تعريف نسخ مستقلّة منها.. طبعا بإمكانك أن تسمح له بذلك، لو كان هذا الأمر سيفيدك.

لاحظ أنّ كلمة Friend تجعل الإجراء متاحا للمشروع كلّه.. لهذا لو أضفت نموذج اختبار وكتبت به بعض الكود، فلن يعمل الأمر كما أوضحناه هنا، وسيكون بإمكانك تعريف نسخ من الخلايا والمجموعات.. ولكن لو أردت أن تجرّب الأمر بصورة مثاليّة، فأنشئ مكتبة خلايا Class Library، وقم ببناء ملف DLL لها، ثمّ اربطه بمشروع ويندوز وجرّب الخليّة من خلاله (راجع الفصل السابع).

على كلٍّ ستجد هذا الكود في تطبيق Education في مجلّد برامج هذا الفصل.

**نطاقات الخلايا المتداخلة Domains Of Nested Classes:**

لماذا اضطررنا لجعل وسيلة الإضافة Add تعيد نسخةً من خليّة الطالب Student التي تمّت إضافتها؟.. لماذا لم نتعامل مع هذه النسخة مباشرةً من خليّة المدرسة School؟.. أليست تحتوي على خليّة الطالب؟

هنا يجب أن تعي جيّدا القاعدة التالية: رغمّ أنّك قد عرّفت الخلايا متداخلة Nested، إلا إنّك لا تستطيع أن تتعامل مع أيّ متغيّر من خارج خليّته، حتّى ولو كان عامّا Public!!.. فمثلا، لا يمكنك استخدام المتغيّر Regions من الخليّة Region رغم أنّه موجود في الخليّة Edu التي تحتوي الخليّة Region، ورغم أنّه كذلك متغيّر عامّ Public؟

أعرف أنّك مندهش، فهذا يخالف قواعد نطاق المتغيّر Variable Domain التي تعرّفنا عليها في الفصل الثالث، والتي تنصّ على أنّ النطاق الداخليّ يستطيع التعامل مع متغيّرات النطاق الذي يحتويه، حتّى لو كانت هذه المتغيّرات خاصّة Private.. فمثلا، يمكن استخدام المتغيّرات المعرّفة على مستوى النموذج من كلّ الإجراءات المعرّفة داخل هذا النموذج، واستخدام المتغيّرات المعرّفة على مستوى الإجراء من كلّ مقاطع If.. End If و For.. Next و Do.. Loop وغيرها، الموجودة داخل هذا الإجراء.

إذن فلماذا لا يعمل الأمر هنا هكذا؟

ببساطة لأنّ الخلايا معزولة Encapsulated، وقد أفضنا في شرح ذلك وأوضحنا أهمّيّته من قبل.. إنّ هذا العزل Encapsulation يقتضي ما يلي:

1- لا يمكن التعامل مع المتغيّرات الخاصّة Private المعرّفة في أيّ خليّة، من أيّ خليّة أخرى، حتّى ولو كانت الأولى تحتوي على الثانية.. بل وحتى لو ورثت إحداهما الأخرى.

2- بل حتّى لا يمكن التعامل مع المتغيّرات العامّة Public المعرّفة في أيّ خليّة، من أيّ خليّة أخرى، إلا من خلال إنشاء نسخة جديدة من الخليّة New Instance، ووضعها في متغيّر من هذه الخلية.. وأنت بالطبع تذكر أنّنا لا نستطيع التعامل مع أدوات النموذج ومتغيّراته العامّة من خلال اسمه مباشرة.. فالجملة التالية غير مقبولة:

**Form1.TextBox1.Text = ""**

ولكن يمكن استخدام الكود التالي كبديل:

**Dim X As New Form1**

**X.TextBox1.Text = ""**

ولا يستثنى من ذلك إلا الوراثة بالطبع، وحالة أخرى، هي حالة المتغيّرات المشتركة Shared Variables.. وأنت بالطبع تعرف أنّ المتغيّر المشترك يمكن استخدامه عبر اسم الخليّة الأساسيّ مباشرةً، أو من أيّ نسخة تمّ تعريفها من هذه الخليّة.

باختصار: إنّ تعريف الخلايا في تراكيب متداخلة لا يخرق قواعد العزل Encapsulation، ولا يجعل هذه الخلايا خاضعة لقواعد النطاقات المتداخلة Nested Domains.

هنا سيتكوّر سؤالٌ ملحٌ ويهدر متدحرجا على منحدرات عقلك:

- إذن كيف يمكن التعامل مع المتغيّرات عبر الخلايا المتداخلة؟

- هناك طريقة واحدة لذلك.. هي أن تمرّر قيم هذه المتغيّرات من خليّة لأخرى، وذلك بإرسال نسخة من الخليّة الخارجيّة (أو متغيراتها)، إلى الخليّة الداخليّة، عند تعريف نسخة جديدة من هذه الأخيرة.

- هلا كنت أكثر إيضاحا؟

- خذها بمثال: نريد أن نكتب بعض الكود في خاصيّة العام الدراسيّ Year في خليّة الطالب.. أنت طبعا تعرف أنّ رقم العام الدراسيّ ينحصر بين 1 و 6.. سهل إذن أن تكتب شرطا يتحقّق من ذلك.. ولكن ما زالت هناك مشكلة: لا يمكن أن يكون هناك طالب بمدرسة إعداديّة أو ثانويّة في العام الدراسيّ الرابع!!.. معنى هذا أنّنا نحتاج للتأكّد من نوع المرحلة التعليميّة Stage لخليّة المدرسة School التي ينتمي إليها الطالب.. وطبعا للأسف، لن يمكن استخدام المتغيّر Stage من داخل خليّة الطالب مباشرةً.. هنا لن يكون أمامنا إلا حلّ وحيد: تعريف معامل لحدث إنشاء خليّة الطالب New()، نمرّر من خلاله قيمة المتغيّر Stage كالتالي:

**Friend Sub New(Stage As Stages, Name As String, Year As String)**

**MyBase.New**

**MyStage = Stage**

**Me.Name = Name**

**Me.Class = Year**

**End Sub**

- مهلا.. ماذا تعني بالمتغيّر MyStage هذا؟

- لو تريّثت قليلا لعرفت.. إنّه متغيّر خاصّ Private، نستخدمه لتخزين قيمة المرحلة التعليميّة.. ويتمّ تعريف هذا المتغيّر على مستوى خليّة الطالب كالتالي:

**Dim MyStage As Stages**

وبهذا يمكنك التحقّق من القيمة المرسلة لخاصيّة العام الدراسيّ، باستخدام المتغيّر MyStage.

- جميل.. ولكنّ هناك شيئا ما زلت لم أفهمه.

- هلمّ به.

- ما زلت أرى أنّ المشكلة لم تُحلَّ بعد.. إنّنا نعرّف نسخةً جديدةً من خليّة الطالب في الوسيلة Add الموجودة في خليّة مجموعة الطلاب StudentCollection.. أليست هي نفسها معزولةً عن خليّة المدرسة School؟!!

- معك حقّ.. هذه بديهةٌ حاضرةٌ منك وتركيزٌ حميد لك.

- لا أريدُ مدحا.. أريد حلا!!

- بسيطة.. اتبع الطريقة نفسها التي استخدمناها في خليّة الطالب، مع خليّة مجموعة الطلاب StudentCollection!

- تعني أن أعرّف معاملا لحدث الإنشاء لها، و.....

- بالضبط.. يمكنك أن تعرف الحدث New في خليّة مجموعة الطلاب كالتالي:

**Friend Sub New(Stage As Stages)**

**MyBase.New**

**MyStage = Stage**

**End Sub**

طبعا مع تعريف المتغيّر الخاصّ MyStage على مستوى الخليّة.. بعد ذلك يمكنك تعديل وسيلة إضافة الطالب كالتالي:

**Public Function Add(Name As String, Year As String) As Student**

**Dim S As New Student(MyStage, Name, Year)**

**List.Add (S)**

**Return S**

**End Function**

الآن يجب أن تعدّل تعريف المتغيّر Students الموجود في خليّة المدرسة School ليصير كالتالي:

**Dim Students As New StudentCollection(Me.Stage)**

ويمكنك أن تفحص هذا الكود في تطبيق Education في مجلّد برامج هذا الفصل.

- هووه.. يا لها من لفّة!!

- ماذا بيدنا لنفعل؟!.. للأسف لا يوجد حلّ آخر!

- ولكن لماذا!.. لقد بدأت أعتاد أن تَفهمني اللغة في سلاسة وبساطة.. فلماذا هذا التعقيد في هذا الموضوع بالذات؟

- لا يوجد أيّ تعقيد.. ولو فكّرت قليلا في الأمر لوجدته منطقيّا.. فبرغم أنّنا نمنع المبرمج من تعريف نسخ من خليّة الطالب إلا باستخدام الوسيلة Add الموجودة في المتغيّر Students (وهو لن يكون متاحا إلا من خلال نسخة من خليّة المدرسة) إلا إنّ هذه ليست قاعدة.. ولقد ذكرنا أنّ تعريف حدث الإنشاء New بكلمة Friend يمنعك من تعريف متغيّرات من هذه الخليّة من خارج المشروع، ولكنّه لا يمنعك من تعريف متغيّرات منها من أيّ نموذج ينتمي لنفس المشروع.. معنى هذا أنّك تستطيع كتابة جملة كالتالية (باستخدام الصيغة القديمة لحدث التعريف، قبل تعديلها):

**Dim S As New Student("حسام", 3)**

فهل تستطيع أن تخبرني الآن لأيّ مدرسة ينتمي هذا الطالب؟.. إنّ هذه نسخة جديدة مستقلة تماما من خليّة الطالب، ولا تنتمي لأيّ خليّة مدرسة على الإطلاق.. لهذا كان عزل الخلايا منطقيّا، حتّى ولو كانت متداخلة Nested.. إنّ هذا يضمن لكلّ نسخة يتمّ تعريفها من الخليّة، استقلالها التامّ.. و... أنت بالخيار!.. أو الاختيار، حتّى لا تسيء فهمي!

- بقيَ شيءٌ آخر: كيف نمرّر المتغيّرات من الخليّة الداخليّة للخليّة التي تحتويها؟

- بسيطة.. استخدم القيم المعادة Return Values.. إنّ هذا هو ما فعلناه في الوسيلة Add الخاصّة بمجموعة الطلاب StudentCollection، حيث تعيد لنا النسخة الجديدة التي تمّت إضافتها من خليّة الطالب.. ولا تنسَ أنّنا أنشأنا المجموعات Collections أساسا لحلّ هذه المشكلة، ففي كلّ خليّة توجد مجموعة تحتوي على كلّ النسخ التي تمّ تعريفها من الخليّة الداخليّة، مثل المجموعات Regions و Schools و Students.. عرفت الآن أهمّية تعريف المجموعات داخل التراكيب Hierarchies؟