برمجة اطار عمل NET. باستخدام

Visual Basic .NET

-- تركي العسيري

قرص شمج مرافق

شبكة المطورون العرب

Arabian Developer Network

yallayon

بسم الله الرحمن الرحيم

((سُبْحَانَكَ لَا عِلْمَ لَنَا إِلَّا مَا
عَلَّمْتَنَا إِنَّكَ أَنْتَ الْعَلِيمُ الْحَكِيمُ))

PDF Factory Pro trial version www.fastio.com

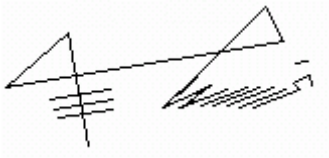
إهداء

الجمال في الحياة شيء يجبر الفؤاد
على ذكره في كل لحظة من لحظات خفقانه ...

وان لم تكوني من الجمال في الحياة،
فحسبي أن الجمال قد نبض إلى الحياة منك ...

أمي الحبيبة،
أهديك هذا الكتاب ...

ابنك المخلص



PDF Factory Pro trial version www.fastio.com

برمجة إطار عمل .NET
باستخدام Visual Basic .NET

الطبعة الاولى 2003

- حقوق كتاب **برمجة إطار عمل .NET. باستخدام Visual Basic .NET** محفوظة للمؤلف، ولا يحق لأي شخص او جهة رسمية من إعادة نشر هذا الكتاب او جزء منه بأي وسيلة دون الإذن الخطي من المؤلف. **وإلا فإنه سيكون معرضا للمطالبة بالتعويض وتطبيق أقصى العقوبة عليه**.

- اسماء البرامج او التقنيات او الشركات (كـ Visual Basic .NET، ADO .NET، Microsoft ....الخ) هي علامات تجارية مسجلة لأصحابها، والمؤلف يحترم هذه العلامات ويقر بها لماليكها سواء كانوا افراد او شركات او اي جهات تنظيمية أخرى، ولم يتم ذكرها للاختصار.

- اسماء الاشخاص او الشركات والمذكورة في أمثلة هذا الكتاب هي اسماء وهمية ولا يقصد بها تحديد هوية أشخاص او جهات معينة.

- تم اختبار المادة العلمية في هذا الكتاب والتحقق منها ومراجعتها، الا ان المؤلف غير مسئول بأي شكل من الأشكال عن الأضرار الناتجة من تطبيق المعلومات في هذا الكتاب.

- جميع الاراء الموجودة في هذا الكتاب تعبر عن رأي المؤلف الشخصي حتى لو لم توثق بأمثلة او أدلة حسية.

- فسح وزارة الاعلام رقم: 3876 -

PDF

المحتويات

تمهيد ..... ا
تقديم ..... ا
شكر وتقدير ..... ج
المقدمة ..... ه
لمن هذا الكتاب؟ ..... و
ماذا عن مبرمجي Visual Basic 1 **è** 6؟ ..... و
المصطلحات تعريب ام تعذيب؟ ..... ز
ماذا يقدم لك هذا الكتاب؟ ..... ح
القرص المدمج CD-ROM ..... ط
صفحة الكتاب على الانترنت ..... ي
الأخطاء (هام جدا) ..... ي
كلمة أخيرة ..... ي

الجزء الأول: **الأساسيات** 1
الفصل الأول: **تعرف على Visual Basic .NET** **3**
الحياة قبل .NET 3
البرمجة تحت DOS 4
الانتقال الى Windows 4
الحلم أصبح حقيقة مع COM 7
تحديات الانترنت 8
عشرات التقنيات لأداء الوظائف 10
الحياة بعد .NET 10
الاستقلالية عن منصات العمل 11
.NET نسخة محسنة من COM 11
تكامل لغات البرمجة 13
خدمات ويب هي مستقبل الانترنت 13
ماذا عن المبرمج العربي؟ 14

محتويات إطار العمل .NET Framework 15
الترجمة على الفور JIT 16
المجمعات Assemblies 17
بيئة التطوير Visual Studio .NET 17
نوافذ بيئة التطوير 18
القائمة الرئيسة 24
أشرطة الأدوات 26
كتابة برنامجك الأول 26
الحلول والمشاريع 26
أنواع المشاريع 28
بناء برنامجك الأول 30
استخدام ArabicConsole 31
الترجمة والتوزيع 32
الفصل الثاني: لغة البرمجة 35
الوحدات البرمجية Modules 35
الإجراء Sub Main() 38
الإجراء Sub New() 39
المتغيرات والثوابت 41
التصريح عن المتغيرات 41
قابلية الرؤية وعمر المتغيرات 43
أنواع البيانات 48
إسناد القيم 52
الثوابت 58
التركيبات والمصفوفات 59
التركيبات من نوع Enum 59
التركيبات من نوع Structure 62
المصفوفات 67
الإجراءات والدوال 69
الإرسال بالمرجع او القيمة 70

تخصيص الوسيطات المرسلة 72
تجاوز الحدود مع Windows API 73
التفرع والتكرار 74
التفرع باستخدام If ... Then 74
التفرع باستخدام Select Case 77
الحلقات التكرارية 79
مجالات الاسماء 81
تعريف مجال اسماء 82
الوصول الى عناصر مجال الاسماء 84
استيراد مجال اسماء باستخدام Imports 85
استيراد مجال اسماء دون استخدام Imports 87
الفصل الثالث: الفئات والكائنات 89
مدخلك السريع للفئات 89
بناء اعضاء الفئات 94
الحقول Fields 94
الطرق Methods 96
الخصائص Properties 105
الأحداث Event 110
استخدام الكائنات 116
ما هي حقيقة الكائن؟ 116
عبارات خاصة بالكائنات 119
اسناد القيم 123
حياة وموت الكائنات 126
ارسال الكائن بالمرجع او القيمة 137
الأعضاء المشتركة 138
الحقول المشتركة Shared Fields 138
الطرق المشتركة Shared Methods 140
الخصائص المشتركة Shared Properties 141
الاحداث المشتركة Shared Events 142

PDF

الفصل الرابع: الوراثة 147
مقدمة الى الوراثة 147
مبدأ الوراثة 147
تطبيق الوراثة بـــ Visual Basic .NET 149
التعامل مع الفئات الوارثة والمورثة 152
وراثة الأعضاء 152
المشيدات Constructors 154
التعامل مع الكائنات 157
إعادة القيادة Overriding 159
اعادة قيادة الطرق والخصائص 161
استخدام MyBase 166
استخدام MyClass 168
التظليل Shadowing 169
الاعضاء المشتركة Shared Members 173
كلمات محجوزة إضافية 174
الكلمة المحجوزة NotInheritable 174
الكلمة المحجوزة MustInherit 174
الكلمة المحجوزة MustOverride 176
محددات الوصول 177
قابلية الرؤية للفئات 177
قابلية الرؤية لاعضاء الفئات 181
تأثير محددات الوصول على المشيدات 184
الفصل الخامس: الواجهات، التفويض، والمواصفات 187
الواجهات 187
بناء واجهة 190
تضمين الواجهة 192
الوصول الى الواجهة 196
وراثة الواجهات 197
واجهات من اطار عمل .NET Framework 198

الواجهة IComparable 199
الواجهة ICloneable 201
الواجهتان IEnumerable و IEnumerator 203
التفويض 207
الاجراءات الستاتيكية 208
اجراءات الفئات 211
محاكاة الاحداث 212
دمج التفويضات 214
المواصفات 217
صيغة كتابة المواصفات في Visual Basic .NET 217
مواصفات من اطار عمل NET Framework. 218
المواصفة Conditional Attribute 218
المواصفة DebuggerStepThrough Attribute 221
المواصفة Obsolete Attribute 221
المواصفة StructLayout والمواصفة FieldOffset 223
بناء مواصفات خاصة 225

الجزء الثاني: **إطار عمل** NET Framework. 229
الفصل السادس: **الفئات الأساسية** **231**
الفئة System.Object 231
طرق الفئة Object 232
البيانات المرجعية والبيانات ذات القيمة مرة اخرى 234
الصندقة واللاصندقة 236
الفئات الحرفية 236
الخصائص والطرق 239
مقارنة الحروف 242
الفئة CultureInfo 243
البحث عن الحروف 246
الفئات من النوع Char 247

PDF

الفئات من النوع StringBuilder 248
الفئات العددية 250
الخصائص والطرق 251
تنسيق الاعداد 252
الفئة Math 254
توليد الاعداد العشوائية Random Numbers 255
فئات اخرى 256
فئات الوقت والتاريخ 256
الفئات من النوع Enum 263
الفئات من النوع Array 265
مجال اسماء System.Collections 270
الواجهات ICollection و IList 270
الفئة Stack 271
الفئة Queue 272
الفئة ArrayList 273
الفصل السابع: اكتشاف الأخطاء 275
فكرة عامة 275
اخطاء وقت التصميم 275
اخطاء وقت التنفيذ 276
الشوائب 278
الكائن Exception 279
تفادي الاستثناءات Catching Exceptions 280
رمي الاستثناءات Throwing Exceptions 284
انشاء فئات استثناءات خاصة Custom Exceptions 287
الكائن Err 288
تفادي الاستثناءات 288
رمي الاستثناءات 290
الاختيار بين Exception و Err 290
ادوات التنقيح من Visual Studio .NET 292

اساليب التنفيذ 292
نوافذ اخرى 294
الاعدادات Configurations 296
الفصل الثامن: الملفات والمجلدات 299
الفئة Directory 299
طرق تعود بمسارات 301
البحث عن الملفات والمجلدات 302
الفئة File 303
الفئة Stream 305
الخصائص والطرق المشتركة 306
التعامل مع الملفات النصية 309
التعامل مع الملفات الثنائية 312
تكوين Custom Streams خاصة 314
فئات اخرى 316
الفئة Path 316
الفئات DirectoryInfo و FileInfo 317
الفصل التاسع: تسلسل الكائنات Object Serialization 321
مدخلك الى تسلسل الكائنات 322
ما هو التسلسل؟ 322
التسلسل بالصيغة الثنائية Binary Serialization 323
تسلسل انواع بيانات مخصصة (غير قياسية) 325
خريطة الكائنات Object Graph 228
نسخ الكائنات 231
انشاء Custom Serialization خاصة 335
الواجهة ISerializable 336
مثال تطبيقي 337
التسلسل بصيغة XML 338
الفئة XmlSerializer 340
مواصفات اضافية 341

PDF

احداث تقع عند عكس التسلسل 345
الفصل العاشر: مسارات التنفيذ Threading 349
مقدمة الى مسارات التنفيذ 349
انواع مسارات التنفيذ 350
متى تستخدم مسارات التنفيذ المتعددة؟ 351
انشاء مسار تنفيذ 353
الطرق والخصائص 354
التعامل مع مسارات التنفيذ 364
مشاركة البيانات 368
المتغيرات المحلية الديناميكية 369
المواصفة ThreadStatic Attribute 370
وحدة التخزين المحلية TLS 372
تبادل البيانات بين مسارات التنفيذ 374
التزامن Thread Synchronization 377
التركيب SyncLock ... End SyncLock 377
المواصفة Synchronization 379
المواصفة MethodImpl 380
فئات اخرى 381
الفئة ThreadPool 383
المؤقتات Timers 386
المؤقت System.Timers.Timer 387
المؤقت System.Threading.Timer 388
الفصل الحادي عشر: المجمعات Assemblies 391
الوحدات المدارة Managed Modules 391
المجمعات Assemblies 392
المجمعات الاحادية والمتعددة الملفات 393
اساليب تنفيذ المجمعات 394
المجمعات الخاصة والمشتركة 395
الاسماء القوية Strong Names 397

المواصفة Assembly 397
ملفات التهيئة Configuration Files 398
انواع ملفات التهيئة 399
تغيير الاعدادات 400
اعدادات لملفات التهيئة 400
استخدام الاداة .NET Framework Configuration 403
ادوات الترجمة، الربط، والتسجيل 405
المترجم VBC.EXE 406
الرابط AL.EXE 408
المسجل SN.EXE 412
المسجل GACUTIL.EXE 413
الفصل الثاني عشر: فئات الانعكاس Reflection Classes 415
التعامل مع المجمعات والوحدات المدارة 415
الفئة Assembly 416
الفئة Module 418
التعامل مع انواع البيانات 419
الفئة System.Type 419
خصائص اضافية 421
التعامل مع الاعضاء 422
الفئة القاعدية MemberInfo 422
التعامل مع الحقول 425
التعامل مع الخصائص 426
التعامل مع الطرق 428
التعامل مع الاحداث 429
الوسيطات Parameters 430
التعامل مع الكائنات 431
الفئة ReflectionExample 431
اسناد/قراءة قيم الحقول 432
اسناد/قراءة قيم الخصائص 433

استدعاء الطرق 433
مواضيع اخرى 435
الانشاء الديناميكي للكائنات..... 435
معرفة الاجراءات المستدعية 436

الجزء الثالث: تطوير تطبيقات Windows 439
الفصل الثالث عشر: نماذج Windows Forms **441**
مدخلك الى نماذج Windows Forms 442
مصمم النماذج Form Designer 442
نظرة حول الشيفرة المولدة 445
التعامل مع اكثر من نموذج 448
محل الفئة Form من الاعراب 449
الخصائص، الطرق، والاحداث 450
خصائص النموذج 450
طرق النموذج 453
احداث النموذج 456
نماذج MDI Forms 460
النوافذ الابناء Child Windows 461
خصائص وطرق اضافية 462
القوائم Menus 463
الخصائص، الطرق، والاحداث 465
القوائم المنبثقة Popup-Menu 466
نماذج MDI مرة اخرى 466
الانشاء الديناميكي للقوائم 469
مواضيع متقدمة 469
التفاعل مع نوافذ Modeless 469
وراثة النماذج Form Inheritance 470
النماذج المحلية 474
الفصل الرابع عشر: الأدوات Controls **479**

PDF Factory Pro trial version www.fastio.com

الخصائص المشتركة 479
اسم الاداة Name 480
خصائص المظهر 480
خصائص الموقع والحجم 482
خصائص الاحتضان 487
خصائص الالوان 488
خصائص التركيز 489
خصائص الجدولة 489
خصائص اخرى 490
الطرق المشتركة 490
الاحداث المشتركة 491
احداث الفأرة 492
احداث لوحة المفاتيح 493
احداث التركيز 494
احداث اخرى 494
عرض سريع للادوات 495
الاداة Label 497
الاداة LinkLabel 497
الاداة TextBox 498
الاداة Button 501
الاداة CheckBox 501
الاداة RadioButton 502
الاداة ListBox 502
الاداة CheckedListBox 504
الاداة ComboBox 505
الاداة ImageList 505
الاداة TreeView 506
الاداة ListView 508
الاداتان ToolBar و StatusBar 509

الاداة Splitter 510
ادوات صناديق الحوار الشائعة 510
ادوات المزودات 514
ادوات اخرى 515
تقنية المرآة 518
الخاصية RightToLeft 518
قصور الخاصية RightToLeft 520
مدخلك الى تقنية المرآة 521
تطبيق تقنية المرآة بـــ Visual Basic .NET 522
مشاكل اضافية 527
ادوات صناديق الحوار الشائعة 527
صناديق الرسائل 528
الفصل الخامس عشر: مبادئ GDI+ 531
الرسم المتقدم 531
الكائن Graphics 532
رسم الخطوط، المستطيلات، والدوائر 533
رسم المنحنيات المعقدة 534
كائن القلم Pen 535
كائن مسار الرسم GraphicsPath 538
التعبئة 537
كائن الفرشاة Brush 540
انظمة القياس 542
التعامل مع الصور 545
تحميل وحفظ الصور 545
عرض الصور 546
عكس، قلب، وسحب الصور 549
تحديد الالوان 552
الرموز Icons 552
المخرجات النصية 554

عوائل الخطوط 554
رسم النصوص 556
التفاف النص 557
الكائن StringFormat 558
عودة الى الادوات Controls 567
الفصل السادس عشر: مواضيع متقدمة 571
تطوير ادوات خاصة 571
وراثة اداة 572
حضن مجموعة من الادوات 578
انشاء اداة مستقلة 581
لمسات فنية اضافية 582
تطوير خدمات Windows 587
مقدمة الى خدمات Windows 587
انشاء مشاريع من نوع Windows Service 588
تصحيح الشيفرة 590
الفئة System.IO.FileSystemWatcher 592
كتابة الشيفرات 593
تسجيل الخدمة 595
الاداة InstallUtil.EXE 596
فئات اخرى 598
الفئة Application 598
الفئة Cursor 599
الفئة SendKeys 600
الفئتان Registry و RegistryKey 601
الفئة Help 603

الجزء الرابع: برمجة قواعد البيانات 605
الفصل السابع عشر: استخدام ADO.NET 607
مدخلك الى ADO.NET 608

PDF

الوضع المتصل والوضع المنفصل 608
مزودات .NET Data Providers. 609
فئات ADO.NET 611
كائن الاتصال Connection 612
انشاء كائن اتصال 612
نص الاتصال 613
فتح واغلاق الاتصالات 614
كائن الاوامر Command 617
انشاء كائن اوامر 618
الربط مع اتصال 618
تنفيذ جمل الاستعلام SQL 620
قراءة السجلات 621
كائن البيانات DataReader 623
انشاء كائن بيانات 623
قراءة السجلات 624
خاص بمستخدمي Microsoft SQL Server ® 626
قراءة نتائج متعددة 627
الفصل الثامن عشر: ADO.NET للوضع المنفصل 629
كائن البيانات DataSet 629
الفئة DataTable 632
الفئة DataRow 633
الفئة DataColumn 634
الفئة DataRelation 634
انشاء كائن بيانات DataSet من الصفر 635
كائن المحول DataAdapter 638
سيناريو الوضع المنفصل 638
انشاء كائن محول DataAdapter 639
الربط مع اتصال 640
قراءة البيانات 641

تحديث البيانات 644
تخصيص افضل للخصائص xxxCommand 648
اتقاء شر التعارضات 649
عرض التعارضات 650
الحدث RowUpdated 651
الفصل التاسع عشر: ربط البيانات والتكامل مع XML 653
ربط البيانات 653
انواع الربط 654
ميكانيكية الربط 655
الربط الى مصفوفة 656
الربط باستخدام ADO.NET 659
الربط المعقد Complex Binding 660
فئات خاصة بـــ XML 662
الفئة XmlTextReader 663
الفئة XmlTextWriter 664
تكامل ADO.NET و XML 665
كتابة البيانات بصيغة XML 666
قراءة البيانات بصيغة XML 668

الجزء الخامس: برمجة ويب 669
الفصل العشرون: تطبيقات ASP.NET (1) 671
الخادم IIS 671
تركيب الخادم IIS 672
الادلة الوهمية 673
الوصول الى الادلة الوهمية 675
مدخلك الى نماذج Web Forms 677
انشاء المشروع 678
ضبط الاعدادات الرئيسة 681
كتابة الشيفرات 682

PDF

تحليل الشيفرة 685
اساليب تنفيذ الصفحة 688
الخلاصة 690
الفئة Page 690
خصائص صفحة النموذج 690
طرق صفحة النموذج 693
احداث صفحة النموذج 693
وسوم اضافية 694
الادوات 695
ادوات Web Forms Controls 695
ادوات HTML Forms Controls 696
ادوات التحقق Validation 697
الفصل الحادي والعشرون: تطبيقات ASP.NET (2) 701
كائنات صفحات ASP.NET الاساسية 701
الكائن HttpRequest 701
الكائن HttpResponse 703
الكائن HttpServerUtility 705
الكائن HttpApplicationState 705
الكائن HttpSessionState 706
الملف Global.asax 707
الاجراءات xxxStart() و xxxEnd() 707
الاجراء Global_Error() 708
الامان Security 709
مدخلك الى الصلاحيات 709
اوضاع التصديق في ASP.NET 708
ملفات التهيئة 709
الوضع Forms للتصديق 709
الوسم <credentials> 713
صفحة تسجيل الدخول Login 714

مواضيع متقدمة 715
التخزين Caching 715
المتغيرات العامة 718
حماية الصور 718
التوليد الديناميكي للصور 720
الفصل الثاني والعشرون: خدمات ويب Web Services 723
مدخلك الى خدمات ويب 723
كيف تعمل خدمات ويب؟ 724
بناء خدمة ويب 725
انشاء المشروع 725
كتابة الشيفرة 727
اختبار الخدمة من المتصفح 728
استخدام الخدمة 731
تحديث الخدمة 735

الملاحق

الملحق أ: لغة وصف البيانات XML ..... م 1
الملحق ب: لغة الاستعلام SQL ..... م 7

PDF

تقديم

لقد أسعدني خبر أن تركي العسيري قام بكتابة كتابا جديدا عن لغة Visual Basic .NET والتي تعتبر لغة المستقبل للمطورين والمبرمجين، وان كنت لا تعلم من هو تركي العسيري، فدع هـذا الكتاب يعرفك به؛ يكفيك مقالاته التي تمتلئ بها المواقع العربية حول تقنية NET.، ولا تنسى انه المؤسس للموقع الشهير **شبكة المطورون العرب** (dev4arabs.com) وكاتب لجميـع الـشيفرات المصدرية التي تنبض الموقع بالحياة، وما يجعلني أشعر بالفخر هو أن الكاتب عربي ويوجه خبرته ومعرفته للعرب.

يتمتع الأخ تركي بمزايا خاصة سواء كان مبرمجا أو كاتبا، فلديه من الأسلوب الذي يجعلك تستوعب الأمور المعقدة وكأنها معادلة 1+1=2 ! والآن أرى أنه أنجز مشروعا (كما يلقب كتابه) يعتبر أول كتاب من مؤلف عربي حول تقنية Microsoft .NET، كما سـطر مجموعـة مـن الصفحات والموجه خصيصا للمطور العربي.

كخبير تصميم وتطوير مواقع، أرى بأن الكتاب سيفيد الكثير من المبرمجين العرب التائهين في تقنية NET.، فهذه التقنية تعتبر تقنية المستقبل لتطوير التطبيقات ومواقع الإنترنت، وهي قوية جدا ولها من الإمكانيات ما يفوق التوقعات، ولكن مسالة استيعابها سيكون صـعبا علـى اغلـب المبرمجين المستجدين والمخضرمين أيضا!، هذا وان علمت أنها من أحدث التقنيات الصادرة من معامل ريدمون بولاية واشنطن (اعني من شركة Microsoft)، ووجود مصدر لإتقانها يحتاج إلى جهد جهيد، ومن النادر الحصول على مرجع لها باللغة العربية، فكل الكتب العربية والخاصة بتقنية NET. مترجمة، فما بالكم إذا كان الكاتب عربي يعرف كيف نفكر وما هي متطلباتنا وإلى مـاذا نتطلع أو نتوقع من كتاب بهذا الحجم والتخصص.

يتميز هذا الكتاب بالنمط الاستيعابي الذي يتعمده المؤلف، فهو يركز على السهولة والمنطق في التسلسل لإيصال المعلومة للقارئ وهذا ما يعرف عن الكاتب، فأسلوبه سلس ودائما ما يحاول النزول لمستوى القارئ ثم يرتقي به كلما استمر في القراءة، كما انه يضفي على الكتاب أسـلوب المحاورة مما يوحي أن المؤلف أمامك ويناقشك في المواضيع وكأنك في قاعة محاضرة، أضـف

إلى ذلك الأمثلة الفورية التي يعرضها عند الحديث بعد شرح كل نقطة، فهي ليست كبيرة ليصعب تتبعها.

أما عن المسالة التجارية، فواضح من الأسلوب في الكتابة ووفرة المعلومات بأن الكاتب حريص على إيصال أكثر قدر من المعرفة ليجعل القارئ يتعلم أكبر قدر ممكن وبالتالي يستطيع تطبيق ما تعلمه والاستفادة من قراءة الكتاب، فلا يقوم بنسخ جداول وثائق NET. Documentation للتكثير من عدد الصفحات وزيادة الدخل كما تفعل اغلب الكتب المنتشرة في الأسواق، فمعظم هذه الكتب تتجاوز صفحاتها الألف صفحة، وبعد شرح كل نقطة يقوم المؤلف بعرض جدول **منسوخ من وثائق NET Documentation.** وإضافته. ولكن مع هذا الكتاب فلن يحدث معك ذلك، حيث انك ستلاحظ جملة المؤلف المتكررة "**راجع مكتبة MSDN لمزيد من التفاصيل**" وهذا خير دليل على أن الكاتب لا يريد زيادة الصفحات المنسوخة من وثائق NET. Documentation وتكرارها في كتابه.

وختاماً أتمنى من الله أن يوفق المؤلف في نشر المعرفة للعرب، وأن يبدأ الشباب بالمضي قدماً في مجال النشر والتعليم لأنه سلاح العصر، وأنا واثق كامل الثقة بأنك لن تشعر بالندم بعد اقتنائك لهذا الكتاب، ولن تضعه مع الكتب الأخرى في رفوف مكتبتك، بل سيضل ناصبا على سطح مكتبك.

تنبيه أخير ناتج من تجربة شخصية: الكثير من المفاجئات والأفكار ستجدها في هذا الكتاب، لذلك لا تقرأ هذا الكتاب قبل موعد النوم بدقائق حيث انك لن تستطيع النوم دون تجربة الشيفرات المصدرية بنفسك!

إياد يحيى مكي زكري،

عضو في الإتحاد العالمي لمصممي ومدراء المواقع **IAWMD** (iawmd.com).

PDF

شكر وتقدير

مجموعة من الأشخاص دعموني بشكل مباشر و غير مباشر لإنجاز كتاب **برمجة إطار عمل NET. باستخدام Visual Basic .NET**. صحيح أن كلمة شكر قد لا تكون كافية للأشخاص التالي ذكر أسمائهم، ولكن ما كان في القلب أعظم.

لنبدأ بالصديق إياد مكي كاتب صفحة **التقديم** في هذا الكتاب، أشكرك على كلماتك الجميلة - والتي قد بالغت في بعضها حولي، كما اقدر عرفانك ووقفات قريبا مني في الفترات السابقة، وبالنسبة لنصائحك وتوجيهاتك فقد أخذتها بعين اعتباري من الصفحة الأولى حتى الأخيرة عند كتابة هذا الكتاب.

اختص بالشكر الجزيل جدا جدا للأستاذ الفاضل سالم المالكي، والذي بنى معي أساسيات هذا الكتاب، كما غطى على ابرز الأخطاء التي اتبعتها منذ البداية والذي علمني كيف تكون البداية الصحيحة لتعليم لغات البرمجة –خاصة إن كانت جديدة كليا.

ثلاثة أسباب تجعلني اشكر الصديق محمد الحلبي، السبب الأول هو مراجعته الدقيقة لكل حرف من حروف شيفراتي المصدرية (رغم انه مبرمج Visual C# .NET وليس Visual Basic .NET)، والثاني وقفاته قريبا مني رغم انشغالاته وظروف السفر التي لم تمنعه من مواصلة المشوار معي، أما السبب الثالث فهو تكلفه بتوفير Visual Studio .NET وإرسالها لي، ولو لم يفعل ذلك فلم يكون هذا الكتاب أمام عينيك.

اعتقد انه من المناسب هنا ذكر مشرفي **شبكة المطورون العرب** (dev4arabs.com) (الموقع الذي أديره مع مجموعة من المحترفين العرب في مختلف مجالات علوم الحاسب) وذلك نظير استمرارهم في إدارة الموقع بينما كنت مشغولا في كتابة هذا الكتاب، اشكر كلا من: طارق موسى، محمد نسمان، محمد العتيبي، أيمن المدهون، عمر رضوان، لمياء حاشي، هند محسن، محمد لبد، واحمد الشمري.

كما يتحتم علي هنا شكر المهندس حسين فاضلي الذي سهل لي عقبة توفير المراجع اللازمة حول Visual Basic .NET وتقنية Microsoft .NET (في أيامي الأولى مع هذا الكتاب لم تكن متوفرة بكثرة في الأسواق). كما اشكر الأستاذ عبدالله العتيق مؤسس موقع (vb4arab.com) (اكبر موقع عربي للغة Visual Basic) على دعمه المعنوي، ليس فقط في الإعلان عن كتابي بل

PDF

تخصيص صفحة خاصة به. ولا أنسى شكر الأستاذ دانيال ريد (حيث انه أكثر سبقا مني في مجال تأليف كتب البرمجة) على إرشاداته الفائقة الروعة لطريقة إعداد مخطط العمل لكتابة هذا الكتاب.

من الأسماء الذي ساهمت في هذا الكتاب أيضا، الصديق صالح الغامدي مصمم غلاف الكتاب، وهناك أيضا فهد العمير الذي سهل لي عملية عرض أمثلة هذا الكتاب واختصارها حتى يتم استيعابها بسهولة. وأما البعيدة القريبة الأستاذة الفاضلة غادة فلا اعلم من أين ابدأ أو انتهي لعرض فضائلها وجمائلها علي والتي تحتاج إلى كتاب كامل لذكرها.

إلى جميع الأسماء السابق ذكرها، اكرر شكري الجزيل وامتناني الكبير لكم، المشاركة والعمل معكم في الفترة الماضية كان فرصة من اسعد الفرص في حياتي، كما اعتقد أنني شخص محظوظ جدا بمعرفتكم الطيبة، عسى أن تجمعنا أعمال أخرى مشرفة.

وأخيرا، مهما سطرت من كلمات الشكر فإنها لم ولن توفي حق أعظم من رأته عيني في حياتي، إليك يا صانع الرجال، إليك يا سيدي، إليك يا والدي العظيم، كم سجدت لله شكرا وحمدا على نعمته علي في أن شرفني وجعلني من صلبك.

-- تركي

PDF

المقدمة

عندما شرعت في كتابة هذا الكتاب، خانتني العبرة ولم تأتي التوقعات كما رسمت في الخاطر، فالموضوع اكبر من كونه شرح لغة برمجة وإجراء تطبيقات بها، حيث ان كتابة كتاب عن لغة البرمجة Visual Basic .NET ليس سوى لهجة بلسان آخر تخاطب بها تقنية Microsoft .NET..

سواء كنت مبرمج Visual Basic .NET او مبرمج Visual C# .NET او اي لغة برمجة أخرى موجه الى إطار العمل NET Framework. فلن تقدم ولن تؤخر ذلك الشيء الكبير، إذ ان اللقب الذي عليك البحث عنه دائما هو **مبرمج NET.** او –لمزيد من التفصيل- مبرمج NET. بلهجة Visual Basic .NET. وان لم يتضح لك المعنى من اللقب مبرمج NET. فعليك معرفة ان لغة البرمجة Visual Basic .NET ليست سوى مفتاح بسيط تستخدمه للباب المعقد والمسمى NET Framework.، فالاحتراف في برمجة Visual Basic .NET لن يتطلب منك ذلك الجهد للوصول إليه في أيام معدودات (قراءة الفصول الخمس الأولى من هذا الكتاب كافية إلى حد كبير)، أما الاحتراف في تقنية NET. فيحتاج إلى تفرغ ذهني من كل شيء في حياتك تقريبا.

إطار عمل NET Framework. معقد جدا، وهو بحر ليس له بداية ولا نهاية، وقد لا أبالغ ان قلت ان كل ميناء من موانئ هذا البحر تحتاج إلى كتاب مخصص. فآلاف الصفحات والخاصة بمستندات تقنية NET. Microsoft الرسمية خير دليل. ولتقديم كتاب عن هذه التقنية، كان علي اختيار احد الحلول الثلاث:

الأول هو كتابة كتب متعددة يختص كل واحد فيها بمجال معين، كتاب عن تطوير تطبيقات Windows وآخر عن برمجة ويب، وثالث عن الاستخدام المتقدم لتقنية ADO.NET لبرمجة قواعد البيانات، ولكن يعيبه الوقت الطويل الذي سيستغرقه بالإضافة ان جميعها تتطلب شرح أساسيات اللغة، مما يعني تكرار معظم الصفحات. الحل الثاني، هو مشاركة كتاب آخرين ومحاولة توسعة وزيادة المادة العلمية في هذا الكتاب، ولكن يعيبه اختلاف الأساليب التي يتبعها كل مؤلف مما قد يسبب التشويش على القارئ. أما الحل الثالث فهو اقتطاف من كل بستان زهرة ووضعها كمقدمة ومدخل لك يمكنك من الانطلاق منه، وهذا ما فعلته في كتاب **برمجة إطار عمل NET. باستخدام Visual Basic .NET**.

PDF

لمن هذا الكتاب؟

هذا الكتاب للمبتدئين ام للمتوسطين او للمحترفين؟ سؤال يراود الكثير ويهتمون في إجابته أكثر من اهتمامهم بمؤلف الكتاب. لكل إنسان مقياس خاص لتصنيف المستويات، وقد يكون مقياسي الشخصي مرفوض من قبل الكثيرين، لذلك لن أعطيك إجابة مباشرة لهذا السؤال، وسأكتفي بذكر صفات الأشخاص الذين اعتقد ان الكتاب سيكون مناسب لهم من نظرتي الشخصية.

أستطيع تصنيف كتب لغات البرمجة إلى أربعة أنواع، النوع الأول وهو **التعليمي Tutorial** حيث يوجه إلى الأشخاص الذين ليس لديهم الوقت الكافي للقراءة، ولا يودون استيعاب المبادئ بقدر ما يهمهم إنجازها. فستجد في مثل هذه النوعية من الكتب عشرات السطور المتمثلة في خطوات Step by Step مرقمة تقوم بإنجاز مهمة معينة دون ذلك الشرح.

النوع الثاني هو **المراجع Reference**، وهي كتب لا تقرأ من الغلاف إلى الغلاف وإنما يرجع لها من وقت لآخر. وكما تعلم ان المراجع تكتب من قبل آلاف الأشخاص، وقد احتاج للكتابة بأصابع قدمي حتى أنجز مرجع قبل ان تشيب شعرات رأسي.

النوع الثالث من أنواع الكتب هو **ورشة العمل Workshop**، وهو عندما يميل الكتاب إلى الجانب التطبيقي أكثر من الشرح، ولكن يفترض المؤلف انك تعلم كل شيء عن لغة البرمجة، حيث يعرض لك الأساليب البرمجية المتعددة لإنجاز المهام وتطبيقها مباشرة.

أما الكتاب الذي أمام عينيك فهو من النوع الرابع وهو **الاستيعابي Comprehensive** فأميل فيه إلى شرح الأساسيات وبناء قواعد معرفية تمكنك من الانطلاق في برمجة إطار عمل NET Framework. من أوسع أبوابها، حيث يمكنك اعتبار ان كل فصل من فصول هذا الكتاب مقدمة او مدخل الى استخدام تقنية من تقنيات إطار عمل NET Framework.، ويبقى الأمر في النهاية عليك لتتعمق وتتخصص في المجال الذي تريده بنفسك.

ماذا عن مبرمجي Visual Basic 1 è 6؟

حسنا، دعني هنا أخاطب تلك الفئة من المبرمجين عشاق النغمة الرنانة Visual Basic. العلاقة بين Visual Basic .NET و الإصدارات السابقة Visual Basic 1->6 لا تكاد ان تكون إلا علاقة تشابه أسماء فقط، فهي مشروع تسويقي أكثر من ما هو تطوير للغة البرمجة المحتكرة من

PDF

قبل Microsoft. حيث ان الضريبة التي كانت ستكلف مشروع Microsoft .NET هو إنتاج لغة البرمجة الجديدة Visual C# .NET والاستغناء عن الملايين من مبرمجي Visual Basic حول العالم. إنتاج لغة البرمجة الجديدة أمر لا بأس منه، أما قضية الاستغناء عن ملايين المبرمجين فهي بحاجة –بكل تأكيد- إلى إعادة نظر.

بوضوح وصراحة مباشرة، أنتجت Microsoft لغة البرمجة الجديدة Visual C# .NET وأكاد أتخيل احد صناع القرار في تلك الشركة يقول: لما لا نقوم باستبدال صيغ لغة C# إلى صيغ شبيهة بلغة BASIC، أي –باختصار- احذفوا الأقواس من عبارة الشرط if واجبروا كتابة Then بعدها، مع إلغاء ضرورة استخدام الفاصلة المنقوطة ";" نهاية كل سطر.

صحيح ان Visual Basic .NET هي نسخة بلسان آخر من Visual C# .NET، إلا أن الفروق طفيفة ولا تكاد تذكر (أستطيع ان اجزم ان Microsoft تعمدت وضع هذه الفروق حتى نقتنع أنهما لغتا برمجة مختلفتين).

بالرغم من تشابه الصيغ بين Visual Basic .NET والإصدارات السابقة Visual Basic 1->6 إلا انه من الخطأ الكبير والجرم العظيم اعتبار Visual Basic .NET إصدار جديد منها، حيث انك ستتعامل مع لغة برمجة جديدة كليا وليس لها اي علاقة –كما ذكرت- مع الإصدارات السابقة من Visual Basic. لذلك، تقبل نصيحتي هذه قبل ان تعاني الكثير من المتاعب –كما عانيت انا- وانسى كل ما تعلمته سابقا في الإصدارات القديمة من Visual Basic، وضع في عين اعتبارك دائما انك تتعامل مع لغة برمجة جديدة وحديثة العهد اسمها Microsoft Visual Basic .NET..

المصطلحات تعريب ام تعذيب؟

لا تزال مشكلة المصطلحات العربية في رمتها، وبما أنني لست في منصب مسئول لتحديد واختيار الترجمات العربية الصحيحة للمصطلحات الأجنبية، فذلك يعني حريتي في اختيار ما أراه مناسبا ورفض ما لا يناسبني.

وجهة نظري الشخصية حول ترجمة المصطلحات تتمحور حول اختيار الكلمة الأكثر شعبية أو التي تميل إلى توضيح المعنى التقني **وليس المعنى الحرفي**، لديك الكلمة **Help** مثلا والتي تترجم –بشكل صحيح- إلى "**مساعدة**"، ولكني فضلت اختيار المصطلح "**تعليمات**" لشعبيته بين

PDF

مستخدمي نظم Windows. من ناحية أخرى، لديك المصطلح **Overloading** والذي تترجمه الكتب العربية "**بالتحميل الزائد**"، وهو تعبير لم استطع تقبله لا من بعيد ولا من قريب، فلا يوحي معناه بالهدف منه، وكان اختياري للتعبير "**إعادة التعريف**" هو الأنسب والأفضل للترجمة التقنية له.

مع ذلك، قدمت الأمر في البداية والنهاية لك أيها القارئ العزيز لاختيار المصطلحات التي تناسب توجهاتك اللغوية، فلا أكاد اذكر مصطلح إلا وأرفق المقابل الإنجليزي له من مستندات NET. Documentation، وان كان اختياراتي لا تناسب توجهاتك اللغوية، فيمكن الشطب على الكلمة وكتابة ما تريد بدالها.

ماذا يقدم لك هذا الكتاب؟

دعنا نضع النقاط على الحروف، ونكون واقعين قدر الامكان، فقبل ان توجه لي النقد حول قصوري في ذكر محتويات إطار عمل NET Framework.، عليك معرفة ان هذا الكتاب **ليس مرجعا من مراجع MSDN**، فلا تتوقع مني شرح كل شيء وذكر كافة التفاصيل عن المواضيع التي تطرقت لها، فانك تخاطب شخص يكتب بعشرة أصابع فقط، ومن غير منطقي مقارنة مجهوده بمجهود آلاف الموظفين في شركة Microsoft. وان كنت شخص تقدر الكم العددي على الكيف المعلوماتي، فيؤسفني إخبارك بان هذا الكتاب ليس مناسب لك.

إذاً هل هذا الكتاب بهذا السوء؟ في الحقيقة ستكون شهادتي –بكل تأكيد- مجروحة إن مدحته، ولكن دعني اعرض لك ماذا ستجد بين ثنايا الـ 700 صفحة والمكونة لهذا الكتاب:

أساسيات لغة البرمجة NET. Visual Basic من الصفر تبدأ بعرض كيفية كتابة أول برنامج لك، مع شرح الصيغ والعبارات المستخدمة في لغة البرمجة كجمل الشرط، التفرع، والتكرار. بالإضافة إلى نظرة كائنية التوجه OOP وطرق تعريف الفئات Classes لإنشاء الكائنات Objects، والاستخدام الأمثل لها لتطبيق مبادئ الوراثة Inheritance وتعدد الواجهات Polymorphism.

عرض سريع لمكتبة فئات إطار عمل NET Framework. إن أردت اخذ جولة سريعة حول مكتبة فئات إطار عمل NET Framework.، فستجد ملخصا لها هنا، حيث اعرض لك مجموعة من الفئات الأساسية، وفئات أخرى تستخدمها لإنجاز مهام معينة، مثل: كائنات الاستثناءات Exceptions، دخل خرج الملفات File IO، تسلسل الكائنات Object

Serialization، مسارات التنفيذ Threading، طريقة التعامل مـــع المجمعـــات Assemblies باستخدام فئات الانعكاس Reflection Classes.

أساسيات تطوير تطبيقات Windows Application باستيعاب فكرة نماذج Windows Forms وطريقة وراثتها، كما آخذك في جولة سريعة حول معظم الأدوات Controls والغرض منها، بالإضافة إلى تطبيق تقنية المرآة Mirroring عليها. ثم اخصص فصل كامل يعتبر مدخلك إلى استخدام تقنية GDI+ وطرق الرسم والتعامل مع الصور والمخرجات النصية. المزيد أيضا، اعرض لك كيفية تكـــوين أدوات خاصـــة Custom Controls وبنـــاء خـــدمات Windows Services.

استخدام ADO.NET لبرمجة قواعد البيانات وشرح لفكرة الوضع المتصل Connected Mode والوضع المنفصل Disconnected Mode، وذلك بذكر الفئات اللازم استخدامها في كلا الوضعين، كما أتحدث عن طريقة ربطها بالأدوات وتكامل تقنية ADO.NET مع XML.

مقدمة لبرمجة ASP.NET لأعرض فيه فلسفة عمل صفحات الخادم ASP.NET وبنـــاء مواقعك المستخدمة لهذه التقنية، كما اعرض لك مجموعة من الكائنات التي لا غنى عنها في اغلب مشاريع ASP.NET، واختم الكتاب بالتحدث عن خدمات ويب Web Services وطريقة إنجازها واستخدامها.

القرص المدمج CD-ROM

يحتوي القرص المدمج المرفق مع هذا الكتاب على جميع الشيفرات المصدرية التي أضيف الرمز في أعلاها، مقسمة الى مجموعة من المجلدات تمثل رقم الفصل الذي سطرت في الشيفرة. كما تعمدت إضافة ملفات صور (من النوع JPG) تمثل الصور والأشكال التوضيحية المعروضة فـــي الفصول، وذلك خشية عدم وضوحها بين صفحات هـــذا الكتـــاب. كمـــا يمكنـــك إيجـــاد الملـــف ArabicConsole.DLL والذي طورته لمحاكاة الكائن Console في الدليل الجذري للقرص - سأخبرك به لاحقا في الفصل الأول **تعرف على Visual Basic .NET**.

PDF

صفحة الكتاب على الانترنت

لا أود إنهاء علاقتي معك مع الصفحة الأخيرة من هذا الكتاب، بل سيكون شرف لي تمديدها إلى فترات أطول. يمكنك عزيز القارئ اصطيادي على شبكة الانترنت إما عن طريق مراسلتي على بريدي الالكتروني او من خلال زيارة صفحة الكتاب إن كنت ترغب في الحصول على آخر التحديثات وتصحيح الأخطاء المتعلقة به، وذلك بتوجيه متصفحك إلى العنوان التالي:

http://www.dev4arabs.com/lib/vbnetbook

الأخطاء (هام جدا)

الوصول إلى الكمال شيء لا يتم إلا بالقدرة الإلهية، والعمل الذي أمام عينك جهد بشري متواضع ومعرض بنسبة كبيرة للخطأ، يهمني جدا التقليل من الأخطاء في هذا الكتاب، حتى يكون مرجعا عربيا سليما من الشوائب. في حالة العثور على أي خطأ (سواء لغوي او تقني)، برجاء إبلاغي فورا عن الخطأ ورقم الصفحة الذي وقع فيها، وبهذا تسدي لي خدمة سأكون شاكرا ومقدرا لها.

كلمة أخيرة

أخي المبرمج العربي من المحيط إلى الخليج، بودي توضيح نقطة ضرورية تتمحور حول مؤلف الكتاب الذي تقرأه الآن. صحيح ان فن الكتابة بعيد كل المبعد عن مجال عملي إلا انه عليك معرفة أنني مبرمج ولست كاتب، فلست من الذين لديهم أعمدة في الصحف والمجلات ويهوى كتابة المقالات، بل أقوم بكتابة العديد من المشاريع والبرامج لمختلف القطاعات. كما لدي الكثير من العلاقات بين عمالقة المبرمجين في أنحاء المعمورة وقد استفدت الكثير والشيء الكبير من احتكاكي معهم. واعتقد –بل أكاد اجزم- انك لم تقتني هذا الكتاب لتتعلم فن البلاغة او الاستمتاع بالتعابير اللغوية، بل تريد أن تتعلم لغة برمجة اسمها Visual Basic .NET. لذلك، حاول قدر الامكان

PDF

تجاهل تعابيري اللغوية الركيكة، وضعف أسلوبي البلاغي والكتابي، ولنجعل لغة الشيفرات المصدرية Source Codes هي القاسم المشترك بيننا.

الفترة السابقة التي قضيتها في كتابة هذا الكتاب كانت طويلة بعض الشيء والوقت قصير جدا، ولم استطع –بصراحة شديدة- كتابة كل التفاصيل التي وددت ذكرها في هذا الكتاب، لذلك وضعت في عين اعتباري قدرة القارئ على التعلم الذاتي، حيث كان هدفي إعطائك مفتاح ومدخل لمسائل عديدة وتركت الباقي عليك للبحث عن التفاصيل الأخرى سواء في مواقع الانترنت او مستندات ومراجع NET Documentation. الرسمية.

برمجة إطار عمل NET. باستخدام Visual Basic .NET ما هو إلا محاولة جادة لتأليف كتاب عربي من مؤلف عربي وموجه إلى مبرمج عربي لتقديم كل ما يحتاجه من معلومات تمكنه من بناء تطبيقات وحلول عملية موجه إلى إطار عمل NET Framework. باستخدام لغة البرمجة Visual Basic .NET، بدئا من توضيح أساسيات لغة البرمجة وانتهائا بتطوير تطبيقات مختلفة المجالات داعمة للغة العربية. أتمنى أن أكون قد وفقت في محاولتي هذه وقدمت ما يرضي ذوق المبرمج العربي ... حظا سعيدا!

-- تركي العسيري
الظهران – فبراير 2003
turki@dev4arabs.com

PDF

الجزء الأول

الأساسيات

PDF

تعرف على Visual Basic .NET

بسم الله نبدأ وعلى بركته نسير مع الجملة Visual Basic .NET، تتكون هذه الجملة من 14 حرفا ونقطة واحدة، الحروف الـ 11 الأولى تعني لغة برمجة اسمها Visual Basic، والنقطة والحروف الثلاث الأخيرة تعني إطار عمل NET Framework.. لذلك، يمكننا ان نطلق على هذه اللغة Visual Basic for .NET Framework (أي لغة البرمجة Visual Basic الموجهة إلى إطار عمل NET Framework.).

رحلة الألف ميل تبدأ بخطوة، وخطوتنا الأولى في رحلتنا مع Visual Basic .NET ستكون نظرية بحتة. فقبل أن تبدأ بفرقعة أناملك لكتابة الشفرات وبناء برامجك، من الجيد اخذ فكرة مبسطة عن الحروف الشهيرة NET. ومعرفة ماذا تعني هذه الحروف، وما الذي تقدمه لك، وما هي الحاجة التي تدعونا –أنت وأنا على الأقل- للانتقال إلى برمجة إطار عمل NET. Framework. وإذا كنت من المبرمجين المخضرمين، فسيكون هذا الفصل مصدر لإفراغ جميع معاناتك السابقة مع عالم البرمجة، أما إن كان هذا الكتاب أول كتاب برمجة تقرأه في حياتك، فاعتبر نفسك مبرمج محظوظ جدا لما ستكتشفه من التعقيدات التي كانت تواجه المبرمجين قبل تقنية NET..

الحياة قبل NET.

لست هنا بصدد تأليف كتاب تاريخ أو التحدث عن بدايات ظهور الحاسبات الشخصية، ففي ذلك الزمن لم أكن شيئاً مذكورا. ولكن ما أنا بصدده الآن هو تقديم عرض مقتضب وسريع لأساليب بناء البرامج منذ نظام التشغيل DOS وحتى لحظة كتابة هذه السطور، وسيكون حديثي موجهاً لمبرمجي نظم Microsoft بشكل حصري.

البرمجة تحت نظم DOS

كان كل ما هو مطلوب منك -كمبرمج- استخدام أمر Input لقنص المدخلات من المستخدم، والأمر Print لعرض المخرجات على الشاشة، بالإضافة إلى استخدام مجموعة من العبارات لتطبيق العمليات الحسابية. كانت في الحقيقة برمجة سهلة وممتعة للمبرمجين، حتى أصبح كل من هب ودب يدعي انه مبرمج، إلا أن النتيجة كانت برامج متشابهة، لا جديد فيها ولا تستخدم تقنيات جديدة.

بعد ذلك، ظهرت الحاجة إلى التحليق إلى مدى ابعد من الأسلوب السابق، فكانت أهداف التحليق -بشكل مبدئي- التفاعل مع الأجهزة Devices التي تركب في الجهاز (كالطابعة، بطاقة الصوت، الفأرة ...الخ)، إلا أن أجنحة المبرمجين في ذلك الوقت كانت تعتمد اعتماداً كلياً على بريمجيات تابعة تسمى **المشغلات Drivers**. معظم هذه المشغلات كانت تنجز بلغة التجميع Assembly، وتتطلب خبرة كبيرة في التعامل مع المعالج وعتاد الكمبيوتر. فلو قمت بعمل برنامج يطبع النتائج على ورق الطابعة، فعليك إرفاق مشغل الطابعة مع البرنامج، وإذا أردت من برنامجك أن يعزف ملف صوتي، عليك إرفاق مشغل الصوت.

قد تبدو فكرة إرفاق ملفات المشغلات مقبولة -إلى حد ما- لبعض المبرمجين، إلا أن المشكلة الحقيقة التي كنا نواجهها هي أن لكل طابعة مشغل خاص بها. وبما أنه ليس لدينا أي فكرة عن نوعية الطابعة التي ستكون على طاولة المستخدم، فإن ذلك يفرض علينا إرفاق مشغلات لجميع أنواع الطابعات الموجودة في السوق. فلو تذكر برنامج Lotus 123 الذي يعمل تحت نظام MS-DOS، ستعلم أن البرنامج يرفق معه أكثر من 200 ملف، هذه الملفات ما هي إلا مشغلات لمختلف أنواع الطابعات. بالنسبة لي، كنت أفضل استخدام يدي لكتابة النتائج على الورق بدلا من تحمل تكاليف نسخ الأقراص (كانت غالية جدا في ذلك الوقت) لوضع مشغلات الطابعات عليها.

الانتقال إلى Windows

أما مع نظام التشغيل Windows فقد حلت المشكلة السابقة، بحيث يتكفل نظام التشغيل بمهمة التعرف على عتاد الكمبيوتر وإرفاق مشغلاتها، فهو يوفر لك إمكانية الطباعة في برنامجك دون الحاجة لمعرفة نوعية الطابعة، ويمكنك من استخدام الصور والرسوم أو عزف ملفات الصوت أو استخدام الفأرة في برنامجك دون الالتزام بإرفاق مشغلات الأجهزة، أي كل ما هو مطلوب منك -كمبرمج- التركيز على برنامجك وصرف النظر عن الأمور التقنية الدنيا كالأجهزة والعتاد، إدارة الذاكرة، إدارة الأقراص وغيرها، والتي يتكفل بها نظام التشغيل بكل اقتدار.

PDF

إلا أن البرمجة تحت بيئة Windows تختلف اختلافا جذرياً عن البرمجة تحت بيئة DOS، فبرنامجك لم يعد يستخدم الطرق التقليدية لقنص المدخلات وعرض المخرجات، فقنص المدخلات أصبحت تتم من قبل نظام التشغيل، والذي يقوم بإرسالها لك على شكل **رسائل Messages** كالنقر Click، الضغط على زر KeyDown ...الخ. لذلك، انقلبت الموازين البرمجية في حياة معظم المبرمجين، لتصبح برامجهم تحتوي على عشرات -بل مئات- الحلقات التكرارية لقنص هذه الرسائل.

أما من ناحية عرض المخرجات، فلم يعد هناك شيء اسمه Print لإظهار المخرجات على الشاشة، حيث يتطلب نظام التشغيل Windows من المبرمجين إنشاء نوافذ وسياقات رسم وتسجيل طبقات ليتمكنوا من عرض المخرجات من خلالها. فلو أراد مبرمج تعلم لغة برمجة جديدة لكتابة أول برنامج شهير Hello World تحت بيئة Windows، عليه كتابة عشرات السطور المعقدة جدا لعمل ذلك، الأمر الذي أدى تقاعد المبرمجين من عالم البرمجة وفضلوا العمل عند شركات سيارات الأجرة (التاكسي)! وحتى اريك الأمر الواقع، انظر إلى هذه الشيفرة المعدة بلغة C والتي تمثل برنامج يقوم بعرض نافذة خالية فقط:

```
#include <windows.h>

LRESULT CALLBACK MainWndProc( HWND, UINT, WPARAM, LPARAM );

HINSTANCE ghInstance;

int PASCAL WinMain( HINSTANCE hInstance,
    HINSTANCE hPrevInstance,
    LPSTR lpszCmdLine,
    int nCmdShow )
}
    WNDCLASS wc;
    MSG msg;
    HWND hWnd;

    if( !hPrevInstance )
    {
        wc.lpszClassName = "ShowWindow";
        wc.lpfnWndProc = MainWndProc;
        wc.style = CS_OWNDC | CS_VREDRAW | CS_HREDRAW;
        wc.hInstance = hInstance;
        wc.hIcon = LoadIcon( NULL, IDI_APPLICATION );
        wc.hCursor = LoadCursor( NULL, IDC_ARROW );
        wc.hbrBackground = (HBRUSH)( COLOR_WINDOW+1 );
        wc.cbClsExtra = 0;
        wc.cbWndExtra = 0;
```

```
        RegisterClass( &wc );
    }

    ghInstance = hInstance;

    hWnd = CreateWindow ( "ShowWindow",
        "ShowWindow",
        WS_OVERLAPPEDWINDOW|WS_HSCROLL|WS_VSCROLL,
        0,
        0,
        600,
        300,
        NULL,
        NULL,
        hInstance,
        NULL);

    ShowWindow( hWnd, nCmdShow );


    while( GetMessage( &msg, NULL, 0, 0 )) {
        TranslateMessage( &msg );
        DispatchMessage( &msg );
    }

    return (int) msg.wParam;
}
LRESULT CALLBACK MainWndProc( HWND hWnd, UINT msg, WPARAM wParam,
    LPARAM lParam )
{
    PAINTSTRUCT ps;
    HDC hDC;

    switch( msg ) {

    case WM_DESTROY:
        PostQuitMessage( 0 );
        Break;

    default:
        return( DefWindowProc( hWnd, msg, wParam, lParam ));
    }

    return 0;
}
```

ولو أنهم صبروا لكان خيرا لهم، فبعد فترة ليست طويلة ظهرت حلول من كبريات شركات صناعة البرمجيات لتسهيل عملية البرمجة تحت نظم Windows، وذلك باختراع الكلمة السحرية **Visual**، فكل ما هو مطلوب من المبرمج تصميم شاشات (نوافذ) برنامجه بالفأرة، وكتابة بضعة

أوامر يتم تنفيذها بمجرد قيام المستخدم بالتفاعل مع برنامجه سواء بالفأرة أو لوحة المفاتيح أو انفه (لقد شاهدت فعلا احد الزملاء يستخدم انفه على الشاشة!).

بعد ذلك، لاحظ المبرمجون أن نسبة كبيرة من شيفرات برامجهم مكررة وقد كتبت في عشرات المشاريع، فلو أمعنت النظر قليلاً، لوجدت ان معظم تطبيقات Windows تتشارك إلى حد كبير في معظم وظائفها الشائعة، لذلك كان على مطوري نظام Windows أيجاد حلول لتبادل البيانات ومشاركة الشيفرات بين البرامج، إلا أن تحقيق هذا الهدف بدا مستحيلاً لبعض الوقت، لان جميع برامج Windows تعمل في مناطق مختلفة ومستقلة بها في الذاكرة تسمى **مساحات العنونة Address Spaces**، لذلك أسس مطورو Windows أسلوبا أو بروتوكول برمجي يسمح للتطبيقات بالتخاطب فيما بينها بمعايير ومواصفات قياسية يسمى **التبادل الديناميكي للبيانات Dynamic Data Exchange - (DDE)**.

إلا أن DDE كانت بها الكثير من العيوب التي حدت بالمبرمجين إلى تجنب استخدامها، ككثرة الانهيارات التي تحدث في البرامج، والاتصالات دائمة الانقطاع بين التطبيقات، بالإضافة إلى صعوبة وتعقيدات الشيفرة المصدرية وغيرها، إلى أن قامت Microsoft بإصدار تقنية **ربط الكائنات وتضمينها Object Linking & Embedding - (OLE)** والتي تعتمد في بنيتها التحتية على DDE، حيث وفرت قابلية لتبادل البيانات بين البرامج والتطبيقات المختلفة لتمكنك – مثلا- من إدراج جدول من Microsoft Excel لتضمينه أو ربطه في مستند Microsoft Word.

في أواخر عام 1993 غيرت Microsoft البنية التحتية لـ OLE حيث لم تعد تعتمد على DDE وتم إعادة بنائها من جديد لتصدر ما سمي **OLE2**، والتي مكنت المبرمجين من تطبيق أسلوب **العمل في نفس المكان In-place Activation** بحيث يمكنك تحرير جدول Excel وأنت بداخل مستند Word في نفس النافذة ودون الحاجة لمغادرة Word.

الحلم أصبح حقيقة مع COM

من الإنجازات التي أحدثت ثورة كبيرة في عالم تطوير البرامج تحت Windows، تقنية **برمجة المكونات Component Object Model - (COM)**، حيث مكنت هذه التقنية المبرمجين – بلغات البرمجة المختلفة- من المشاركة في تطبيقاتهم بأسلوب **كائني التوجه Object Oriented**. ليس هذا فقط، بل تعدى الأمر أكثر من ذلك ليصل إلى **المكونات الموزعة Distributed COM - (DCOM)**، لتصبح مكونات البرامج موزعة على أجهزة مختلفة، ويتم تبادل البيانات عن طريق شبكة الانترنت بشكل مذهل، فيستطيع صديقي في منغوليا من

PDF Factory Pro trial version www.fastio.com

استخدام بعض أجزاء برنامجي الموجود في جهازي المحمول الذي اصطحبه معي في رحلاتي البرية بصحراء الربع الخالي.

أثرت COM بشكل إيجابي كبير في عالم تطوير البرامج تحت Windows، لدرجة ظهور شركات متخصصة فقط في تطوير مكونات COM (كأدوات التحكم ActiveX Controls، مكتبات فئات ActiveX DLL ....الخ)، وأصبحت عملية بناء البرامج تعتمد على **البرمجة مكونية التوجه Component Oriented Programming** بشكل كبير، ولا تكاد تجد أي برنامج الآن إلا ويستخدم مكونات COM.

مع ذلك، فإن استيعاب البنية التحتية لبرمجة المكونات COM مسالة صعبة جدا، فهي تتطلب التوغل في تفاصيل معقدة لاستخدام ما يسمى **الواجهات Interfaces**، وكثرة الأخطاء والشوائب البرمجية أصبحت أمراً طبيعياً، وعند الحديث عن مصادر النظام System Resources فحدث ولا حرج، فهي تستهلك الكثير من المساحات الغير مستخدمة لعدم تفريغ أجزاء الذاكرة من الكائنات المنشأة، إما بسبب الانهيارات المفاجئة للبرامج، أو نسيان حذف مؤشرات الكائنات التي انشأها أو استخدمها البرنامج.

من ناحية أخرى، فإن مكونات COM تعتمد اعتماداً كلياً على **مسجل النظام Windows Registry**، وأي مشكلة تحدث في هذا المسجل تؤثر على باقي المكونات المثبتة في الجهاز، ولن تستطيع استخدامها إلا بإعادة تركيب Reinstall البرامج التابعة لها. وعملية تركيب البرامج بحد ذاتها معقدة جدا، إذ تتطلب منك نسخ ملفات المكونات ومن ثم تسجيلها في المسجل وإعدادها والتحقق من الإصدارات الأقدم ومن ثم تعريفها على الشبكة المحلية (إن كانت DCOM)، وأي خطأ في عملية تثبيت البرامج، يؤدي إلى حدوث كارثة في جهاز المستخدم والتأثير على باقي البرامج المثبتة في الجهاز، ليكون الحل الوحيد إعادة تهيئة Format القرص الصلب. وعند الحديث عن التوافقية، فلا يمكنك استخدام إصدارين مختلفين لنفس المكون بسبب مشكلة تسمى **Versioning** ليس هذا مجال تفصيلها.

تحديات الانترنت

مع ظهور الانترنت أصبحت مسألة تكامل التطبيقات مع هذه الشبكة أمر ضروري إن لم يكن إلزامي، فعمليات تحديث البرامج وتبادل البيانات فيما بينها يمكن أن تُخفض تكاليفها عن طريق الانترنت، أضف إلى ذلك مدى السهولة التي تمكن مستخدمين الكمبيوتر –حول العالم- من الوصول إلى المعلومات بالتصفح في المواقع المختلفة. هذه الشبكة لم يحصر مجالها في عرض

PDF Factory Pro trial version www.fineprint.com

المواقع فقط، بل تعدى الأمر أكثر من ذلك إلى أن يصل لمستوى تطوير تطبيقات ويب باستخدام تقنيات معالجة الصفحات على أجهزة الخوادم كتقنية **Active Server Pages - (ASP)**.

مع ذلك، فإن تطوير النظم الكبيرة باستخدام ASP أمر معقد جدا، ويتطلب عشرات التنقيحات والتعديلات للصفحة الواحدة، خاصة إن علمت أن برمجة ASP ليست كائنية توجه OOP، حيث أنها كانت تعتمد على لغة البرمجة المصغرة VBScript لكتابة شيفرات الصفحات.

إلا أن دعم صفحات ASP لبرمجة المكونات COM أدى إلى تكامل حقيقي وتسهيل اكبر ومرونة أكثر في تطوير النظم –خاصة الكبيرة، ليتمكن المستخدمون من تطوير تطبيقات متعددة الطبقات nTied Applications حقيقية، تعتمد على كائنات ADO كخلفية النظام لإدارة قواعد البيانات، ومكونات COM لأداء وظائف النظام كطبقة وسطى، وصفحات ASP لتمثل واجهة المستخدمين.

انتشرت نظم ASP انتشارا كبيرا بين المواقع العالمية، إلا أن المشكلة التي واجهت مطوري نظم ويب تظهر عند تبادل وتشارك البيانات بين المواقع المختلفة، فمثلا لو كنت تصمم موقع لبيع المادة الخام المستخدمة في إنتاج الفلافل (الطعمية) وأردت من هذا النظام ان يستقبل الطلبات من مختلف الدول العربية، وعلى الأرجح ستتم عملية الشراء بالعملة المحلية للدولة، أضف إلى ذلك مسألة التسعير حيث أنها تعتمد اعتماداً كلياً على القوة الشرائية لعملة تلك الدولة، وسوق المطاعم المنافسة في إنتاج سندويتشات الفلافل، لذلك قد تحتاج إلى تبادل البيانات مع مواقع أخرى تحدد لك السعر المناسب في وقت معين، ومواقع أخرى تمكن المستخدمين من مراقبة عمليات شحن الفلافل وبيان مواقعها الحالية.

إن أردت تبادل البيانات مع هذه المواقع بشكل تقليدي، فالعملية تتم بارسال صفحات HTML من موقع إلى آخر، وعليك القيام بجميع الخطوات اللازمة لحذف الوسوم Tags (التي تستخدم لتنسيق صفحات HTML)، وهذه بحد ذاتها عملية معقدة جدا، وتتطلب جهد ووقت إضافي. ليس هذا فقط، بل أن أي تعديل بسيط في صفحات احد المواقع قد يؤدي إلى كارثة في عملية نقل البيانات وعدم دقتها وقد يصل الأمر إلى أن تكون الطلبية فلافل ويتم شحن سندويتشات كبدة بلدي!

لذلك اعتمد المطورون على تقنية **(SOAP) – Simple Object Access Protocol** بحيث يتم نقل البيانات عن طريق لغة XML (وهي لغة وصف البيانات وليس تنسيق البيانات كـ HTML) واستخدام بروتوكولات الاتصال TCP/IP.

قد تكون فكرة SOAP ممتعة جدا، ولكن -مع الأسف الشديد- استخدام هذه التقنية معقد جدا، ويتطلب الكثير من الخبرة البرمجية، بحيث انحصر استخدامها على المبرمجين المحترفين

PDF Factory Pro trial version www.fastio.com

فقط، كذلك كثرة الأخطاء والشوائب البرمجية Bugs في تطوير نظم تعتمد على SOAP أمـــر لا مفر منه.

عشرات التقنيات لأداء الوظائف

كما رأيت في الفقرات السابقة، فإن تطوير البرامج مسألة معقدة جدا وتتطلب درايـــة كافيـــة فـــي التعامل مع التقنيات المختلفة، فلكي تطور مواقع ويب ديناميكية عليك تعلم VBScript (إن كانت من جهة العميل) وتعلم ASP (إن كانت من جهة الخادم)، وان أردت بناء نظـــم قواعـــد بيانـــات عملاقة عليك إتقان لغات الاستعلام المتقدمة كـــ T-SQL للحصول على اكبر قدر مـــن تحـــسين للكفاءة Optimization، وان أردت تطوير مكونات COM بفاعلية أكثر ودون حدود عليك تعلم احد لغات البرمجة المتقدمة كـــــ Visual C++، وان أردت مخاطبـــة تطبيقـــات Microsoft Office الشهيرة فلا مخرج لك إلا باستخدام VBA، أما إن أردت تطوير برامج تعمل تحت نظم Windows بسهولة وكسر حاجز الوقت فستجد ضالتك في Visual Basic.

ليس هذا فقط، بل حتى الوظائف المتشابهة تنجز بتقنيات مختلفة، فلـــديك مـــثلا التقنيـــات ADO، DAO، وRDO لتطوير التطبيقات المعتمدة على قواعد البيانات Databases، وهنـــاك أيضا مجموعة من التقنيات كـــ GDI، DirectX، و OpenGL لتطوير النظم التي تعتمد علـــى الصور والرسوم بكثرة.

كانت هذه جولة سريعة حول بناء البرامج في السنوات السابقة، تذكر أن كلامـــي موجـــه لتقنيات Microsoft فقط. وان لم تكن لديك أي معرفة بما سبق ذكره من تقنيات برمجية، فأرجو أن لا يصيبك ذلك بالإحباط، بل على العكس من ذلك، يمكنك أن تعتبر نفسك مبرمج محظوظ جدا، حيث ستبدأ حياتك الجديدة من حيث انتهى الآخرون، ومع أحدث وأفضل تقنية تـــستخدم لتطـــوير التطبيقات في القرن الحادي والعشرين وهي Microsoft .NET.

الحياة بعد .NET

لا ادعي هنا علم الغيب ووصف مستقبل تطوير التطبيقات، ولكن يمكنك اعتبار ما ســـأذكره فـــي الفقرات التالية عرض سريع لاستراتيجية NET. وبيان مدى تأثيرها على صناعة البرمجيات.

الاستقلالية عن منصات العمل

اكتب البرنامج مرة واحدة فقط وسيتم تنفيذه على مختلف منصات العمل المختلفة: كالأجهزة المحمولة Notebooks، خادمات Servers، هواتف جوالة Mobiles، تليفزيونات رقمية Digital TVs، ثلاجات، طائرات، أبواب كراج، سيارات، وكل شيء رقمي Digital. وان كنت شخص تسكن في منطقة بعيدة عن عائلتك ولا تجيد الطبخ، فيمكنك طلب شيفرة مصدرية من الوالدة لكتابة برنامج لتحظير الكبسة ومن ثم تركيبه في الفرن (الذي سيكون رقمي لاحقا) لإنجاز الكبسة. وهذا بفضل استقلالية برامجك عن منصات العمل الذي تقدمه NET..

الاستقلالية عن منصات العمل لا تنحصر حول العتاد Hardware فقط، بل تشمل نظم التشغيل المختلفة، فحاليا برامجك يمكنها أن تعمل على مختلف إصدارات نظام التشغيل Windows، وقريبا قد نرى ان إطار عمل NET Framework. سيدعم في أنظمة التشغيل الأخرى كـ ®Linux وحتى ®Macintosh.

النتيجة الإيجابية من استقلالية برامجك عن منصات العمل تقتضي منك –كمبرمج- التركيز على برامجك فقط وصرف النظر عن العالم الخارجي أو المكان الذي سيتم تنفيذ البرنامج فيه، مع ذلك قد تسأل سؤال بديهي وتقول: ان كانت البرامج مستقلة عن منصات العمل، فكيف سيتم تنفيذ الوظائف المختلفة والتي لا تتوفر في منصة عمل معينة، والجواب بكل بساطة يعتمد على نوعية البرنامج الذي تصممه، فبكل تأكيد الشيفرة المصدرية لبرنامج الوالدة (تحضير الكبسة) لن تقوم بتشغيله في غسالة الملابس (التي ستكون رقمية أيضا)، وإنما موجهه إلى الفرن الرقمي. لذلك أريد توضيح أن المقصد من قضية استقلالية البرنامج عن منصات العمل ميزة من إطار عمل NET. Framework وليس للمبرمج أي علاقة مباشرة بها، فكل ما هو مطلوب منه كتابة البرنامج فقط بحيث يلائم البيئة التي سيعمل بها.

NET. نسخة محسنة من COM

قد تُفاجأ إن أخبرتك أن الاسم الابتدائي لمشروع NET. كان يسمى COM 2.0، أي الجيل التالي من برمجة المكونات COM, وهذه بحد ذاتها حقيقة إن أخذتها بشكل نظري. فالفكرة من COM و NET. تقريبا متطابقة –نظريا- من منطلق توزيع الشيفرات والاستقلالية الشبه تامة عن منصات العمل، إلا أن NET. تختلف اختلافاً جوهرياً كبيراً في بنيتها التحتية عن COM، حيث أن تقنية NET. تم إعادة بنائها من جديد وعولجت العشرات من المشاكل التي واجهت مبرمجي COM سابقا.

PDF

أول مشكلة ابتدائية تم حلها هي الاستغناء عن مسجل النظام System Registry، حيث أن مكونات NET. تصل إليها وتستعلم عنها مباشرة عن طريق ملفاتها، دون الحاجة إلى المرور بمسجل النظام كما كنا نفعل سابقا مع COM، وهذا يعني أن عملية تثبيت البرامج لا تتطلب جهد إضافي لإنجازها، فيكفي نسخ الملفات من القرص المدمج إلى القرص الصلب وسيعمل البرنامج دون أية مشاكل، مع ذلك قد تحتاج إلى برامج التركيب لتنفيذ بعض اللمسات الخفيفة (كوضع الملفات في أماكنها المناسبة، تخصيص العناصر المطلوب تثبيتها، اعدادات بسيطة قبيل عملية تنفيذ البرنامج ....الخ).

وبالنسبة للمكونات الموزعة DCOM فلن تحتاج إلى العبث في نظام التشغيل Windows ومحتويات المكون لتجري عشرات الاعدادات الإضافية حتى يتم توزيعه، إذ أن مكونات NET. هي موزعة بحد ذاتها.

أما مشكلة التوافقية Versioning فلن تحدث بعد الآن، حيث يمكن تثبيت اصدارين مختلفين من نفس المكون دون أن يؤثر احدهما على الآخر.

ميزة عظيمة أخرى في مكونات NET. لم تكن مدعومة سابقا مع مكونات COM وهي **الوراثة Inheritance**، فمكونات COM لم يكن متاحاً اشتقاقها وراثيا وتطوير فئاتها، أما مكونات NET. فلديك القدرة الكاملة لاشتقاق فئات المكونات وراثيا دون الحاجة للحصول على شيفراتها المصدرية.

انظر أيضا

الوراثة والاشتقاق الوراثي مواضيع دسمة، سأتحدث عنها لاحقا في الفصل الرابع **الوراثة** .

صحيح أن مكونات COM كانت تزيل حاجز الفروقات بين لغات البرمجة المختلفة، إلا أن هذا الحاجز لم يتم إزالته بشكل كامل، فما زال مبرمجو بعض لغات البرمجة (كـ Visual Basic) يواجهون مشاكل وصعوبات في استخدام بعض مكونات COM المنجزة بلغات متقدمة أخرى (كـ Visual C++) خاصة مع المكونات التي تتعامل مع أنواع بيانات ليست مدعومة في Visual Basic (كالمؤشرات Pointers مثلا)، ولكن مع مكونات NET. أمست كل هذه التعارضات من الماضي، ومرد ذلك ان جميع لغات NET. موحدة بفضل معايير CRL كما سترى لاحقا.

PDF

تكامل لغات البرمجة

جميع لغات NET. متكاملة فيما بينها، فبرنامجك المصمم بـــ Visual Basic .NET يمكن إضافة بعض العناصر والشيفرات المصدرية إليه من لغة Visual C# .NET دون أي مشاكل، بل يمكن للمشروع الواحد أن يدمج شيفرات مصدرية من لغات متعـــددة مثـــل: Delphi .NET، Java .NET، Visual C++ .NET، Turki .NET (إن وجدت)، Fortran .NET، ...الخ وذلـــك بفضل معايير CRL التي توحد لغات البرمجة.

قد تتساءل وتقول، ما دامت لغات البرمجة المختلفة موحدة بهذا الشكل فما الفائدة من تعلـــم أكثر من لغة؟ والجواب هو انه ما زالت كل لغة برمجة تحتوي على سمات ومميزات خاصة بها، واعني بكلمة خاصة بها في هذا السياق هو عدم إمكانية تكاملها مع لغات NET. الأخرى أن تـــم تفعيل هذه المزايا. من ناحية أخرى، جميع لغات NET. يتم تحويلها إلـــى لغـــة MSIL لحظـــة الترجمة Compiling كما سترى لاحقا.

خدمات ويب هي مستقبل الانترنت

منذ أن نشبك سلك الهاتف ونتصل بالانترنت، فإن معظم وقتنا نقضيه على المتـــصفح Browser للوصول إلى المواقع المختلفة، الفعل السابق سيكون من أساطير الأولين في المـــستقبل القريـــب، وذلك بعد انتشار **خدمات ويب Web Services** حيث ستغير الكثير من أسلوب تعاملنا مع شبكة الانترنت بطرق لم تخطر على بال انس ولا جان. الجزء الخامس **برمجة ويب** من هذا الكتاب يقدم لك شرحاً وافياً حول خدمات ويب، ولكن دعني هنا اعرف لك ما هي خدمة ويب Web Service بشكل سريع.

خدمة ويب (تسمى احياناً XML Web Service) ما هي إلا برنـــامج يـــستقبل طلبـــات Requests ومن ثم يستجيب لها Response باستخدام بروتوكول HTTP تحـــت معـــايير لغـــة XML القياسية، وبذلك تتمكن ملايين المواقع المنتشرة حول العالم من تبادل البيانات فيما بينهـــا وإنجاز الأعمال المطلوبة. يمكنك اعتبار خدمات ويب على أنها برنامج يتصل بالعالم الخـــارجي عن طريق شبكة الانترنت مما يمكنك من إرسال واستقبال البيانات بصيغة XML دون أن يتطلب الأمر منك أي خبرة في لغة XML أو بروتوكولات TCP/IP، حيث يوفر لك إطار عمل NET. Framework كل ما تحتاجه لتطوير وبناء خدمات ويب.

ستنتشر خدمات ويب انتشاراً لا مثيل له في مواقع الانترنت، (سنوفر خدمة ويب في موقع **شبكة المطورون العرب** بمشيئة الله)، بل ستصل إلى تغيير معظم الأنظمة الحالية لتعتمـــد علـــى خدمات ويب وتصبح مؤجرة، فيمكنك مثلا توفير خدمة ويب للتذكير بالمواعيد، أو خدمة تـــوفير

أسعار العملات التي تحدث على مدار الساعة، أو خدمة الحصول على دروس ومقالات من موقع آخر، أو خدمة بحث كخدمة Goggle Search ...الخ.

ماذا عن المبرمج العربي؟

إطار عمل NET Framework. موجه إلى جميع المبرمجين حول العالم باختلاف لغاتهم الطبيعية، واللغة العربية إحدى هذه اللغات، فالنصوص مثلا جميعها تتبع الترميز UNICODE وتوزيع محارف لوحات المفاتيح العربية مأخوذ في عين الاعتبار، كذلك الحال مع مواصفات البيئة كتنسيق العملة، الوقت والتاريخ، الأرقام، نظام الفرز ...الخ، فهي مدعومة لجميع الدول العربية باستثناء -مع الأسف الشديد- السودان، فلسطين، جيبوتي، الصومال وموريتانيا فلم اجدها في احد روابط مستندات NET Documentation. الرسمية.

وعند الحديث عن المسائل التقنية الأخرى، فتقنية **المرآة Mirroring** مدعومة بشكل جيد في نماذج Windows Forms (كما سترى في الجزء الثالث **تطوير تطبيقات Windows**)، وبالنسبة للتاريخ الهجري فيوفر لك إطار عمل NET Framework. الفئة HijriCalendarClass التي تمكنك من استخدام التاريخ الهجري في برامجك وتوفر لك طرق وخصائص لتعديل وضبط القيمة الصحيحة لليوم والشهر.

انظر أيضا

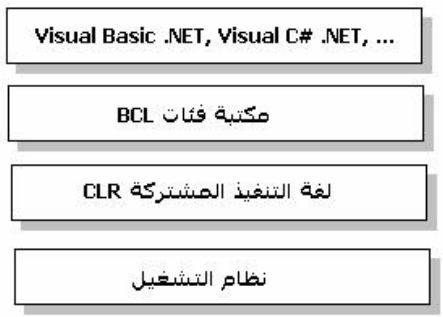
سترى تطبيقات وأمثلة حول استخدام الفئة HijriCalendarClass في الفصل السادس **الفئات الأساسية**.

باختصار، وفرت Microsoft منصة تطوير قوية تدعم اللغة العربية بشكل رائع، لتنحصر المسؤولية علينا نحن كعرب –سواء في شركات عربية أو مطورون عرب- لتقديم كافة الحلول الفعالة للمستخدم العربي.

PDF

محتويات إطار العمل .NET Framework

والآن سيتمحور حديثي حول معمارية ومحتويات إطار عمل .NET Framework.، يمكنني ان اقسم لك محتوياته إلى أكثر من 10 طبقات، ولكني فضلت تقليص العدد –للتسهيل عليك- كما ترى في (**الشكل 1-1**):



شكل 1-1: محتويات إطار عمل .NET Framework.

مكتبة فئات Base Class Library (تسمى أيضا مكتبة فئات .NET Framework Class Library) هي عبارة عن مئات الفئات الموجودة في عشرات ملفات DLL تعتبر كنزا غاليا يسيل له لعاب المبرمجين الجادين، حيث تحتوي كل ما تحتاجه لإنجاز برامجك ومشاريعك بدءاً بتقديم فئات لوصف البيانات الأساسية (كـ String، Integer) إلى إدارة خرج ودخل الملفات I/O File Processing، مسارات التنفيذ Threading، الصور والرسوم، نماذج Windows Forms، نماذج Web Forms، الاتصال بقواعد البيانات ADO .NET وغيرها الكثير.

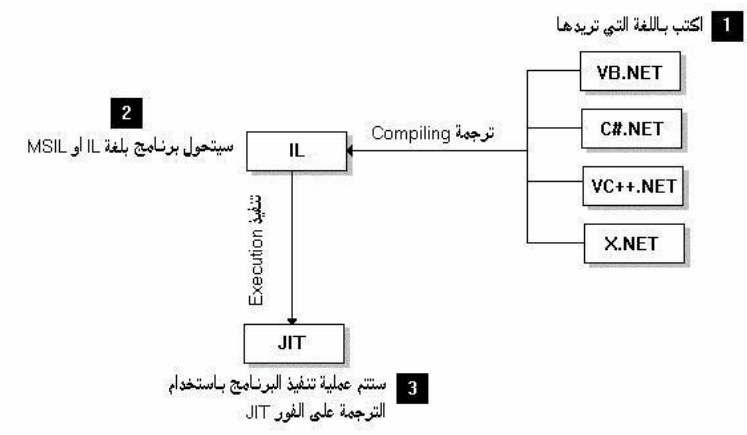
بالنسبة **للغة التنفيذ المشتركة Common Language Runtime – (CLR)** فهي موحدة لمعايير جميع لغات .NET. الأخرى، كما أنها المسئولة عن عمليات إدارة الذاكرة Memory Management، تفريغ مصادر النظام باستخدام Garbage Collection، أخطاء وقت التنفيذ Exception Handling (سأتحدث عن كل ما ذكرته في الفصول اللاحقة من هذا الكتاب).

أخيرا، نظام التشغيل الذي يمكنك من تثبيت إطار عمل .NET Framework. عليه هو Windows فقط (لحظة كتابة هذه السطور)، ولكن قد ترى في القريب العاجل نظم تشغيل أخرى داعمة له.

PDF Factory Pro trial version www.fastio.com

الترجمة على الفور JIT

من أجمل الأشياء التي ستكتشفها والتي تعتبر فتح كبير -في رأيي الشخصي- في عالم برمجة NET. هو أسلوب **الترجمة على الفور Just In Time Compiling - (JIT)**، وهي تقنية تقوم بترجمة البرنامج عند تنفيذه حيث ينتج أفضل شيفرة تتناسب مع الجهاز الذي سيعمل عليه البرنامج مما ينتج عنه نتائج إيجابية جيدة جدا (هذا عند الحديث عن تحسين الكفاءة Optimization) وحتى تعلم كيف يحدث ذلك تابع (**الشكل 1-2**):



شكل 1-2: مراحل ترجمة وتنفيذ البرنامج.

اختر أي لغة تناسب مزاج أناملك واكتب الشيفرة بها، وعند قيامك بعملية الترجمة سيتم تحويل ملف البرنامج إلى ملف شبيه بالملفات التنفيذية Executable File مكتوب بلغة جديدة اسمها **Microsoft Intermediate Language (تختصر IL أو MSIL)**. حيث تحتوي على شيفرات البرنامج ولكنها غير قابلة للتنفيذ مباشرة، بل يشترط وجود مترجم على الفور JIT Compiler حيث يقوم بترجمة هذا الملف الثنائي إلى لغة الآلة معطيا أفضل التعليمات بحيث تناسب الجهاز الحالي، فعندما تصمم برنامجك مرة واحدة فقط باستخدام Visual Basic .NET فاعلم ان البرنامج سيستفيد من كل مصادر العتاد ونظام التشغيل الذي يتم تنفيذ البرنامج فيه، أي لو شغلت برنامج تحت خادم Server يحتوي على 9504378250470592 معالج فثق ثقة تامة ان برنامجك سيستفيد من كل هذه المعالجات رغم انك لم تهتم بهذه النقطة على الأرجح.

PDF Factory Pro trial version www.fineprint.com

عملية الترجمة JIT تقوم بترجمة كل جزء من البرنامج عند الحاجة أي عند استدعاء وظيفة معينه فيه، ويمكنك ترجمة البرنامج من أوله إلى اخره أيضا عن طريق مترجم آخر يسمى Native Image Generator – (NGEN) (يسمى أيضا Pre-JIT Compiler) مع معرفة أن الترجمة تتم مرة واحدة فقط، ولن يشعر المستخدم بأي بطء في عملية تنفيذ البرنامج عند تشغيله مرة أخرى، فقد تمت ترجمته بالشكل المناسب لجهازه. (راجع مستندات .NET Documentation لمزيد من التفاصيل حول المترجم NGEN).

المجمعات Assemblies

المجمع Assembly ما هو إلا ملف (قد يكون EXE أو DLL) يحتوي على كل شيء يتعلق بالبرنامج سواء شيفراته المصدرية بعد الترجمة Compiling، الصور والرسوم، ملفات المصادر Resource Files، صفحات HTML، وغيرها. كل هذه العناصر يمكنك أن تضعها جميعا في ملف واحد فقط.

في اغلب الأحوال، يمثل المجمع برنامج واحد، مع ذلك يمكن للمجمعات Assemblies أن تحتوي على مجمعات أخرى، أي -بعبارة أخرى- يمكن لبرنامجك أن يدمج في داخله برنامج آخر -كما سترى لاحقا في الفصل الحادي عشر **المجمعات Assemblies**.

أخيرا، كل ما ذكرته من جمل وعبارات مبهمة في الفقرات السابقة، سأعود للحديث عنها بالتفصيل الممل في باقي فصول هذا الكتاب، فلست بحاجة لان تكون مستوعبا لكل شيء الآن، إذ أن كل ما كنت أريده الآن هو تقديم جولة سريعة حول محتويات إطار عمل .NET Framework.

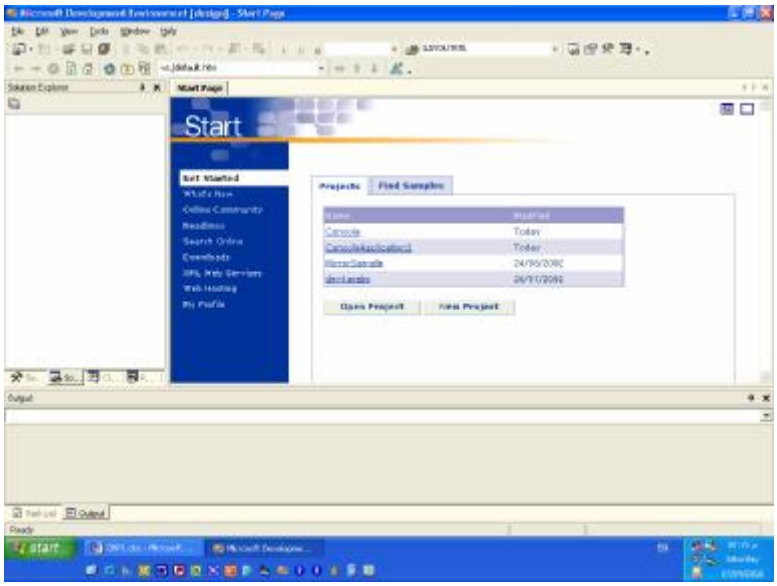
بيئة التطوير Visual Studio .NET

قد أغير العرف المتبع في كتب البرمجة (والتي تبدأ بشرح بيئة التطوير بالتفصيل)، ولكن من منطلق إيماني الشديد بان الشخص –الذي يقرأ كتابي الآن- هو مبرمج وليس مستخدم، فاعتقد انه قد وصل إلى مرحلة تمكنه من تعليم نفسه ذاتيا لاستخدام بيئة التطوير. لذلك، التمس منك العذر الشديد يا عزيزي القارئ في مسالة شرح بيئة التطوير، فلن تجد هنا إلا عرض سريع ومختصر لبعض محتويات البيئة. مع ذلك، قد أتطرق في فصول لاحقة إلى تفصيل بعض النوافذ متى ما دعت الحاجة لذلك.

تتشارك معظم لغات البرمجـــة (Visual Basic .NET، Visual C#.NET، Visual C++ .NET وغيرها) في بيئة تطوير متكاملة من Microsoft تسمى Visual Studio. NET، حيث توفر لك كل ما تحتاجه من خدمات وأدوات في قمة الروعة تسهل لك حياتك البرمجية. شرح جميع محتويات Visual Studio .NET يتطلب –دون مبالغة- كتاب كامل، وقد أصدرت بالفعل مطابع Microsoft Press كتابا يشرح كل صغيرة وكبيرة حول هذه البيئة، أما الكتاب الذي تقرأه فهو يتعلق بلغة البرمجة Visual Basic .NET فقط، ولـــن اذكـــر إلا النوافـــذ والأدوات التـــي ستستخدمها باختصار هنا.

نوافذ بيئة التطوير

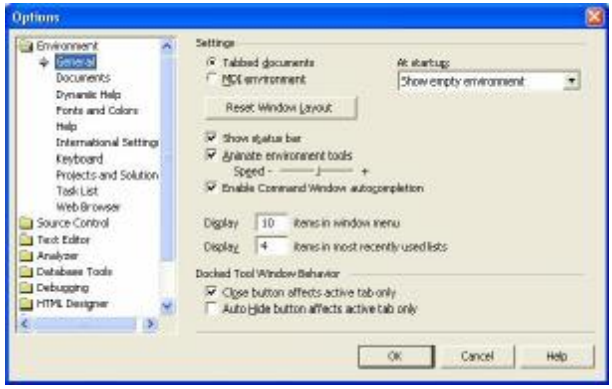
عند تشغيل بيئة Visual Studio .NET لأول مرة، مستظهر لك النافذة Start Page (**شكل 1-3**)، تستطيع الوصول إلى آخر مشاريع تم فتحها أو إنشاء مشاريع جديدة عن طريق هذه النافـــذة، كما انك تستطيع الوصول إلى آخر الأخبار المتعلقة بـــ Framework .NET. بالـــضغط علـــى الرابط Headlines وذلك لان النافذة Start Page ما هي إلا صفحة ويب تقليدية.



شكل 1-3: الشاشة الافتتاحية لبيئة التطوير Visual Studio .NET.

نافذة الخيارات Options:

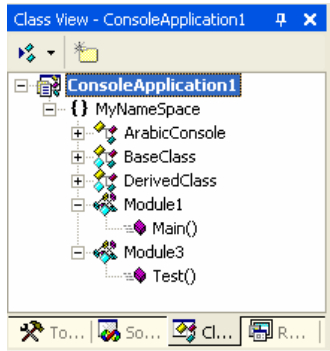
اختيارك للأمر Options من قائمة Tools يؤدي إلى ظهور هذه النافذة (**الشكل 1-4**)، حيث تمكنك من تخصيص وإعداد عشرات الأوضاع والخيارات كاعدادات بيئة التطوير، محرر الشيفرات، المترجمات Compilers، قواعد البيانات Database، التنقيح Debugging، محرر صفحات HTML، مصمم نماذج Windows Forms وغيرها.



شكل 1-4: نافذة الخيارات Options.

نافذة عرض الفئات Class View:

الغرض الرئيسي من هذه النافذة هو عرض فئات البرنامج Project Classes على شكل شجري (**شكل 1-5**)، مع العلم أن الفئات التي تعرض في هذه النافذة هي الفئات التي تعرفها في الشيفرة المصدرية للمشروع الحالي فقط، فلا تتوقع ظهور فئات من مكتبات أخرى لم يتم تضمينها في نفس المشروع.

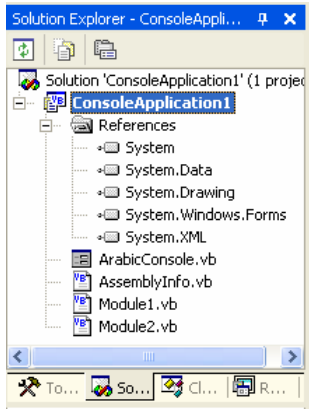


شكل 1-5: نافذة عرض الفئات Class View

الضغط المزدوج على عنصر من عناصر هذه الشجرة، يؤدي إلى فتح نافذة المحرر ونقل مؤشر الكتابة Cursor إلى منطقة تعريف الفئة أو العضو في الفئة. اختر الأمر Class View من قائمة View لعرض هذه النافذة.

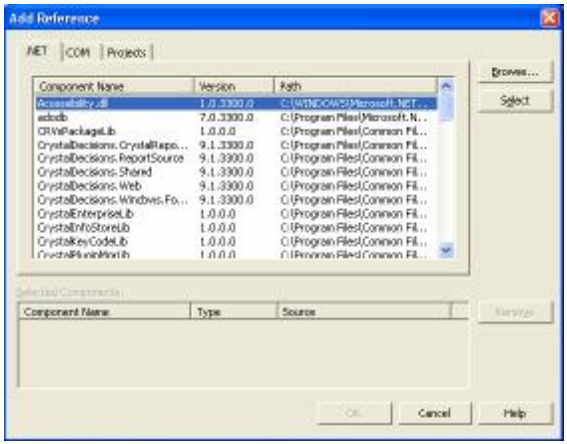
نافذة مستكشف الحل Solution Explorer:

إن كانت نافذة عرض الفئات السابقة تعرض فئات المشروع الحالي، فان نافذة مستكشف الحل تعرض ملفات المشروع الحالي والمشاريع الأخرى (**شكل 1-6**). اختر الأمر Solution Explorer من قائمة View لعرض هذه النافذة.



شكل 1-6: نافذة مستكشف الحل Solution Explorer

في أعلى شجرة ملفات المشروع تلاحظ وجود عنصر المراجع References، يقصد بهذه المراجع المكتبات الخارجية التي تريد تضمينها في المشروع الحالي والوصول إلى فئاتها، انقر بزر الفأرة الأيمن على هذا العنصر واختر الأمر Add Reference من القائمة المنبثقة، لتظهر لك نافذة المراجع (**شكل 1-7**).



شكل 1-7: نافذة المراجع Reference.

مع العلم انه يمكنك إضافة أو حذف المراجع من هذه النافذة.

نافذة خصائص المشروع Project Property Pages:

عند البدء في إنشاء مشروع جديد، ينصح دائما بتعديل اعدادات المشروع أولا عن طريق هذه النافذة، تستطيع الوصول لها بالنقر بزر الفأرة الأيمن على عنصر المشروع في نافذة مستكشف الحل (**شكل 1-6 بالصفحة السابقة**) ومن ثم اختيار الأمر Properties من القائمة المنبثقة لتظهر هذه النافذة (**شكل 1-8 بالصفحة التالية**). اكتب اسم المشروع الحالي تحت خانة Assembly Name.

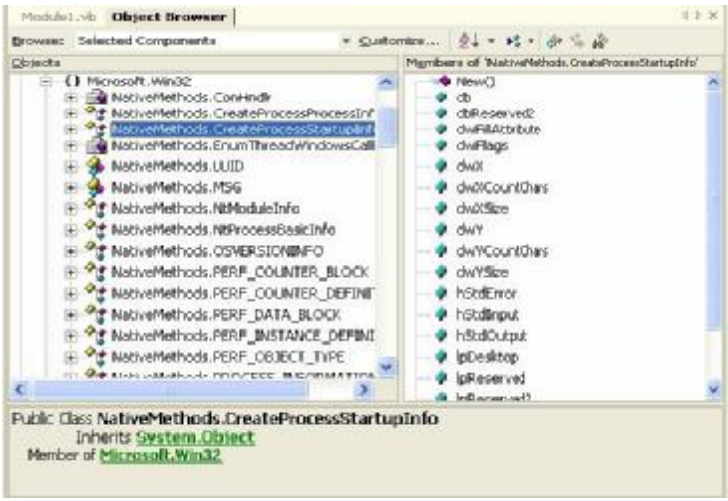
شكل 1-8: نافذة خصائص المشروع.

ملاحظة

في الحقيقة الاسم الرسمي لهذه النافذة هو xxx Property Pages (حيث ترمز الحروف xxx إلى اسم المشروع الحالي)، مع ذلك ستلاحظ أني استخدم الاسم Project Property Pages في هذا الكتاب، وذلك لجهلي باسم المشروع الحالي عند تطبيقك للأمثلة في جهازك.

نافذة مستعرض الكائنات Object Browser:

إذا أردت معرفة جميع الفئات والتركيبات والأعضاء التابعة لها سواء كانت هذه الفئات معرفة في المشروع الحالي أو مضمنة من قائمة المراجع Reference، فاختر الأمر View->Other Windows->Object Browser لعرض نافذة مستعرض الكائنات (**الشكل 1-9 بالصفحة المقابلة**).



شكل 1-9: نافذة مستكشف الكائنات.

نافذة محرر الشيفرة Code Editor:

توفر نافذة محرر الشيفرة **(شكل 1-10 بالصفحة التالية)** قائمة IntelliSense والتي تظهر بمجرد كتابة النقطة "." بعد اسم الكائن لتعرض جميع أعضائه، كما توجد في أعلى نافذة المحرر قائمتين الأولى لنقل المؤشر إلى الفئات والوحدات البرمجية في الملف الحالي، والثانية للطرق والخصائص. المزيد أيضا، يمكن إخفاء جزء من الشيفرات المصدرية وإظهاره بالضغط على الرموز "+" الموجودة في يسار النافذة.

PDF

شكل 1-10: نافذة محرر الشيفرة.

يمكنك تخصيص وتغيير اعدادات هذه النافذة (كالخطوط، الألوان، المحاذاة ....الخ) بالانتقال إلـــى خانة التبويب Text Editor في صندوق الحوار Option (**شكل 1-4 صفحة 19**).

القائمة الرئيسة

فيما يلي عرض سريع لمحتويات القائمة الرئيسة بشكل مختصر.

القائمة File:

معظم أوامر هذه النافذة تتعلق بملفات المشروع الحالي، حيث تمكنك من حفظها، فتحها، إغلاقهـــا وإضافة عناصر وملفات أخرى إضافية. كما انك تستطيع الوصول إلى وظائف الطباعة Printing عن طريق هذه القائمة.

القائمة Edit:

عمليات التحرير كالنسخ Copy، القص Cut، واللصق Paste موجودة في هذه القائمة، بالإضافة إلى مجموعة من أدوات البحث وعلامة الملاحظات Bookmarks.

القائمة View:

تتعلق بإظهار وإخفاء مجموعة كبيرة من النوافذ متعددة الوظائف والأغراض.

القائمة Project:

تتعلق بالمشروع الحالي حيث توفر أوامر لإضافة عناصر وملفات أخرى للمشروع، كما تستطيع الوصول إلى نافذة المراجع Reference أيضا من خلال هذه القائمة. بالنسبة للأمر Set As Startup Project فهو يجعل المشروع الحالي هو المشروع الابتدائي، وذلك في حالة وجود أكثر من مشروع Project في نفس الحل Solution.

القائمة Build:

تمكنك هذه القائمة من ترجمة المشروع Compiling، وبالنسبة للأمر Configuration Manager فهو يحدد الاعدادات المسبقة الحفظ للمترجم.

القائمة Debug:

لحظة تصميم البرنامج تكون عدد عناصر هذه القائمة لا تتجاوز 9 أوامر، ولكن عند التنفيذ سيمتد هذا العدد ليصل إلى 13 أمر (بعضها غير مفعل)، تجد أوامر التنفيذ والإيقاف النهائي والمؤقت في هذه النافذة، كما تحتوي على جميع وظائف التنقيح Debugging للبرنامج.

انظر أيضا

لعرض بضعة أوامر من هذه القائمة وأدوات التنقيح، انتقل إلى الفصل السابع **اكتشاف الأخطاء**.

القائمة Tools:

تحتوي على أوامر إضافية مختلفة الوظائف، كما يمكنك جعلها منصة لتشغيل برامج اخرى تستخدمها بشكل متكرر عن طريق اختيار الأمر Externals Tools. وبالنسبة **للإضافات Add-Ins**، فيمكن الوصول لها عن طريق اختيار الأمر Add-In Manager.

PDF Factory Pro trial version www.fastio.com

القائمة Window:

لا تعليق!

القائمة Help:

لابد من أن تكون قد ثبت نسخة من مكتبة MSDN أو NET Documentation. حتى تتمكن من الوصول إلى التعليمات. وبالنسبة للأمر Dynamic Help، فهو يعرض لك نافذة تظهر تعليمات فورية بمجرد النقر بزر الفأرة على أي عنصر أو نافذة من بيئة التطوير. أنصحك بإغلاق هذه النافذة عند عدم الحاجة إليها، فهي تسبب بطء نسبي في الجهاز خاصة ان كان جهازك بطيء.

أشرطة الأدوات

أشرطة الأدوات Toolbars ما هي إلا أوامر مثل الموجودة في القائمة الرئيسية تقريبا، يمكن إضافتها تحريرها وحذفها بالضغط بزر الفأرة الأيمن على أي شريط واختيار الأمر Customize من القائمة المنبثقة، تماماً كما تفعل مع طاقم تطبيقات Microsoft Office.

كتابة برنامجك الأول

والان سنبدأ بكتابة أول برنامج لك بلغة NET. Visual Basic حتى تتمكن من استخدام بيئة Visual Studio .NET بشكل مبدئي.

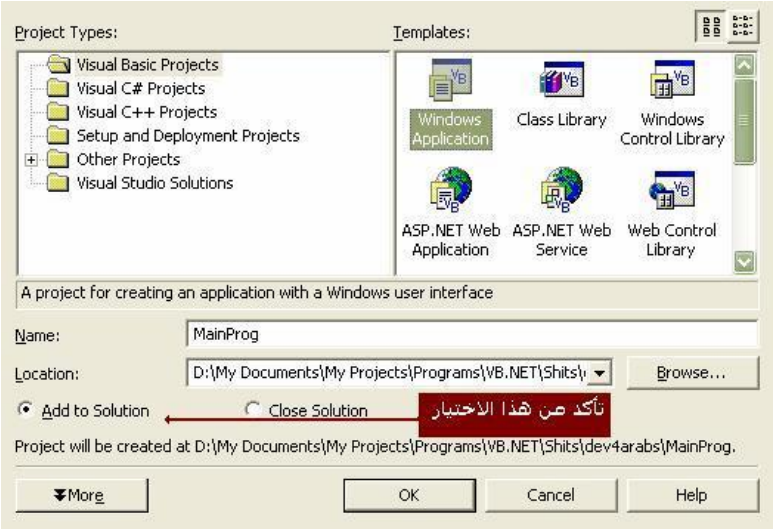
الحلول والمشاريع

الحل Solution هو عبارة عن حاوي لعنصر أو مجموعة عناصر تسمى **المشاريع Projects**، والمشروع هو البرنامج الذي تود إنشائه والذي بدوره يحتوي على عدة عناصر تسمى **ملفات المشروع Project Files** أو وحدات المشروع Project Items.

سأبدأ معك بالحل Solution، يمكنك إنشاء حل جديد باختيار الأمر الفرعي Blank Solution من الأمر New من قائمة File حيث سيظهر لك صندوق حوار بعنوان New Project يطلب منك اسم الحل ومسار مجلده. بعد الضغط على زر OK ستنشئ لك بيئة التطوير Visual Studio .NET ملفين -في نفس المجلد الذي حددته سابقاً- بالامتدادين sln. و suo. الملف sln هو ملف نصي يمثل مراجع إلى المشاريع المضمنة في الحل الحالي، اما الملف suo

فيحتوي على الاعدادات وخيارات التخصيص التي تحددها في هذا الحل كمواقع النوافذ المفتوحة وغيرها.

بعد إنشائك لحل جديد قد تبدأ بإضافة مشروع أو عدة مشاريع إليه، اختر الأمر الفرعي Project من الأمر New من قائمة File سيظهر لك نفس صندوق الحوار السابق، انقر على العنصر Node في الشجرة اليسرى الذي يحمل الاسم Visual Basic Projects حتى تظهر لك في القائمة اليمنى عدة قوالب Templates جاهزة للاستخدام. حدد نوع، اسم، ومسار المشروع ثم تأكد من الاختيار Add to solution قبل الضغط على زر OK (**شكل 1-11**):



شكل 1-11: القوالب Templates الجاهزة والتي تمثل مشاريع يمكنك البدء فيها.

يمكنك إضافة المزيد من المشاريع بمختلف أنواعها، والطريقة الظريفة التي تمكنك من التحويل بين المشاريع هي نافذة مستكشف الحال Solution Explorer (**شكل 1-6 صفحة 27**).

بالنسبة لملفات المشروع المنجزة بلغة Visual Basic .NET فجميعها تنتهي بالامتداد vb. مهما كان نوعها (سواء ملفات أدوات التحكم UserControls، نماذج Windows Forms، فئات Classes ...الخ)، وفي الحقيقة يمكن للملف الواحد أن يحتوي على جميع العناصر السابقة.

أنواع المشاريع

يمكنك Visual Basic .NET من دمج عدة أنواع مختلفة من المشاريع كأدوات التحكم User Controls، تطبيقات قياسية Windows Application، مكتبات الفئات Class Library وغيرها، ربما تكون قد لاحظتها في صندوق الحوار السابق New Projects، واليك يا قارئي العزيز ملخص عنها:

Windows Application:

وهي مشاريع تشابه تطبيقات Windows القياسية (أي Standard Application). الجزء الثالث **تطوير تطبيقات Windows** مخصص لهذا النوع من المشاريع.

Class Library:

هذا النوع من المشاريع يحتوي على مكتبة فئات يمكنك الاستفادة منها في برامج أخرى، كما يمكنك ترجمتها إلى ملفات من النوع DLL.

Windows Control Library:

يمكنك هذا النوع من المشاريع من إنشاء أدوات تحكم User Controls تستخدمها في تطبيقات Windows Application. سنطبق أدوات التحكم بعد مئات الصفحات إلى أن نصل للفصل السادس عشر **مواضيع متقدمة**.

ASP .NET Web Application:

يمكنك من إنشاء مشاريع ASP .NET بحيث تعمل في جهة الخادم Server ويتم عرض صفحاتها عن طريق عملاء Clients بأحد المتصفحات Browsers. الفصلان العشرون والحادي والعشرون **تطبيقات ASP .NET** مخصص لهذا النوع من المشاريع.

ASP .NET Web Service:

هذا النوع من المشاريع يسهل عليك عملية تبادل البيانات عبر الانترنت عن طريق استخدام بروتوكولات HTTP و XML القياسية دون الحاجة إلى تطوير المكونات الموزعة DCOM – كما سترى لاحقا في الفصل الثاني والعشرون **خدمات ويب Web Services**.

PDF Factory Pro trial version www.fineprint.com

Web Control Library:

يمكنك تطوير مشاريع شبيهة بأدوات التحكم لكنها خاصة للعرض على صفحات HTML، وهي مشابهة إلى حد كبير بمشاريع أدوات التحكم User Controls، ولكنها تعرض في المستعرض Browser.

Console Application:

إن كنت تشعر بالحنين إلى تطوير التطبيقات تحت بيئة DOS فهذا النوع مناسب لك تماماً. بالنسبة لنا، سنستمر في تطوير هذا النوع من المشاريع حتى نهاية الجزء الثاني من هذا الكتاب لنتعلم أساسيات لغة البرمجة Visual Basic .NET.

Windows Services:

نوع خاص من تطبيقات Windows القياسية بحيث يعمل في الخلفية Background دائما منذ بدأية تحميل نظام التشغيل حتى إغلاق جهاز الكمبيوتر. لي عودة إلى هذا الموضوع في الفصل السادس عشر **مواضيع متقدمة**.

Empty Project:

سهلة جدا ولا تحتاج إلى تفاصيل.

Empty Web Project:

أسهل من سابقتها.

بعد هذه الجولة السريعة حول أنواع المشاريع، يؤسفني إخبارك بان ما ذكرناه هو مجرد قوالب Templates تقوم بتوليد الشيفرات الضرورية لعمل ما تريد، بل حتى يمكنك إنشاء المزيد من هذه القوالب أو حذف الحالية، فالحل الواحد قد يشمل خدمة Windows Service و برنامج قياسي Windows Application و أداة تحكم User Control، السر كله يكمن في شيفرات ملفات المشروع.

PDF

بناء برنامجك الأول

كل الطرق تؤدي إلى روما قيلت سابقا، لديك عشرات الطرق والوسائل التي تمكنك من كتابة برنامجك الأول بـ Visual Basic .NET ابتداء من ملفات نصية باستخدام المفكرة Notepad أو حتى استخدام احد القوالب الجاهزة -كقالب Console Application.

ما يهمني في هذا الفصل إعطائك الخطوات الأساسية لبناء برامجك الأول باستخدام Visual Basic .NET وليس البدء الفعلي بشرح قواعد ومفردات اللغة، فلن أقدم لك الكثير من التفاصيل، ما انشده هنا هو توضيح عملية بناء المشاريع وتنفيذها فقط.

أنشئ أي مشروع جديد ولنقل Console Application على سبيل المثال، ستلاحظ أن نافذة محرر الشيفرة قد فُتحت وكتب بها الشيفرة التالية:

```
Module Module1
    Sub Main()

    End Sub
End Module
```

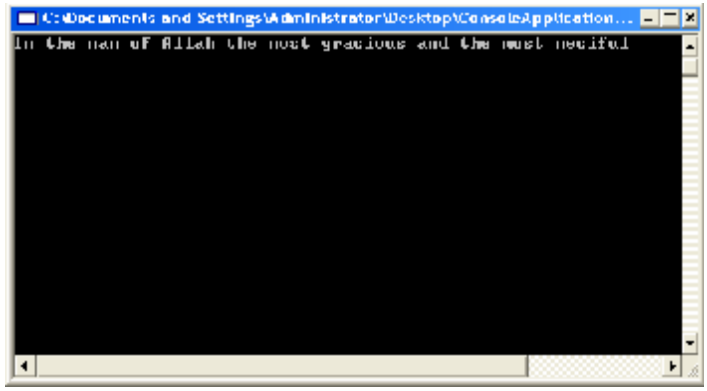
غير اسم الوحدة البرمجية Module1 إلى FirstProg، استخدم الكائن Console لعرض المخرجات على الشاشة. هذا برنامجك الأول وهو يقوم بعرض البسملة لتكون فاتحة خير علينا بمشيئة الله:

```
Module FirstProg
    Sub Main()
       Console.WriteLine("In the name of Allah the most gracious
                 and the most merciful")
    End Sub
End Module
```

اضغط على المفتاح [F5] أو اختر الأمر Start من قائمة Debug لتنفيذ المشروع، (مع العلم أن عملية التنفيذ تؤدي إلى عملية الترجمة Compiling بشكل تلقائي) ستلاحظ أن نافذة سوداء ظهرت واختفت بسرعة، ولكي تتمكن من إيقافها أضف الأمر التالي في البرنامج السابق، بحيث يسمح للمستخدم بالضغط على مفتاح [ENTER] قبل اخفاء النافذة:

```
Module FirstProg
    Sub Main()
        …
        Console.Read()
    End Sub
End Module
```

اعد تنفيذ البرنامج لترى مخرجاته (**شكل 1-12**).



شكل 1-12: مخرجات البرنامج الأول.

استخدام ArabicConsole

اضطررت في الفقرة السابقة لاستخدام اللغة الإنجليزية لعرض مخرجات البرنامج الأول وذلك لان الكائن Console لا يدعم الحروف العربية، وبما أني سأعتمد على هذا الكائن في شرح الشيفرات المصدرية فلابد من استخدامه، وبدلا من جعل أمثلة الكتاب تعتمد على الكلمات الإنجليزية، فكرت بتطوير كائن بسيط جدا يحمل الاسم ArabicConsole يحاكي الكائن Console يمكنك من استخدام الحروف العربية. ضع في عين الاعتبار، أن الكائن ArabicConsole لا يحتوي إلا على طريقة واحدة هي WriteLine() فقط، وهي الطريقة الوحيدة التي نحتاجها لعرض المخرجات. ان كنت تود استخدام هذا الكائن في مشاريعك، فيمكنك إضافته عن طريق نافذة المراجع (**شكل 1-7**) والضغط على الزر Brows ومن ثم البحث عن الملف ArabicConsole.DLL في الدليل الجذري للقرص المدمج.

PDF

بعد اضافتك لمرجع الكائن ArabicConsole في مشروعك، تستطيع استخدامه مباشرة وتكتب شيئا مثل:

```
' اضف هذا السطر قبل استخدام الكائن
' ArabicConsole
Imports ArabicConsoleProject

Module FirstProg
    Sub Main()
        ArabicConsole.WriteLine ("بسم الله الرحمن الرحيم")
    End Sub
End Module
```

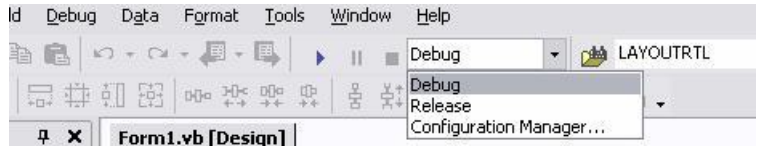
انظر أيضا

ستفهم الغرض من استخدام العبارة Imports ArabicConsoleProject في الفصل التالي **لغة البرمجة** بمشيئة الله.

لست بحاجة لاستخدام الطريقة ()Read حتى تمنع النافذة من الإغلاق التلقائي، حيث أن الكائن ArabicConsole يعطيك فرصة لإغلاقها بنفسك.

الترجمة والتوزيع

النقطة الأخيرة التي أريد التطرق لها هي عملية ترجمة البرنامج Compiling وتوزيعه، بالنسبة للترجمة فتوجد عشرات الخيارات المعقدة -بالنسبة لي- تحدد بها سلوك المترجم Compiler، إلا أن فريق التطوير لـ Visual Studio .NET قد سهلوا علينا عمل ذلك بتجهيز هذه الخيارات بشكل مبدئي بحيث يناسب اغلب الحالات، تعرف هذه الاعدادات بالـ **Configurations**. بشكل مبدئي يمكنك رؤية احد هذه الاعدادات وهو النص المعنون Debug في شريط الأدوات العلوي لبيئة التطوير Visual Studio .NET (**شكل 1-13**):



شكل 1-13: اعدادات الترجمة.

بالنسبة للاعدادات Release فحددها إذا كانت هذه آخر مرة تقوم فيها بعملية ترجمة البرنامج وعندما تكون جاهز لتوزيع البرنامج. يمكنك إضافة اعدادات جديدة أو تحرير هذه الاعدادات والتي تجدها في قائمة Configuration Properties من أداة الشجرة الموجودة في صندوق حوار خصائص المشروع Project Property Pages.

أخيرا، إذا أردت توزيع برنامجك إلى أجهزة أخرى عليك إرفاق مكتبة NET. Framework مع البرنامج حتى تعمل، يمكنك إنزال هذه المكتبات من موقع microsoft.com أو يفضل إرفاقها في اسطوانة مستقلة لان حجمها يزيد عن 100 ميجا بايت.

قد تكون أصبت بالإحباط المبدئي من هذا الفصل ولم تستفد منه الشيء الكثير، ولكن صدقني البداية الحقيقية ستلمسها اعتباراً من الفصل التالي وحتى نهاية الكتاب، وسترى مئات الأسطر من الشيفرات المصدرية مع شرح وافي لكل شيء مبهم لم تفهمه في هذا الفصل، فهدفي هنا تقديم جولة سريعة ومبسطة جدا حول تقنية NET.، بيئة التطوير Visual Studio .NET، وطريقة كتابة أول برنامج لك بـ Visual Basic .NET. اقلب الصفحة لتبدأ تعلم لغة البرمجة Visual Basic .NET. من الصفر في الفصل التالي **لغة البرمجة**.

PDF

لغة البرمجة

إن كنت تعتقد بأنني سأبدأ معك في تصميم النوافذ ووضع الأدوات عليها، فيؤسفني إخبارك أن الوقت ما زال مبكرا جدا للحديث عنها، حيث لن أتطرق إلى هذه المواضيع إلا مع بداية الجزء الثالث من هذا الكتاب **تطوير تطبيقات Windows**. ولو كان الأمر بيدي، لنصحت جميع مبرمجي Visual Basic .NET العرب بأن لا يقفزوا إلى برمجة نماذج Windows Forms حتى يتقنوا أساسيات اللغة، فمسألة إتقان لغة البرمجة Visual Basic .NET أمر في غاية الأهمية قبل الانتقال إلى تطوير التطبيقات المختلفة كـ Windows Applications، أو Web Applications، أو Windows Services ...الخ، خاصة إن علمت أن Visual Basic .NET ليس موجهاً لتطوير تطبيقات Windows وحسب، بل يمكنك من إنجاز الأنواع المختلفة من المشاريع التي تطرقت إليها في الفصل السابق.

اعتباراً من هذا الفصل وحتى الفصل الثاني عشر سنتعامل مع الكائن ArabicConsole لعرض المخرجات على الشاشة، وتوضيح نتائج الشيفرة لتتعلم لغة البرمجة Visual Basic .NET بشكل سليم. وسأفترض أن لديك خلفية –ولو بسيطة- في البرمجة بغض النظر عن اللغة التي استخدمتها، حيث لن أضيع الكثير من وقتي ووقتك في شرح المسائل النظرية (كالتكرار، التفرع، المتغيرات، الإجراءات ...الخ) فلدي الكثير من المسائل التطبيقية والتي ستوضح لك قضايا أهم بكثير في لغة برمجتك الجديدة Visual Basic .NET.

الوحدات البرمجية Modules

أول مصطلح سنتعرف عليه هو **الوحدة البرمجية Module**، وهي عبارة عن حاوية تمكنك من كتابة جميع شيفراتك المصدرية بداخلها، فلو عدت إلى البرنامج الأول في الفصل السابق، ستلاحظ أننا عرفنا وحدة برمجية باسم FirstProg باستخدام الكلمة المحجوزة Module:

```
Module FirstProg
        Sub Main()
                ArabicConsole.WriteLine("بسم الله الرحمن الرحيم")
        End Sub
End Module
```

لا يمكنك كتابة أي شيفرة خارج نطاق الوحدة البرمجية (أي فوق السطر Module X وتحت السطر End Module). ولو تجرأت وصرحت عن متغير أو أعلنت عن إجراء خارج نطاق الوحدة البرمجية، فأنت عملياً تكتب خارج نطاق الحاوية، وسيظهر لك المترجم رسالة خطأ Statement is not valid in a namespace:

```
' لن يتم تنفيذ الشيفرة التالية لعدم وجودها
' Module داخل وحدة برمجية
Dim X As Integer

Sub Test ()
    …
    …
End Sub

Module FirstProg
    …
    …
End Module
```

يمكن للملف الواحد أن يحتوي على أكثر من وحدة برمجية، فقد ترغب –مثلا- في تقسيم وحداتك البرمجية استنادا إلى تصنيف وظائفها:

```
Module DrawingFunctions
    …
    …
End Module

Module InternetFunctions
    …
    …
End Module

Module SystemFunctions
    …
    …
End Module

…
…
```

بالنسبة لشروط تسمية الوحدات البرمجية فهي مثل شروط تسمية باقي المعرفات Identifiers الأخرى (كالمتغيرات، التركيبات، الفئات ...الخ) وهي:

- أن لا يزيد عدد حروف المعرف عن 16383 حرف -لا اعتقد انك بحاجة إلى كل هذا العدد!
- أن يبدأ المعرف بحرف أبجدي، مع ذلك يمكنك استخدام الشرطة السفلية "_" كبداية لاسم المعرف ولكن عليك إتباعها بحرف أبجدي كي يتمكن المترجم من تمييزها عن المعامل "_" (الذي يستخدم لتقسيم الأمر إلى أكثر من سطر).
- لا يمكنك استخدام اسم يمثل كلمة محجوزة Keyword لتعريف معرف جديد. وان كان لابد من ذلك، فاكتب اسم المعرف داخل القوسين [و] (مثال: Dim [Dim] As Integer). يمكنك استخدام القوسين [و] أيضاً لتعريف معرفات أخرى غير الكلمات المحجوزة.

ملاحظة

تسمح لك لغة البرمجة Visual Basic .NET باستخدام الحروف العربية لكتابة أسماء المعرفات، فهي داعمة لجدول الرموز UNICODE، إلا أنني لم ولن استخدمها لا في هذا الكتاب ولا في مشاريعي الخاصة، فلا اعتقد انك تود رؤية شيفرة مشابهة للشيفرة التالية:

```
Dim الكائن As New فئة
Dim س As Integer

If الكائن.خاصيته = 23 Then
   س = الكائن.خاصيته2 = 0
End If
الكائن.طريقة.كائن_محضون.الكائن (20, س)
```

أخيراً، لا تحاول تعريف أكثر من وحدة برمجية Module بنفس الاسم في داخل المشروع، حتى وان اختلفت الملفات التابعة لها، فهذا يسبب خطأ تعارض الأسماء:

```
' لن تتم ترجمة الشيفرة التالية لتعارض
' اسم الوحدة MyModule في الملفين
```

```
' First.vb في الملف
Module MyModule
    …
    …
End Module

' Second.vb في الملف
Module MyModule
    …
    …
End Module
```

ملاحظة

يمكنك استخدام نفس الاسم لتعريف أكثر من وحدة برمجية شريطة تعريفها في مجالات أسماء Namespaces مختلفة. سأتطرق إلى مجالات الأسماء في القسم الأخير لهذا الفصل.

الإجراء Sub Main()

تستطيع تعريف عدد غير محدود من الإجراءات في داخل الوحدة البرمجية Module بما لذ وطاب لك من الأسماء التي تريدها، إلا أن الإجراء الذي يحمل الاسم Main() له طابع خاص، فهو يمكنك من تخصيص المترجم ليقوم باستدعاء هذا الإجراء مع بداية تنفيذ البرنامج:

```
Module Module1
    Sub Main()
        ArabicConsole.WriteLine("بداية البرنامج من هنا")
    End Sub
End Module
```

المزيد أيضا، تستطيع تعريف أكثر من إجراء Main() في أكثر من وحدة برمجية:

```
Module Module1
    Sub Main()
        ArabicConsole.WriteLine("من الوحدة البرمجية الاولى")
    End Sub
End Module



Module Module2
    Sub Main()
```

PDF

```
        ArabicConsole.WriteLine("من الوحدة البرمجية الثانية")
    End Sub
End Module
```

والسؤال الذي يطرح نفسه، أي من الإجرائين ()Main السابقين سيتم استدعائه مع بداية تنفيذ البرنامج؟ والجواب هو كائن الوحدة الذي تحدده في خانة Startup Object من نافذة Project Property Pages (**شكل 2-1**). ستظهر رسالة خطأ إذا اخترت وحدة برمجية Module لم يعرف بها إجراء باسم ()Main، كما ستظهر نفس رسالة الخطأ ان حددت الاختيار Sub Main (الموجود في نفس القائمة) إن وجد الإجراء ()Main في أكثر من وحدة برمجية أو لم يتم تعريفه في أي وحدة برمجية.

شكل 2-1: تحديد الإجراء الابتدائي للمشروع.

الإجراء Sub New()

إجراء آخر له طابع خاص يحمل الاسم New، يعرف هذا النوع من الإجراءات **بالمشيد Constructor**، وهو عبارة عن إجراء يتم تنفيذه بمجرد إنشاء نسخة من الكائن التابع له. فلو كانت الوحدة البرمجية التالية هي أول وحدة يتم تنفيذها في البرنامج، فسيتم تنفيذ الإجراء ()New قبل ()Main:

```
Module Module1
    Sub New()
```

PDF

```
        ArabicConsole.WriteLine("سيتم تنفيذ المشيد New اولا")
    End Sub

    Sub Main()
        ArabicConsole.WriteLine("ثم تنفيذ الإجراء الرئيسي Main")
    End Sub
End Module
```

انظر ايضا

تجد المزيد من التفاصيل والتطبيقات حول المشيدات Constructors في الفصل الرابع **الفئات والكائنات**.

عليك معرفة أن المشيدات في الوحدات البرمجية لا يتم تنفيذها إلا إن قمت باستدعاء احد إجراءات الوحدة البرمجية أو الوصول إلى احد متغيراتها، فالمشيد الموجود في الوحدة البرمجية Module2 التالية لن يتم تنفيذه، وذلك لأننا لم نستخدم أي عضو من أعضاء الوحدة البرمجية التابعة له:

```
Module Module1
    Sub New()
        ArabicConsole.WriteLine("سيتم تنفيذ المشيد New اولا")
    End Sub

    Sub Main()
        ArabicConsole.WriteLine("ثم تنفيذ الإجراء الرئيسي Main")
    End Sub
End Module

Module Module2
    Sub New()
        ArabicConsole.WriteLine("لن يتم تنفيذ هذا المشيد")
    End Sub
End Module
```

المتغيرات والثوابت

لا يختلف مبرمجان اثنان على اهمية موضوع المتغيرات في أي لغة برمجة، وإذا كان أساس إتقان اللغات الطبيعية هو تعلم حروف ومفردات تلك اللغة، فان أساس إتقان لغات البرمجة هو تعلم المتغيرات والثوابت التي تبني بها إجراءات برامجك. نظريا، لا تختلف فكرة المتغيرات في Visual Basic .NET عن لغات البرمجة القديمة، ولكنها تختلف اختلافاً جذرياً في بنيتها التحتية عما كانت عليه في السابق كما سترى لاحقا.

التصريح عن المتغيرات

ما زالت الكلمة المحجوزة Dim تستخدم للتصريح عن متغير جديد برفقة المعامل As الذي يحدد نوع المتغير:

```
' Integer متغير من النوع
Dim Age As Integer

' String متغيران من النوع
Dim FirstName As String
Dim LastName As String
```

في الشيفرة السابقة عرفت متغيرين من النوع String في سطرين منفصلين، مع ذلك يمكّنك Visual Basic .NET من دمجهما في سطر واحد:

```
Dim Age As Integer
Dim FirstName, LastName As String
```

أو دمج تصاريح المتغيرات الثلاثة كلها في سطر واحد –بالرغم من اختلاف أنواعها:

```
' لا انصحك بتعريف انواع مختلفة من
' المتغيرات في سطر واحد
Dim FirstName, LastName As String, Age As Integer
```

ينصح دائما بتحديد نوع المتغير عند التصريح عنه، وان لم تحدد نوع المتغير فسيكون نوعه بشكل مبدئي Object، وسيتم تحويله إلى نوع آخر تماثل نوع القيمة التي تسندها إليه:

```
Dim X

X = 10      ' Integer اصبح المتغير هنا من النوع
X = "10"  ' String وهنا اصبح
```

PDF

مع أن الطريقة السابقة تعطيك مرونة كبيرة في تشكيل وتغيير نوع المتغير من وقت لآخر، إلا أنها تسبب بطئاً كبيراً في عملية تنفيذ البرنامج، والسبب هو اضطرار المترجم إلى القيام بجهد إضافي لتحويل نوع المتغير.

العبارة Option Explicit:

مبدئيا، عملية التصريح عن المتغيرات أمر إلزامي عليك قبل استخدام المتغير، أما إن كانت العبارة Option Explicit Off مسطورة في أعلى الملف، فيمكنك استخدام المتغيرات والتعامل معها مباشرة دون الحاجة للتصريح عنها بـ Dim:

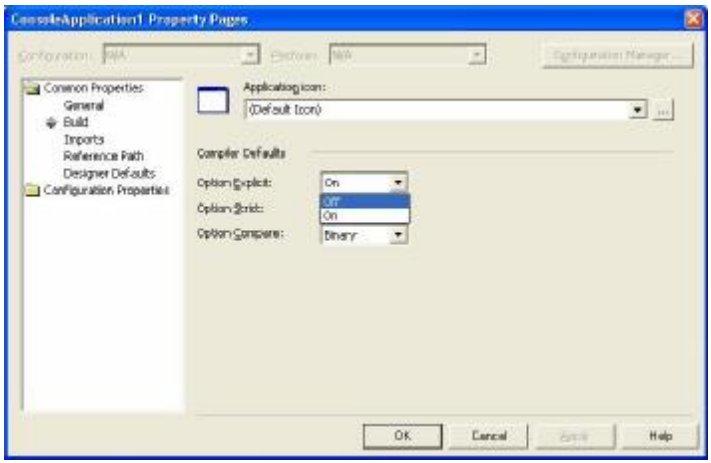
```
Option Explicit Off

Module Module1
    Sub Main()
        ' متغير جديد استخدمته مباشرة
        ' دون تعريفه بـ Dim
        programmerName = "تركي العسيري"
        ArabicConsole.Writeline ( programmerName )
    End Sub
End Module
```

صحيح أن الشيفرة السابقة ستوفر عليك عناء التصريح عن المتغيرات، إلا أن هذا الأسلوب غير محبذ بشكل كبير لدى المبرمجين الجادين، الأخطاء الإملائية هي احد الأسباب:

```
' مخرجات الامر التالي لا شئ بسبب الخطأ الاملائي في
' كتابة اسم المتغير السابق
ArabicConsole.Writeline ( programerName )
```

سبب آخر قد يجعلك ترفض استخدام العبارة Option Explicit Off وهو أن جميع المتغيرات ستكون بشكل ابتدائي من النوع Object وفي كل مرة تسند قيمة جديدة سيتم تحويل نوع المتغير إلى النوع المماثل للقيمة المسندة إليه، مما يسبب بطئاً في عملية التنفيذ.

عليك الأخذ بعين الاعتبار أن تأثير العبارة Option Explicit Off يشمل الملف الحالي الذي سطرت فيه العبارة فقط. وبدلا من كتابتها في كافة ملفات المشروع الأخرى، يمكنك اختيار القيمة Off من قائمة Option Explicit في خانة التبويب Build من نافذة خصائص المشروع Project Property Pages (**شكل 2-2 بالصفحة المقابلة**).



شكل 2-2: تغيير قيمة Option Explicit من On إلى Off.

قابلية الرؤية وعمر المتغيرات

قابلية الرؤية Visibility أو **المدى Scope** للمتغير تمثل قدرة شيفرة البرنامج على الوصول إلى المتغير واستخدامه، فالمتغير X الموجود في الإجراء MySub1() التالي، لا يمكنك الوصول إليه واستخدامه من خارج الإجراء:

```
Sub MySub1 ()
    Dim X As Integer

    X = 20
End Sub

Sub MySub2 ()
    ' لا يمثل المتغير X السابق
    ArabicConsole.WriteLine (X)
End Sub
```

أما عمر Lifetime المتغير، فتمثل الفترة التي يظل فيها المتغير محتفظا بقيمته، فالمتغير X الموجود في الشيفرة السابقة، سينتهي ويفقد القيمة 20 التي كان محتفظا بها بمجرد الانتهاء من تنفيذ الإجراء MySub1(). وحتى تفهم الأسلوب الذي يتبعه Visual Basic .NET لتطبيق مفهومي قابلية الرؤية والعمر للمتغير، عليك معرفة أنواع المتغيرات من منظور الرؤية والعمر:

المتغيرات المحلية الديناميكية:

المتغيرات المحلية الديناميكية Dynamic Local Variables هي متغيرات يتم الإعلان عنها داخل الإجراءات، وعمر المتغير يبدأ من السطر الذي تصرح فيه عن المتغير وينتهي بعد الانتهاء من تنفيذ الإجراء. أما بالنسبة لقابلية الرؤية فهي محصورة داخل الإجراء الذي صرحت فيه فقط. تستخدم الكلمة المحجوزة Dim أيضا للتصريح عن متغير محلي ديناميكي.

تقترح عليك مستندات Microsoft .NET إتباع أسلوب يسمى smallCase في تسمية هذا النوع من المتغيرات، بحيث تكون الكلمة الأولى صغيرة الحروف small والحرف الأول من الكلمات الأخرى كبير Capital. أمثلة:

```
Dim programmerName As String
Dim userID as Integer
Dim employeeSalary As Decimal
```

بعيدا عن موضوع التسمية، يوجد نوع خاص من المتغيرات المحلية الديناميكية يعرف بالـ Block level Variables، وهي متغيرات يتم تعريفها داخل تركيب Block (كحلقة For … Next، جملة If … Then، حلقة Do … Loop وغيرها). مدى هذه المتغيرات يكون محصوراً داخل التركيب الذي أعلنت فيه عن المتغير، وعمرها مثل عمر المتغيرات المحلية الديناميكية السابقة. هذا متغير يحمل الاسم y عرف داخل حلقة For … Next:

```
Dim counter As Integer

For counter = 1 To 10
    Dim y as integer
    …
    …
Next
```

لا تحاول استخدام المتغير المعرف في داخل تركيب خارج هذا التركيب، فمدى هذا النوع من المتغيرات –كما قلت- محصور داخل التركيب فقط:

```
Dim x As Integer

If x = 0 Then
    Dim y As Integer
    …
    …
End If

x = y     ' رسالة خطأ
```

PDF

كما أنك لن تستطيع استخدام اسم متغير ديناميكي محلي لتسمي به متغير داخل تركيب في نفس الإجراء:

```
Dim a As Integer

Do
    Dim a as integer     ' رسالة خطأ
    …
    …
Loop
```

مع ذلك، يسمح لك Visual Basic .NET باستخدام نفس أسماء المتغيرات العامة أو على مستوى الوحدة أو حتى أسماء متغيرات أخرى معرفة في تركيب آخر:

```
' افترض انه متغير على مستوى الوحدة أو عام
Dim x As Integer
…
…

If x = 0 Then
    Dim x As String     ' ممكن جدا
    …
    …
End If

Do
    Dim x As Long       ' خذ راحتك
    …
    …
Loop
```

نقطة أخيرة هامة، عمر هذا النوع من المتغيرات مستمر حتى نهاية الإجراء وليس نهاية التركيب الذي عرفت فيه، فالمتغير x التالي سيحتفظ بقيمته حتى وان خرجت من تركيب الحلقة For counter2 الذي عرف فيها:

```
Dim counter As Integer
Dim counter2 As Integer

For counter = 1 To 3
    For counter2 = 1 To 3
        Dim x As Integer  ' سيستمر في الاحتفاظ بقيمته
        x = x + 1
        ArabicConsole.WriteLine(x)
    Next
Next
```

PDF

مخرجات الشيفرة السابقة ستكون:

```
1
2
3
4
5
6
7
8
9
```

المتغيرات المحلية الستاتيكية:

المتغيرات المحلية الستاتيكية Static Local Variables هي نفس المتغيرات المحلية الديناميكية، لذلك كل ما قلته في الفقرة السابقة ينطبق هنا دون أي اختلاف، باستثناء أن عمرها الافتراضي ابدي (أي يستمر المتغير الاستكاتيكي محتفظاً بقيمته حتى نهاية البرنامج أو موت الكائن التابع له)، كما انك ستستخدم الكلمة المحجوزة Static عوضا عن Dim للتصريح عن متغير ستاتيكي:

```
Static staticVariable As Integer
```

لا تحاول استخدام المتغيرات الستاتيكية كثيرا، فهي أبطأ من المتغيرات الديناميكية، كما أنها تحجز مساحة في الذاكرة طوال فترة عمل البرنامج دون أن يكون هناك حاجة ماسة إليها. قد تستخدم المتغيرات الستاتيكية مثلا للاحتفاظ بقيمة عداد أو تنفيذ إجراء مرة واحدة:

```
Sub Counter ()
    Static counter As Integer

    counter = counter + 1
    …
    …
End Sub

Sub PrintData ()
    Static isPrinting As Boolean

    If isPrinting Then
        Exit Sub
    Else
        isPrinting = True
    End If
    …
    …
End Sub
```

أخيرا، الكلمة المحجوزة Static لا تطبق إلا على المتغيرات المحلية، فلا تحاول استخدامها مع المتغيرات على مستوى الوحدة أو المتغيرات العامة فهي ستاتيكية بطبيعتها.

المتغيرات على مستوى الوحدة البرمجية والمتغيرات العامة:

قد تود من الإجراءات المختلفة التابعة لوحدة برمجية معينة مشاركة المتغيرات فيما بينها، يمكنك Visual Basic .NET من عمل ذلك عن طريق تصريح متغيرات على مستوى الوحدة Module Level Variables، وبهذا يكون مدى هذه المتغيرات شاملاً لجميع إجراءات الوحدة البرمجيـــة. استخدم الكلمة المحجوزة Private أو Dim لتعريف متغير على مستوى الوحدة شريطة أن يـــتم التصريح عن المتغير خارج الإجراءات:

```
Module Module1
    ' متغيرات على مستوى الوحدة
    Dim x As Integer
    Private y As Integer

    Sub Main()
        x = 50
        …
    End Sub

    Sub Test ()
        y = 10
        …
    End Sub
End Module
```

أما المتغيرات العامة Global Variables فمداها يشمل جميع شوارع وأوديـــة وملفـــات البرنامج، وليست محصورة لوحدة برمجية معينة. استخدم الكلمة المحجـــوزة Public لتعريـــف متغير عام:

```
Module Module1
    Public x As Integer     ' متغير عام

    Sub Main()
        x = 5
        …
    End Sub
End Module
```

PDF

```
' وحدة برمجية اخرى
Module Module2
    Sub Test()
        x = 1     ' يمكن الوصول إلى المتغير
    …
    End Sub
End Module
```

ملاحظة

يمكنك أيضا استخدام الكلمة المحجوزة Friend لتعريف متغير عام، ولكنها تختلف عن الكلمة المحجوزة Public في قابلية الوصول إلى المتغير من مشروع خارجي. حيث تحصر Friend مدى المتغير على المشروع الحالي فقط.

وكمعيار للتسمية، تقترح عليك مستندات .NET استخدام أسلوب PascalCase عند تسمية المتغيرات العامة والأسلوب smallCase للمتغيرات على مستوى الوحدة:

```
Public ProgrammerName As String        ' متغيرات عامة
Friend ClientAge As Integer

Dim programmerName As String           ' متغيرات على مستوى الوحدة
Private clientAge As Integer
```

أخيرا، عمر المتغيرات على مستوى الوحدة أو المتغيرات العامة مستمر حتى نهاية البرنامج أو الكائن التابع الذي صرحت فيه.

أنواع البيانات

يبدو أن الوقت قد حان لأخذ جولة تعريفية حول أنواع البيانات المختلفة التي تستطيع استخدامها في برامجك، ولكن دعني أوضح لك نظرتي الشخصية حول هذه الفقرة:

أنواع البيانات (كـ String، Integer، Long، Date ....الخ) لا تتبع -تقنيا- للغة البرمجة Visual Basic .NET، فهي عبارة عن فئات Classes وتركيبات Structures عرفت في مكتبة فئات BCL (التابعة لإطار عمل NET Framework. والتي تحدثت عنها في الفصل السابق)، وبعبارة أخرى: كل شيء تراه في شيفراتك عبارة عن كائن Object، فان كنت من المبرمجين المخضرمين عليك أن تعلم علم اليقين أن جميع المتغيرات التي تصرح عنها وتستخدمها

في برامجك، ما هي إلا كائنات منشأة من فئات أو تركيبات معرفة مسبقاً. فحتى ابسط أنواع المتغيرات مثل Byte، عبارة عن كائن له طرق وخصائص تابعه له. وبما أننا ما زلنا في بداية تعلم اساسيات لغة البرمجة Visual Basic .NET، فلا اعتقد انه من المناسب –حاليا على الأقل- التحدث عن هذه الأنواع قبل استيعاب الكائنات والفئات (وهو موضوع الفصل الثالث **الفئات والكائنات**)، لذلك كان قراري النهائي هو تأجيل تفصيل هذه الأنواع إلى الفصل السادس **الفئات الأساسية**. وإلي أن نلتقي هناك، يعرض لك الجدول التالي ملخص سريع **لأنواع البيانات الأولية Primitive Data Types** التي يدعمها Visual Basic .NET:

النوع	الحجم	مجال القيمة
Boolean	2 بايت	True (صح) أو False (خطأ).
Byte	1 بايت	عدد صحيح من 0 إلى 255.
Char	2 بايت	حرف واحد من نوع UNICODE.
Date	8 بايت	وقت من الساعة 0:00:00 إلى الساعة 11:59:59، كما يشمل تاريخ من يوم 1 يناير لعام 0001 إلى 31 ديسمبر لعام 9999.
Decimal	16 بايت	عدد صحيح من 0 إلى ±79,228,162,514,264,337,593,543,950,335 أو عشري من 0 إلى ±7.9228162514264337593543950335
Double	8 بايت	عدد عشري من 1.79769313486231570E+308 إلى 4.94065645841246544E-324 بالنسبة للأعداد السالبة. ومن 4.94065645841246544E-324 إلى 1.79769313486231570E+308 بالنسبة للأعداد الموجبة.
Integer	4 بايت	عدد صحيح من – 2,147,483,648 إلى 2,147,483,647.

PDF

النوع	الحجم	مجال القيمة
	4 بايت	
Long	8 بايت	عدد صحيح من - 9,223,372,036,854,775,808 إلى 9,223,372,036,854,775,807.
Object	4 بايت	جميع القيم والأنواع يمكن حفظها هنا.
Short	2 بايت	عدد صحيح من – 32,768 إلى 32,767.
Single	4 بايت	عدد عشري من 3.4028235E+38 إلى -E 1.40129845 بالنسبة للأعداد السالبة. ومن -E 1.40129845 إلى 3.4028235E+38 بالنسبة للأعداد الموجبة.
String	10 + (2 * عدد الحروف) بايت	من 0 إلى 2 مليار حرف من نوع UNICODE.

عندما تمعن النظر في الجدول السابق، ستلاحظ وجود نوعين من البيانات الحرفية هما Char و String. بالنسبة للنوع الأول فهو يمثل حرف واحد فقط من حروف Unicode، لذلك فالمتغيرات من النوع Char لا يمكن أن تحمل قيمة حرفيه تزيد عن حرف واحد، كما يشرط استخدام حرف الذيل **"c"** حتى تميز القيمة الحرفية من النوع Char عن قيمة سلسلة الحروف من النوع String:

```
Dim A As Char

A = "ت"c
A = "تركي"c      ' رسالة خطأ هنا
```

قد تستغرب مدى الجدوى من الاعتماد على المتغيرات من نوع Char بدلا من المتغيرات من نوع String رغم إمكانياتها المحدودة، السبب ببساطة السرعة في التنفيذ والاقتصاد في استهلاك مصادر النظام. حيث أن المتغيرات من نوع Char هي متغيرات من النوع ذات القيمة Value Type Variables بينما المتغيرات من النوع String هي متغيرات مرجعية Reference Type Variables. الفروق بين المتغيرات المرجعية والمتغيرات ذات القيمة هو موضوع الصفحة المقابلة.

PDF

المتغيرات المرجعية والمتغيرات ذات القيمة:

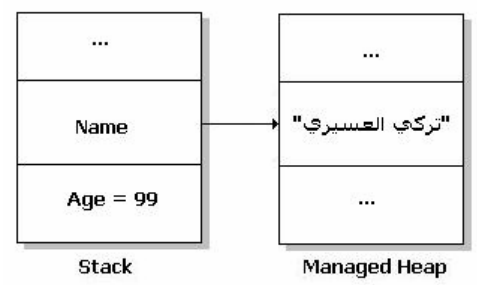
توجد مسائل تقنية بحتة تعتبر في غاية الأهمية تتعلق بالمتغيرات، حيث أني ذكرت -في بداية هذا القسم من الفصل- أن البنية التحتية للمتغيرات قد تغيرت تغيراً جذرياً عما كانت عليه في لغات البرمجة السابقة، إذ أن القضية ابعد من أن تكون مجرد منطقة محجوزة في الذاكرة تحفظ بها قيمة لتمثل المتغير، فكل شيء تراه في Visual Basic .NET هو كائن Object كما اتفقنا سابقا. ولكن عليك معرفة أن المتغيرات –أو أنواع البيانات بصفة عامة- في لغة البرمجة Visual Basic .NET تنقسم إلى قسمين رئيسين مهما اختلفت أنواعها هما: **المتغيرات المرجعية Reference Type Variables والمتغيرات ذات القيمة Value Type Variables**.

سأبدأ معك بالمتغيرات ذات القيمة Value Type Variables، هذا النوع من المتغيرات مشتق وراثيا من الفئة System.ValueType (الوراثة والاشتقاق الوراثي موضوع الفصل الرابع **الوراثة**). تقنيا، هذا النوع من المتغيرات شبيه بفكرة المتغيرات الموجودة في لغات البرمجة السابقة، حيث أن المتغيرات تحفظ قيمها في قسم من ذاكرة البرنامج (قد تكون Stack في معظم الأحوال)، وستمحى من الذاكرة مباشرة بعد نهاية عمر المتغير. جميع البيانات العددية Numbers، والبيانات من النوع Boolean، Char، و Date، والمتغيرات المعرفة من التركيبات Structures أو Enums هي بيانات من النوع ذات القيمة Value Type.

أما المتغيرات المرجعية Reference Type، فيوجد الكثير لأخبرك به عنها لاحقا، ولكن كل ما أريد منك أن تعلمه عنها في الوقت الحالي هو أن قيمة المتغير (يسمى **مؤشر Pointer** في هذه الحالة) يتم حفظه كما تحفظ المتغيرات ذات القيمة، بينما تحفظ البيانات الحقيقة للكائن في قسم خاص من ذاكرة البرنامج يسمى **Managed Heap**، وفي الحقيقة لا تتم عملية إزالة قيمها من الذاكرة مباشرة بعد نهاية عمرها الافتراضي، فهي تتطلب عملية تسمى إفراغ المصادر عن طريق المجموعة **Garbage Collection** مقدمة من إطار عمل NET. فلو كان لدينا هذين المتغيرين:

```
Dim Name As String = "تركي العسيري"      ' متغير مرجعي
Dim Age As Integer = 99                  ' متغير ذا قيمة
```

يمكننا تخيل مواقعهما بالذاكرة كما في الشكل التالي:



شكل 2-3: أماكن المتغيرات ذات القيمة والمتغيرات المرجعية في الذاكرة.

انظر أيضا

لي عودة أخرى حول المتغيرات ذات القيمة في الفصل السادس **الفئات الأساسية**، بينما سيكون لي حديث مطول عن المتغيرات المرجعية في الفصل الثالث **الفئات والكائنات**.

المتغيرات من النوع String، المصفوفات Arrays، والمتغيرات المعرفة من الفئات Classes جميعها متغيرات مرجعية Reference Type.

من منطلق التسلسل التعليمي الذي اتبعه في هذا الكتاب، لا أريد إعطائك جميع الفروقات بين المتغيرات ذات القيمة والمتغيرات المرجعية هنا، حيث أني أفضل ذكرها في أماكن متفرقة من هذا الكتاب لكي تناسب الفقرات التابعة لها، ولكن دعني أخبرك هنا بان المتغيرات ذات القيمة أسرع بكثير من المتغيرات المرجعية، كما أنها اقتصادية جدا في استهلاك مصادر النظام، لذلك حاول الاعتماد عليها عوضا عن المتغيرات المرجعية إلا إن دعتك الحاجة لغير ذلك.

إسناد القيم

قد تستغرب من تخصيص فقرة كاملة عن عملية إسناد القيم إلى المتغيرات، إلا انك ستكتشف أن الأمر بحاجة إلى التطرق لبعض التفاصيل الدقيقة حول هذه المسألة.

بادئ ذي بدء، أنت تعلم وأنا اعلم أننا نستطيع إسناد القيم إلى المتغيرات باستخدام معامل إسناد القيم "="، يمكنك إسناد قيمة للمتغير بعد التصريح عنه مباشرة، أو أثناء عملية التصريح بكل انسيابية:

```
Dim X As Integer = 10   ' اسناد قيمة لحظة التصريح
Dim Y As Integer
Dim Z As Long

Y = 20
Z = 30
```

إن أسندت قيم للمتغيرات أثناء عملية التصريح لأكثر من متغير في سطر واحد، عليك تحديد نوع المتغير لكل تصريح و إلا ستظهر رسالة خطا:

```
' هنا ممكن
Dim X As Integer, Y As Integer = 20, Z As Long = 30

' رسالة خطأ
Dim X, Y As Integer = 20, Z As Long = 30
```

لست بحاجة لإخبارك انك المسئول الأول والأخير عن مجال القيم التي تسندها إلى المتغيرات، وان أضفت قيمة خارج نطاق مجال القيم المسموح به لنوع معين من المتغيرات، ستظهر رسالة خطأ وقت التنفيذ:

```
' رسالة خطأ
Dim X As Byte = 256
```

المزيد أيضا، يوفر لك Visual Basic .NET معاملات إضافية لإسناد القيم إلى المتغيرات تعتبر اختصار لعمليات رياضية شائعة توضحها لك هذه الشيفرة:

```
Dim X As Integer = 5 + 5

X += 1  ' x = x + 1
X -= 2  ' x = x - 2
X *= 3  ' x = x * 3
X \= 6  ' x = x \ 6
X ^= 2  ' x = x ^ 2
```

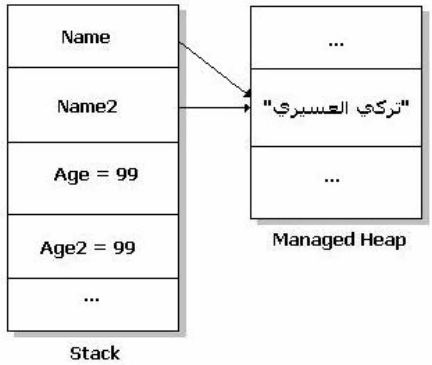
مع العلم أن المعاملات السابقة لا يمكنك استخدامها أثناء عملية التصريح عن المتغير:

```
' لن يسمح لك Visual Basic .NET بكتابة الشيفرة
' بالشكل التالي حتى لو توسط بيل جيتس
Dim X As Integer += 10
Dim Y As Long ^= 20
```

PDF

دعني أوضح لك قضية تقنية تتعلق بإسناد القيم حيث تبين لنا احد الفروق بين المتغيرات ذات القيمة Value Type والمتغيرات المرجعية Reference Type. عملية إسناد القيم بين المتغيرات ذات القيمة تقوم بنسخ فعلي للمتغيرات ليستقل كل متغير بقيمته، أما إسناد القيم بين المتغيرات المرجعية، فهي لا تؤدي إلى نسخ قيم المتغيرات، بل كل ما تقوم به هو نسخ المؤشرات لتشير إلى نفس الكائن الذي يحمل بيانات المتغيرات الفعلية (والموجودة في القسم Managed Heap). إن لم تفهم شيئا مما سبق، ركز في الشيفرة التالية والموضحة **(بالشكل 2-4)**:

```
' متغيرات مرجعية
Dim Name As String = "تركي العسيري"
Dim Name2 As String = Name

' متغيرات ذات قيمة
Dim Age As Integer = 99
Dim Age2 As Integer = Age
```



شكل 2-4: توضيح الفرق في عملية إسناد القيم بين المتغيرات ذات القيمة والمتغيرات المرجعية.

كما ترى في **(الشكل 2-4)**، رغم أني عرفت متغيرين مختلفين (Name و Name2) إلا أنهما لا يزالان يشيران إلى نفس القيمة الفعلية للمتغير في القسم Managed Heap، بينما تستقل المتغيرات (Age و Age2) بقيمها في مناطق مختلفة من الذاكرة.

ملاحظة

مؤشرات المتغيرات المرجعية (الموجودة في قسم Stack) حجمها 4 بايت مهما اختلف نوع القيم التي تشير لها.

العبارة Option Strict:

على مر الأجيال السابقة من لغات البرمجة، واجه المبرمجون معاناة كبيرة في مسألة تحويل البيانات المختلفة من القيم. مع ذلك، لن تواجه أي مشاكل عند إتباع أسلوب **التحويل الواسع Widening Conversion**، حيث أن التحويل الواسع يقوم بنسخ قيمة من نوع صغير (Short مثلا) إلى نوع اكبر منه (كـ Double) مما لا يؤدي إلى التضحية بدقة القيمة:

```
Dim A As Single = 3.9999
Dim B As Double = A

ArabicConsole.WriteLine(A)     ' 3.9999
ArabicConsole.WriteLine(B)     ' 3.9999
```

وعلى العكس من ذلك، لو كان لدينا متغير من النوع Double وأردت إسناد قيمته إلى متغير آخر من النوع Single، ستفقد هذه العملية الدقة العددية الموجودة في المتغير السابق، السبب واضح لان المتغيرات من النوع Double تستطيع حمل قيم أكثر دقة من النوع Single:

```
Dim A As Double = 3.9999999
Dim B As Single = A

ArabicConsole.WriteLine(A)     ' 3.9999999
ArabicConsole.WriteLine(B)     ' 4
```

في الشيفرة السابقة، يضطر Visual Basic .NET إلى إجراء عملية التحويل التلقائية حتى لا يتعدى حدود مجال القيم التي تسمح للمتغيرات من النوع Single حملها، مما يؤدي إلى التضحية بدقة الرقم المحفوظة في النوع Double. يعرف هذا النوع من التحويل **بالتضييق Narrowing Conversion**، أي انك تضيق القيمة من متغير كبير (كـ Double) إلى متغير اصغر منه (وهو Single). أمثلة أخرى: التحويل من Long إلى Integer إلى Short إلى Byte ...الخ.

PDF

التضييق سبب رئيسي لأمراض البرامج (الشوائب Bugs)، إلا انك تستطيع استخدام العبارة Option Strict On في أعلى الملف حتى تمنع نفسك كمبرمج و Visual Basic .NET كمترجم من إجراء عملية التضييق التلقائية بين الأنواع المختلفة من البيانات، لذلك الشيفرة السابقة ستظهر لك رسالة خطأ إن كانت العبارة Option Strict On مسطورة:

```
' منع عملية التضييق
Option Strict On

Module Module1
    Sub Main()
        Dim A As Double = 1
        Dim B As Single

        A = B     ' التحويل الواسع ممكن

        B = A     ' رسالة خطأ بسبب التضييق

    End Sub
End Module
```

ملاحظة

تأثير العبارة Option Strict يشمل الملف الذي سطرت فيه فقط، وان أردت شملها في كافة ملفات المشروع الأخرى -دون الحاجة لكتابتها- يمكنك تعديل اعدادات المترجم في نافذة Project Property Pages - كما فعلنا سابقا مع Option Explicit (**شكل** 2-2 **صفحة** 43).

لا يقتصر تأثير العبارة Option Strict على البيانات العددية فقط، بل يمتد ليصل إلى المتغيرات الأخرى كـ String، Date، Boolean ...الخ:

```
' ممكن في حالة
' فقط Option Strict Off
Dim A As String
Dim B As Boolean

A = "True"
B = A
```

PDF

وعند الحديث عن المعاملات، عليك التفريق بين معامل القسمة \ الخاص بالأعداد الصحيحة، وبين معامل القسمة / والذي يستخدم للأعداد العشرية لان عملية التحويل لن تتم تلقائيـــاً، كمـــا أن معامل الأس ^ يحول القيم إلى Double:

```
' رسالة خطأ في حالة
' Option Strict On
Dim X As Integer

X = 10 / 2
X = 2 ^ 3
```

في المقابل، تفعيلك للعبارة Option Strict On لا يعني انك لا تستطيع إســناد الأنـــواع المختلفة من القيم، بل يمكنك الاستمرار على ذلك شريطة ان تكون لبق وتستخدم دوال التحويل:

```
' ممكن عمل ذلك حتى لو فعلت العبارة
' Option Strict On

Dim X As Double = 3.2
Dim Y As Integer = CInt(X)
```

يعرض لك الجدول التالي مجموعة من دوال التحويل لأنواع البيانات الأخرى:

الدالة	القيمة التي تعود بها
CBool	Boolean
CByte	Byte
CChar	Char
CDate	Date
CDbl	Double
CDec	Decimal
CInt	Integer
CLng	Long
CObj	Object
CShort	Short
CSng	Single
CStr	String

أخيرا، ما ذكرته في السطور السابقة حول تأثير العبارة Option Strict On كان موجها إلى المتغيرات ذات القيمة Value Type بشكل مباشر، وبالنسبة للكائنات الحقيقية –اقصد المتغيرات المرجعية Reference Type- فلها سواليف وعلوم رجاجيل أخرى نذكرها لاحقا في الفصل الثالث **الفئات والكائنات** بمشيئة الله.

الثوابت

بشكل افتراضي، الثوابت العددية الصحيحة يتعامل معها المترجم على أنها من النوع Integer، والأعداد العشرية من النوع Double:

```
ArabicConole.WriteLine (10)   ' Integer قيمة من النوع
ArabicConole.WriteLine (5.5)  ' Double قيمة من النوع
```

مع ذلك، يمكنك تحديد نوع الثابت لزيادة سرعة إسناد القيم، فتستطيع استخدام الذيل **"L"** للنوع Long، الذيل **"S"** للنوع Short، الذيل **"D"** للنوع Decimal، والذيل **"F"** للنوع Single (راجع بقية الاختصارات في مكتبة MSDN):

```
Dim X As Long
Dim Y As Long

X = 100
Y = 100L        ' الإسناد التالي أسرع وذلك لعدم إجراء التحويل الواسع
```

وبالنسبة للثوابت التي تحمل النوع Date، فضع قيمة الوقت و/أو التاريخ بين الرمزين # و #:

```
Dim X As Date
Dim Y As Date

x = #1/29/2003#
Y = #2/15/2003 9:30:00 PM#
```

فكرة الثوابت المسماة شبيه بفكرة المتغيرات، إلا أن قيم الثوابت المسماة لا يمكن تعديلها وقت التنفيذ، وذلك لأنها تستبدل بقيمها أثناء عملية الترجمة للبرنامج، ويتم حفظها في ملف البرنامج النهائي (كـ EXE مثلا). استخدم الكلمة المحجوزة Const لتعريف ثابت جديد:

```
Const PROGRAMMER_NAME = "عباس السريع"
ArabicConsole.WriteLine(PROGRAMMER_NAME) ' عباس السريع
```

PDF Factory Pro trial version www.fastio.com

تحديد نوع الثابت أمر مفضل لزيادة السرعة، بينما يكون إلزامي إن فعلت العبارة Option Strict On:

```
Const PROGRAMMER_NAME As String = "عباس السريع"
```

عودة إلى الثوابت العدية، يمكنك كتابة الأعداد بالصيغة الست عشرية Hexadecimal أو الثمانية Octal باستخدام الرموز **&H** و **&O** –على التوالي- قبل العدد:

```
' صيغة ست عشرية
ArabicConsole.WriteLine(&HFF)  ' 255

' صيغة ثمانية
ArabicConsole.WriteLine(&O10)  ' 8
```

تذكر أن الأعداد –بشكل افتراضي- تكون من النوع Integer، لذلك لا تنسى استخدام الذيل المناسب للقيمة المناسبة، فالعدد التالي يفضل إسناد الذيل **"L"** له حتى نخرج بالنتيجة المناسبة:

```
ArabicConsole.WriteLine(&HFFFFFFFF)     ' -1
ArabicConsole.WriteLine(&HFFFFFFFFL)    ' 4294967295
```

أخيرا، لا يمكنك استخدام الصيغة الست عشرية Hexadecimal أو الثمانية Octal للأعداد العشرية:

```
' بودي ولكن للاسف غير ممكن
ArabicConsole.WriteLine(&HFF.25)
```

التركيبات والمصفوفات

عبر الزمن ومع الأيام، ستبدأ بتعريف أنواع خاصة بك في برامجك الجدية تعرف بالتركيبات، والتي يدعمها Visual Basic .NET بقوة. في هذا القسم من الفصل سأتحدث عن التركيبات من نوع Enums و التركيبات من نوع Structures، كما سأخصص فقرة كاملة حول المصفوفات.

التركيبات من نوع Enums

يمكنك تعريف نوع معين من أنواع المتغيرات بحيث تحصر مجال القيم التي تسندها إليها تعرف بالـ Enumeration. استخدم الكلمة المحجوزة Enum لتعريف تركيب جديد إما على مستوى

PDF

الوحدة البرمجية Module، خارج الوحدة البرمجية، أو داخل تركيب آخر ولكن من النوع Structure. هذا المثال عرفت فيه تركيب يمثل أيام الأسبوع:

```
Enum Day
    Saturday
    Sunday
    Monday
    Tuesday
    Wednesday
    Thursday
    Friday
End Enum
```

والآن يمكنك استخدام التركيب السابق وتعريف متغيرات جديدة منه:

```
Dim x As Day
Dim y As Day

x = Day.Friday
y = x
```

ملاحظة

تقنيا، تصنف التركيبات من النوع Enums ضمن الثوابت، فهي كالثوابت المسماة -التي تطرقت لها سابقا- حيث أن قيمها تستبدل أثناء عملية الترجمة.

عمليا، ستستخدم التركيبات من نوع Enums كوسيطات ترسلها إلى الإجراءات:

```
Sub ShowDay(ByVal CurrentDay As Day)

    If CurrentDay = Day.Friday Then
        ArabicConsole.WriteLine("اجازة")
    End If

    …
    …
End Sub
```

ثم ترسل إليها المتغيرات من نفس نوع التركيب أو قيم التركيب مباشرة:

```
Dim X As Day

X = Day.Friday

ShowDay(X)
ShowDay(Day.Friday)
```

تبدأ قيم عناصر التركيب من الرقم 0، مع ذلك يمكنك تخصيص قيم أخرى بكل انسيابية:

```
Enum Day
    Saturday = 10
    Sunday = 20
    Monday
    Tuesday
    Wednesday
    Thursday
    Friday
End Enum
```

مع العلم أن مقدار الزيادة لباقي عناصر التركيب هو واحد. أي أن Monday سيحمل القيمة 21، و Tuesday سيحمل القيمة 22، ... وهكذا.

جميع القيم التي عرفتها في التركيبات السابقة هي من النوع Integer، مع ذلك يمكنك Visual Basic .NET من تغييرها إلى Byte، Short، أو Long رغم أن مستندات NET. لا تنصحك بعمل ذلك إلا عند وجود سبب مقنع لعمل ذلك:

```
Enum Day As Long
    Saturday
    Sunday
    …
    …
End Enum
```

وللحديث عن مدى هذا النوع من المتغيرات، فاختصر عليك الكلام بالقول: ان عرفت التركيب باستخدام الكلمة المحجوزة Private، فان مدى هذا التركيب سيكون محصوراً داخل الوحدة البرمجية الذي عرف فيها هذا التركيب، أما إن استخدمت الكلمة المحجوزة Public أو حتى تجاهلتها فسيكون المدى شاملاً لباقي ملفات المشروع:

```
Module Module1
    ' عام
    Enum GlobalEnum
        Enum1
        Enum2
        …
    End Enum

    ' عام ايضا
    Public Enum GlobalEnum2
        Enum1
        Enum2
        …
    End Enum

    'على مستوى الوحدة البرمجية
    Private Enum PrivateEnum
        Enum1
        Enum2
        …
    End Enum
    …
    …
End Module
```

ملاحظة

يمكنك أيضا تعريف التركيبات من نوع Enum خارج الوحدات البرمجية، إلا انك لن تستطيع استخدام الكلمة المحجوزة Private في هذه الحالة. أما إن عرفت التركيبات من نوع Enum داخل تركيبات من نوع Structure فالوضع سيكون مثل ما كان عليه مع الوحدات البرمجية Modules.

المزيد أيضا، تستطيع استخدام الكلمة المحجوزة Friend والتي تماثل الكلمة المحجوزة Public، إلا أن الأولى لا تسمح لك استخدام التركيب من خارج حدود المشروع.

التركيبات من نوع Structures

يعرف هذا النوع من التركيبات بالأنواع **المعرفة من قبل المستخدم User Defined Types– (UDT)**، بحيث تمكنك من دمج أنواع مختلفة من المتغيرات وضمها في تركيب أو كتلة واحدة. استخدم الكلمة المحجوزة Structure لتعريف تركيب جديد:

```
Structure Person
    Dim Name As String
    Dim Age As Integer
End Structure
```

ثم تعرف متغيرات جديدة من هذا التركيب وتتعامل معها كالمتغيرات العادية:

```
Dim Turki As Person

Turki.Name = "تركي العسيري"
Turki.Age = 99

ArabicConsole.WriteLine(Turki.Name) ' تركي العسيري
ArabicConsole.WriteLine(Turki.Age)  ' 99
```

المزيد أيضا، يمكنك نسخ قيم التركيبات بانسيابية كاملة كما تفعل مع المتغيرات العادية، شريطة أن تكون التركيبات متطابقة:

```
Dim Turki2 As Person

Turki2 = Turki

ArabicConsole.WriteLine(Turki2.Name) ' تركي العسيري
ArabicConsole.WriteLine(Turki2.Age)  ' 99
```

لا تنسى أن التركيبات من نوع Structure يمكن أن تكون **متداخلة Nested** -أي يحتوي بعضها بعضا:

```
Structure Person
    Structure AddressStruct
        Dim City As String
        Dim Countrey As String
    End Structure
    Dim Name As String
    Dim Age As Integer
    Dim Address As AddressStruct
End Structure
```

الوصول إلى عناصر التركيب المحضون يتم من خلال التركيب الحاضن لها بكل منطقية:

PDF

```
Dim Turki As Person

Turki.Name = "تركي العسيري"
Turki.Age = 99
Turki.Address.City = "الظهران"
Turki.Address.Countrey = "المملكة العربية السعودية"
```

بالإضافة إلى المتغيرات، عليك معرفة أن التركيبات من نوع Structure في Visual Basic .NET هي تركيبات مطورة ومرنة جدا (مثل التركيبات الموجودة في لغة C++)، فهي تمكنك من تعريف عناصر إضافية في داخل التركيب كالطرق Methods والخصائص Properties:

```
Structure Person
    Dim Name As String
    Dim Age As Integer

    ' تعريف طريقة
    Sub ShowData()
        ArabicConsole.WriteLine(Name)
        ArabicConsole.WriteLine(Age)
    End Sub
End Structure
```

مرة أخرى، يمكنك الوصول إلى عناصر التركيب واستدعاء طرقه بنفس الطريقة الانسيابية:

```
Dim Turki As Person

Turki.Name = "تركي العسيري"
Turki.Age = 99

Turki.ShowData()
```

انظر أيضا

سأتناول الطرق والخصائص بشكل مفصل في الفصل الثالث **الفئات والكائنات**.

لا أريد أن أشتت تفكيرك الآن بموضوع الطرق والخصائص (فهي حديث الفصل الثالث كما ذكرت في المربع الأعلى)، ولكن دعني المح لك هنا أن المشيدات Constructers مدعومة بشكل مخفي في التركيبات من النوع Structures. يا الهي! ماذا تقصد يا تركي بكلمة مخفي؟! اقصد يا

PDF Factory Pro trial version www.fineprint.com

عزيزي أن الإجراء Sub New() معرف بشكل تلقائي في التركيب دون أن تـــراه. ولمـــاذا تـــم إخفائه؟ السبب يا سيدي تقني بحت ولا أود أن أبينه إلا في الفصول اللاحقة. حسنا وما الفائدة منه؟ الفائدة ببساطة إسناد قيم ابتدائية لمتغيرات التركيب، فلو حاولت إسناد القيم وقت التصريح كما فعلنا سابقا عند التصريح عن المتغيرات:

```
Structure MyStruct
    Dim x As Integer = 0
    Dim y As Integer = 10
    …
    …
End Structure
```

سيظهر لك المترجم رسالة خطأ تفيد بأنك لا تستطيع فعل ذلك (رغم أني لم أجد سبب منطقي مقنع لا يسمح لي بفعل ذلك)، وهنا يأتي دور المشيد المخفي Sub New() الذي يقوم بإسناد قيم ابتدائية للمتغيرات (0 للمتغيرات العددية، لا شيء للمتغيرات الحرفية، والقيمة Nothing للكائنات). مـــع ذلك، يمكنك تعريف مشيد Sub New() بنفسك عن طريق تطبيق مبدأ يعرف **بإعـــادة التعريـــف Overloading** (وهو حديث الفصل الثالث أيضا). لعمل ذلك، أضف وســـيطات Parameters إضافية مع الإجراء Sub New():

```
Structure Person
    Dim Name As String
    Dim Age As Integer

    ' Overload إجراء تمت اعادة تعريفه
    Sub New(ByVal PersonName As String)
        Name = PersonName
        ArabicConsole.WriteLine("تم تنفيذ المشيد")
    End Sub
End Structure
```

رغم أن الوظيفة الأساسية للإجراء Sub New() هي العمل كمـــشيد، إلا انـــه لـــن يـــتم استدعائه بمجرد إنشاء كائن من التركيب فيما لو صرحت عن متغير جديـــد بـــالطرق التقليديـــة، والدليل جرب هذا السطر:

```
' مع الاسف الشديد، لن يتم تنفيذ المشيد
Dim Turki As Person
```

PDF

ستلاحظ أن Visual Basic .NET لم يقم بتنفيذ ذلك المشيد، والسبب قد يبدو بديهيا إن تذكرت انه يوجد مشيد Sub New() آخر (لكنه مخفي) تم تنفيذه بدلا من مشيدنا الظريف. وحتى نبلغ مترجم اللغة أن عليه تنفيذ مشيدنا الجديد، بدلاً من المشيد المخفي علينا استخدام الكلمة المحجوزة New وإرسال الوسيطات التي توافق ذلك المشيد:

```
' سيتم تنفيذ المشيد بمجرد التصريح عن المتغير هنا
Dim Turki As New Person("تركي العسيري")
```

لماذا؟ كيف؟ وما السبب؟ كل هذه الاستفسارات سأتطرق لها في الفصل الثالث **الفئات والكائنات** بمشيئة الله، لذلك اطلب منك عزيزي القارئ أن لا تقلق نفسك كثيرا بالأشياء الغير مفهومه هنا، حيث أنها تتعلق بالكائنات وطريقة إنشائها، ولا أود نقل مواضيع الفصل الثالث هنا، فنحن ما زلنا نتعلم الأساسيات.

لننتقل إلى موضوع آخر يتمحور حول قابلية الوصول إلى عناصر التركيب، فجميع المتغيرات المحضونة في التركيبات السابقة عرفناها باستخدام الكلمة المحجوزة Dim، لذلك تمكنا من الوصول إلى عناصرها. ولكنك في بعض الأحيان قد تود تعريف متغيرات مخفية لا يمكن الوصول إليها إلا من داخل التركيب نفسه، لذلك عليك استخدام الكلمة المحجوزة Private أثناء التصريح عن متغير في داخل التركيب:

```
Structure Person
    Public Name As String              ' هنا Dim مثل Public
    Dim Age As Integer
    Private MotherName As String

    Sub Test()
        MotherName = "احم احم!"        ' يمكن الوصول إلى المتغير المخفي
        …                              ' من داخل التركيب فقط
        …
    End Sub
End Structure
```

الكلمات المحجوزة Public و Private في الشيفرة السابقة، تسمى **محددات الوصول Access Specifiers**. يوجد نوع ثالث من محددات الوصول يستخدم الكلمة المحجوزة Friend وظيفته تماثل وظيفة محدد الوصول Public، إلا أن عناصر التركيب من هذا النوع لا يمكن الوصول اليها من خارج حدود المشروع الحالي (الذي عرف فيه التركيب).

PDF Factory Pro trial version www.fastio.com

طولناها وهي قصيرة! اختم فقرة التركيبات من نوع Structure بتوضيح مدى و قابلية الرؤية لها والتي تماثل المدى و قابلية الرؤية للتركيبات من نوع Enums. وحتى أغنيك من عناء قلب الصفحات للبحث عنها، دعني أعيد صياغتها لك هنا: إن عرفت التركيب باستخدام الكلمة المحجوزة Private، فان قابلية الرؤية لهذا التركيب ستكون محصورة داخل الوحدة البرمجية الذي عرف فيها هذا التركيب، أما إن استخدمت الكلمة المحجوزة Public (أو حتى تجاهلتها) فستكون قابلية الرؤية شاملة لباقي ملفات المشروع. مع العلم انك تستطيع استخدام الكلمة المحجوزة Friend أيضا.

المصفوفات

يمكنك Visual Basic .NET من تعريف المصفوفات سواء كانت أحادية البعد أو متعددة الأبعاد والتي قد تصل إلى 32 بعدا:

```
Dim OneDim (9) As Integer            ' عناصر 10
Dim TwoDims (1, 1) As String         ' عناصر 4 = 2 * 2
```

يمكنك فورا البدء بعملية إسناد القيم لها –كما تفعل مع المتغيرات العادية- مع العلم أن بدء الترقيم لفهرس المصفوفات يبدأ بالرقم 0:

```
OneDim (0) = 100
OneDim (1) = 200
…
OneDim (9) = 900

TwoDims (0, 0) = "تركي"
TwoDims (0, 1) = "العسيري"
TwoDims (1, 0) = "عباس"
TwoDims (1, 1) = "السريع"
```

وان كنت مستعجلاً في عملية إسناد القيم، فإن هذا متاح لك في سطر التصريح مباشرةً، شريطة عدم تحديد عدد عناصر المصفوفة:

```
Dim OneDim() As Integer = {1, 2, 3, 4, 5, 6, 7, 8, 9}
Dim TwoDims(,) As String = {{"تركي", "العسيري"}, {"عباس", "السريع"}}
```

المصفوفات السابقة تسمى مصفوفات ديناميكية Dynamics Arrays لأننا لم نحدد عدد عناصرها، الميزة في هذا النوع من المصفوفات هو إمكانية تغيير حجمها من وقت لأخر باستخدام

PDF

الكلمة المحجوزة ReDim، مع الإشارة إلى أن عناصر المصفوفة المعاد تغيير حجمها باستخدام ReDim سوف تلغى:

```
ReDim OneDim (99)
ReDim TwoDims (10, 10)

ArabicConsole.WriteLine ( OneDim(0) ) ' 0
```

مع ذلك، يمكنك تغيير حجم المصفوفة دون المخاطرة بفقد بياناتها باستخدام الكلمة المحجوزة Preserve، ولكن –مع الأسف الشديد- لا يمكن تغيير إلا عدد عناصر البعد الأخير فقط في هذه الحالة:

```
' ممكن جدا
ReDim Preserve OneDim (500)
ReDim Preserve TwoDims (10, 500)

' رسالة خطأ
ReDim Preserve TwoDims (500, 500)
```

ومع الأسف الشديد أيضا، لا يمكنك تغيير عدد أبعاد المصفوفة الديناميكية سواء استخدمت Preserve أو لم تستخدمها:

```
' رسالة خطأ
ReDim Preserve OneDim (500, 500)
ReDim TwoDims (100)
```

في المقابل، تستطيع تدمير المصفوفة الديناميكية لتحرير المساحة في الذاكرة في أي وقت تريده باستخدام الأمر Erase:

```
Erase OneDim
Erase TwoDims
```

على صعيد آخر، المصفوفات تعتبر من البيانات المرجعية Reference Type فلا يمكنك نسخ قيمها باستخدام معامل إسناد القيم "=". لي عودة حول المصفوفات في الفصل السادس **الفئات الأساسية**، أما الآن دعني اعرض لك كيف ننسخ قيمة مصفوفة إلى أخرى باستخدام الطريقة Clone():

```
Dim X () As Integer = {1, 2, 3, …}
Dim Y () As Integer

' Y نسخ المصفوفة X إلى
Y = X.Clone()
```

انهي حديثي عن المصفوفات بذكر الدالة UBound() التي تعود بـــرقم فهـــرس العنـــصر الأخير للمصفوفة، والدالة LBound() برقم الفهرس للعنصر الأول:

```
For counter = LBound(OneDim) To UBound(OneDim)
    …
    …
Next
```

وبالنسبة للمصفوفات المتعددة الأبعاد، يتوجب عليك إرسال رقم البعد الذي تود معرفة فهرسته:

```
UBound(OneDim)        ' البعد الاول
UBound(OneDim, 1)     ' البعد الاول
UBound(OneDim, 2)     ' البعد الثاني
```

الإجراءات والدوال

يمكنك Visual Basic .NET من تعريف الإجراءات إما بالكلمة المحجوزة Sub أو Function، حيث أن استخدامك للكلمة المحجوزة الثانية يجعل الإجراء قادرا على العودة بقيمة نوعها تحـــدده عند تعريف الإجراء:

```
' إجراء لا يعود بقيمة
Sub MySub()
    ArabicConsole.WriteLine ("إجراء لا يعود بقيمة")
End Sub

' دالة تعود بقيمة من
' Long النوع
Function Abs (ByVal X As Integer) As Long
    If X < 0 Then
        Return -X
    Else
        Return X
    End If
End Function
```

عند استدعاء الإجراءات، عليك كتابة الأقواس حتى لو لم توجد وسيطات Parameters ترسلها لها:

```
' سيقوم المحرر باضافة الاقواس ان لم تضفها
MySub ()

ArabicConsole.WriteLine (Abs (-5)) ' 5
```

تستطيع إنهاء الإجراء في أي وقت باستخدام الأمر Exit Sub إن تم التعريف باستخدام Sub أو Exit Function إن تم التعريف باستخدام Function:

```
Function Abs (ByVal X As Integer) As Long
    If X = 0 Then
        Exit Function
    End If
    …
    …
End Function
```

ملاحظة

إن استخدمت الأمر Exit Function دون تعيين قيمة للدالة، فستعود الدالة بقيمة 0 إن كانت عددية، لا شيء إن كانت حرفية، أو Nothing إن كانت كائنية.

وبالنسبة لقابلية الرؤية فهي إما تكون Private، أو Friend، أو Public كالمتغيرات.

الإرسال بالمرجع أو القيمة

بشكل افتراضي، الوسيطات التي يستقبلها الإجراء هي متغيرات أرسلت بالقيمة، وان أردت من إجراءاتك أن تستقبل قيمها بالمرجع لتتمكن من تعديل قيم المتغيرات المرسلة، استخدم الكلمة المحجوزة ByRef:

```
' هنا بالقيمة ولن تتأثر المتغيرات المرسلة
Sub swapByVal(ByVal a As Integer, ByVal b As Integer)
    Dim temp As Integer
    temp = a
    a = b
    b = temp
End Sub
```

```
' اما هنا بالمرجع وستتأثر المتغيرات المرسلة
Sub swapByRef(ByRef a As Integer, ByRef b As Integer)
    Dim temp As Integer
    temp = a
    a = b
    b = temp
End Sub
```

يتضح الفرق في الشيفرة التالية:

```
Dim A As Integer
Dim B As Integer

A = 10
B = 20

' ارسال بالقيمة
swapByVal (A, B)

ArabicConsole.WriteLine (A) ' 10
ArabicConsole.WriteLine (B) ' 20

' ارسال بالمرجع
swapByRef (A, B)

ArabicConsole.WriteLine (A) ' 20
ArabicConsole.WriteLine (B) ' 10
```

وللحديث عن المسائل التقنية، سأبدأ بعملية إرسال المتغيرات بالقيمة، فهي أبطأ من الإرسال بالمرجع وذلك لأنه سيتم إنشاء نسخة من البيانات المرسلة في كل مرة تستدعي الإجراء. من ناحية أخرى، توجد ميزة في عملية إرسال المتغيرات بالقيمة، وهي عدم التأثير على باقي أجزاء البرنامج أن قمت بتعديل قيمها بطريق الخطأ. أما الإرسال بالمرجع، –كما قلت قبل قليل- هو أسرع من الإرسال بالقيمة، فأنت ترسل مؤشر للمتغير مما يمكنك من تعديل قيمة المتغير المرسل.

ملاحظة

بالنسبة للمتغيرات المرجعية Reference Type Variables المرسلة إلى الإجراءات، ستتأثر بالتغييرات حتى وان أرسلت بالقيمة، أي باستخدام الكلمة المحجوزة ByVal.

تخصيص الوسيطات المرسلة

المزيد من التخصيص حول الوسيطات يوفره لك Visual Basic .NET، إذ يمكنك من التصريح عن الوسيطات الاختيارية Optional والغير محدودة العدد ParamArray.

الوسيطات الاختيارية:

أحيانا تود من إجراءاتك أن تكون مرنة بما فيه الكفاية بحيث لا تشترط توافق عـــدد المتغيـــرات المرسلة مع عدد وسيطات الإجراء، تستطيع استخدام الكلمة المحجوزة Optional قبل كل وسيطة اختيارية مع ضرورة تحديد قيمة افتراضية لها في حالة عدم إرسال قيمة للإجراء:

```
Sub MySub(Optional ByVal X As Integer = -1)
    If X = -1 Then
        ArabicConsole.WriteLine ("لم ترسل قيمة")
    End If
    …
    …
End Sub
```

نقطة أخرى، لا يمكنك استخدام الكلمة المحجوزة Optional إلا في الوسيطات الأخيرة (أي الموجودة في جهة اليمين) فلا يمكن لوسيطة اختيارية أن تسبق وسيطة عادية:

```
' هكذا ممكن
Sub MySub(ByVal Y As Byte, Optional ByVal X As Integer = -1)
    …
    …
End Sub

' انسى هذه الفكرة
Sub MySub(Optional ByVal X As Integer = -1, ByVal Y As Byte)
    …
    …
End Sub
```

الوسيطات غير محدودة العدد:

في هذه الحالة فانك لا تحدد عددا معينا للوسيطات التي يستقبلها الإجراء، لان القيم سترسل وتحفظ في مصفوفة تعرفها باستخدام الكلمة المحجوزة ParamArray:

```
Function Sum(ByVal ParamArray Nums() As Integer) As Integer
    Dim counter As Integer

    For counter = 0 To UBound(Nums)
        Sum += Nums(counter)
    Next
End Function
```

تطبيقيا، كل هذه الإستدعاءات صحيحة باستثناء الأخير الذي يتوقع انه اختياري:

```
ArabicConsole.WriteLine ( Sum (1) )                  ' 1
ArabicConsole.WriteLine ( Sum (2, 2) )               ' 4
ArabicConsole.WriteLine ( Sum (1, 2, 3, 4, 5) )      ' 15

' خطأ هنا
ArabicConsole.WriteLine ( Sum (1, , 3) )
```

تجاوز الحدود مع Windows API

إن كنت لا تعرف ما هي إجراءات Windows API، فاعتبر نفسك مبرمج محظوظ جدا! وبما انني لست من المبرمجين الشجعان، فلن أتحدث عنها. أما إن كنت من مبرمجي Windows المخضرمين، فتستطيع التصريح عن إجراءات Windows API لتتجاوز حدود عالم إطار عمل NET Framework.. لعمل ذلك، صرح عن الإجراء باستخدام الكلمة المحجوزة Declare مع تحديد نوع صفحة المحارف إما Ansi، Unicode، أو Auto. إن استخدمت Auto، سيتم تحويل الحروف إلى Unicode تحت جميع الأنظمة باستثناء Windows 98 و Windows ME حيث ستتحول إلى Ansi:

```
Module Module1

    Declare Auto Function GetUserName Lib "advapi32.dll" Alias _
        "GetUserNameA" (ByVal lpBuffer As String, _
         ByRef nSize As Integer) As Integer

    Sub Main ()
        …
        …
        GetUserName (x, y)
    End Sub

End Module
```

التفرع والتكرار

الخوارزميات من الصعب تطبيقها برمجيا دون استخدامك لجمل التفرع وحلقات التكرار. في هذا القسم من الفصل سنتوغل في عبارات التفرع If ... Then و Select Case، كما سأتطرق إلـــى الحلقات التكرارية المختلفة المتوفرة في لغة البرمجة Visual Basic .NET.

التفرع باستخدام If ... Then

استخدم الكلمة المحجوزة If لتضيف إليها جملة شرطية ثم تلحقها بكلمة Then، ولا تنسى استخدام End If إن وزعت أوامر الشرط في أكثر من سطر (وهو المفضل):

```
' في سطر واحد
If X = 0 Then Y = 1
If X = 1 Then X = 2 : Y = 4
If Y = 1 Then X = 0 Else X = 2

' يفضل توزيعها هكذا
If X = 0 Then
    Y = 1
End If

If X = 1 Then
    X = 0
    Y = 4
End If

If Y = 1 Then
    X = 0
Else
    X = 2
End If
```

ملاحظة

المعامل ":" عكس المعامل "_" بحيث يمكنك من دمج عدة أوامر فـي سطر واحد.

ذكرت مرتين أن المفضل استخدام الصيغة الموزعة وإغلاقها بـــ End If حيث أنها تسهل عليك قراءة وفهم منطق التفرع خاصة إن كانت جمل الشرط متداخلة، ركز معي يا حلو في هـــذه الجمل:

```
If X = 0 Then
    If Y = 0 Then
        X = 100
    End If
Else
    Y = 1
End If
```

قد يأتي شخص مصمم على اختصار الجمل السابقة في سطر الواحد (فهو مبرمج محترف كما يدعي) ويكتب شيئا مثل:

```
If X = 0 Then If Y = 0 Then X = 100 Else Y = 1
```

أنصحك بعدم الاستماع له مدى الدهر مادام الحمام يغرد! فمنطق التفرع في جملة أخينا في الله خاطئة، حيث أن كلمة Else الأخيرة تتبع للشرط الثاني وليس الأول، أي أن Visual Basic .NET سيفهمها على أنها:

```
If X = 0 Then
    If Y = 0 Then
        X = 100
    Else
        Y = 1
    End If
End If
```

وحتى لا نضيع وقتنا الثمين في مثل هذه السجالات، سأغلق الموضوع بنصيحة: استخدم الصيغة المفرقة If … Then … End If دائما حتى لو كان جواب الشرط يحتوي على أمر واحد فقط.

أدوات الربط المنطقي:

يمكنك استخدام أدوات الربط المنطقي (And، Or، Not …الخ) بطلاقة كاملة كما تفعل مع لغات البرمجة الأخرى، حيث أنها مدعومة في Visual Basic .NET:

```
If x > 0 And t < 1 Then
    …
    …
End If

If Not Y > 1 Then …
```

دعنا نلهو قليلا في علم المنطق الرياضي، واطلب منك التركيز في الشرط التالي:

```
If X <> 0 And 10 \ X = 2 Then
```

لغويا، الشرط السابق يختبر قيمة المتغير X ما إذا كانت تساوي الصفر أم لا، وان كانت لا تساوي صفر فستختبر ناتج القسمة. مع ذلك، فإن الشيفرة السابقة ستظهر رسالة خطأ إن كانت قيمة المتغير X تساوي صفر، والسبب أن Visual Basic .NET سيجري عملية القسمة دائما. منطقيا، يفترض من Visual Basic .NET أن لا يتعب نفسه ويجري عملية القسمة إن كانت قيمة المتغير X تساوي صفر، والسبب أن الشرط سيكون دائما False (خاطئ). تقنيا، المعامل And يقوم باختبار جميع الجمل الشرطية التي حوله، لذلك ينصح باستخدام المعامل AndAlso في مثل هذه الحالات:

```
If X <> 0 AndAlso 10 \ X = 2 Then
```

إن كانت قيمة المتغير X في الجملة السابقة تساوي صفر، فان Visual Basic .NET لن يكمل عملية التحقق من عملية القسمة مما يجنبنا ظهور رسالة الخطأ.

إلى جانب المعامل AndAlso يوجد معامل آخر هو OrElse والذي سيتخطى الشرط الثاني إن كان الأول True:

```
' لن يتم التحقق من الشرط الثاني
' ان كان العدد في المتغير X
' موجب
If X > 0 OrElse Y < 0 Then …
```

عليك معرفة أن المعاملات AndAlso و OrElse تتعامل مع القيم المنطقية فقط، وان استخدمت الأعداد (ستجرى عملية التحويل التلقائي في حالة Option Strict Off) فستعتبر اي قيمة غير الصفر True، بينما المعاملات And و Or تختبر البتات التي تكون العدد –لذلك تسمى bit-wise operators، فجملة الشرط التالية:

```
x = 3
y = 12
If x <> 0 And y <> 0 Then … ' True
```

يمكنك اختصارها بالمعامل AndAlso لتعطي نتيجة مماثلة:

```
' عملية المقانة تختبر القيم
' True And True = True
If x AndAlso y Then …
```

بينما يؤدي استخدام المعامل And إلى اختبار البتات المكونة للأعداد، لتعطي نتيجة خاطئة:

```
' عملية المقارنة تختبر البتات
' 0011 And 1100 = 0000 (False)
If x And y Then …
```

أخيرا، يمكنك اختبار مجموعة جمل شرطية وتنفيذ أوامر معينه إن أخفقت كلها باســتخدام ElseIf:

```
If X = 1 Then
    …
ElseIf X = 2 Then
    …
ElseIf X = 3 Then
    …
Else
    …
End If
```

التفرع باستخدام Select Case

التفرع باستخدام Select Case يمكن تطبيقه بسهولة تامة:

```
Dim X As Integer
…
…
Select Case X
    Case 1
        ArabicConsole.WriteLine ("محرم")
    Case 2
        ArabicConsole.WriteLine ("صفر")
    …
    …
    Case 12
        ArabicConsole.WriteLine ("ذو الحجة")
    Case Else
        ArabicConsole.WriteLine ("غير معرف")
End Select
```

PDF Factory Pro trial version www.fineprint.com

تكمن قوة العبارة Select Case في تطبيق المعاملات المنطقية أو تحديد مجـــالات للقــيم المطلوب التحقق منها:

```
Dim Grade As Integer
…
…
Select Case Grade
    Case Is < 60
        ArabicConsole.WriteLine ("راسب")
    Case 60 To 69
        ArabicConsole.WriteLine ("مقبول")
    Case 70 To 79
        ArabicConsole.WriteLine ("جيد")
    Case 80 To 89
        ArabicConsole.WriteLine ("جيد جدا")
    Case Is >= 90
        ArabicConsole.WriteLine ("ممتاز")
End Select
```

المزيد أيضا، يمكنك استخدام الفاصلة بمرونة كاملة، وبنفس المنطق السابق:

```
Dim Letter As Char
…
…

Select Case Letter
    Case "A"c To "Z"c, "a"c To "z"c
        ArabicConsole.WriteLine ("حرف ابجدي")
    Case "0"c To "9"c
        ArabicConsole.WriteLine ("عدد")
    Case "."c, ":"c, " "c, ";"c, "?"c
        ArabicConsole.WriteLine ("رمز")
    Case Else
        ArabicConsole.WriteLine ("غير معروف")
End Select
```

الفاصلة التي استخدمناها في المثال السابق تمثل أداة الربط Or المنطقية:

```
Select Case True
    Case x > 0, Y < 0
    ' تعادل
    ' If (X > 0) Or (Y < 0)
    …
End Select
```

```
Select Case False
    Case x > 0, Y < 0
    ' تعادل
    ' If ( Not (X > 0) ) Or ( Not (Y < 0) )
    …
End Select
```

الحلقات التكرارية

حدد القيمة الابتدائية والنهائية لحلقة For … Next:

```
Dim counter As Integer

For counter = 2 To 4
    ArabicConsole.WriteLine(counter) ' ثلاث مرات
Next
```

تستطيع التحكم في مقدار الزيادة أو النقصان باستخدام Step:

```
For counter = 5 To 1 Step -1
    …
Next
```

ضع في اعتبارك أن متغير الحلقة سيزيد أو ينقص بالمقدار المحدد حتى بعد نهاية الحلقة:

```
For counter = 5 To 1 Step -1
    …
Next
' 1 قيمة العداد 0 وليس
ArabicConsole.WriteLine(counter) ' 0
```

المزاح مع متغير الحلقة داخل الحلقة فيه شيء من الخطر، فعدد مرات التكرار للحلقة التالية هو واحد فقط:

```
For counter = 1 To 100
    counter = 100
    …
Next
```

وقبل انتهاء المدة الافتراضية للحلقة، يمكنك قطعها وإنهائها بعبارة Exit For:

```
For counter = 1 To 50
    …
        If y = 10 Then
            Exit For
        End If
    …
Next
```

حلقة أخرى جميلة جدا تعرف بـــ For Each تطبــق علـــى المـــصفوفات Arrays أو المجموعات Collections:

```
Dim x(5) As Integer
Dim y As Integer

…
…

For Each y In x
    ArabicConsole.WriteLine(y)
Next
```

ملاحظة

إن كنت مبتدءاً، فلا تستخدم الحلقة For Each كثيرا هـذه الأيـام حتـى تصل إلى الفصل الخامس **الواجهات، التفويض، والمواصفات**، حيـث ستجد المزيد من التفاصيل عن استخدام هذه الحلقة.

وعند الحديث عن الحلقات اللانهائية، فلن تجد أفضل من حلقة Do … Loop المرنة جدا، حيث تمكنك من وضع الشرط إما في أعلى الحلقة أو في أسفلها (ليتم تنفيذ أوامر الحلقة مرة واحدة على الأقل في حالة وضع الشرط أسفل الحلقة). إن استخدمت الكلمة المحجــوزة Until، ســيتم تكرار الحلقة حتى يصبح الشرط True، أما إن كانت الكلمة المحجوزة While هـــي المرافقـــة، فسيتم تكرار الحلقة ما دامت قيمة الشرط True:

```
Do Until MsgBox("انهي الحلقة؟", MsgBoxStyle.YesNo) = MsgBoxResult.Yes
    …
Loop

Do While MsgBox("انهي الحلقة؟", MsgBoxStyle.YesNo) = MsgBoxResult.No
    …
Loop
```

PDF

أخيرا، تستطيع في أي لحظة الخروج من الحلقة باستخدام العبارة Exit Do.

التبديل بين For … Next و Do … Loop:

تستطيع تحويل حلقة For … Next إلى حلقة Do … Loop والعكس صحيح، لكن عليك الانتباه إلى ان القيم التي تحددها في بداية الحلقة For … Next تمثل عدد التكرار حتى وان تغيرت داخل الحلقة، فبالرغم من أن الحلقتين التاليتين متشابهتين:

```
A = 5

' For … Next حلقة
For counter = 1 To A
    …
Next

' Do … Loop تحويلها إلى
counter = 1
Do
    …
    counter = counter + 1
Loop Until counter > A
```

إلا أن الاختلاف سيظهر في حال ما إذا تم تغيير قيمة المتغير A، فالحلقة الأولى For … Next سيتم تنفيذها دائما خمس مرات حتى وان تغيرت قيمة المتغير A في داخل الحلقة، بينما تغيير قيمة المتغير يؤثر بشكل كبير على عدد مرات تكرار الحلقة الأخرى Do … Loop.

مجالات الأسماء Namespaces

الفكرة من مجالات **الاسماء Namespaces** تقتضي توزيع الأسماء المتشابهة لمعرفات البرنامج (كأسماء الفئات Classes، الوحدات البرمجية Modules، التركيبات Structures … الخ) إلى كتل مختلفة تسمى مجال أسماء Namespace، بحيث تسهل عليك ترتيب أسمائها على مجموعات، وتمكنك أيضا من تعريف أسماء متشابهة لمعرفات مختلفة. فلو عرفنا تركيب للفأر باسم Mouse:

```
Structure Mouse
    …
    …
End Structure
```

PDF

فلا يمكننا استخدام نفس الاسم لتعريف تركيب آخر -يمثل جهاز الفأرة- بنفس الاسم Mouse. لذلك سنقوم بتعريف مجالات أسماء مختلفة.

تعريف مجال أسماء

قبل أن تبدأ بتعريف مجالات أسماء خاصة بك، عليك معرفة أن المشروع Project الحالي الذي تصممه قد عرف مجال اسم جديد، ويكون اسمه -بشكل مبدئي- نفس اسم المشروع. يمكنك تغيير مجال الاسم من نافذة Project Property Pages، ومن ثم كتابة اسم مجال الأسماء في خانة Root namespace (**شكل 2-5**).

شكل 2-5: تسمية مجال الأسماء الجذري للمشروع.

هذا الاسم الذي اخترته يمثل مجال الأسماء الجذري والرئيسي للمشروع الحالي، وجميع المعرفات ومجالات الأسماء الأخرى تابعة أو داخلة -إن صح التعبير- ضمن حيز هذا المجال.

برمجيا، يمكن تعريف المزيد من مجالات الأسماء في داخل مشروعك باستخدام الكلمة المحجوزة Namespace وتذييل المجال بالعبارة End Namespace:

```
Namespace Devices
    …
    …
End Namespace
```

PDF Factory Pro trial version www.fastio.com

يمكنك البدء بإضافة كل المعرفات التي ترغب بحضنها داخل هذا المجال، والمعرفات التي يمكنك تعريفها في داخل مجال الأسماء هي إما أن تكون فئات Classes، أو وحدات برمجية Modules، أو تركيبات Structures، أو واجهات Interfaces أو تركيبات من النوع Enumd فقط:

```
Namespace Devices
    Structure Mouse
        …
        …
    End Structure

    Structure Printer
        …
        …
    End Structure
…
…
End Namespace

Namespace Animals
    Structure Mouse
        …
        …
    End Structure

    Structure Cat
        …
        …
    End Structure
…
…
End Namespace
```

انظر أيضا

سيتم الحديث عن الفئات Classes في الفصل الثالث **الفئات والكائنات**، والواجهات Interfaces في الفصل الخامس **الواجهات، التفويض، والمواصفات**. أما الوحدات البرمجية Modules والتركيبات - سواء كانت Enums أو Structures فقد فصلتها سابقا في هذا الفصل.

PDF

أخيرا، مجالات الأسماء ممكن أن تكون متداخلة Nested:

```
Namespace Devices
    Namespace Inputs
        Structure Mouse
            …
            …
        End Structure

        Structure Keyboard
            …
            …
        End Structure
        …
        …
    End Namespace

    Namespace Outputs
        Structure Monitor
            …
            …
        End Structure

        Structure Printer
            …
            …
        End Structure
        …
        …
    End Namespace
…
…
End Namespace
```

الوصول إلى عناصر مجال الأسماء

كل ما هو مطلوب منك تحديد اسم مجال الأسماء ومن ثم ذكر المعرف الذي ترغب في استخدامه:

```
Dim X As Animals.Mouse
Dim Y As Devices.Mouse
…
…
```

وبالنسبة لمجالات الأسماء المتداخلة، عليك ذكر جميع المجالات الحاضنة لها، وبنفس الترتيب المنطقي الذي تتبعه للوصول إلى عناصر تركيبات Structures متداخلة:

```
Dim X As Devices.OutPuts.Printer
Dim Y As Devices.OutPuts.Screen
Dim Z As Devices.Inputs.Keyboard
```

استخدم الاسم الكامل لمجال الأسماء إن كنت خارج المجال فقط (كما في الأمثلة الســابقة)، أما إن كنت داخل المجال فلا يوجد داعي لتحديد اسم المجال الحالي:

```
' Devices.Inputs داخل مجال الاسماء
Dim X As Keyboard
Dim Y As Mouse
Dim Z As OutPuts.Printer ' هنا استخدمت مجال خارجي اخر
```

استيراد مجال أسماء باستخدام Imports

يقصد بجملة استيراد مجال أسماء أي تضمين مجال أسماء معين ودمجه في مجال الأسماء الحالي بحيث يمكنك الوصول إلى جميع عناصره دون الحاجة للالتزام بالصيغة الكاملة لاســم المجــال، فالشيفرة التالية:

```
Dim X As Devices.OutPuts.Printer
Dim Y As Devices.OutPuts.Screen
Dim Z As Devices.Inputs.Keyboard
```

يمكنك اختصارها باستيراد المجال Devices باستخدام الكلمة المحجوزة Imports:

```
Imports MyNameSpace.Devices
…
…
Dim X As OutPuts.Printer
Dim Y As OutPuts.Screen
Dim Z As Inputs.Keyboard
```

بل تستطيع اختصارها أكثر من ذلك أيضا باستيراد المجالات الفرعية:

```
Imports MyNameSpace.Devices.OutPuts
Imports MyNameSpace.Devices.Inputs
…
…
Dim X As Printer
Dim Y As Screen
Dim Z As Keyboard
```

PDF

ملاحظة

عند استيراد مجال أسماء باستخدام Imports، لابد من كتابة الاسم الكامل لمجال الأسماء (بما في ذلك اسم مجال الأسماء الذي يحتضنه). ففي الشيفرات السابقة، استخدمنا مجال الأسماء الجذري للمشروع MyNameSpace في كل مرة استوردنا فيها مجال أسماء.

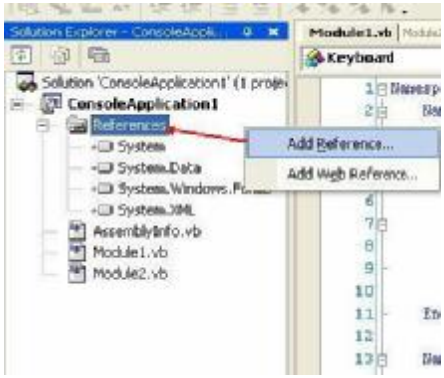
قد تفتح شركة استيراد وتصدير في احد الأيام، وتحاول استيراد جميع مجالات الأسماء في برامجك، ولكنك ستصاب بخيبة أمل كبيرة إن حدثت تعارضات، فلو حاولت استيراد هذين المجالين:

```
Imports MyNameSpace.Animals
Imports MyNameSpace.Devices.Inputs

Dim X As Mouse
```

سيظهر لك المترجم رسالة خطأ بسبب تعارض اسم التركيب Mouse في كلا المجالين. وان كان هذا سيسبب خسارة لشركة الاستيراد والتصدير الخاصة بك، فيمكنك الالتفاف حول هذا التعارض بتعريف مجال أسماء مؤقت:

```
Imports MyNameSpace.Animals
Imports tmp = MyNameSpace.Devices.Inputs

Dim X As Mouse
Dim Y As tmp.Mouse
```

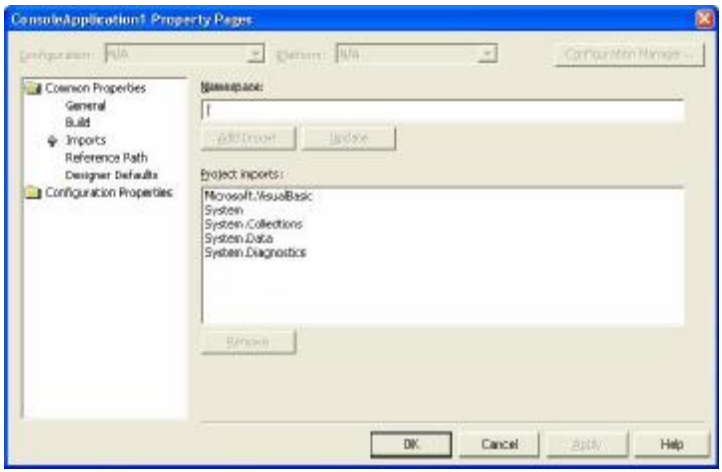
أخيرا، إن أردت استيراد مجالات أسماء لمشاريع وبرامج أخرى (كمكتبة فئات .NET Framework) عليك إضافة مرجع لهذه المجالات في خانة المراجع من نافذة مستكشف المشروع Solution Explorer (**شكل 2-6 بالصفحة المقابلة**).



شكل 2-6: إدراج مراجع إضافية لمجالات أسماء أخرى في المشروع الحالي.

استيراد مجال أسماء دون استخدام Imports

طريقة غير برمجية أخرى تمكنك من استيراد مجالات أسماء (أي دون الحاجة لاستخدام عبارة Imports) وذلك بإضافة جميع مجالات الأسماء التي تود استيرادها في برنامجك عن طريق خانة التبويب Imports في صندوق الحوار Project Property Pages (**شكل 2-7**).



شكل 2-7: استيراد مجال أسماء دون استخدام Imports.

الميزة في هذه الطريقة، هي أن عملية ستشمل كآفة ملفات المشروع وليس كاستخدام الكلمة المحجوزة Imports والتي تشمل الملف المسطورة فيه فقط.

كان هذا الفصل نهاية البداية لإتقان لغة البرمجة Visual Basic .NET. تبقى لنا مجموعة من المواضيع الأخرى كالوراثة Inheritance، الواجهات Interfaces، المواصفات Attributes، والتفويض Delegates لنكمل مرحلة تعلم الأساسيات. ولكن قبل ذلك، من المهم جدا استيعاب فكرة **الفئات والكائنات** عنوان الفصل التالي.

PDF Factory Pro trial version www.fastio.com

الفئات والكائنات

عند الحديث عن OOP، قد تكون تعاني من عقدة النقص من جملة Visual Basic بين قبائل المبرمجين، أما مع Visual Basic .NET فارفع راسك لفوء أوي أوي أوي! حيث أن Visual Basic .NET لغة برمجة كائنية التوجه OOP حقيقية داعمة لكل سمات لغات OOP الأخرى (باستثناء مبدأ **الوراثة المتعددة Multiple Inheritance** والذي أيضا لا تدعمه أي لغة .NET. أخرى إلا Visual C++ على حد علمي لحظة كتابة هذه السطور).

إن سئمت يوما من الأيام من هذا الكتاب وقررت حرق صفحاته، فدعني اطلب منك عزيز القارئ الاحتفاظ بهذا الفصل على الأقل، حيث انه أهم فصول الكتاب والذي ستحتاجه طيلة حياتك البرمجية مع إطار عمل .NET Framework. وكنصيحة أخوية، لا تنتقل إلى الفصول الأخرى حتى تتقن هذا الفصل إتقان صحيح وتحاول تطبيق كل ما ستتعلمه. مرة أخرى، سأفترض أن لديك خلفية –ولو متواضعة- في البرمجة كائنية التوجه OOP حيث سأبدأ مباشرة في التطبيق دون تقديم مسائل نظرية. حظا سعيدا!

مدخلك السريع للفئات

سيكون هذا القسم من الفصل مدخلك السريع لفهم فكرة الفئات وطرق بنائها. بشكل مبدئي، الفئات Classes في Visual Basic .NET مشابهة إلى حد كبير مع التركيبات من النوع Structures، فلو كان لدينا هذا التركيب:

```
Structure Person
    Dim Name As String
    Dim Age As Integer
End Structure
```

يمكنك تحويله إلى فئة باستخدام الكلمة المحجوزة Class عوضا عن Structure:

```
Class Person
    Dim Name As String
    Dim Age As Integer
End Class
```

مع ذلك، فإن نقطة الاختلاف الرئيسية بين الفئات والتركيبات هي أن الفئات من النوع المرجعي Reference Type بينما التركيبات من النوع ذات القيمة Value Type. وبالنسبة لقابلية الوصول لمحتويات الفئة، عليك معرفة أن قابلية الوصول الافتراضية في الفئات هي Private أما مع التركيبات فهي Public، لذلك عليك تحديد محدد الوصول بشكل واضح:

```
Class Person
    ' Public محدد الوصول
    Public Name As String
    Public Age As Integer
End Class
```

والآن يمكنك تعريف متغير (يسمى **مؤشر** إلى كائن في هذه الحالة) من الفئة Person السابقة والبدء بإسناد قيمه، ولكن عليك استخدام الكلمة المحجوزة New عند التصريح عن المؤشر:

```
Dim Turki As New Person()

Turki.Name = "تركي العسيري"
Turki.Age = 99

ArabicConsole.WriteLine(Turki.Name)   ' تركي العسيري
ArabicConsole.WriteLine(Turki.Age)    ' 99
```

أما إن أنشأت إجراء (سواء Sub أو Function) داخل هذه الفئة، فمحدد الوصول الافتراضي هو Public كما في التركيبات:

```
Class Person
    Sub PublicMethod ()                ' Public
        …
        …
    End Sub

    Public Sub PublicMethod2 ()        ' Public
        …
        …
    End Sub
```

PDF Factory Pro trial version www.fastio.com

```
    Friend Sub FriendMethod ()        ' Friend
        …
        …
    End Sub

    Private Sub PrivateMethod ()      ' Private
        …
        …
    End Sub
    …
    …
End Class
```

على عكس التركيبات، يمكن للفئات أن لا تحتوي على أية أعضاء:

```
' ممكن جدا
Class PersonClass

End Class

' رسالة خطأ
Structure PersonStructure

End Structure
```

ملاحظة

قد تستغرب مدى الجدوى من تعريف فئة لا تحتوي على أية أعضاء، ولكنك قد تفعل ذلك يوما من الأيام إن رغبت بتعريف فئة لاشتقاق واجهات Interfaces من فئات أخرى، سترى لاحقا في الفصل الخامس **الواجهات، التفويض، والمواصفات**.

لا تنسى انه يمكن للفئات أن تكون متداخلة Nested أيضا:

```
Class Car
    Class Engine
        Public Cylinder As Integer
    End Class

    Public Model As String
    Public CarEngine As New Engine()
End Class
```

PDF

يمكنك الوصول إلى جميع عناصر الفئة والفئة المحضونة بكل منطقية، شـــريطة اســـتخدام New عند تعريف متغير من فئة:

```
Dim BMW As New Car()

BMW.Model = "2003"
BMW.CarEngine.Cylinder = 12

ArabicConsole.WriteLine(BMW.Model)                    ' 2003
ArabicConsole.WriteLine(BMW.CarEngine.Cylinder)       ' 12
```

وللحديث عن قابلية الرؤية Visibility للفئة، فهي افتراضيا Friend إن لم تحدد شيء قبل اسم الفئة، حيث يمكنك الوصول إلى الفئة من داخل المشروع الحالي فقط، بينما Public تجعـــل الفئة قابلة للاستخدام من مبرمجين آخرين في مشاريعهم الأخرى، أما Private فستحصر قابليـــة الرؤية على المكان الذي عرفت فيه الفئة مع العلم انك لا تستطيع اســـتخدام الكلمـــة المحجـــوزة Private إلا أن عرفت الفئة داخل وحدة برمجية Module أو فئة Class أخرى، أو تركيب من النوع Structure:

```
' Friend
Class FriendClass
    …
End Class

Module Module1
    ' Friend
    Class FriendClass2
        …
    End Class

    ' Friend
    Friend Class FriendClass3
        …
    End Class

    ' Public
    Public Class PublicClass
        …
    End Class

    ' Private
    Private Class PrivateClass
        …
    End Class
    …
    …
End Module
```

PDF

انظر أيضا

بالإضافة إلى Private، Friend، و Public توجد كلمات محجوزة أخرى تحدد فيها قابلية الرؤية للفئات هي Protected و Protected Friend. مع ذلك، فضلت تأجيل التحدث عنها إلى الفصل الرابع **الوراثة** حيث أنها تتعلق بمبدأ قابلية تطبيق الاشتقاق الوراثي بشكل مباشر. (كل شيء في وقته حلو)!

تستطيع إرسال الفئات على شكل وسيطات إلى الإجراءات، ولكن من الضروري معرفة انك حتى لو أرسلت كائن من فئة بالقيمة (باستخدام ByVal) فالإجراء سيتمكن من تعديل قيمة المتغير المرسل رغم إرساله بالقيمة. وكإثبات لكلامي تحقق من الشيفرة التالية (ستعرف السبب لاحقا في هذا الفصل):

```
Module Module1
    Class TestClass
        Public X As Integer
    End Class

    Sub Main()
        Dim TestObject As New TestClass()

        TestObject.X = 10
        SendByValue(TestObject)
        ArabicConsole.WriteLine(TestObject.X) ' 10 وليس 1-
    End Sub
    Sub SendByValue(ByVal obj As TestClass)
        obj.X = -1
    End Sub
End Module
```

أعيد واكرر، الفئات Classes ليست كالتركيبات Structures رغم الشبه الكبير بينهما، وقد ذكرت بضعة فروق بينهما في السطور السابقة لعل أبرزها أن الفئات من النوع المرجعي Reference Type بينما التركيبات من النوع ذات القيمة Value Type، كما أود أن أضيف هنا أن قابلية تطبيق مبدأ الوراثة والاشتقاق الوراثي على الفئات ممكنة، بينما التركيبات لا تصل إلى هذا المستوى الرفيع من البرمجة -كما سترى في الفصل القادم.

أخيرا، تقترح عليك مستندات NET. بعدم ارفاق حرف C عند تسمية الفئات (وهو الأسلوب السائد قديما بين المبرمجين)، حيث أنها تفضل استخدام أسلوب PascalCase لتسمية الفئات

PDF

وأعضائها على مستوى Public أو Friend، بينما تستخدم camelCase للأعضاء على مستوى Private:

```
' لا تستخدم طريقة التسيمة القديمة
' CEmployeeData
Class EmployeeData
    Public EmployeeName As String
    Friend EmployeeAge As Integer
    Dim motherName As String          ' Private
    Private salaryAmount As Decimal   ' Private
    …
    …
End Class
```

بناء أعضاء الفئات

في القسم السابق عرفتك على الفئات وطريقة بنائها بشكل سريع ومبسط حتى أكون علاقة لطيفة بينك وبينها، أما الآن حان دور التحدث عن كافة التفاصيل المتعلقة ببناء أعضاء الفئات Class Members وهي: الحقول، الطرق، الخصائص، والأحداث.

الحقول Fields

ابسط أنواع الأعضاء التي يمكنك تعريفها في الفئات هي **الحقول Fields**، والحقول -في عالم فئات NET.- ما هي إلا متغيرات تقليدية تعرفها في الفئات مهما اختلف نوعها. لغويا، الفئة التالية تحتوي على 5 حقول:

```
Class SimpleClass
    Public Field1 As String
    Freind Field2 As Integer
    Dim field3 As Double
    Dim field4 As Boolean
    Private field5 As PersonClass
End Class
```

تتميز الحقول المعرفة في الفئات عن الحقول المعرفة في التركيبات Structure بالقدرة على إسناد قيم لها إثناء عملية التصريح:

```
Class SimpleClass
    Public Field1 As String = "تركي العسيري"
    Freind Field2 As Integer = 99
    …
End Class
```

PDF

شيء جميل آخر، يمكنك تعريف حقل من نفس نوع الفئة، وهذا الأسلوب سيفيدك كثيرا لإجراء خوارزميات برمجية شهيرة (كـ LIFO و FIFO):

```
Class SimpleClass
    Public Field1 As SimpleClass
    Public Field2 As Integer = 0
End Class
```

وعند الرغبة في التعامل مع الحقول السابقة، فكن منطقياً قدر الامكان:

```
Dim SimpleObject As New SimpleClass()

SimpleObject.Field1 = New SimpleClass()
SimpleObject.Field1.Field2 = 10

ArabicConsole.WriteLine(SimpleObject.Field2)  ' 0
ArabicConsole.WriteLine(SimpleObject.Field1.Field2)  ' 10
```

انظر أيضا

LIFO (Last-In-First-Out) و FIFO (First-In-First-Out) ما هي إلا خوارزميات برمجية تتبع لتنظيم سلسلة من البيانات في كتلة واحدة. يمكنك تطبيقها يدويا بنفسك إن كنت مستوعباً تماما للمنطق البرمجي لها. مع ذلك، يوفر لك إطار عمل NET. Framework فئتين هما Stack و Queue لإنجاز هذه الخوارزميات مباشرة تجد شرحها في الفصل السادس **الفئات الأساسية**.

يمكن للحقول أن تصرح بين الأقواس (و)، وبعبارة أخرى الحقول قابلة لان تعرف على شكل مصفوفات ستاتيكية كانت أم ديناميكية دون أي مشاكل تذكر:

```
Class ArrayClass
    Public X() As Integer
    Public Y(9) As Integer
End Class
```

وعند استخدام المصفوفات الديناميكية، فلست بحاجة لتذكيرك باستخدام ReDim قبل إسناد القيم لها:

PDF

```
Dim ArrayObject As New ArrayClass()

' حقل يمثل مصفوفة ستاتيكية
ArrayObject.X(0) = 100
ArrayObject.X(1) = 200
…

' حقل يمثل مصفوفة ديناميكية
ReDim ArrayObject.Y(99)
ArrayObject.Y(0) = 10
ArrayObject.Y(1) = 20
…
```

أخيرا، يمكنك حماية حقول الفئة من العبث بها بجعلها للقراءة فقط، استخدم الكلمة المحجوزة ReadOnly عند التصريح عن الحقل:

```
Class SimpleClass
    Public ReadOnly CreatedDate As Date = Now()
    …
End Class
```

استخدامك للكلمة المحجوزة ReadOnly سيمنعك منعا باتا من الكتابة على الحقل بإسناد أي قيمة إليه، مع ذلك فإن فرصة قراءة قيمة الحقل لازالت قائمة:

```
Dim SimpleObject As New SimpleClass()

' رسالة خطأ
SimpleObject.CreatedDate = Now()

' ممكن
ArabicConsole.WriteLine(SimpleObject.CreatedDate)
```

الطرق Methods

كما أن الحقول Fields ما هي إلا متغيرات تقليدية، فان **الطرق Methods** أيضا ما هي إلا إجراءات (Subs أو Functions) تقليدية. فالقضية ليست سوى مصطلحات برمجية مقدمة من إطار عمل Framework .NET. وبما أني –بكل تأكيد- لن أعيد فقرات الفصل السابق، دعني أحاول البحث عن أي تلميحة من هنا وهناك لها علاقة مباشرة من بعيد أو من قريب بالطرق Methods.

قد تفيدك فكرة الطرق بإسناد قيم للخصائص والحقول الأكثر استخدامًا مع الفئة، حيث توفر عليك كتابة السطور المكررة لتؤدي إلى زيادة السرعة. فمثلا، يمكنك تعريف هذه الطريقة التي تسند قيم الحقول الأكثر استخدامًا مع الفئة:

```
Class PersonRecord
    Public Name As String
    Public Age As Integer
    Public Address As String

    Sub SetValues(ByVal PersonName As String, _
        ByVal PersonAge As Integer, ByVal PersonAddress As String)

        Name = PersonName
        Age = PersonAge
        Address = PersonAddress
    End Sub
End Class
```

فبدلا من إسناد قيمة كل حقل على حده، استدعي هذه الطريقة في خطوة واحدة:

```
Dim PersonObject As New PersonRecord

' بدلا من تعيين قيمة كل حقل على حده
PersonObject.Name = "تركي العسيري"
PersonObject.Age = 99
PersonObject.Address = "المملكة العربية السعودية"

' يفضل استدعاء الطريقة بخطوة واحدة
PersonObject.SetValues("تركي العسيري", 99, "المملكة العربية السعودية")
```

فرق أخير بين الطرق المعرفة في الفئات والطرق المعرفة في التركيبات Structures يتعلق بقدرة تعريف المتغيرات الستاتيكية في داخل الطريقة، فاستخدام الكلمة المحجوزة Static داخل طرق الفئات ممكن، بينما لا تستطيع استخدامها داخل طرق التركيبات:

```
Class TestClass
    …
    Sub Method()
        Static X As Integer     ' ممكن
    End Sub
    …
End Class

Structure TestStructure
    …
    Sub Method()
        Static X As Integer     ' رسالة خطأ
    End Sub
    …
End Structure
```

PDF

أما الحديث عن مقترحات مستندات .NET. للتسمية، فهي تقترح استخدام الـــ PascalCase عند تسمية الطرق، والـــ camelCase عند تسمية وسيطاتها:

```
Sub ShutDown (computerName As String)
    …
    …
End Sub
```

إعادة التعريف Overloading:

من المبادئ الجميلة في لغات OOP هو مبدأ **إعادة التعريف Overloading**، والذي يمكنك من تسمية طرق مختلفة بنفس الاسم، شريطة اختلاف **نوع** وسيطات الإجراء:

```
Class SimpleClass
    ' SameName اعادة تعريف الاجراء
    ' ثلاث مرات
    Sub SameName()
        …
    End Sub

    Sub SameName(ByVal X As Integer)
        …
    End Sub

    Sub SameName(ByVal Y As String)
        …
    End Sub
    …
    …
End Class
```

عليك الانتباه انه لا يمكنك إعادة تعريف الإجراء إلا عن طريق تغيير **نوع** الوسيطات المرسلة وليس اسمها، فإعادة تعريف الإجراء SameName() التالي سيظهر رسالة خطأ رغم اختلاف أسماء الوسيطات:

```
Sub SameName(ByVal X As String)
    …
End Sub

' رسالة خطأ
Sub SameName(ByVal Y As String)
    …
End Sub
```

PDF

ليس هذا فقط، فحتى لو حاولت تغيير طريقة إرسال الوسيطة (إما بـــالمرجع ByRef أو بالقيمـــة ByVal) فإن ذلك سيتسبب في ظهور رسالة خطأ أيضا:

```
Sub SameName(ByVal X As String)
    …
End Sub

' رسالة خطأ
Sub SameName(ByRef Y As String)
    …
End Sub
```

وكتأكيد لكلامي، لا تحاول أيضا إعادة تعريف الطرق بتغيير محدد الوصول، فذلك لن يفيد أيضا:

```
Public Sub SameName(ByVal X As String)
    …
End Sub

' رسالة خطأ
Friend Sub SameName(ByVal X As String)
    …
End Sub
```

حالة أخرى لا تمكنك من تطبيق مبدأ إعادة التعريف، وهي عند اختلاف نـــوع الوســـيطات الاختيارية Optional فقط للطرق. فمثلا، إعادة تعريف الطرق التالية سيظهر رسالة خطأ:

```
Sub SameName(Optional ByVal X As Integer = 0)
    …
End Sub

' رسالة خطأ
Sub SameName(Optional ByVal X As String = "...")
    …
End Sub
```

والسبب أن الاختلاف في وسيطات الطرق هي وسيطات اختيارية Optional فقط. لذلك، لابد من وجود اختلاف في وسيطات غير اختيارية لتتمكن من إعادة تعريفها.

كما رأيت في الأمثلة السابقة، يمكنك إعادة تعريف الطرق بمجرد تعريفها مباشرة، مع ذلك يفضل استخدام الكلمة المحجوزة Overloads حتى تسهل على مترجم اللغة معرفة الإجراء الذي تود استدعائه، مما يزيد من سرعة التنفيذ:

PDF

```
Overloads Sub SameName(ByVal X As String)
    …
End Sub

Overloads Sub SameName(ByVal X As Integer)
    …
End Sub
```

ملاحظة

كتابة الكلمة المحجوزة Overloads -كما رأيت سابقا- أمر اختياري، ولكنها تصبح أمر إلزامي في باقي الإجراءات (التي تم إعادة تعريفها) إن استخدمتها لأول مرة.

على الجانب الآخر، لنلقي الضوء حول عملية استدعاء الطرق المعاد تعريفها، فلو كانت لدينا هاتين الطريقتين:

```
Class TestClass
    Overloads Sub SameName(ByVal X As Integer)
        ArabicConsole.WriteLine("النسخة التي تستقبل قيمة عددية")
    End Sub

    Overloads Sub SameName(ByVal X As String)
        ArabicConsole.WriteLine("النسخة التي تستقبل قيمة حرفية")
    End Sub
End Class
```

سيتم استدعاء النسخة التي توافق نوع الوسيطة المرسلة:

```
Dim TestObject As New TestClass()
Dim A As Integer = 100
Dim B As String = "100"

TestObject.SameName(A) ' النسخة التي تستقبل قيمة عددية
TestObject.SameName(B) ' النسخة التي تستقبل قيمة حرفية
```

ضع في اعتبارك أن نوع القيمة المرسلة لا يتمثل في نوع المتغير المرسل، وإنما في نوع القيمة النهائية. فمثلا لو استخدمت دوال التحويل سيتم استدعاء الطريقة التي توافق نوع القيمة التي تعود بها الدالة بغض النظر عن نوع المتغير المرسل:

PDF

```
Dim TestObject As New TestClass()
Dim A As String = "100"

' سيتم استدعاء النسخة الاولى من
' الطريقة وليس الثانية
TestObject.SameName(CInt(A))
```

وفي حالة إرسالك قيمة لا تكافئ وسيطات الطرق المعرفة، فسيتم استخدام التحويل الواسع Widening Conversion للقيمة ومن ثم استدعاء الطريقة الأقرب للقيمة بعد التحويل، أما التضييق Narrowing Conversion فسيظهر رسالة خطأ:

```
Dim A As Byte = 10
Dim B As Char = "A"c
Dim C As Long = 10

TestObject.SameName(A) ' النسخة التي تستقبل قيمة عددية
TestObject.SameName(B) ' النسخة التي تستقبل قيمة حرفية
TestObject.SameName(C) ' (رسالة خطأ)
```

انظر أيضا

لمزيد من التفاصيل حول دوال التحويل، التحويل الواسع، والتضييق، راجع الفصل الثاني **لغة البرمجة**.

تطبيقياً، يفيدك مبدأ إعادة التعريف في عدم تكرار كتابة الطرق المتشابهة بأسماء مختلفة، كما انه سبب رئيسي في تقليص عدد سطور جمل الشرط (كـ If … Then أو Select Case). فمثلا، قد تود تعريف طريقة لفتح سجل من قاعدة بيانات، تحديد الوسيطة يعتمد اعتماداً كلياً على طريقة البحث (إما بالاسم أو رقم المعرف مثلا)، فلو قمت بتعريف هاتين الطريقتين:

```
Sub OpenByID (ByVal id As Integer)
    …
End Sub

Sub OpenByName (ByVal name As String)
    …
End Sub
```

فإن ذلك سيكلفك الكثير من السطور البرمجية في كل مرة تود فتح سجل من قاعدة البيانات، حيث يتوجب عليك أولاً اختبار نوع القيمة (إما حرفية String أو عددية Integer) ومن ثم استدعاء

الطريقة المناسبة، أما مع تطبيق مبدأ إعادة التعريف، فيكفي استخدام الاسم Open لتعريف الطريقة، ومن ثم تقوم باستدعائها مباشرة دون الحاجة للتحقق من نوع القيمة بنفسك:

```
Overloads Sub Open (ByVal id As Integer)
    …
End Sub

Overloads Sub Open (ByVal name As String)
    …
End Sub
```

أخيرا، يمكنك تطبيق مبدأ إعادة التعريف أيضا على الإجراءات المعرفة في الوحدات البرمجية Modules والتركيبات من النوع Structures.

المشيدات Constructors:

برمجيا، **المشيد Constructor** هو طريقة Method يتم تنفيذها بمجرد إنشاء نسخة كائن جديدة من الفئة، فلو عرفت إجراء باسم Sub New() في أي فئة:

```
Class TestClass
    Sub New()
        ArabicConsole.WriteLine("تم تنفيذ مشيد الفئة")
    End Sub
End Class
```

فان هذا الإجراء هو مشيد الفئة TestClass وسيتم تنفيذه بمجرد إنشاء نسخة كائن جديدة من الفئة باستخدام الكلمة المحجوزة New:

```
' سيتم تنفيذ المشيد بمجرد انشاء نسخة
' كائن من الفئة كما بالسطر التالي
Dim TestObject As New TestClass()
```

تذكر أن المشيد لا يتم تنفيذه إلا عند إنشاء نسخة كائن جديدة من الفئة، وان حاولت إسناد قيم بين متغيرات الكائنات فلا تتوقع تنفيذ المشيد (سأفصل أكثر في موضوع إنشاء وموت الكائنات لاحقا في هذا الفصل):

```
Dim TestObject As New TestClass()     ' سيتم تنفيذ المشيد هنا
Dim TestObject2 As TestClass          ' لن ينفذ المشيد هنا
TestObject2 = TestObject              ' ولا حتى هنا
```

PDF Factory Pro trial version www.fastio.com

المزيد أيضا، يمكنك تمرير وسيطات Parameters إلى المشيد لحظة إنشاء الكائن. فمثلا، قد تود إجبار المبرمج (الذي يستخدم فئتك) أن يعين قيم لبعض حقول الفئة:

```
Class PersonClass
    Public FirstName As String
    Public LastName As String

    ' مشيد يحمل وسيطات
    Sub New(ByVal firstN As String, ByVal lastN As String)
        FirstName = firstN
        LastName = lastN
    End Sub
End Class
```

وعند إنشاء نسخة كائن جديدة من الفئة، عليك إرسال كافة الوسيطات المطلوبة والا فسيكون لرسالة الخطأ نصيب من الظهور:

```
' رسالة خطأ هنا
Dim TestObject As New PersonClass()

' لابد من ارسال الوسيطات للمشيد
Dim TestObject As New PersonClass("تركي", "العسيري")
```

كما تلاحظ في السطر الأول من الشيفرة السابقة، ستظهر رسالة خطأ إن لم ترسل وسيطات لحظة إنشاء الكائن، مع ذلك يمكنك جعل هذه المسألة اختيارية لمستخدم الفئة، إما بجعل الوسيطات اختيارية Optional، أو تطبيق مبدأ إعادة التعريف Overloading (وهو الحل الأفضل) على المشيد:

```
' يمكنك جعل وسيطات المشيد اختيارية
Class PersonClass
    …
    Sub New(Optional ByVal firstN As String = "",
        Optional ByVal lastN As String = "")

        FirstName = firstN
        LastName = lastN
    End Sub
End Class
```

PDF

```
' أو تطبيق مبدأ إعادة التعريف
' وهو المستحسن
Class PersonClass
    …
    Sub New()
        …
    End Sub

    Sub New(ByVal firstN As String, ByVal lastN As String)
        FirstName = firstN
        LastName = lastN
    End Sub
End Class
```

ملاحظة

لا يمكنك استخدام الكلمة المحجوزة Overloads عند إعادة تعريف المشيدات.

على صعيد آخر، استدعاء المشيدات تراجعيا Recursively غير ممكن، بعبارة أخرى، لا تستطيع استدعاء المشيد من داخل المشيد نفسه لتطبيق **الخوارزميات التراجعية Recursion**:

```
Class TestClass
    Sub New()
        …
        New()    ' رسالة خطأ
        …
    End Sub
End Class
```

مع ذلك، يسمح لك Visual Basic .NET باستدعاء مشيد آخر تم إعادة تعريفه بإرسال الوسيطات المطابقة له من مشيد آخر مع ضرورة استخدام الكلمة المحجوزة Me (سأتحدث عن كلمة Me لاحقا في هذا الفصل):

```
Class TestClass
    Sub New()
        Me.New(100) ' ممكن
        …
    End Sub

    Sub New(ByVal X As Integer)
        …
    End Sub
End Class
```

PDF Factory Pro trial version www.fastio.com

وضع في عين الاعتبار، انك لن تستطيع استدعاء المشيد الرئيسي Sub New() من داخل المشيد المعاد تعريفه Sub New(X As Integer)، حيث أن Visual Basic .NET لن يسمح لك بذلك.

أخيرا، المشيد هو المكان الوحيد الذي يسمح لك بإسناد قيمة لحقل معرف باستخدام الكلمة المحجوزة ReadOnly:

```
Class TestClass
    Public ReadOnly X As Integer = -1

    Sub New()
        X = 100     ' ممكن جدا
    End Sub
End Class
```

الخصائص Properties

الخصائص Properties ما هي إلا حالة تهجين بين الحقول Fields والطرق Methods، والذي اقصده من كلمة تهجين في هذا السياق هو أن الخصائص تعمل عمل الحقول حيث تمكنك من إسناد وقراءة قيم منها واليها. وفي الوقت نفسه، الخصائص هي إجراءات كالطرق Methods، حيث تستطيع كتابة الشيفرة بها. استخدم الكلمة المحجوزة Property لتعريف تركيب لخاصية جديدة ولا تنسى تحديد نوع قيمة الخاصية:

```
Class PersonClass

    Property BirthDate() As Date ' Date خاصية من النوع
        …
    End Property

End Class
```

يمكنك إضافة اجرائين في هذه الخاصية، الإجراء الأول هو Get والذي يتم استدعائه عند قيام مستخدم الفئة بقراءة قيمة الخاصية، أما الإجراء Set فسيتم استدعائه عند إسناد قيمة جديدة للخاصية، كما يفضل استخدام متغير وسيط خاص Private يحمل قيمة الخاصية:

```
' متغير وسيط يحمل قيمة الخاصية
Private m_BirthDate As Date

Property BirthDate() As Date
    Get
        Return m_BirthDate
    End Get
```

PDF

```
    ' Value لا بد من ان تطابق نوع قيمة الخاصية مع الوسيطة
    ' ByVal وهو Date كما يشترك ارسالها بالقيمة
    Set(ByVal Value As Date)
        m_BirthDate = Value
    End Set
End Property
```

في الجهة المقابلة، تستطيع التعامل مع الخصائص كما تتعامل تماما مع الحقول، حيث يمكنك إسناد القيم لها وقراءتها في أي وقت:

```
Dim Turki As New PersonClass()

Turki.BirthDate = #1/1/1903#
ArabicConsole.WriteLine(Turki.BirthDate)
```

سيناريو تنفيذ الاجرائين Get و Set السابقين سيكون كالتالي: في كل مرة تسند قيمة إلى الخاصية BirthDate سيتم استدعاء الإجراء Set وإرسال القيمة التي أسندت للخاصية إلى وسيطة الإجراء Value (والتي لابد من ان تتوافق مع نوع الخاصية وان ترسل بالقيمة ByVal)، وعلى العكس، في كل مرة تقوم فيها بقراءة قيمة الخاصية BirthDate سيتم تنفيذ الإجراء Get والذي –في اغلب الأحوال- سيعود بقيمة الخاصية باستخدام الامر Return كما تفعل مع الدوال Functions.

وكحماية لقيمة الخاصية من الكتابة، تستطيع جعلها للقراءة فقط لتسمى **Read Only Property**، يتم ذلك باستخدام الكلمة المحجوزة ReadOnly وحذف إجراء الخاصية Set (فلم يعد هناك داعي من وجودة لأنك لن تستطيع إسناد قيمة للخاصية). وكمثال تطبيقي، يمكنك تعريف خاصية للقراءة فقط تمثل عمر الشخص Age لتعود بقيمة استنادا إلى تاريخ ميلاده (والموجود في الخاصية BirthDate):

```
ReadOnly Property Age() As Integer
    Get
        Return DateDiff(DateInterval.Year, BirthDate(), Now)
    End Get
End Property
```

وكحماية أيضا لقيمة الخاصية من القراءة، يمكنك تعريف خاصية للكتابة فقط، بحيث تمنع مستخدم الخاصية من قراءتها باستخدام الكلمة المحجوزة WriteOnly وحذف الإجراء Get (فلست بحاجة إليه لأنك لن تستطيع إسناد قيمة للخاصية)، قد تستخدم هذا النوع من الخصائص لتمنع مستخدمي الفئة من سرقة كلمة المرور مثلا:

```
WriteOnly Property Password() As String
    Set(ByVal Value As String)
        m_Password = Value
    End Set
End Property
```

المزيد أيضا، يمكن للخصائص أن تستقبل وسيطات Parameters كما في الطرق، وذلك بذكر جميع الوسيطات في بداية تعريف الخاصية:

```
Private m_Address(2) As String
Property Address(ByVal index As Integer) As String
    Get
        If index >= 0 And index <= UBound(m_Address) Then
            Return m_Address(index)
        End If
    End Get

    Set(ByVal Value As String)
        If index >= 0 And index <= UBound(m_Address) Then
            m_Address(index) = Value
        End If
    End Set
End Property
```

وعند الرغبة في إسناد قيم للخصائص أو قراءتها، أرسل الوسيطات المطلوبة بين الأقواس:

```
Dim Turki As New PersonClass()
Dim counter As Integer

Turki.Address(0) = "شارع الحزن"
Turki.Address(1) = "حي السعادة"
Turki.Address(2) = "ولاية نيويورك - منغوليا"

For counter = 0 To 2
    ArabicConsole.WriteLine(Turki.Address(counter))
Next
```

أخيرا، يمكنك تطبيق مبدأ إعادة التعريف Overloading على الخصائص أيضا، كما يمكنك تعريف الخصائص في وحدات برمجية Modules وتركيبات من نوع Structure، مع العلم انك لن تستطيع التصريح عن متغيرات محلية ستاتيكية Static في الخصائص المعرفة في تركيبات Structures.

PDF

الخصائص الافتراضية Default Properties:

الخصائص الافتراضية هي خصائص يمكنك إسناد القيم لها أو قراءة قيمها باستخدام اسم الكائن التابعة له فقط ودون الحاجة لذكر اسم الخاصية، وكل ما هو مطلوب منك لجعل الخاصية افتراضية استخدام الكلمة المحجوزة Default لجعل الخاصية افتراضية:

```
Default Property Address(ByVal index As Integer) As String
    …
    …
End Property
```

وحتى تصل إلى الخاصية Address (التي أصبحت افتراضية)، تستطيع الاكتفاء بذكر اسم الكائن التابع لها دون الحاجة لتحديد اسم الخاصية:

```
Turki(0) = "شارع الحزن"
Turki(1) = "حي السعادة"
Turki(2) = "ولاية نيويورك - منغوليا"

For counter = 0 To 2
    ArabicConsole.WriteLine(Turki(counter))
Next
```

لا تعتقد أني جعلت الخاصية Address هي الافتراضية لأنها كانت آخر خاصية أدرجت في الفئة السابقة، ولكن السبب الحقيقي هو أنها الخاصية الوحيدة التي تحمل وسيطات Parameters، فلو حاولت جعل الخاصية Age أو الخاصية BirthDate أو حتى الخاصية Password هي الافتراضية، سيظهر لك مترجم Visual Basic .NET رسالة خطأ، فالخصائص السابقة لا توجد بها وسيطات Parameters.

معلومة أخيرة حول الخاصية Address السابقة، إن خطر على بالك حالة معينة لحظة تعريف الكائن Turki السابق، فسأقف ضاربا التحية لعقليتك النبيهة والتي تخرج منها مثل هذه التساؤلات! الحالة التي اقصدها هنا هي عند تعريفك للكائن Turki في مصفوفة:

```
Dim Turki(5) As PersonClass

Set Turki(0) = New PersonClass()
```

PDF

كيف ستصل إلى الخاصية Address بشكل افتراضي (أي دون الحاجة لكتابتها)؟ حيث أن الأقواس التي تلي الكائن تمثل رقم فهرس الكائن في المصفوفة، لذلك عليك إضافة أقواس إضافية لتمثل وسيطات الخاصية Address الافتراضية:

```
Turki(0)(0) = "شارع الحزن"
Turki(0)(1) = "حي السعادة"
Turki(0)(2) = "ولاية نيويورك - منغوليا"

For counter = 0 To 2
    ArabicConsole.WriteLine(Turki(0)(counter))
Next
```

أيهم انسب الحقل، الطريقة، أم الخاصية؟

بصراحة شديدة وبدون مقدمات، لا أستطيع إعطائك إجابة منصفة لهذا السؤال، حيث أن القضية تعتمد على الجانب الفكري أكثر من الجانب التقني، إذ أن جميع مشاكلك التي تواجهها يمكن إنجازها برمجيا باستخدام الأنواع الثلاث. فلو أردنا إضافة عضو في فئة يمثل عمر الشخص Age، فجميع هذه الحلول ممكنة:

```
Class PersonClass
    ' حقل
    Public Age As Integer

    ' خاصية
    Private m_Age As Integer
    Property Age() As Integer
        Get
            Return m_Age
        End Get
    Set(ByVal Value As Integer)
        m_Age = Value
    End Set
    End Property

    ' طريقة
    Function Age(Optional ByVal newValue As Integer = -1) As Integer
        Static AgeValue As Integer

        If newValue <= 0 Then
            Return AgeValue
        Else
            AgeValue = newValue
        End If
    End Function
End Class
```

PDF

كما ذكرت أن القضية يغلب عليها الطابع الفكري أكثر من التقني، فتحديد نوع العضو يعتمد اعتماداً كلياً على الوظيفة الأساسية لهذا العضو، ليكون القرار النهائي للمبرمج الذي يحدد ما هو النوع المناسب، الحقول في العادة تستخدم لإسناد قيم دون الحاجة إلى أي شيفرة اختبار أو تحقق إضافية (كحقل يحمل قيمة تمثل ملاحظة Note)، وبالنسبة للخصائص فهي كالحقول تقريبا إلا أنها تتطلب شيفرات إضافية للتحقق من القيمة المرسلة أو تنفيذ مجموعة من الأوامر لتظهر التأثير المباشر الصادر من عملية إسناد القيم لها (كخاصية عدد الضفادع NumOfFrogs في المستنقع حيث لا يمكن لها أن تكون قيمة سالبة)، أما الطرق فيغلب عليها الجانب العملي والذي يؤدي إلى فعل Action للكائن (كالطريقة Smile لرسم ابتسامة على شفتى كائن).

الأحداث Events

جميع الأعضاء التي تطرقت لها (الحقول، الطرق، والخصائص) يتم بنائها من مؤلف الفئة، وهو المسئول الأول والأخير عن الشيفرات المصدرية التي ملأت الفئة، ولكن في اغلب الأحوال يفضل إعطاء فرصة كبيرة لمستخدم الفئة من تخصيص شيفرات معينة وكتابتها بالطريقة التي يريدها لتجعل الفئة قابلة أكثر لإعادة الاستخدام Reusable. ولكي يتحقق ذلك، عليك إضافة أعضاء تسمى **الأحداث Events**.

الأحداث في الفئات ما هي إلا مؤشرات مفوضة إلى إجراءات تقليدية يتم استدعائها لحظة وقوع حدث معين يحدده مؤلف الفئة، ولكن نقطة الاختلاف الجوهرية بين الأحداث وباقي الأنواع الأخرى من أعضاء الفئات، هي ان ردة الفعل أو –بعبارة تقنية- الشيفرات التابعة لتلك الإجراءات لا ينجزها إلا مستخدم الفئة وليس مؤلفها. فمثلا، لو أردت تعريف حدث في الفئة السابقة PersonClass يمثل موت الشخص Die، سأعطي كامل الحرية لمستخدم الفئة لكتابة الشيفرات المصدرية التي يريدها لحظة وقوع الموت (وقوع الموت يسمى –برمجيا- **إطلاق الحدث Fire Event** في هذا السياق). لتعريف هذا الحدث، استخدم الكلمة المحجوزة Event:

```
Class PersonClass
    Event Die()
    …
End Class
```

والآن قد أعطيت حرية كبيرة لمستخدم الفئة من تحديد ردة الفعل وكتابة الشيفرات التي يريدها لحظة انطلاق الحدث. وحتى يتمكن المستخدم من فعل ذلك، عليه استخدام الكلمة المحجوزة

WithEvents عند التصريح عن متغير الكائن، ومن ثم تحديد الإجراء الذي يود تنفيذه لحظة انطلاق الحدث باستخدام الأمر Handles وإلحاقها بالحدث المراد قنصه:

```
Module Module1
    Dim WithEvents Turki As New PersonClass()

    Sub Main()
        …
    End Sub

    ' Die قنص الحدث
    Sub PersonHasDied() Handles Turki.Die
        ArabicConsole.WriteLine("لقد توفي الشخص")
    End Sub
End Module
```

ملاحظة

لا تستطيع استخدام الكلمة المحجوزة WithEvents على المتغيرات المحلية Local Varaibles، حيث يشترط أن يكون المتغير على مستوى الوحدة البرمجية Module، أو على مستوى الفئة Class، أو عام Global.

والسؤال الذي يطرح نفسه بكل تأكيد، متى سيموت Turki؟ والجواب المؤكد هو في علم الغيب والذي لا يعلمه إلا الله عز وجل، لذلك دعني أعيد صياغة السؤال حتى نتمحور حول الشيفرة السابقة ليكون: متى سيتم إطلاق الحدث Die حتى يتم تنفيذ الإجراء PersonHasDied()؟ والجواب هو في أي وقت يحدده مؤلف الفئة عن طريق الأمر RaiseEvent، فلو عدنا إلى مرحلة بناء الفئة يمكنني ان اعرف الطريقة KillHim() التي تؤدي إلى اطلاق الحدث Die:

```
Class PersonClass
    Event Die()

    Sub KillHim()
        RaiseEvent Die()
    End Sub
    …
End Class
```

ما أن يقوم مستخدم الفئة باستدعاء الطريقة KillHim() فسيتم إطلاق الحدث Die، وحتى نشاهد هذه اللحظة الحزينة، دعنا نكتب هذه الشيفرة التي تستدعي الطريقة KillHim():

```
Module Module1
    Dim WithEvents Turki As New PersonClass()

    Sub Main()

        ' استدعاء الطريقة يؤدي إلى اطلاق الحدث
        Turki.KillHim()

    End Sub

    Sub PersonHasDied() Handles Turki.Die
        ArabicConsole.WriteLine("لقد توفي الشخص")
    End Sub
End Module
```

المزيد أيضا، الكلمة المحجوزة Handles لا تستخدم لقنص حدث واحد فقط، بل يمكنك إضافة أحداث مختلفة من كائنات مختلفة بحيث تؤدي إلى تنفيذ نفس الإجراء عند انطلاق احد أحداث هذه الكائنات:

```
Sub TestEvents() Handles Turki.Die, Ali.Die, Apple.Stink
    …
End Sub
```

مع ذلك، لن تستطيع استخدام Handles لقنص أحداث لكائنات مختلفة في حالة واحدة وهي عندما تكون الوسيطات Parameters للحدث مختلفة، فمثلا لو عرفت حدث في فئة يستقبل وسيطات إضافية:

```
Class TestClass
    Event TestEvent(ByVal x As Integer)
    …
End Class
```

عليك استخدام Handles في إجراء يطابق عدد ونوع الوسيطات التي يتطلبها الحدث:

```
' لا يمكن هنا
Sub TestSub() Handles TestObject.TestEvent
    …
End Sub

' هنا مناسب
Sub TestSub(ByVal Y As Integer) Handles TestObject.TestEvent
    …
End Sub
```

PDF

ملاحظة

بالنسبة للأحداث التي تحتوي على وسيطات إضافية، فلابد من إرسال قيم الوسيطات عند إطلاق الحدث باستخدام RaiseEvent، فالحدث السابق يتم اطلاقة هكذا:

RaiseEvent TestEvent(10)

قنص الأحداث باستخدام AddHandler:

بالإضافة إلى الكلمة المحجوزة WithEvents لقنص الأحداث، يوفر لك Visual Basic .NET طريقة أخرى أكثر مرونة لقنص الأحداث وهو الأمر AddHandler. الشيء الجميل والذي يعجبني جدا في هذا الأمر، هو انك تحدد الإجراء الذي تود تنفيذه لحظة إطلاق الحدث وقت تنفيذ وليس التصميم. يتطلب الامر AddHandler اسم الحدث الذي تود قنصه و مؤشر إلى الإجراء الذي تود تنفيذه (للحصول على مؤشر الإجراء استخدم AddressOf):

```
Module Module1
    Dim Turki As New PersonClass()

    Sub Main()
        ' قنص الحدث
        AddHandler Turki.Die, AddressOf PersonHasDied
        Turki.KillHim()
    End Sub

    Sub PersonHasDied()
        ArabicConsole.WriteLine("لقد توفي الشخص")
    End Sub
End Module
```

سيستمر الإجراء PersonHasDied() السابق في انتظار الحدث Die حتى موت الكائن Turki نهائيا (سأتحدث بالتفصيل الممل حول موت الكائنات لاحقا)، وان كنت لا ترغب في الانتظار طيلة هذه الفترة، يمكنك إيقاف عملية قنص الحدث في أي وقت باستخدام الأمر RemoveHandler والذي يستدعى بنفس صيغة AddHandler:

```
RemoveHandler Turki.Die, AddressOf PersonHasDied
```

PDF

بمجرد تنفيذ الامر RemoveHandler السابق، فلن يتم استدعاء الإجـــراء ()PersonHasDied حتى لو تم إطلاق الحدث Die ملايين المرات.

لا يقتصر الاختلاف في قنص الأحداث بين WithEvents و AddHandler في مسألة أن الأولى وقت التصميم والثانية وقت التنفيذ، بل يتعدى الأمر أكثر من ذلـــك، إذ أن عمليـــة قـــنص الأحداث باستخدام WithEvents يكون موجه إلى متغير الكائن بشكل حصري، أما قنص الأحداث باستخدام AddHandler فهو موجه إلى الكائن نفسه وليس المتغير الذي يشير إليه. قد يبدو لي أن الفرق لم يتضح لك بعد، لذلك سأحاول توضيح الفرق باستخدام أكثر من مؤشر إلى نفس الكـــائن، والبداية ستكون مع WithEvents:

```
Module Module1
    Dim WithEvents Turki As New PersonClass()

    Sub Main()
        ' Turki و Turki2 يشيران إلى نفس الكائن
        Dim Turki2 As PersonClass = Turki

        ' سيتم تنفيذ الإجراء
        ' PersonHasDied
        Turki2.KillHim()

        ' ( WithEvents الغاء المؤشر الاول (وهو الذي عرف بـــ
        Turki = Nothing

        ' لن يتم تنفيذ الإجراء
        ' PersonHasDied
        Turki2.KillHim()
    End Sub

    Sub PersonHasDied() Handles Turki.Die
        ArabicConsole.WriteLine("لقد توفي الشخص")
    End Sub
End Module
```

من المثال السابق يتضح لنا أن استخدام الكلمة المحجوزة WithEvents لقنص أحـــداث الكـــائن مرتبطة ارتباطاً كاملاً بمتغير (مؤشر) الكائن الذي استخدم لحظة التعريف، والدليل أننا بعدما ألغينا متغير الكائن الذي يستخدم WithEvents، لم تتم عملية تنفيذ الإجراء ()PersonHasDied، رغم أن الحدث Die قد تم إطلاقه بالفعل.

من ناحية أخرى، قنص الأحداث باستخدام AddHandler لا يتعامل مع مؤشر الكائن (متغير الكائن)، وإنما يتعامل مع الكائن نفسه (الذي يشير إليه المتغير)، فلو حاولت إعادة تطبيق الشيفرة السابقة ولكن باستخدام AddHandler:

```
Module Module1
    Dim Turki As New PersonClass()

    Sub Main()
        AddHandler Turki.Die, AddressOf PersonHasDied

        ' Turki و Turki2 يشيران إلى نفس الكائن
        Dim Turki2 As PersonClass = Turki

        ' سيتم تنفيذ الإجراء
        ' PersonHasDied
        Turki.KillHim()

        ' الغاء المؤشر الاول
        Turki = Nothing

        ' سيتم تنفيذ الإجراء
        ' أيضا PersonHasDied
        Turki2.KillHim()
    End Sub

    Sub PersonHasDied()
        ArabicConsole.WriteLine("لقد توفي الشخص")
    End Sub
End Module
```

ستكتشف أن الإجراء PersonHasDied() تم استدعائه لحظة انطلاق الحدث Die مرتين بالرغم من اختفاء مؤشر الكائن الأول له، لذلك سيستمر الإجراء PersonHasDied() في عملية قنص الحدث حتى موت الكائن نهائيا واختفاء جميع المؤشرات التي تشير إليه (أذكرك مرة أخرى، سأتحدث عن موت الكائنات بالتفصيل الممل قريبا في هذا الفصل).

ميزة أخرى في عملية قنص الأحداث باستخدام AddHandles غير ممكنة في WithEvents وهي عند التعامل مع المصفوفات، فلو حاولت استخدام WithEvents مع مصفوفة:

```
' غير ممكن
Dim WithEvents Turki(9) As PersonClass
```

PDF

سيظهر لك مترجم اللغة رسالة خطأ والسبب ان الكلمة المحجوزة WithEvents لا يمكن تطبيقها على المصفوفات، الشيء الذي يضطرك إلى استخدام AddHandles:

```
Module Module1
    Dim Turki(5) As PersonClass

    Sub Main()
        Dim counter As Integer
        Dim Turki(5) As PersonClass

        For counter = 0 To UBound(Turki)
            Turki(counter) = New PersonClass()
            ' ممكن قنص الحدث
            AddHandler Turki(counter).Die, AddressOf PersonHasDied
        Next

        Turki(0).KillHim()
        Turki(5).KillHim()
    End Sub

    Sub PersonHasDied()
        ArabicConsole.WriteLine("لقد توفي الشخص")
    End Sub
End Module
```

استخدام الكائنات

تعرفنا في الصفحات السابقة على الفئات وطريقة بنائها من منطلق انك مؤلف الفئة، أما الآن سنذهب إلى الجهة المقابلة ونستخدم هذه الفئة لننشئ بها كائنات ونتحدث من منطلق انك مستخدم لهذه الفئة، البداية ستكون مع توضيح حقيقة الكائن.

ما هي حقيقة الكائن؟

حديثي هنا محصور حول الكائنات من النوع المرجعي Reference Type، فكما قلت سابقا ان المتغيرات المنشأة من الفئات Classes، أو المصفوفات Arrays، أو الحروف Strings هي متغيرات من النوع المرجعي، وعليك أن تعلم علم اليقين أن الكائنات من النوع المرجعي لا تتعامل معها بشكل مباشر في شيفراتك المصدرية، حيث انك تصل إليها عن طريق متغير معرف منها يسمى **مؤشر Pointer**، فالمتغير Turki التالي:

```
Dim Turki As New PersonClass
```

PDF

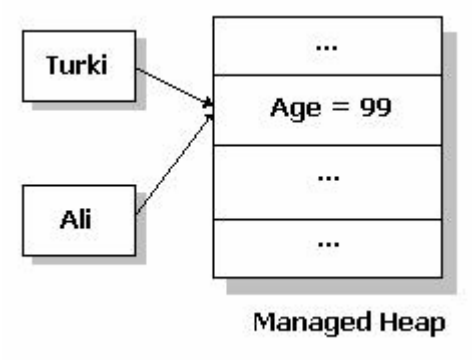
لا يحتوي على البيانات الحقيقية للكائن، فهو ليس سوى مؤشر إلى كائن في منطقة من الذاكرة تسمى Managed Heap تدار عن طريق إطار عمل NET. وحجم المؤشر Turki سيكون دائما 4 بايت بغض النظر عن نوع الفئة التي أنشئ منها.

عندما تتعامل مع المؤشرات عليك تغيير منطقك البرمجي فهي ليست كالمتغيرات التقليدية، حيث أن المتغيرات التقليدية تحمل القيمة الفعلية دائما بينما المؤشرات لا تحمل إلا قيمة تمثل موقع البيانات الحقيقية للكائن في الذاكرة، اقلب الصفحة وادرس الشيفرة التي بأعلاها:

```
Class PersonClass
    …
    Public Age As Integer
    …
End Class

…
…

Dim Turki As New PersonClass()
Dim Ali As New PersonClass()

Ali.Age = 25

' الان كلاهما يشيران إلى نفس الكائن
Ali = Turki

Turki.Age = 99

ArabicConsole.WriteLine(Ali.Age) ' 99
```

من المثال السابق يتضح لنا أن Turki و Ali مؤشرين يشيران إلى نفس الكائن، والدليل أنني عندما قمت بتغيير قيمة الحقل Age في المؤشر Turki، تأثر نفس الحقل عندما وصلت إليه عن طريق المؤشر Ali، تجد توضيح المثال السابق في (**الشكل 3-1 في أعلى الصفحة التالية**).

PDF

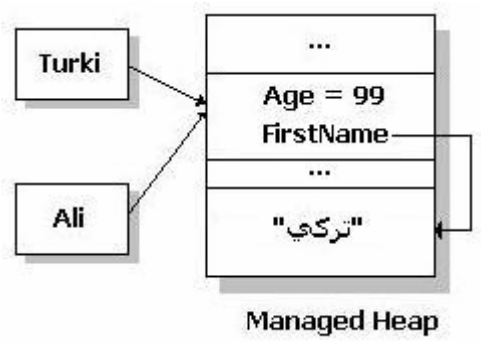


شكل 3-1: كلا المؤشران Turki و Ali يشيران إلى نفس الكائن.

دعني اجعل الأمور أكثر تعقيدا ونضيف حقل آخر من النوع String، التعامل مع هذا الحقل هو نفس التعامل مع متغيرات الفئات Classes، فالمتغيرات من النوع String ومتغيرات الفئات – كما قلت- هما من النوع المرجعي Reference Type:

```
Turki.FirstName = "تـركـي"
```

عليك معرفة أن الحقل FirstName ما هو إلا مؤشر أيضا، والقيمة التي يحملها تمثل بيانات لكائن وهي مستقلة في القسم Managed Heap (**شكل 3-2**).



شكل 3-2: الحقل FirstName ما هو إلا مؤشر آخر يشير إلى كائن مستقل آخر.

أردت من الشكل السابق أن أوضح لك بان المؤشرين Turki و Ali يشيران إلى بيانات الكائن والتي تحتوي على مؤشر FirstName آخر يشير بدوره إلى بيانات كائن آخر. برمجيا، ستستخدم المؤشر Turki الذي يشير إلى المؤشر FirstName لتصل إلى بيانات الكائن (والذي يحتوي القيمة الفعلية "**تركي**").

عبارات خاصة بالكائنات

حتى تتعامل مع المؤشرات أو الكائنات بطرق صحيحة، من الضروري استيعاب بعض الكلمات المحجوزة والعبارات التي تتعلق بالكائنات بشكل مباشر أو غير مباشر. البداية ستكون مع الكلمة المحجوزة New.

الكلمة المحجوزة New:

الكلمة المحجوزة New تستخدم للقيام بعملية الإنشاء الفعلي للكائن وحجز منطقة له في ذاكرة Managed Heap، يمكنك استخدامها لحظة التصريح عن المتغير (المؤشر) أو عند إسناد قيمة له بمعامل المساواة "=":

```
' انشاء الكائن لحظة التصريح عن المتغير
Dim TestObject As New TestClass ()

' احذف الاقواس التي قد يضفها المحرر
Dim TestObject As TestClass
' تم انشاء الكائن الان
TestObject = New TestClass
```

مع ذلك، لا يمكنك استخدام New لحظة التصريح عن متغير إن كان احد حقول Fields التركيبات من النوع Structures:

```
Structure TestStructure
    Dim TestObject As New TestClass () ' رسالة خطأ
End Structure
```

الكلمة المحجوزة New مرنة جدا بحيث يمكنك استخدامها في أي مكان تقريبا من شيفراتك المصدرية، فيمكنك مثلا إنشاء الكائن لحظة إرساله إلى إجراء:

PDF Factory Pro trial version www.fastio.com

```
Sub Main()
    …
    ' انشاء الكائن لحظة ارسال الوسيطة
    TestSub(New TestClass())
    …
End Sub

Sub TestSub(ByVal obj As TestClass)
    obj.DoMethod ()
    …
End Sub
```

سيتم تنفيذ المشيدات مباشرة بعد إنشاء نسخة جديدة من الكائن باستخدام New، لذلك إن كان للفئة مشيدات، فلا بد من إرسال وسيطات المشيدات لحظة استخدام New، وان لم تكن للفئة أي مشيدات فيمكنك الاكتفاء بوضع الأقواس خالية:

```
' لاتوجد وسيطات لمشيد الفئة
Dim TestObject As New TestClass ()

' يحتوي مشيد الفئة على وسيطتين
Dim AnotherObject As New AnotherClass(2, 5)
```

التركيب With … End With:

يستخدم التركيب With … End With لتسريع عملية الوصول إلى أعضاء الكائن دون الحاجة لكتابة اسم الكائن، فبدلا من الوصول إلى الأعضاء بهذا الشكل:

```
Dim TestObject As New TestClass()
…
…
TestObject.Field1 = 10
TestObject.Field2 = 20
TestObject.Method1 (10)
TestObject.Property1 (2, 2) = 10
```

يمكنك استخدام With … End With بهذه الطريقة:

```
Dim TestObject As New TestClass()
…
…
With TestObject
    .Field1 = 10
    .Field2 = 20
    .Method1 (10)
    .Property1 (2, 2) = 10
End With
```

PDF Converter

القيمة Nothing:

إسناد القيمة Nothing إلى المؤشر تؤدي إلى إلغاء العلاقة بين ذلك المؤشر والكائن الذي يشير له، وضع في عين الاعتبار أن المؤشر الذي لا يشير إلى الكائن لا يمكنك الوصول إلى أعضائه:

```
Dim Turki As New PersonClass()

Turki.Name = "تركي العسيري"

Turki = Nothing

Turki.Age = 99 ' رسالة خطأ
```

نقطة هامة جدا: إسناد القيمة Nothing للمؤشر لا تؤدي إلى موت الكائن الذي كان يشير إليه وإلغائه من الذاكرة، فالمسألة تحتاج إلى تفصيل تجده قريبا في فقرة **حياة وموت الكائنات**.

المعامل Is:

استخدم المعامل Is مع المؤشرات إن كنت تود التحقق من أن المؤشر يشير إلى نفس الكائن:

```
Dim Turki As New PersonClass()
Dim Khaled As New PersonClass()

ArabicConsole.WriteLine(Turki Is Khaled) ' False

' المؤشران يشيران إلى نفس الكائن الان
Khaled = Turki

ArabicConsole.WriteLine(Turki Is Khaled) ' True
```

كما تلاحظ، القيمة التي يعود بها المعامل Is منطقية (من النوع Boolean) مما يجعلها مناسبة جدا لاستخدامها داخل جملة شريطة:

```
If Not Turki Is Khaled Then
    Turki = Khaled
End If
```

يمكنك الاستفادة أيضا من المعامل Is لمعرفة ما اذا كان المؤشر يشير حقا إلى كائن ام لا، وذلك باستخدامه مع القيمة Nothing:

```
' هل المؤشر Turki لا يشير إلى كائن؟
If Turki Is Nothing Then
    Turki = New PersonClass()
End If
```

العبارة TypeOf ... Is:

العبارة TypeOf ... Is شبيهه إلى حد كبير مع المعامل Is السابق، ولكنها تستخدم للتحقق من نوع الفئة التي يشير اليها متغير الكائن:

```
' هل الكائن Turki منشئ من الفئة PersonClass ؟
If TypeOf Turki Is PersonClass Then
    Turki.Name = "تركي العسيري"
End If
```

الكائن Me:

يمكنك استخدام الكائن Me داخل الفئة للتعبير عن الكائن الحالي، بعبارة أخرى الكائن Me هو مؤشر للنسخة الحالية من الكائن، فعندما أقول: أنا، فإنني اقصد نفسي وعندما تقول أنت: أنا، فانك تقصد نفسك:

```
Class PersonClass
    Public Name As String

    Sub PrintName ()
        ' طباعة قيمة الحقل Name للكائن الحالي
        ArabicConsole.WriteLine ( Me.Name )
    End Sub
    …
End Class
```

صحيح أن الكائن Me يمثل مؤشر الكائن الحالي، إلا انه يختلف في بنيته التحتية ومسائل تقنية أخرى عن مؤشرات (متغيرات) الكائنات التقليدية. فمثلا، عن طريق Me تستطيع الوصول إلى الأعضاء من النوع Private، ولكنك لن تستطيع إسناد أي قيمة إلى Me:

```
Class PersonClass
    Private Name As String

    Sub PrintName ()
        ArabicConsole.WriteLine (Me.Name) ' ممكن
        Me = Nothing ' رسالة خطأ
    End Sub
    …
End Class
```

إسناد القيم

استخدام معامل إسناد القيم "=" بين متغيرات الكائنات لا يقوم بنسخ بيانات الكائن، وإنما زيادة عدد المؤشرات التي تشير إلى الكائن -كما أخبرتك أكثر من مرة سابقا:

```
Dim X As New ClassA
Dim Y As ClassA

Y = X
```

في حالة اختلاف نوع الفئات التي تشير لها الكائنات، فلن تستطيع إسناد القيم إليها بنسخ مؤشراتها:

```
Dim X As New ClassA
Dim Y As ClassB

' خطا لاختلاف نوع فئات الكائنات
Y = X
```

كذلك الحال عند استخدام الكلمة المحجوزة New لحظة إسناد القيم، فلابد من توافق نوع الفئة مع نوع المتغير الذي أعلنت عنه:

```
Dim X As ClassA

X = New ClassA () ' ممكن
X = New ClassB () ' رسالة خطأ
```

مع ذلك تستطيع إنشاء الكائنات باستخدام New أو إسناد القيم لها دون الحاجة لتوافق نوع الفئة مع المتغير، وذلك في حالة تصريحك للمتغيرات من النوع Object:

```
Dim X As Object
Dim Y As Object

' ClassA انشاء كائن من الفئة
X = New ClassA ()

' ممكن
Y = X
Y.MethodInClassA ()

' ClassB الكائن اصبح منشئ من الفئة
Y = New ClassB ()
```

PDF

```
' ممكن
X = Y
X.MethodInClassB ()
```

رغم أن تصريح المتغيرات بالنوع Object يعطي مرونة كبيرة عند إسناد القيم لها أو إنشاء الكائنات منها، إلا أن ذلك يسبب الكثير من البطء في عملية التنفيذ، حيث تتطلب من مترجم Visual Basic .NET تحويل نوع بيانات المتغير والتحقق من كافة الطرق والخصائص الموجودة في الكائن قبل استدعائها والبحث أيضا عن مواقعها. وحتى لا أتوغل في التفاصيل التقنية أكثر، دعني أخبرك أن هذه الطريقة تسمى **الربط المتأخر Late Binding**، وجميع الطرق التي ذكرتها سابقا في هذا الفصل هي **الربط المبكر Early Binding** وهو أسرع بعشرات -إن لم يكن مئات أحيانا- المرات من الربط المتأخر:

```
' الربط المبكر
Dim X As TestClass = New TestClass ()

' الربط المتاخر
Dim X As Object = New TestClass ()
```

العبارة Option Strict مرة أخرى:

عند تفعيلك للعبارة Option Strict On في احد ملفات المشروع، سيمنعك Visual Basic .NET من استخدام أسلوب الربط المتأخر Late Binding:

```
' رسالة خطأ في حالة تفعيلك للعبارة
' Option Strict On
Dim X As Object

X = New ClassA ()        ' ليس هنا السبب

X.MethodInClassA()       ' هنا السبب
```

أخي الكريم، عليك الانتباه عن أن سبب الخطأ ليس بسبب عملية الإسناد السابقة، وإنما بسبب استدعاء الطريقة MethodInClassA() عن طريق الربط المتأخر، قد تستغرب مدى الجدوى من المتغير X السابق ما دمت لن تستطيع استدعاء أعضائه، ولكن في اغلب الأحوال يلجأ المبرمجون إلى هذه العملية لجعل وسيطة الإجراء Parameter تستقبل قيم مهما كان نوعها (كما هو الحال مع الكائن ArabicConsole)، كما تفيدهم هذه الطريقة أيضا لإبقاء الكائن على قيد الحياة والاحتفاظ بمتغير يشير إليه حتى تتمكن من إعادة إسناده إلى متغير آخر من نفس الفئة:

PDF

```
Dim X As Object
Dim Y As ClassA
…
…

X = New ClassA ()

Y = X ' عملية الإسناد
```

مع ذلك، سيظهر مترجم Visual Basic .NET رسالة خطأ بسبب تعارض أنواع المتغيرات، فالمتغير X السابق من النوع Object (رغم أن القيمة التي يحتفظ بها من النوع ClassA) بينما المتغير Y من النوع ClassA، والحل الوحيد الذي يمكنك من إنجاز هذه العملية هو باستخدام معامل التحويل CType والذي يتطلب القيمة ومن ثم النوع المراد تحويله إليها:

```
Dim X As Object
Dim Y As ClassA
…
…

X = New ClassA ()

Y = CType (X, ClassA) ' CType لابد من استخدام المعامل

Y.MethodInClassA() ' توكل على الله واستدعي الطرق كما تريد
…
…
```

ملاحظة

التعبير "CType **المعامل**" ليس خطأ مطبعي، لان CType هو **معامل** Operator وليست دالة Function كدوال التحويل CInt()، CLng() ....الخ، فالمعامل CType هو احد سمات لغة البرمجة Visual Basic .NET.. مع ذلك، يمكنك استخدام الدالة CType احم احم اقصد المعامل CType ليعمل عمل دوال التحويل الأخرى:

```
Dim Y As Integer = CInt ("10")
Dim Y As Integer = CType ("10", Integer)
```

PDF

حياة وموت الكائنات

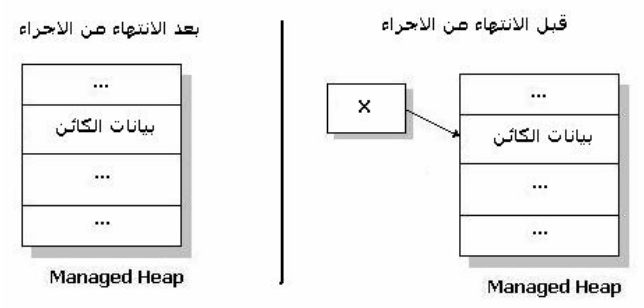
إن تحدثنا عن المتغيرات ذات القيمة Value Type، فهي ستبقى محتفظة بقيمتها استنادا لعمرها Lifetime كما ذكرت في الفصل السابق، فلو كان لدينا المتغير x التالي:

```
Sub MySub ()
    Dim X As Integer = 10
    …
End Sub
```

سيبقى هذا المتغير محتفظا بقيمته حتى الخروج من الإجراء ويختفي ويزال نهائيا من الذاكرة. أما الحديث عن المتغيرات المرجعية Reference Type:

```
Sub MySub ()
    Dim X As New SimpleClass
    …
End Sub
```

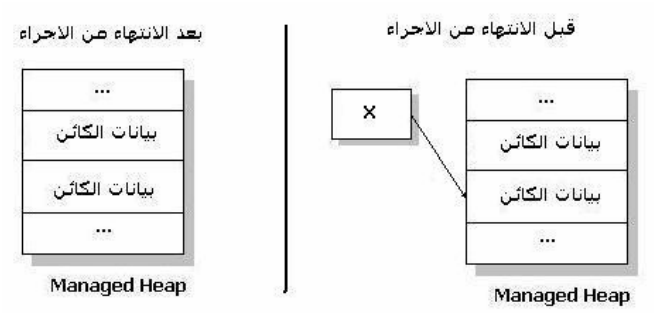
فسيبقى المؤشر X محتفظا بقيمته (القيمة هنا عنوان بيانات الكائن في ذاكرة Managed Heap) حتى نهاية الإجراء ومن ثم سيزال المؤشر X من الذاكرة وتلغى قيمته **ولكن الكائن سيستمر في الذاكرة! (شكل 3-3).**



شكل 3-3: الكائن ما زال موجود في ذاكرة Managed Heap

ازيدك من الشعر بيت، هل تصدق انك لو قمت باستدعاء الإجراء MySub() السابق مرة أخرى سيتم إنشاء نسخة جديدة من الكائن وستبقى حتى بعد نهاية الإجراء إلى جانب النسخة السابقة **(شكل 3-4 بالصفحة المقابلة).**

PDF Factory Pro trial version www.fastio.com



شكل 3-4: نسخة أخرى من الكائن في ذاكرة Managed Heap لم يتم ازالتها أيضا.

عجبا على هذا الغباء! لماذا لا تزال الكائنات من الذاكرة رغم أننا لا نريدها ولن نستطيع الوصول لها بسبب فقدان مؤشراتها؟ الجواب يتعلق بالبنية التحتية لإطار عمل NET. Framework وطريقة إدارته للذاكرة Managed Heap وتعامله معها. وحتى تستوعب ذلك، عليك أن تتعرف –شئت أم أبيت- على المجموعة Garbage Collection.

المرجعية الدائرية Circular Reference:

قبل التوغل في تفاصيل المجموعة Garbage Collection بودي توضيح قضية تعتبر معاناة كبيرة واجهها المبرمجون –وواجهتها أنا شخصيا- في لغات البرمجة السابقة، ألا وهي **المرجعية الدائرية Circular Reference**. في السابق، كانت الكائنات تموت مباشرة بعد اختفاء جميع المؤشرات التي تشير لها ومن ثم تتم عملية تفريغ مساحتها من الذاكرة، بعبارة أخرى، عندما يصل عدد المؤشرات التي تشير إلى الكائن (يسمى **العداد Counter**) إلى صفر، تتم عملية إلغاء بيانات الكائن الفعلية من الذاكرة.

وبما أن برامجنا كانت تعتمد اعتمادا كبيرا جدا على الكائنات، ظهرت لنا مشكلة المرجعية الدائرية Circular Reference والتي تبقي كائنين على قيد الحياة رغم اختفاء جميع مؤشراتها. (إن كنت من المبرمجين المبتدئين، فاعذرني على ذكر هذه المسائل ولكنها ستعرفك على احد الأسباب التي حدت بمطوري NET. أن يعتمدوا على المجموعة Garbage Collection)، وحتى تفهم كيف تحدث مشكلة المرجعية الدائرية افترض أن لدينا هذه الفئة والمسطورة في أعلى الصفحة التالية:

```
Class PersonClass
    Public Name As String
    Public Brother As PersonClass
End Class
```

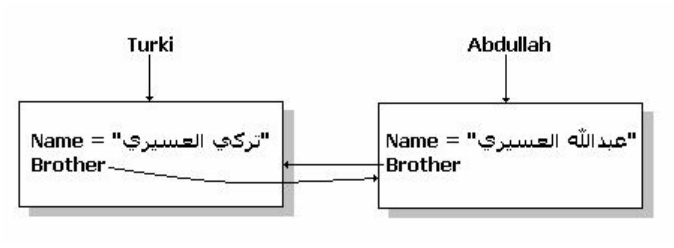
سننشئ كائنين ونسند اليهما قيم للحقول:

```
Dim Turki As New PersonClass()
Dim Abdullah As New PersonClass()

Turki.Name = "تركي العسيري"
Abdullah.Name = "عبدالله العسيري"
```

حتى الآن الكلام جميل جدا ولا توجد به أي مشاكل، ولكن عندما تسند القيم إلى الحقل Brother ستبدأ رحلة العناء والشقاء مع عالم الكائنات:

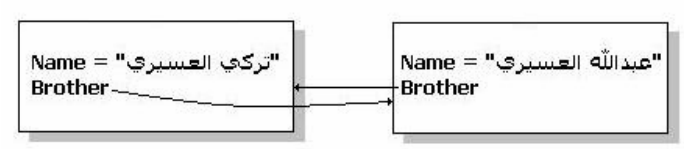
```
Turki.Brother = Abdullah
Adbdullah.Brother = Turki
```

في السطر السابق قمت بربط الكائنين Turki و Abdullah بحيث يشيران إلى بعضهما البعض (**شكل 3-5**)، هذه العملية تسمى المرجعية الدائرية Circular Reference.



شكل 3-5: المرجعية الدائرية Circular Reference.

والآن ركز معي يا عسل، تخيل أن الشيفرات السابقة كانت داخل إجراء معين وانتهى الإجراء بعد إسناد القيم، منطقيا ستفقد المؤشرات Turki و Abdullah قيمها وتزال من الذاكرة، ولكن الكائنات ما زالت موجودة وذلك بسبب الحقل Brother الذي لا يزال يشير إلى الكائن ويحميه من الموت (**شكل 3-6**) وهذه هي مشكلة المرجعية الدائرية.



شكل 3-6: مشكلة المرجعية الدائرية Circular Reference حدثت.

كما ترى في (**الشكل 3-6**)، الكائنات ستستمر بالذاكرة، ويمكنك التأكد بنفسك باستخدام هذه الشيفرة:

```
Dim Turki As New PersonClass()
Dim Abdullah As New PersonClass()

Turki.Name = "تركي العسيري"
Abdullah.Name = "عبد الله العسيري"

Turki.Brother = Abdullah
Adbdullah.Brother = Turki

Turki = Nothing ' قتل الكائن الاول

' الكائن الاول لا يزال حيا يرزق
ArabicConsole.WriteLine(Abdullah.Brother.Name) ' تركي العسيري
```

قد تتباهى بذكائك وتعطيني حلا سريعا لمشكلة المرجعية الدائرية، وذلك بإلغاء جميع الحقول المسببة لهذه المشكلة قبل الخروج من الإجراء ثم إلغاء المؤشرات الرئيسية:

```
' لنلغي الحقول اولا
Turki.Brother = Nothing
Abdullah.Brother = Nothing

' ثم المؤشرات الرئيسية
Turki = Nothing
Abdullah = Nothing
```

كلامك صحيح ولا غبار عليه، ولكنه ليس عملياً إلا في مثالنا السابق فقط، ففي البرامج الكبيرة ستتعامل مع مئات الكائنات، وكل كائن يحتوي على عشرات الحقول والتي تشير إلى كائنات أخرى لتصبح إمكانية نسيان تنظيف احد الحقول مسألة لا إرادية. وصدقني، مشكلة المرجعية الدائرية يقع فيها كبار المبرمجين المحترفين والدليل هو استهلاك البرامج للمصادر الكبيرة من الذاكرة والتي لا تستخدم.

PDF

المجموعة Garbage Collection:

بعدما عرضت عليك مشكلة المرجعية الدائرية، قد تفضل العودة إلى البرمجة الإجرائية التقليدية، ولكن أريدك أن تتغلب على الخوف الذي يصيبك من هذه المعلومات وتصبح مبرمج شجاع (كلمة الشجاعة لا يشترط ربطها ببرمجة Windows API **J**) وتستمر في استخدام الكائنات، لأنه يسرني إخبارك أن مشكلة المرجعية الدائرية لن تصادفها أبدا وذلك بفضل المجموعة Garbage Collection.

حسنا، يوفر لك إطار عمل NET. Framework المجموعة Garbage Collection والهدف الرئيسي منها هو تفريغ الذاكرة من الكائنات الغير مرغوبة، واعني بالكائنات الغير مرغوبة هي الكائنات التي لا يشير إليها أي مؤشر أو متغير ظاهر في شيفرات البرنامج وليس جميع المؤشرات، فلو أمعنت النظر في (**الشكل 3-6**) ستلاحظ ان الكائنين لهما مؤشران باسم Brother، وبما أن هذا المؤشر ليس ظاهر في شيفراتك المصدرية، ستقوم المجموعة Garbage Collection بإلغاء الكائنات وتحرير مساحتها من الذاكرة.

كلام مريح جدا ويزيح عن المبرمج هم التفكير في مسالة تحرير كائناته من الذاكرة، ولكن متى ستتم هذه العملية؟ إن كان السؤال موجه لي شخصيا، فانا لا اعلم متى ستتم عملية التحرير الفعلي للذاكرة، حيث انك في كل مرة تنشئ فيها كائن جديد، سيستمر هذا الكائن في ذاكرة Managed Heap حتى تمتلئ تماما، وبعد ذلك تتم عملية التحرير تلقائيا من قبل المجموعة Garbage Collection، وان كانت الكائنات المستخدمة قليلة ولا تستهلك تلك المساحة الكبيرة، فعملية التحرير لن تتم حتى نهاية البرنامج.

مع ذلك، يمكنك طلب عملية التحرير من المجموعة Garbage Collection في أي وقت تريده يدويا باستدعاء الطريقة Collect() التابعة للكائن GC:

```
GC.Collect()
GC.WaitForPendingFinalizers()
```

الكائن GC معرف في إطار عمل NET. Framework وهو يمثل المجموعة Garbage Collection، وبالنسبة للطريقة WaitForPendingFinalizers() فهي توقف عملية تنفيذ البرنامج مؤقتاً حتى الانتهاء من عملية تحرير المساحة.

مع ذلك، أنصحك بشدة عدم استدعاء الطريقة GC.Collect() بنفسك يدويا، فعملية تحرير الذاكرة تتطلب وقت طويل -خاصة ان كانت الكائنات كثيرة- مما يؤدي إلى بطء ملحوظ في تنفيذ

البرنامج، لذلك دع إطار عمل NET Framework. يحدد الوقت المناسب للقيام بهـذه العمليـة، واطلع منها أنت!

ملاحظة

ذكرت قبل قليل أن الطريقة WaitForPendingFinalizers() توقف عملية تنفيذ البرنامج بشكل مؤقت، ولكن الحقيقة أنها توقف مسار التنفيذ Thread الحالي فقط. سأتحدث بالتفصيل المملّ عن مسارات التنفيذ Threading في الفصل العاشر **مسارات التنفيذ** Threading.

الموت المنطقي والموت الحقيقي للكائنات:

ببساطة شديدة، الموت المنطقي للكائنات يحدث عندما تختفي جميع المؤشرات التي تشير إلى الكائن، أما الموت الحقيقي للكائن فيحدث عندما تقوم المجموعة Garbage Collection بتحرير مساحته من الذاكرة Managed Heap، فالكائن التالي:

```
Sub Test ()
    Dim TestObject As New TestClass()
    …
End Sub
```

سيموت منطقيا عند نهاية الإجراء وذلك لاختفاء المؤشر TestObject الذي يشير له، ولكنه لن يموت بشكل حقيقي إلا إن تمت عملية التحرير من قبل المجموعة Garbage Collection.

كيف يمكننا معرفة الوقت الذي يموت فيه الكائن، وذلك لان لغات البرمجة القديمة كانت توفر إجراء يسمى **المهدم Destructor** يتم استدعائه لحظة موت الكائن. يمكنك Visual Basic NET. من تعريف الإجراء Finalize() والذي يتم استدعائه من قبل المجموعة Garbage Collection (أي يتم تنفيذه لحظة الموت الحقيقي وليس المنطقي):

```
Class TestClass
    …
    Protected Overrides Sub Finalize()
        ' اكتب الشيفرة هنا
        …
    End Sub
End Class
```

PDF Factory Pro trial version www.fastio.com

انظر أيضا

استخدمت محدد الوصول المحمي Protected والمعامل إعادة القيادة Overrides وذلك لأني قمت باشتقاق الإجراء ()Finalize من الكائن System.Object، الفصل القادم **الوراثة** سيوضح لك السبب.

مع ذلك، فإن الاعتماد على الإجراء ()Finalize السابق فيه شيء من الخطأ، وذلك لأننا لا نعلم بالضبط متى سيكون الموعد الحقيقي لموت الكائن، ففي لغات OOP السابقة كنا نستخدم هذا النوع من المهدمات لحظة الموت المنطقي للكائن، أما الموت الحقيقي فقد يتم بعد الموت المنطقي بثواني، دقائق، أو حتى ساعات فهو مرتبطاً ارتباطاً كلياً بعملية التحرير التي تقوم بها المجموعة Garbage Collection.

تنبيه

لا تحاول الوصول إلى كائنات أخرى من داخل الإجراء ()Finalize (كـ ArabicConsole وغيرها) وذلك لان هذه الكائنات قد تكون تمت عملية موتها الحقيقي وأخليت من الذاكرة، مما قد يتسبب في ظهور رسالة خطأ. بصفة عامة، يستخدم الإجراء ()Finalize للقيام بعمليات أخيرة لا تستخدم كائنات أخرى في نفس المشروع أو الكائن نفسه.

بما أننا لا نستطيع تعريف مهدم في الفئة يتم تنفيذه لحظة الموت المنطقي للكائن، اعتمد مبرمجوا NET. على محاكاة المهدم وتعريف إجراء باسم ()Dispose في الفئة:

```
Class TestClass
    Implements IDisposable

    Public Sub Dispose() Implements System.IDisposable.Dispose

    End Sub
    …
    …
End Class
```

انظر أيضا

الكلمة المحجوزة Implements تتعلق بالواجهات Interfaces وهو موضوع الفصل الخامس **الواجهات، التفويض، والمواصفات**.

على المبرمجين الذين ينشئون كائنات من هذه الفئة، استدعاء الطريقة Dispose() لحظة الموت المنطقي للكائن، حتى يتمكن من القيام بما عليه القيام به (كإغلاق الملفات، قطع الاتصالات مع قواد البيانات، تحرير مصادر النظام ...الخ):

```
Dim TestObject As New TestClass

…
…

' استدعي الطريقة اولا
TestObject.Dispose()

' ثم اقتل الكائن منطقيا
TestObject = Nothing
```

بكل تأكيد قد تنسى –أو ينسى احد المبرمجين- استدعاء الطريقة Dispose() لحظة إلغاء المؤشر الأخير للكائن، لذلك لنجعل الطريقة Finalize() تستدعي Dispose() كنوع من الاحتياط:

```
Protected Overrides Sub Finalize()
    Dispose()
End Sub
```

وحتى لا يتم استدعاء الطريقة Dispose() أكثر من مرة بالخطأ، يفضل استخدام متغير ستاتيكي لمنع ذلك:

```
Public Sub Dispose() Implements System.IDisposable.Dispose
    Static CancelDisposing As Boolean

    If CancelDisposing Then
        Exit Sub
    End If
    …
    …
    CancelDisposing = True
End Sub
```

PDF

أسلوب أكثر امانا لكتابة المهدمات Finalize() و Dispose():

ذكرت سابقا، أن المبرمج يستدعي الطريقة Dispose() قبل لحظة موت الكائن منطقيا، مع ذلك لدى المبرمج فرصة كبيرة لاستدعاء طرق وخصائص الكائن بعد موته منطقيا (وهي فرصة كفيلة بحدوث عشرات المشاكل والشوائب البرمجية):

```
Dim TestObject As New TestClass

…
…

' قام المبرمج باستدعاء المهدم
TestObject.Dispose()

' لديه فرصه لاستدعاء إجراءات الكائن
' بعد موته المنطقي
TestObject.MethodInClass ()
```

لذلك عليك منع المبرمج من استدعاء طرق وخصائص الكائن بعد استدعاء المهدم Dispose()، قد تفعل ذلك باستخدام العبارة Exit Sub:

```
Class TestClass
    …
    …
    Private CancelDisposing As Boolean

    Sub MethodInClass ()
        ' منع المبرمج من استدعاء الإجراء بعد
        ' Dispose() بعد استدعاء المهدم
        If CancelDisposing Then
            Exit Sub
        End If
        …
        …
    End Sub

    Public Sub Dispose() Implements System.IDisposable.Dispose
        If CancelDisposing Then
            Exit Sub
        End If
        …
        …
        CancelDisposing = True
    End Sub
End Class
```

PDF

أو **رمي استثناء Throw an Exception** كما تفعل سائر كائنات إطار عمل .NET. الأخرى:

```
Class TestClass
    …
    …
    Sub MethodInClass ()
        ' منع المبرمج من استدعاء الإجراء بعد
        ' Dispose() بعد استدعاء المهدم
        If CancelDisposing Then
            Throw New ObjectDisposedException("TestClass")
        End If
        …
    End Sub
    …
    …
End Class
```

انظر أيضا

رمي استثناء Throw an Exception هي عملية أحداث خطأ وقت التنفيذ Run Time Error في البرنامج، الفصل السابع **اكتشاف الأخطاء** خاص بتدارك ورمي الاستثناءات.

المزيد أيضا، قد تستدعي المهدم Dispose() من داخل المهدم Finalize() كنوع من الاحتياط –كما ذكرت- وذلك في حالة عدم استدعائه من قبل المبرمجين لحظة الموت المنطقي للكائن:

```
Protected Overrides Sub Finalize()
    Dispose()
End Sub
```

مشكلة أخرى –ليست كبيرة- في الأسلوب السابق، وهي أن الإجراء Finalize() سيتم استدعائه دائما من قبل المجموعة Garbage Collection حتى وان قام المبرمج باستدعاء المهدم Dispose() يدويا، ولا تنسى أن عملية تحرير المصادر من قبل المجموعة Garbage Collection تستغرق وقت طويل دون حاجة، مما يعني أننا لسنا بحاجة لاستدعاء الإجراء Finalize() من قبل المجموعة Garbage Collection إن تم استدعاء الإجراء Dispose() من

قبل المبرمج. لذلك، قد تستدعي الطريقة GC.SuppressFinalize() من داخل الإجراء Dispose() لتمنع المجموعة Garbage Collection من استدعاء المهدم Finalize():

```
Class TestClass
    …
    …
    Public Sub Dispose() Implements System.IDisposable.Dispose
        …
        …
        ' Garbage Collection منع المجموعة
        ' Finalize من استدعاء المهدم
        GC.SuppressFinalize(Me)
    End Sub
End Class
```

نقطة أخرى حول الشيفرة قبل السابقة، قد تبدو فكرة استدعاء المهدم Dispose() من داخل المهدم Finalize() ذكية، ولكن توجد بها مشكلة قد تسبب في أحداث كوارث في البرنامج (خصوصا مع الفئات الكبيرة والكائنات المعقدة). قد تحدث هذه المشكلة لحظة الموت الحقيقي للكائن، حيث سيتم استدعاء الإجراء Dispose() والذي قد تصل شيفراته المصدرية إلى كائنات أخرى ميتة بسبب المجموعة Garbage Collection.

يمكنك الالتفاف حول هذه المشكلة بإعادة تعريف Overloads المهدم Dispose() بحيث يستقبل وسيطة إضافية تحدد فيها ما إذا تم استدعاء المهدم من قبل المبرمج (لحظة الموت المنطقي) أو من قبل المهدم Finalize() (لحظة الموت الحقيقي)، وبذلك يكون الأسلوب الأكثر امانا لكتابة المهدمات Finalize() و Dispose() (والذي تتبعه جميع كائنات إطار عمل NET. Framework) كالتالي:

```
Class TestClass
    Implements IDisposable
    …
    Private CancelDisposing As Boolean

    Public Overloads Sub Dispose() Implements IDisposable.Dispose
        Dispose(True)
        GC.SuppressFinalize(Me)
    End Sub
```

```
    Private Overloads Sub Dispose(ByVal disposing As Boolean)
        If CancelDisposing Then
            Exit Sub
        End If

        If disposing Then
            ' الشيفرات التي يتم تنفيذها لحظة استدعاء
            ' الإجراء من قبل المبرمج (الموت المنطقي)
        End If

        ' الشيفرات التي يتم تنفيذه لحظة استدعاء
        ' Finalize () الإجراء من قبل المهدم
        ' (الموت الحقيقي)

        CancelDisposing = True
    End Sub

    Protected Overrides Sub Finalize()
        Dispose(False)
    End Sub
End Class
```

إرسال الكائن بالمرجع أو القيمة

تحدثت سابقا عن عملية إرسال الكائن إلى وسيطات الإجراءات بالمرجع أو بالقيمة، وذكرت انه في الحالتين سيتمكن الإجراء من تعديل قيم أعضاء الكائن المرسل. اما الفرق بين استخدام ByVal و ByRef عند وسيطات الإجراءات فهو تقني بحت، إذ ان إرسال متغير الكائن باستخدام ByVal سيؤدي إلى نسخ قيمة المؤشر إلى مؤشر آخر، مما يزيد عدد المؤشرات التي تشير إلى نفس الكائن. أما إرسال متغير الكائن باستخدام ByRef، فهي ترسل عنوان ذلك المؤشر وليس قيمة المؤشر (والتي تحمل عنوان الكائن نفسه)، المثال التالي قد يوضح احد الفروق:

```
' لا تؤدي إلى موت الكائن ابدا
Sub ByValSub(ByVal TestObject As TestClass)
    TestObject = Nothing
End Sub

' تؤدي إلى موت الكائن ان ارسل
' المؤشر الوحيد
Sub ByRefSub(ByRef TestObject As TestClass)
    TestObject = Nothing
End Sub
```

إن كنت تخشى إرسال كائناتك إلى إجراءات تستقبل وسيطات بـــالمرجع ByRef مـــن أن تهلكها، فهذا من حقك ولا يلومك أي مبرمج، لذلك أضف أقواس إضافية حول القيمة المرسلة لتجبر Visual Basic .NET على إرسال المتغير بالقيمة (حتى وان كانت الوسيطة تستقبل بالمرجع):

```
Dim TestObject As New TestClass

' تم الارسال هنا بالقيمة رغم ان الوسيطات
' يتوقع انها تكون بالمرجع، فلا خوف على الكائن من الموت
ByRefSub ((TestObject))
```

الأعضاء المشتركة

كما رأيت سابقا، الأعضاء التابعة للكائنات تكون مستقلة بالكائن التابعة له، إلا انك فـــي حـــالات كثيرة تود من الكائنات المنشأة من فئة معينة أن تتشارك البيانات فيما بينها. في الفقـــرات التاليـــة سألقي الضوء على **الأعضاء المـــشتركة Shared Members** وكيفية تطبيقها علـــى الحقـــول، الطرق، الخصائص، والأحداث.

الحقول المشتركة Shared Fields

الحقول المشتركة Shared Fields هي حقول تكون قيمها مشتركة بين كافة الكائنات المنشأة من الفئات، وحتى تجعل الحقل مشترك استخدم الكلمة المحجوزة Shared عند التصريح عن الحقـــل. في الفئة التالية عرفت حقلين الأول تقليدي والثاني مشترك:

```
Class TestShared
    Public FirstName As String
    Shared Public LastName As String
End Class
```

عند التعامل مع الحقل الأول FirstName، سيستقل كل كائن بقيمة مختلفة لهذا الحقل، وهذا أمر منطقي:

```
Dim Meshari As New TestShared()
Dim Turki As New TestShared()

Meshari.FirstName = "مشاري"
Turki.FirstName = "تركي"

ArabicConsole.WriteLine(Meshari.FirstName)          ' مشاري
ArabicConsole.WriteLine(Turki.FirstName)            ' تركي
```

أما إن حاولت التعامل مع الحقل الثاني LastName، فعليك الأخذ بعين الاعتبار انه مشترك بين جميع الكائنات المنشئة من الفئة. لمعرفة الفرق في هذه الحالة، راقب الشيفرة التالية:

```
Meshari.LastName = "القحطاني"
Turki.LastName = "العسيري"

ArabicConsole.WriteLine(Meshari.LastName)          ' العسيري
ArabicConsole.WriteLine(Turki.LastName)            ' العسيري
```

نستنتج من ذلك، ان الحقول المشتركة ما هي إلا متغيرات قابلة للوصول من جميع الكائنات المنشأة من فئة معينة، بحيث تتشارك هذه الكائنات في قيمها، لذلك تستطيع الوصول إلى الحقــول المشتركة مباشرة من اسم الفئة دون الحاجة لإنشاء كائن منها:

```
' TestShared لاحظ استخدام الفئة
' Turki وليس الكائن
TestShared.LastName = "العسيري"
```

قد تفيدك الحقول المشتركة –مثلا- في معرفة عدد الكائنات المنشأة من فئة معينة:

```
Class TestShared
    Public Shared NumOfObjects As Integer = 0

    Sub New()
        NumOfObjects += 1
    End Sub
End Class
```

ستلاحظ ان الحقل NumOfObjects مشترك بين كافة الكائنات المنشئة، وستزداد قيمته في كــل مرة تنشئ كائن جديد من الفئة TestShared:

```
Dim ObjectOne As New TestShared()
Dim ObjectTwo As New TestShared()

ArabicConsole.WriteLine(ObjectOne.NumOfObjects)      ' 2

Dim ObjectThree As New TestShared()

ArabicConsole.WriteLine(ObjectThree.NumOfObjects)    ' 3
```

بل يمكنك تقليص عدد الكائنات التي تمكن مستخدم الفئة من إنشائها، فقد تظهر رسالة خطأ إن تجاوز عدد الكائنات المنشأة من الفئة عن عدد معين تحدده في جملة شرطية:

PDF

```
Sub New()
    NumOfObjects += 1

    If NumOfObjects > 10 Then
        ' الامر التالي سيؤدي إلى ظهور رسالة خطأ
        ' انتقل للفصل السابع: اكتشاف الاخطاء
        ' لمزيد من التفاصيل
        Throw New Exception()
    End If
End Sub
```

مئات التطبيقات والأفكار التي تستطيع إنجازها بفضل الحقول المشتركة، كل ما يهمني هنا توضيح الفكرة لك حتى تتمكن من عمل ما تريد، وقد تجد في فصول هذا الكتاب الكثير من الأمثلة التي تستخدم الحقول المشتركة.

الطرق المشتركة Shared Methods

الفكرة من **الطرق المشتركة Shared Methods** قد تبدو غريبة بعض الشيء، فالطرق العادية دائما ما تكون مشتركة بين كائنات الفئات، ولكن الفرق بينهما تقني بحت، فالطرق المشتركة لا تنتمي إلى أي نسخة كائن Object Instance معينة، لذلك يمكنك استخدامها دون الحاجة لإنشاء وتعريف كائن جديد، فلو أضفنا هذه الطريقة إلى الفئة السابقة:

```
Class TestShared
    …
    Public Shared Function CheckNumOfObjects() As Integer
        Return NumOfObjects
    End Function
End Class
```

يمكنك استدعاء الطريقة CheckNumOfObjects() مباشرةً دون الحاجة لتعريف كائن من الفئة:

```
Dim X As TestShared

If TestShared.CheckNumOfObjects < 10 Then
    X = New TestShared()

    ArabicConsole.WriteLine(X.NumOfObjects)
End If
```

من الضروري توضيح نقطة هامة، الشيفرة الموجودة داخل الطرق المشتركة ليس لها صلاحيات كالشيفرة الموجود في الطرق العادية، المقصد من كلمة الصلاحيات في هذا السياق هو

أن الشيفرة الموجودة داخل الطرق المشتركة لا يمكن أن تستخدم الأعضاء (الحقول، الطرق، الخصائص، والأحداث) العادية والتابعة لنفس الفئة، ولكن يمكن أن تصل إلى الأعضاء المشتركة فقط. وهذا الأمر يبدو منطقيا إن علمنا أن الطرق المشتركة لا تتبع لكائن معين فكيف تريد أن تصل إلى أعضاء كائن لم يتم إنشائه:

```
' ستظهر رسالة خطأ لان الطريقة المشتركة
' SharedFun()
' تستخدم عضو غير مشترك
Class My_Class
    Public X As Integer

    Public Shared Function SharedFun() As Integer
        Return X
    End Function
End Class
```

يمكنك الاستفادة من فكرة الطرق المشتركة ان كنت تطور مكتبة فئات وتود توفير بعض الطرق لمستخدمي الفئات دون الحاجة لتعريف كائنات جديدة، ولا بد أنك لاحظت أننا طيلة هذه الفترة استخدمنا الفئة ArabicConsole واستدعينا الطريقة WriteLine() دون الحاجة لإنشاء نسخة كائن جديدة، أي ان كل الذي فعله مؤلف الكتاب هو جعل الطريقة WriteLine() مشتركة، وبذلك سمحت لك باستخدامها مباشرة:

```
ArabicConsole.WriteLine("طريقة مشتركة")
```

إن كانت جميع أعضاء الفئة مشتركة، فيمكنك اعتبار الفئة كالوحدة البرمجية Module حيث ان الوحدات البرمجية ما هي إلا فئات جميع أعضائها مشتركة، ولكن نقطة الاختلاف الرئيسية بينها وبين الفئات هي أن الفئات يمكننا إنشاء نسخ منها تتمثل في كائنات.

أخيرا، إن عرفت الإجراء Sub Main كوظيفة مشتركة داخل الفئة، يمكنك استدعائها مع بداية تنفيذ البرنامج من صندوق الحوار Project Property Pages (**شكل 2-1 صفحة 32**).

الخصائص المشتركة Shared Properties

لا توجد أي إضافات أخبرك بها هنا عن الخصائص المشتركة فهي بالضبط مثل الطرق المشتركة، أي يمكن استدعائها دون الحاجة إلى تعريف كائن ولا يمكن ان تستخدم الشيفرة التي بداخلها أعضاء غير مشتركة، فسأكتفي بعرض هذا المثال لتعريف خاصية مشتركة:

```
Shared Property SharedProp() As Integer
    Get
        …
    End Get

    Set (ByVal Value As Integer)
        …
    End Set
End Property
```

الأحداث المشتركة Shared Events

أيضا، الأحداث المشتركة ما هي إلا أحداث تتأثر بها جميع الكائنات المنشئة من فئة لحظة انطلاقها، وحتى أوضح لك المنطق البرمجي لها، دعني اضرب مثال واقعي: عندما يموت احد الأشخاص فان الموت حدث يقع على ذلك الشخص لوحده، وردة الفعل أو تأثير الحدث سيكون على الشخص وحده فقط، أما إن وقعت كارثة جماعية (كوقوع زلزال) فان جميع الكائنات ستتأثر بهذا الحدث ويلقوا حتفهم. لتعرف حدث مشترك استخدم الكلمة المحجوزة Shared أيضا:

```
Class PersonClass
    Event Die()
    Shared Event AllDie() ' حدث مشترك

    Sub KillHim()
        RaiseEvent Die()
    End Sub

    Shared Sub EarthQuick()
        RaiseEvent AllDie()
    End Sub
End Class
```

ستلاحظ أن الطريقة المشتركة EarthQuick() هي التي ستفجر حدث الزلزال (لا يشترط أن تكون الطريقة مشتركة حتى يتم إطلاق حدث مشترك)، والنقطة الرئيسية التي عليك معرفتها هو أن حدث الزلزال AllDie سيتم إطلاقه في كافة الكائنات المنشئة من نفس الفئة، دعنا نرى تأثير الحدث Die العادي اولا:

```
Module Module1
    Sub Main()
        Dim Turki As New PersonClass()
        Dim Ali As New PersonClass()

        ' قنص الحدث لكلا الكائنين
        AddHandler Turki.Die, AddressOf PersonDie
        AddHandler Ali.Die, AddressOf PersonDie
```

PDF

```
        Turki.KillHim()

        Ali.KillHim()
    End Sub

    Sub PersonDie()
        ArabicConsole.WriteLine("توفي شخص")
    End Sub
End Module
```

ستلاحظ أن عملية إطلاق الحدث Die مرتبطة بكل كائن بشكل مستقل، فالإجراء PersonDie() السابق سيتم استدعائه مرتين بسبب استخدام الطريقة KillHim() مع كل كائن على حده، أما لو جربنا إطلاق الحدث المشترك AllDie فلن نحتاج إلا إلى عملية قنص واحدة فقط، وسيتم استدعائها لحظة إطلاق الحدث بغض النظر عن الكائن التابع له:

```
Module Module1
    Sub Main()
        Dim Turki As New PersonClass()
        Dim Ali As New PersonClass()

        ' قنص واحد فقط للحدث المشترك
        AddHandler PersonClass.AllDie, AddressOf PersonDie

        Turki.EarthQuick()
        Ali.EarthQuick()

        ' طريقة مشتركة لذلك يمكن EarthQuick
        ' استدعائها مباشرة
        PersonClass.EarthQuick()
    End Sub

    Sub PersonDie()
        ArabicConsole.WriteLine("توفي شخص")
    End Sub
End Module
```

تفيدك فكرة الأحداث المشتركة كثيرا إذا أردت قنص حدث واحد فقط لمجوعة كائنات مختلفة، فعندما نصل إلى الجزء الثالث **تطوير تطبيقات Windows** من هذا الكتاب، سننشئ مجموعة من الأدوات بحيث تتشارك جميعها في حدث واحد.

الصيغة القياسية للأحداث في عالم NET. :

ان كانت لي معزة صغيرة في قلبك أتمنى من هذه اللحظة حتى نهاية حياتك البرمجية مع Visual Basic .NET استخدام صيغة عالم NET. القياسية لتعريف الأحداث (سواء كانت مشتركة أو عادية)، فكما لاحظت سابقا أن كل الإجراءات يمكن أن تقنص الأحداث وتكون قابلة للتنفيذ لحظة انطلاق الحدث، وكما علمت أيضا أن جميع الإجراءات يمكنها أن تستقبل أحداث الكائنات المختلفة، لذلك اعتمد مبرمجو NET. على توحيد صيغة تعريف الأحداث حتى تكون قابلة للعمل على جميع الإجراءات.

ليس هذا فقط، بل حتى عندما تتوغل في عالم برمجة إطار عمل NET Framework. ستلاحظ ان جميع الأحداث في جميع الكائنات المختلفة صيغتها موحدة. صحيح ان تطبيق هذه الصيغة قد يكلفك وقت ومجهود إضافي، إلا انك ستوفر الكثير من هذا المجهود لاحقا، وبالذات ان كبر حجم برامجك ومشاريعك.

القاعدة الأولى التي عليك معرفتها لتطبيق الصيغة القياسية هي: جميع الأحداث التي تعرفها ترسل لها وسيطتين الأولى تسمى sender والثانية e:

```
Class PersonClass
    Event Die(ByVal sender As Object, ByVal e As System.EventArgs)
    …
End Class
```

حيث sender هو الكائن الذي أطلق فيه الحدث، بينما e تمثل جميع الوسيطات Parameters التي يرسلها الحدث، وبما ان الحدث Die السابق لن يقوم بإرسال أية وسيطات إضافية، لذلك استخدمنا الفئة System.EventArgs والمقدمة من إطار عمل NET Framework. حيث لن تحتوي على اية وسيطات، لذلك عند إطلاق الحدث أنشئ الكائن عند إرساله للوسيطة مباشرة:

```
Class PersonClass
    …
    Sub KillHim()
        RaiseEvent Die(Me, New System.EventArgs())
    End Sub
End Class
```

والآن يمكنك قنص الحدث بنفس الطريقة إما باستخدام WithEvents أو AddHandler:

```
Module Module1
    Sub Main()
        Dim Turki As New PersonClass()
        AddHandler Turki.Die, AddressOf PersonHasDied

        Turki.KillHim()
    End Sub

    Sub PersonHasDied(ByVal sender As Object, ByVal e As _
        System.EventArgs)

        ArabicConsole.WriteLine("لقد توفي الشخص")
    End Sub
End Module
```

من ناحية أخرى، إذا أردت من الحدث إرسال وسيطات إضافية وكنت تود الاستمرار على صيغة NET. القياسية للأحداث، عليك تعريف فئة جديدة خاصة لوسيطات الحدث (مع ضرورة اشتقاق الفئة System.EventArgs وراثيا باستخدام الكلمة المحجوزة Inherits)، مثلا حدث السفر Travel:

```
Class TravelEventArgs
    Inherits System.EventArgs

    Public FromCity As String
    Public ToCity As String

    Sub New(ByVal fromCity As String, ByVal toCity As String)
        Me.FromCity = fromCity
        Me.ToCity = toCity
    End Sub
End Class
```

والآن سنعود إلى الفئة PersonClass مرة أخرى لنعرف فيها الحدث، وطريقة تمكننا من إطلاقه:

```
Class PersonClass
    Event Travel(ByVal sender As Object, ByVal e As TravelEventArgs)

    Sub Move(ByVal fromCity As String, ByVal toCity As String)
        RaiseEvent Travel(Me, New TravelEventArgs(fromCity, toCity))
    End Sub
    …
End Class
```

أخيرا، الشيفرة المسطورة في الصفحة التالية توضح لك طريقة قنص الحدث Travel السابق وهي شبيهه تماماً بالطرق السابقة، ولا تنسى تعديل الوسيطات حتى تتم عملية القنص بشكل صحيح، (يمكنك أيضا استخدام WithEvents أو AddHandler):

```
Module Module1
    Sub Main()
        Dim Turki As New PersonClass()
        AddHandler Turki.Travel, AddressOf PersonHasTraveld

        Turki.Move("الظهران", "الرياض")
    End Sub

    Sub PersonHasTraveld(ByVal sender As Object, ByVal e As
TravelEventArgs)

        ArabicConsole.WriteLine("سافر الشخص من " & e.FromCity _
        & "إلى " & e.ToCity)

    End Sub
End Module
```

بعد هذا الفصل الطويل جدا والمعقد جدا عرفتك على أهم مواضيع برمجة Visual Basic .NET والذي يتعلق بالفئات Classes والكائنات Objects وكيفية التعامل معها. لا تحاول الانتقال إلى الفصل التالي حتى تتأكد من مدى استيعابك لهذا الفصل، حيث أن جميع فصول الكتاب وبرمجة إطار عمل Framework .NET. بشكل عام تعتمد اعتماد كلي على الفئات والكائنات. إن كنت مستوعباً لمحتويات هذا الفصل، فمرحبا ألف بك مع عالم **الوراثة** عنوان الفصل التالي.

PDF Factory Pro trial version www.fastio.com

الوراثة

لا أستطيع إعطائك نسبة معينة تمثل أهمية موضوع الوراثة في Visual Basic .NET، ولكنـــي سأخبرك ببساطة ان جميع فئات مكتبة إطار عمل NET. Framework متوارثة فيما بينها. وان لم تكن جادا في استيعاب هذا المبدأ، فستصادف الكثير من المتاعب عندما تتوغل في استخدام مكتبة فئات إطار عمل NET Framework. عاجلا أو آجلا.

هذا الفصل هو مدخلك الرئيسي للوراثة ومواضيع أخرى لها صلة، ولا أذيع سرا إن أخبرتك أنني استمتع جدا في الحديث عن هذا الموضوع، لذلك سأبدأ معك من الصفر وسأفترض انـــك لا تعلم أي شيء عن هذا المبدأ، وأعدك أني سأحاول جعل مادة هذا الفصل خفيفة وممتعة وقابلة للفهم السريع بمشيئة الله، وكل ما اطلبه منك هو التركيز في الشيفرات المصدرية، فهي مفتاح المعرفـــة لاستيعاب مبدأ الوراثة.

مقدمة إلى الوراثة

هذا القسم من الفصل هو مدخلك المبدئي إلى الوراثة وكيفية تطبيقها بـــ Visual Basic .NET.

مبدأ الوراثة

إن كنت على دراية كافية بمبدأ الوراثة، يمكنك الانتقال إلى فقرة **تطبيق الوراثـــة بـــــ Visual Basic .NET** دون أي مشاكل، أما إن كان هذا المصطلح جديدا عليك، فأستطيع أن اعرف لـــك **الوراثة Inheritance** على أنها قدرة الفئة (**الفئة المشتقة Derived Class**) علـــى اشـــتقاق أعضاء من فئة أخرى (**الفئة القاعدية Base Class**) دون الحاجة لإعادة تعريفها من جديد.

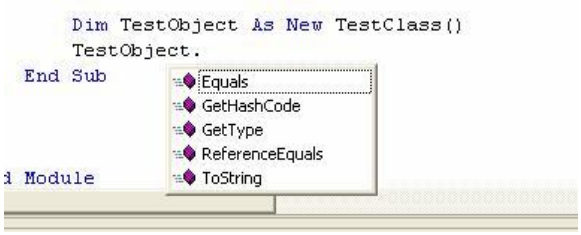
فلو كانت لدينا فئة تمثل شخص Person بها خصائص كالاســـم Name والعمـــر Age، يمكننا اشتقاق فئة أخرى منها Employee لترث الخاصيتين من الفئة القاعدية (Name و Age) بالإضافة إلى بعض الأعضاء الخاصة بها، لنستطيع كتابة شيئا مثل:

```
Dim Turki As New Empolyee

' أعضاء من الفئة القاعدية
' Person
Turki.Name = "تركي العسيري"
Turki.Age = 99

' أعضاء اضافية من الفئة
' Employee
Turki.Job = "مطور برامج ومواقع ويب"
Turki.Salary = 100
…
```

المزايا التي تجنيها من الوراثة لا تعد ولا تحصى، فيكفي انك تستطيع تطوير فئة معينة وذلك باشتقاقها وراثيا لتعريف فئة أخرى. من ناحية أخرى، ان اكتشفت احد الأخطاء في تصميم الفئة القاعدية (الفئة Person مثلا)، فلست بحاجة إلى تنقيح الفئة المشتقة من جديد (Employee)، حيث ان الفئة المشتقة ستتأثر تلقائيا بجميع التعديلات التي تجريها على الفئة القاعدية. المزيد أيضا، لست بحاجة إلى إعادة تكرار عملية بناء الفئات للأعضاء القياسية لتوفير الوقت، فعندما تنشئ الفئة Employee لن تضطر إلى إعادة تعريف الخصائص Age و Name فانك ستشتقها من الفئة Person، ويكفي إضافة الأعضاء التي تود بنائها داخل الفئة المشتقة فقط.

جميع الفئات المعرفة في مكتبة فئات إطار عمل NET. Framework تستخدم الوراثة، والدليل على ذلك لو أنك عرفت فئة خالية وأنشئت كائن منها، ستلاحظ وجود طرق إضافية (**شكل 4-1**):

```
Class TestClass
    ' فئة لا تحتوي على اية أعضاء
End Class
```



شكل 4-1: أعضاء إضافية تابعة للكائن رغم أن الفئة لا تحتوي على أية أعضاء.

من أين أتت هذه الأعضاء الإضافية؟ والجواب بكل بساطة من الفئة System.Object والتي ترث منها جميع مكتبة فئات إطار عمل NET. Framework الأخرى –كما سترى لاحقا في الفصل السادس **الفئات الأساسية**.

لغويا، تسمى العلاقة -في لغات OOP- بين الفئات المشتقة بعلاقة **هو Is a**، فعند الحديث عن فئة الموظف Employee يمكننا ان نطلق عليه عبارة **هو** شخص Person.

تطبيق الوراثة بـ Visual Basic .NET

والآن دعني اعرض عليك مثالا مبسطا يمكنك من استيعاب الوراثة بشكل تطبيقي، عرف هذه الفئة التي تمثل شخص في احد ملفات المشروع:

```
Class Person
    Public Name As String
    Public Age As Integer
End Class
```

كل ما عليك فعله لتطبيق مبدأ الوراثة بـــ Visual Basic .NET هو استخدام الكلمة المحجـــوزة Inherits وإلحاقها باسم الفئة المراد اشتقاقها وراثيا:

```
Class Employee
    Inherits Person ' Person وراثة أعضاء الفئة

    Public Job As String
    Public Salary As Double
End Class
```

يمكنك البدء فورا الآن بإنشاء كائن من الفئة المشتقة ليصل إلى جميع أعضاء الفئة القاعدية والفئة المشتقة:

```
Sub Main()
    Dim Turki As New Employee()

    ' أعضاء الفئة القاعدية
    Turki.Name = "تركي العسيري"
    …

    ' أعضاء الفئة المشتقة
    Turki.Job = "مطور برامج ومواقع ويب"
    …
```

PDF

```
    ArabicConsole.WriteLine(Turki.Name)        ' تركي العسيري
    ArabicConsole.WriteLine(Turki.Job)         ' مطور برامج ومواقع ويب
    …
End Sub
```

ضع في اعتبارك أن الفئة Employee هي الوارثة من الفئة Person والعكس غير صحيح، فلو حاولت تعريف كائن من الفئة القاعدية، لن تتمكن من الوصول إلى الفئة المشتقة (لان الفئة Person لا تعلم انه تم اشتقاقها):

```
Dim Turki As New Person()

' ممكن كما علمنا
Turki.Name = "تركي العسيري"

' مستحيل
Turki.Job = "مطور برامج ومواقع ويب"
```

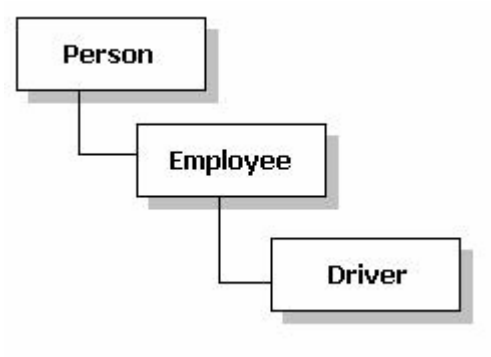
من ناحية أخرى، تستطيع اشتقاق الفئة Person أكثر من مرة لتعرف فئة أخرى بنفس الطريقة:

```
Class Student
    Inherits Person

    Public Department As String
    Public Grade As Integer
End Class
```

بل يمكنك أيضا اشتقاق فئة مشتقة أخرى، فيمكنك مثلا تعريف الفئة Driver المشتقة من الفئة Employee المشتقة من الفئة Person (**شكل 4-2 بالصفحة المقابلة**):

```
Class Driver
    Inherits Employee

    Public CarModel As String
    Public LicenseNumber As String
End Class
```

PDF Factory Pro trial version www.fastio.com



شكل 4-2: شكل يوضح العلاقة الوراثية بين الفئات.

وبشكل منطقي، الكائنات المنشئة من الفئة Driver يمكنها الوصول إلى أعضاء الفئة Employee والفئة Person:

```
Dim Abbas As New Driver()

' Person أعضاء من
Abbas.Name = "عباس السريع"
…

' Employee أعضاء من
Abbas.Job = "سائق خاص"
…

' Driver أعضاء من
Abbas.CarModel = "BMW - 7 Class"
…
```

ملاحظة

إن كنت من مبرمجي OOP المخضرمين، دعني أوضح لك أن (**شكل** 4-2) يبين العلاقة الوراثية بين الفئات وليس علاقة **الاحتواء** Containment بين الكائنات.

مع ذلك، لا يمكن للفئة الواحدة أن تشتق أعضاء من أكثر من فئة وتطبيق ما يسمى **الوراثة المتعددة Multiple Inheritance** (والذي لا تدعمه كل لغات .NET. الأخرى باستثناء Visual C++):

```
Class Thing
    Inherits Person
    Inherits Animal     ' رسالة خطأ
    …
    …
End Class
```

التعامل مع الفئات الوارثة والمورثة

في الصفحات السابقة عرفتك على الفكرة الأساسية من الوراثة والفئات القاعدية Base Classes والفئات المشتقة Derived Classes، ووضحت لك كيف تطبق مبدأ الوراثة بـــ Visual Basic .NET عن طريق استخدام العبارة Inherits، مع ذلك عليك معرفة الكثير من المسائل والحالات الخاصة التي ستواجهها عند تطبيق الوراثة على الفئات. الفقرات التالية ستخبرك بهذه التفاصيل.

وراثة الأعضاء

لا تقتصر عملية الاشتقاق الوراثي على حقول الفئات فقط، بل جميع الأعضاء الأخرى (الأحداث، الخصائص، والطرق) يتم اشتقاقها أيضا:

```
Class Person
    ' حدث
    Event Die()

    ' خاصية
    Private m_BirthDate As Date
    Property BirthDate() As Date
        Get
            Return m_BirthDate
        End Get
        Set(ByVal Value As Date)
            m_BirthDate = Value
        End Set
    End Property

    ' طريقة
    Sub KillHim()
        RaiseEvent Die()
    End Sub
    …
End Class
```

سيتم اشتقاق هذه الأعضاء في الفئة بمجرد استخدام العبارة Inherits، لتتمكن من الوصول إليهـــا واستخدامها بنفس الأساليب السابقة:

```
Module Module1
    Sub Main()
        Dim Turki As New Employee()

        ' الحدث
        AddHandler Turki.Die, AddressOf PersonHasDied

        ' الخاصية
        Turki.BirthDate = #1/1/9999#

        ' الطريقة
        Turki.KillHim()
    End Sub

    Sub PersonHasDied()
        ArabicConsole.WriteLine ("لقد توفي الشخص")
    End Sub
End Module
```

ليس هذا فقط، بل حتى الأعضاء المشتركة Shared Members يتم وراثتها بنفس الطريقة، ولكن ضع في عين الاعتبار أن الأعضاء المشتركة شاملة لكل الفئات الوارثة والمورثة، فلو عرفت هذا الحقل المشترك:

```
Class Person
    Public Shared LastName As String
End Class
```

عليك معرفة ان هذا الحقل سيتبع الفئة القاعدية وجميع الفئات المشتقة، فلا تتوقع وجود نسختين منه (نسخة لـــ Person وأخرى لـــ Employee)، وكإثبات لكلامي أسوق لك هذا المثال:

```
Dim Turki As New Person()              ' كائن من الفئة القاعدية
Dim Abdullah As New Employee()         ' كائن من الفئة المشتقة

' تعديل قيمة الحقل المشترك للفئة القاعدية
Turki.LastName = "العسيري"

' هو نفسه الحقل الموجود في الفئة المشتقة
ArabicConsole.WriteLine(Abdullah.LastName) ' العسيري
```

الكلام السابق يطبق أيضا على الطرق المشتركة، الخصائص المشتركة، والأحداث المشتركة.

على صعيد آخر، يمكن للفئة المشتقة الوصول إلى جميع أعضاء الفئة القاعدية، شريطة ان تكون الأعضاء عرفت على مستوى Public أو Friend، أما الأعضاء الخاصة Private فلا يمكن الوصول لها إلا من داخل الفئة القاعدية نفسها:

```
Class Employee
    Inherits Person

    …
    …
    Sub MethodInEmployee()
        ' الوصول إلى أعضاء الفئة القاعدية
        Me.Name = "تركي العسيري"
        Me.Age = 99
    End Sub
End Class
```

مع ذلك، لن تستطيع الوصول إلى أعضاء الفئة القاعدية بالطريقة السابقة عند تشابه أسماء أعضاء الفئة القاعدية والفئة المشتقة، خاصة عند تطبيق ما يسمى إعادة القيادة Overriding كما سترى لاحقا في القسم **إعادة القيادة Overriding** من هذا الفصل.

المشيدات Constructors

عند الحديث عن المشيدات، انسى كل ما ذكرته في الفقرة السابقة، وذلك لأن المشيدات لا يتم اشتقاقها بين الفئات كباقي الأعضاء لحظة الوراثة، ولكنها تتطلب منك كتابة شيفرات إضافية لتفعيلها مع الفئات القاعدية والمشتقة. وحتى اسهل عليك الأمر، دعنا نأخذ ثلاث حالات: الحالة الأولى عند وجود مشيد في الفئة المشتقة فقط دون الفئة القاعدية:

```
Class Person
    …
    ' لا يوجد مشيد
    …
End Class

Class Employee
    Inherits Person

    …
    …
    ' مشيد في الفئة المشتقة
    Sub New(ByVal Job As String)
        Me.Job = Job
    End Sub
End Class
```

PDF Factory Pro trial version www.fastio.com

في هذه الحالة، الغي كلمة الوراثة من خيالك وتعامل مع الفئة Employee وكأنها مستقلة ولم ترث أي شيء لحظة تمرير الوسيطات إلى المشيد:

```
Sub Main()
    ' وسيطات مشيد الفئة المشتقة
    Dim Turki As New Employee("مطور برامج ومواقع ويب")

    Turki.Name = "تركي العسيري"
    ArabicConsole.WriteLine(Turki.Job)
    …
    …
End Sub
```

الحالة الثانية تكون عند وجود مشيد في الفئة القاعدية فقط دون الفئة المشتقة، عليك هنا بذل المستحيل وعمل كل ما تستطيع عمله حتى تتمكن من تنفيذ مشيد الفئة القاعدية باستخدام الكائن MyBase وليس Me (سأتحدث عن MyBase لاحقا). وحتى تتمكن من عمل ذلك، عليك تعريف مشيد في الفئة المشتقة حتى لو لم تكن هناك حاجة إليه:

```
Class Person
    …
    ' مشيد الفئة القاعدية
    Sub New(ByVal Name As String)
        Me.Name = Name
    End Sub
End Class

Class Employee
    Inherits Person

    …
    …
    ' علينا تعريف هذا المشيد حتى نتمكن
    ' من استدعاء مشيد الفئة القاعدية
    Sub New()
        MyBase.New("تركي العسيري")
    End Sub
End Class
```

إن لم تطبق ما أخبرتك به في هذه الحالة، سيظهر لك مترجم Visual Basic .NET رسالة خطأ ولن تتمكن من تنفيذ البرنامج. أخيرا، عند إنشاء الكائن من الفئة Employee فلست بحاجة إلى إرسال أي وسيطات لمشيدها:

PDF

```
Sub Main()
    ' مشيد الفئة المشتقة لا يتطلب أي وسيطات
    Dim Turki As New Employee()

    Turki.Job = "مطور برامج ومواقع ويب"
    ArabicConsole.WriteLine(Turki.Name)
    …
    …
End Sub
```

الحالة الثالثة والأخيرة التي أود إخبارك بها تكون عند وجود مشيدات في كلا الفئتين (الفئة القاعدية والفئة المشتقة)، عليك إرسال وسيطات مشيد الفئة القاعدية من داخل الفئة المشتقة، وإرسال وسيطات مشيد الفئة المشتقة لحظة إنشاء الكائن:

```
Class Person
    …
    …
    Sub New(ByVal Name As String)
        Me.Name = Name
    End Sub
End Class

Class Employee
    Inherits Person

    …
    …
    Sub New(ByVal Name As String, ByVal Job As String)
        ' استدعي مشيد الفئة القاعدية اولا
        MyBase.New(Name)

        ' ثم قم بعمل ما تريد
        Me.Job = Job
    End Sub
End Class

Module Module1
    Sub Main ()
        ' لاستخدام الكائن ارسل وسيطات
        ' مشيد الفئة المشتقة
        Dim Turki As New Employee("تركي العسيري", "مطور برامج ومواقع ويب")
        …
        …
    End Sub
End Module
```

قبل ان اختم فقرة المشيدات، علي التنويه بضرورة إرسال القيم لوسيطات مشيد الفئة القاعدية باستخدام MyBase قبل كتابة أي حرف من شيفرات مشيد الفئة المشتقة، فالمشيد التالي سيظهر رسالة خطأ:

```
Sub New(ByVal Name As String, ByVal Job As String)
    ' رسالة خطأ لاننا لم نرسل القيم لوسيطات
    ' مشيد الفئة القاعدية اولا
    Me.Job = Job
    MyBase.New(Name)
End Sub
```

التعامل مع الكائنات

لننتقل من مرحلة تأليف وبناء الفئات إلى مرحلة استخدامها وإنشاء الكائنات، أنت تعلم وأنا اعلم أن المتغيرات من النوع Object يمكن إسناد أي قيمة لها:

```
Dim Turki As Object

Turki = New Employee()
…
…
```

إذا أردت معرفة السبب الذي يسمح لنا بفعل ذلك، يمكنني أن ألخصه لك بعبارة: جميع أنواع البيانات –بما فيها الفئات- في عالم NET. مشتقة من النوع Object (الاسم الكامل له هو System.Object), ومن الثمار الابتدائية التي تجنيها من الوراثة هو قدرتك على إسناد كائن من فئة مشتقة إلى كائن من فئة قاعدية، فالعملية التالية:

```
Dim Turki As New Employee()
Dim Turki2 As Person

Turki.Name = "تركي العسيري"

Turki2 = Turki ' ممكن جدا

ArabicConsole.WriteLine(Turki2.Name) ' تركي العسيري
```

صحيحة مية مية يا باشا، والسبب ان جميع أعضاء الفئة القاعدية Person موجودة ايضا في الفئة المشتقة Employee، لذلك سمح لنا NET. Visual Basic من إسناد قيمة الكائن Turki إلى الكائن Turki2. مع ذلك، لا يزال الكائن Turki2 منشأ من الفئة القاعدية فقط، فلا تحاول الوصول إلى أعضاء مشتقة في الفئة Employee –كما تفعل مع الكائن Turki:

```
' رسالة خطأ
Turki2.Salary = 1
```

بفضل قدرة إسناد قيم مشتقة إلى كائنات منشئة من فئات قاعدية، يمكنك –مثلا- تعريف إجراء واحد فقط ليستقبل وسيطات من كائنات لفئات مشتقة مختلفة:

```
Sub PrintInfo(ByVal personObject As Person)
    ArabicConsole.WriteLine(personObject.Name)
    ArabicConsole.WriteLine(personObject.Age)
End Sub
```

لتتمكن من إرسال كائنات من النوع Person أو من جميع الأنواع الأخرى بشرط ان تكون مشتقة من Person:

```
Dim Turki As New Person()
Dim Ali As New Employee()
Dim Umar As New Student() ' بافتراض ان Student مشتق من Person ايضا

…
…

' كل هذه الاستدعاءات صحيحة
PrintInfo(Turki)
PrintInfo(Ali)
PrintInfo(Umar)
```

ولمزيد من التوضيح، يسمح لك Visual Basic .NET بإسناد هذا النوع من القيم من كائن فئة مشتقة إلى كائن فئة ام فقط، فان حاولت عمل العكس ستظهر رسالة خطأ:

```
Dim Abood As New Person()
Dim Abood2 As Employee

Abood.Name = "عبود اللوح"

Abood2 = Abood ' رسالة خطأ
```

مع ذلك، يمكنك الالتفاف حول رسالة الخطأ عن طريق انشاء نسخة جديدة من كائن الفئة Employee وإسناد الكائن إلى متغير من النوع Person:

```
' بافتراض ان
' Option Strict Off
Dim Abood As Person
Dim Abood2 As Employee
```

PDF

```
Abood = New Employee()
Abood.Name = "عبود اللوح"

Abood2 = Abood ' ممكن

ArabicConsole.WriteLine(Abood2.Name) ' عبود اللوح
```

كما هو موضح في تعليق الشيفرة السابق، لابد من تعطيل العبارة Option Strict Off حتى تتمكن من إسناد القيمة، اما ان كنت مبرمج لبق جدا وتفعل Option Strict On دائما في مشاريعك (كما هو الحال مع شيفرات هذا الكتاب)، فعليك استخدام المعامل CType:

```
' في حالة تفعيل
' Option Strict On
…
Abood2 = CType(Abood, Employee)
…
```

انظر أيضا

لمزيد من التفاصيل حول المعامل CType، راجع الفصل الثاني **لغة البرمجة**.

إن كنت من مبرمجي OOP المخضرمين، فقد تطلق على العمليات السابقة التعبير **تعدد الواجهات Polymorphism**، وفي الحقيقة تعبيرك في محله ولكن ينقصه شيء بسيط حتى يقبل على هذا التعبير، سأريك النقص في الفصل القادم **الواجهات، التفويض والمواصفات** بمشيئة الله.

إعادة القيادة Overriding

افترض أننا نريد تعريف طريقة ()ShowName في فئة Person لتعرض لنا الاسم الأول والأخير للشخص، ومن ثم تقوم باشتقاقها في فئة Employee:

```
Class Person
    Public FirstName As String = "عباس"
    Public LastName As String = "السريع"

    Sub ShowName()
        ArabicConsole.WriteLine(Me.FirstName & " " & Me.LastName)
    End Sub
End Class
```

PDF

```
Class Employee
    Inherits Person
    …
    …
End Class
```

وبعد ان قمت باشتقاق هذه الفئة لتعرف فئة الموظف Employee اكتشفت لاحقا أن الشركة التي تستخدم برنامجك لا تعتمد على هذه الصيغة في كتابة الأسماء، حيث أنها تبدأ باسم القبيلة أو العائلة أولا ومن ثم الاسم الأول للشخص، قد تقوم بإعادة كتابة الطريقة ()ShowName في الفئة القاعدية Person:

```
Class Person
    …
    …
    Sub ShowName()
        ArabicConsole.WriteLine(Me.LastName & ", " & Me.FirstName)
    End Sub
End Class
```

لقد اقترفت إثما برمجيا وبهتانا عظيما في التعديل السابق! والسبب انه قد توجد مئات الفئات الأخرى في البرنامج تشتق الفئة السابقة Person لتؤثر على طرق تنسيقها بشكل سلبي، وما هي فائدة الوراثة إذاً إن قمت بتعديل الفئة القاعدية وأنت تريد تطوير فئة منها (الفئة المشتقة) دون المساس بالفئة القاعدية؟

أفضل حل سنقوم بتطبيقه يتم بإعادة كتابة الطريقة ()ShowName في الفئة المشتقة Employee بحيث تكون هذه الطريقة خاصة بها ولن تتأثر لا الفئة القاعدية ولا الفئات الأخرى المشتقة منها بهذه الإضافة، وهذا بالضبط ما يسمى **إعادة القيادة Overriding**. حتى تسمح للطريقة المعرفة في الفئة القاعدية من إعادة قيادتها، عليك استخدام الكلمة المحجوزة Overridable:

```
Class Person
    Public FirstName As String = "عباس"
    Public LastName As String = "السريع"

    ' لنسمح بإعادة قيادة هذه الطريقة
    ' في الفئات المشتقة من هذه الفئة
    Overridable Sub ShowName()
        ArabicConsole.WriteLine(Me.FirstName & " " & Me.LastName)
    End Sub
End Class
```

PDF Factory Pro trial version www.fastio.com

والآن يمكنك إعادة قيادة هذه الطريقة متى ما شئت لحظة الاشتقاق الوراثي بإلحاق الكلمة المحجوزة Overrides قبل اسم الطريقة:

```
Class Employee
    Inherits Person

    ' إعادة قيادة الطريقة
    Overrides Sub ShowName()
        ArabicConsole.WriteLine(Me.LastName & ", " & Me.FirstName)
    End Sub
End Class
```

عند استدعاء هذه الطريقة لكائن من نوع Person سيتم تنفيذ الطريقة التابعة للفئة القاعدية، وعند استدعائها لكائن من نوع Employee سيتم تنفيذ الطريقة المعاد قيادتها:

```
Dim Abbas As New Person()
Dim Abbas2 As New Employee()

Abbas.ShowName()        ' عباس السريع
Abbas2.ShowName()       ' السريع، عباس
```

إعادة قيادة الطرق والخصائص

عملية إعادة القيادة تقتضي إعادة بناء وكتابة الإجراء كي يناسب الفئة المشتقة، فلذلك من البديهي انك لن تستطيع إعادة قيادة إلا الطرق Methods والخصائص Properties فقط. بالنسبة للطرق، فقد عرضت في الفقرة السابقة مثالا لإعادة قيادتها، ولن تتمكن من فعل ذلك إلا عند استخدام الكلمة المحجوزة Overridable قبل اسم الطريقة في الفئة القاعدية، وفي كل مرة تود إعادة قيادة طريقة في فئة مشتقة استخدم الكلمة المحجوزة Overrides.

استخدامك للكلمة Overrides في الفئة المشتقة شبيه باستخدام Overridable في الفئة القاعدية من ناحية صلاحية إعادة القيادة، والدليل انك لو حاولت اشتقاق الفئة المشتقة Employee لتعرف فئة جديدة (Driver مثلا) فستتمكن أيضا من إعادة قيادة طرقها والمعرفة بـ Overrides:

```
Class Driver
    Inherits Employee

    Overrides Sub ShowName()
        ' يمكنك إعادة قيادة الطريقة للمرة الثالثة
        …
    End Sub
End Class
```

PDF

نستنتج من كل ذلك، ان Overrides تؤدي عمل Overridable ولكـــن Overridable لا تؤدي عمل Overrides. ففي المثال السابق تمكنا من إعادة قيادة الطريقة ()ShowName مـــن الفئة Employee، وحتى تمنع حدوث ذلك يمكنك استخدام الكلمة المحجوزة NotOverridable:

```
Class Employee
    Inherits Person

    ' منع الفئات المشتقة من هذه الفئة
    ' من إعادة قيادة هذه الطريقة
    NotOverridable Overrides Sub ShowName()
        ArabicConsole.WriteLine(Me.LastName & ", " & Me.FirstName)
    End Sub
End Class
```

ملاحظة

ذكرت قبل قليل أن مبدأ إعادة القيادة قابل للتطبيق على الطرق والخصائص فقط، وهذا لا يشمل الطرق والخصائص المشتركة حيث انك لن تستطيع إعادة قيادتها.

بالنسبة للخصائص، فجميع ما ذكرته حول الطرق سابقا يطبق عليها تماما، مع العلم انك لن تتمكن من تغير قابلية القراءة والكتابة عند إعادة قيادة الخاصية، فمثلا لو عرفت خاصية في الفئة القاعدية للقراءة فقط ReadOnly:

```
Class Person
    …
    …
    ' خاصية للقراءة فقط
    Overridable ReadOnly Property BirthDate() As Date
        Get
            …
        End Get
    End Property
End Class
```

فلن تتمكن من جعلها للكتابة فقط WriteOnly لحظة إعادة قيادتها:

PDF Factory Pro trial version www.fastio.com

```
Class Employee
    Inherits Person
    …
    …
    ' رسالة خطأ في تعريف الخاصية
    Overrides WriteOnly Property BirthDate() As Date
        Set(ByVal Value As Date)
            …
        End Set
    End Property
End Class
```

والعكس صحيح، فلن تتمكن من إعادة قيادتها وجعلها للقراءة فقط إن كانت معرفة للكتابة فقط في الفئة القاعدية، وحتى لو كانت قابلة للقراءة والكتابة، فلابد من أن تكون أيضا قابلة للقراءة والكتابة عند إعادة قيادتها. وبالنسبة للخصائص الافتراضية Default Properties، فلابد من أن تكون الخاصية افتراضية عند إعادة قيادتها في الفئة المشتقة.

انظر أيضا

لمزيد من التفاصيل حول الخصائص الافتراضية Default Properties، راجع الفصل الثالث **الفئات والكائنات**.

نقطة أخيرة حول إعادة القيادة وهي ان استدعاء الإجراء المعاد قيادته يعتمد على نوع الكائن المنشأ وليس نوع المتغير المعلن (المؤشر):

```
Dim Abbas As Person = New Employee()
Dim Abbas2 As Employee = New Employee()

Abbas.ShowName()        ' السريع، عباس
Abbas2.ShowName()       ' السريع، عباس
```

فكما تلاحظ، رغم ان المؤشر Abbas السابق من النوع Person الا انه استدعى الطريقة المعاد قيادتها في الفئة المشتقة Employee.

حالة إعادة التعريف Overloading:

دائماً ضع في اعتبارك، أن الفئة المشتقة والفئة القاعدية كلاهما –تقريبا- فئة واحدة (من نظرة الفئة المشتقة)، فيمكنك مثلا إعادة كتابة طريقة بإعادة تعريفها Overloads عوضا عن إعادة قيادتها Overrides:

```
Class BaseClass
    Overloads Sub TestMethod()
        ArabicConsole.WriteLine("الفئة القاعدية")
    End Sub
End Class

Class DerivedClass
    Inherits BaseClass

    Overloads Sub TestMethod(ByVal x As Integer)
        ArabicConsole.WriteLine("الفئة المشتقة")
    End Sub
End Class
```

وكما ذكرت في الفصل السابق، يمكنك استخدام الكلمة المحجوزة Overloads لإعادة تعريف طريقة ما شريطة اختلاف نوع وسيطاتها مع الطرق الأخرى، ولكن عند الحديث عن الفئات المشتقة فالوضع يختلف قليلا، قد تفاجئك هذه الشيفرة:

```
Class BaseClass
    Overloads Sub TestMethod()
        ArabicConsole.WriteLine("الفئة القاعدية")
    End Sub
End Class

Class DerivedClass
    Inherits BaseClass

    ' لاحظ عدم تغيير وسيطات الطريقة
    ' المعاد تعريفها
    Overloads Sub TestMethod()
        ArabicConsole.WriteLine("الفئة المشتقة")
    End Sub
End Class
```

كما ترى، سمح لنا مترجم Visual Basic .NET بإعادة تعريف الطريقة TestMethod() دون تغيير وسيطاتها! وحتى تتأكد من ان هذا ليس خطأ، يمكنك تعريف الكائنات وتجرب ذلك بنفسك:

PDF

```
Dim obj As New BaseClass()
Dim obj2 As New DerivedClass()

obj.TestMethod()     ' الفئة القاعدية
obj2.TestMethod()    ' الفئة المشتقة
```

قد تتساءل عن الفرق بين إعادة القيادة Overriding وإعادة التعريف Overloading في هذه الحالة، حيث أن الطريقتين تعملان بنفس الأسلوب في الحالتين، مع ذلك فالاختلاف بين استخدام Overrides واستخدام Overloads تقني من نظرة قابلية الوصول إلى الكائنات. حتى أوضح لك الفرق، أضف هذه الطرق إلى الفئات السابقة:

```
Class BaseClass
    …
    …
    Overridable Sub TestMethod2()
        ArabicConsole.WriteLine("الفئة القاعدية")
    End Sub
End Class

Class DerivedClass
    Inherits BaseClass
    …
    …
    Overrides Sub TestMethod2()
        ArabicConsole.WriteLine("الفئة المشتقة")
    End Sub
End Class
```

عند استدعاء الطرق من الكائنات، فان الطرق المعاد قيادتها Overrides سيتم استدعائها اعتمادا على الكائن المنشأ وليس نوع المؤشر، اما استدعاء الطرق المعاد تعريفها Overloads فسيعتمد على نوع مؤشر الكائن وليس نوع الكائن:

```
Dim obj As BaseClass = New DerivedClass()
Dim obj2 As DerivedClass = New DerivedClass()

obj.TestMethod()     ' الفئة القاعدية
obj2.TestMethod()    ' الفئة المشتقة

obj.TestMethod2()    ' الفئة المشتقة
obj2.TestMethod2()   ' الفئة المشتقة
```

PDF

ملاحظة

استخدامك للكلمة المحجوزة Overloads عوضا عن Overrides كما في المثال السابق، يؤدي إلى ما يسمى **بالتظليل** Shadowing كما سترى بعد فقرتين.

استخدام MyBase

إن كان استخدام المؤشر Me يؤدي إلى الوصول إلى عضو في الكائن الحالي، فان المؤشر MyBase يؤدي إلى الوصول إلى أعضاء الفئة القاعدية. مبدئيا، عمل المؤشر Me يشابه عمل المؤشر MyBase:

```
Class BaseClass
    Sub MethodInBase()
        …
    End Sub
End Class

Class DerivedClass
    Inherits BaseClass

    Sub MethodInDerived()

        ' استدعاء الطريقة في الفئة القاعدية
        MyBase.MethodInBase()

        ' استدعاء الطريقة في الفئة القاعدية أيضا
        Me.MethodInBase()

    End Sub
End Class
```

استنادا إلى المثال السابق، صحيح ان Me تمكننا من استدعاء أعضاء الفئة القاعدية ولكن في حالة عدم وجود طريقة معاد قيادتها Overrided، اما هنا:

```
Class BaseClass
    …
    …
    Overridable Sub MyMethod()
        …
    End Sub
End Class
```

```
Class DerivedClass
    Inherits BaseClass
    …
    …
    Overrides Sub MyMethod()
        …
    End Sub

    Sub TestMethod()
        ' استدعاء الطريقة في الفئة القاعدية
        MyBase.MyMethod()

        ' استدعاء الطريقة في الفئة المشتقة
        Me.MyMethod()
    End Sub
End Class
```

فيتضح لنا ان Me استدعت الطريقة –المعاد قيادتها- في الفئة الحالية، بينما MyBase موجهة للفئة القاعدية بشكل حصري.

فرق آخر بين MyBase و Me يتعلق بقدرة الوصول إلى الأعضاء الخاصة Private، حيث –كما أخبرتك في الفصل السابق- تمكنك Me من الوصول لها، بينما MyBase لا تسمح لك بذلك:

```
Class BaseClass
    Private fieldBase As Integer
End Class

Class DerivedClass
    Inherits BaseClass

    Private fieldDerived As Integer

    Sub Test()
        ' ممكن جدا
        Me.fieldDerived = 99

        ' رسالة خطأ
        MyBase.fieldBase = 99
    End Sub
End Class
```

بصفة عامة، الأعضاء الخاصة Private لا يمكن اشتقاقها من الفئة القاعدية.

الذي كنت أود أن أوصله لك من هذه الفقرة، هو ضرورة استخدام MyBase دائما إن قمت بإعادة تعريف المهدمات ()Finalize أو ()Dispose في الفئة المشتقة حتى تستدعي نفس المهدم التابع للفئة القاعدية:

```
Class DerivedClass
    Inherits BaseClass
    …
    …
    Protected Overrides Sub Finalize()
        ' استدعاء المهدم Finalize في الفئة القاعدية
        MyBase.Finalize()
        …
    End Sub
End Class
```

انظر أيضا

تحدثت عن المهدمات ()Finalize و ()Dispose في الفصل السابق **الفئات والكائنات**، وبالنسبة لمحدد الوصول المحمي Protected فسأتطرق له لاحقا في هذا الفصل.

إن لم تقم باستدعاء المهدم ()MyBase.Finalize فلن يتم تنفيذ مهدم الفئة القاعدية أبدا، لذلك احرص على استدعائه دائما ان قمت بإعادة قيادة المهدم Finalize فقط، أما إن لم تعيد قيادته فسيتم تنفيذ مهدم الفئة القاعدية تلقائيا –دون الحاجة لاستدعائه.

استخدام MyClass

من المعروف أن استخدام Me يمكنك من الوصول إلى أعضاء الفئة الحالية، ولكن عند إعادة قيادة العضو (أثناء عملية الوراثة بكل تأكيد)، فان العضو المعاد قيادته هو الذي سيتم استدعائه وليس العضو التابع للفئة الحالية:

```
Class BaseClass

    Sub Test()
        ' رغم استخدام Me الا انه سيتم استدعاء
        ' طريقة الفئة المشتقة
        Me.OverridedMethod()
    End Sub
```

```
    Overridable Sub OverridedMethod()
        ArabicConsole.WriteLine("من الفئة القاعدية")
    End Sub
End Class

Class DerivedClass
    Inherits BaseClass

    Overrides Sub OverridedMethod()
        ArabicConsole.WriteLine("من الفئة المشتقة")
    End Sub
End Class
```

عند إنشاء نسخة كائن من الفئة المشتقة، ستلاحظ ان الطريقة المعاد قيادتها هي التي تم استدعائها من داخل الفئة القاعدية باستخدام Me:

```
Dim obj As New DerivedClass()

obj.Test() ' من الفئة المشتقة
```

من هنا يأتي دور الكلمة المحجوزة MyClass، حيث أنها تجبر مترجم اللغة على استدعاء العضو التابع للفئة الحالية ان تمت إعادة قيادته، اما استخدامها فهو مثل استخدام Me:

```
Class BaseClass
    Sub Test()
        ' هنا سيتم استدعاء طريقة الفئة الحالية
        MyClass.OverridedMethod()
    End Sub
    …
    …
End Class
…
…
```

التظليل Shadowing

قد تلجأ إلى اعتماد **التظليل Shadowing** في اغلب الأحوال، وبالذات إن كانت قدرة الوصول إلى الشيفرة المصدرية للفئة القاعدية غير ممكنة (كوجود الفئة القاعدية في مكتبة DLL مثلا)، فقد يكون مؤلف الفئة قد عرف طريقة دون استخدام الكلمة المحجوزة Overridable:

```
Class BaseClass
    Sub Method()
        …
    End Sub
End Class
```

PDF Factory Pro trial version www.fastio.com

الطريقة السابقة لا يمكنك إعادة قيادتها (باستخدام Overrides) في الفئة المشتقة، والسبب ان مؤلف الفئة لم يسمح بذلك عن طريق الكلمة المحجوزة Overridable، مع ذلك يمكنك Visual Basic .NET من استخدام نفس اسم الطريقة في الفئة المشتقة (سيظهر المترجم تنبيه Warnning لحظة الترجمة، ولكن الشيفرة سيتم تنفيذها دون مشاكل):

```
Class DerivedClass
    Inherits BaseClass

    Sub MyMethod()
        …
    End Sub
End Class
```

لغويا نقول: ان الطريقة MyMethod() في الفئة المشتقة **ظللت Shadows** الطريقة MyMethod() في الفئة القاعدية، ومن نظرة كائنية، فسيتم تنفيذ الطريقة الموجودة في الفئة المشتقة:

```
Dim obj As New DerivedClass()
obj.MyMethod() ' استدعاء الطريقة في الفئة المشتقة
```

استخدم الكلمة المحجوزة Shadows في كل مرة تريد تظليل طريقة (لن تظهر الإنذارات Warnings في هذه الحالة):

```
Class DerivedClass
    Inherits BaseClass

    Shadows Sub MyMethod()
        …
    End Sub
End Class
```

ملاحظة

رغم ان ظهور الإنذارات Warnings أثناء ترجمة البرنامج لا تؤدي إلى وقف التنفيذ، إلا أني أنصحك بعنف بتعديل الشيفرات التي تسبب هذه الإنذارات دائما قبل توزيع البرنامج.

PDF Converter

حالة إعادة التعريف Overloading مرة أخرى:

استخدام الكلمة المحجوزة Overloads يؤدي إلى تظليل الطريقة أيضا –كما تفعل الكلمة المحجوزة Shadows:

```
Class DerivedClass
    Inherits BaseClass

    ' Overloads تظليل الطريقة باستخدام
    Overloads Sub MyMethod()
        …
    End Sub
End Class
```

مع ذلك توجد فروق بين استخدام Shadows و Overloads عند التظليل، الفرق الأول هو أن Shadows يمكنك استخدامها مع كافة أنواع أعضاء الفئة، بينما Overloads موجهه للطرق والخصائص فقط. الفرق الثاني –وهو الأهم- أن Shadows تقوم بتظليل كافة الطرق والخصائص المعاد تعريفها في الفئة القاعدية، بينما Overloads تظلل الطريقة أو الخاصية التي تحمل نفس وسيطاتها فقط، وحتى تستوعب ما اقصده، افترض هذه الفئة:

```
Class A
    Sub MyMethod()
        ArabicConsole.WriteLine("A.MyMethod")
    End Sub

    Sub MyMethod(ByVal x As Integer)
        ArabicConsole.WriteLine("A.MyMethod (x)")
    End Sub

    Sub MyMethod2()
        ArabicConsole.WriteLine("A.MyMethod2")
    End Sub

    Sub MyMethod2(ByVal x As Integer)
        ArabicConsole.WriteLine("A.MyMethod2 (x)")
    End Sub
End Class
```

والآن سأقوم باشتقاق الفئة وتظليل طريقتين منها –الأولى باستخدام Overloads والثانية بـ Shadows:

PDF

```
Class B
    Inherits A

    Overloads Sub MyMethod()
        ArabicConsole.WriteLine("B.MyMethod")
    End Sub

    Shadows Sub MyMethod2()
        ArabicConsole.WriteLine("B.MyMethod2")
    End Sub
End Class
```

عند إنشاء الكائن واستدعاء الطرق، لاحظ ماذا سيجري:

```
Dim obj As New B()

obj.MyMethod() ' B.MyMethod
obj.MyMethod(10)       ' A.MyMethod (x)
obj.MyMethod2()        ' B.MyMethod2
```

اعلم انك ستتساءل عن عدم استدعائي للطريقة obj.MyMethod2 (x) والسبب اني لو حاولت استدعائها ستظهر رسالة خطأ، لان الكلمة المحجوزة Shadows تؤدي إلى تظليل جميع الطرق المعاد تعريفها في الفئة القاعدية باختلاف وسيطاتها، وبعبارة أخرى لن تكون هناك إلا طريقة MyMethod() واحدة فقط موجودة في الفئة المشتقة.

التصنيف الفرعي Subclassing:

كما رأيت سابقا، التظليل يمكنك من إعادة قيادة الطريقة في الفئة المشتقة حتى لو لم يسمح مؤلف الفئة القاعدية بذلك، وفي العادة لن تقوم بعمل التظليل إلا عند الحاجة (فمؤلف الفئة القاعدية لن يمنعك من إعادة قيادة الطريقة حتى لا تؤثر على سلوك الفئة القاعدية بشكل سلبي)، يمكنك مثلا استدعاء الطريقة في الفئة القاعدية دون إجراء أي تعديل عليها في الفئة المشتقة:

```
Class B
    Inherits A

    …
    …
    Shadows Sub MyMethod2()
        MyBase.MyMethod2 ()
    End Sub
End Class
```

PDF

أما إن قمت بكتابة حرف واحد إضافي من الشيفرة المصدرية في داخل الطريقة السابقة، فاعلم انك تطبق ما يسمى **التصنيف الفرعي Subclassing** لطريقة الفئة القاعدية:

```
Class B
    Inherits A

    …
    …
    Shadows Sub MyMethod2()
        If … Then
            MyBase.MyMethod2 ()
        Else
            …
        End If
    End Sub
End Class
```

الأعضاء المشتركة Shared Members

كما ذكرت سابقا، الأعضاء المشتركة لا يمكن إعادة قيادتها، ولا يمكن أيضا استخدام الكلمات المحجوزة Overrides أو Overridable عليها، ولكنك تستطيع تظليلها باستخدام Shadows:

```
Class BaseClass
    Shared Sub MyMethod()
        …
    End Sub
End Class

Class DerivedClass
    Inherits BaseClass

    ' يمكنك تظليل الطريقة المشتركة
    Shared Shadows Sub MyMethod()
        …
    End Sub
End Class
```

نقطة إضافية أخرى، لا يمكنك استخدام MyBase للوصول إلى أعضاء الفئة القاعدية من داخل الأعضاء المشتركة لحظة الاشتقاق الوراثي، حيث سيعترض مترجم اللغة ويظهر لك رسالة خطأ، والحل الوحيد هو استخدام اسم الفئة القاعدية:

PDF

```
Class DerivedClass
    Inherits BaseClass

    Shared Shadows Sub MyMethod()

        MyBase.MyMethod ()       ' رسالة خطأ

        BaseClass.MyMethods () ' استخدم اسم الفئة القاعدية
    End Sub
End Class
```

كلمات محجوزة إضافية

في هذا القسم سأعرض لك ثلاث كلمات محجوزة لم أتطرق إليها سابقا، اثنتان تتعلق بقابلية الوراثة بين الفئات والثالثة بإعادة القيادة.

الكلمة المحجوزة NotInheritable

لأسبابك الشخصية، قد تمنع احد المتطفلين من اشتقاق فئاتك التي سهرت الليالي في بنائها، يمكنك بكل سهولة عمل ذلك باستخدام الكلمة المحجوزة NotInheritable عند تعريف الفئة:

```
' لا يمكن اشتقاق هذه الفئة
NotInheritable Class MyClass
    …
    …
End Class
```

في اغلب الأحوال، الفئات التي لا تحتوي الا على أعضاء مشتركة يفضل ان تجعلها غير قابلة للوراثة، لديك مثلا الفئات ArabicConsole و Console لن تتمكن من وراثتها:

```
Class DerivedClass
    Inherits ArabicConsole ' في المشمش !
    …
    …
End Class
```

الكلمة المحجوزة MustInherit

كما هو واضح من اسمها، الكلمة المحجوزة MustInherit تجبر المبرمج على اشتقاقها أولا ومن ثم إنشاء كائنات من الفئات المشتقة منها:

```
MustInherit Class BaseClass
    …
    …
End Class
```

لن تستطيع إنشاء نسخة كائن من هذه الفئة مباشرة، بل عليك اشتقاقها أولا في فئة (كـ DerivedClass مثلا) ومن ثم إنشاء كائن من تلك الفئة المشتقة:

```
Dim X As New BaseClass() ' رسالة خطأ
Dim X As New DerivedClass() ' ممكن
```

يعرف هذا النوع من الفئات **بالفئات المجردة Abstract Classes**، قد تحتاج إلى الفئات المجردة في حياتك البرمجية ان ردت تكوين فئات لابد من اشتقاقها ولا يمكن استخدامها مباشرة. مثلا، في موقعنا **شبكة المطورون العرب** (dev4arabs.com) عرفنا فئة مجردة تعبر عن وحدة أو سجل:

```
MustInherit Class Item
    Public Title As String
    Public PostedDate As Date
    Public Author As String
    …
    …

    Sub Delete()
        …
    End Sub

    Overridable Sub Update()
        …
    End Sub

    Overridable Sub AddNew()
        …
    End Sub
End Class
```

هذه الفئة لا يمكن استخدامها مباشرة حيث أنها بحالتها الراهنة لا ترمز إلى نوع معين من البيانات، فالذي قمنا به هو تعريف فئات أخرى تشتق منها، بعضها يقوم بإعادة قيادة طرقها عند الحاجة:

PDF Converter

```
' قسم الشيفرات المصدرية
Class SourceCode
    Inherits Item
    …
    …
End Class

' قسم التلميحات
Class Tip
    Inherits Item
    …
    …
End Class

' قسم المقالات
Class Article
    Inherits Item

    Public Introduction As String
    …
    Public Overrides Sub Update()
        …
    End Sub

    Public Overrides Sub AddNew()
        …
    End Sub
    …
End Class
```

الكلمة المحجوزة MustOverride

بعض الطرق والخصائص في الفئات لابد من أن يتم إعادة قيادتها حتى تتمكن من محاكاة الفئة المشتقة منها، فمثلا لو أردنا تعريف طريقة في الفئة Item السابقة لعرض البيانات:

```
MustInherit Class Item
    …
    …

    Sub ShowDate()

    End Sub
End Class
```

فلا يمكن كتابة شيفرات لعرض البيانات لاختلاف نوع البيانات واختلاف طريقة عرضها في صفحات الموقع، لذلك أضفنا الكلمة المحجوزة MustOverride حتى تجبر مستخدم الفئة من إعادة قيادة الطريقة لحظة اشتقاقها:

PDF Factory Pro trial version www.fineprint.com

```
MustInherit Class Item
    …
    …
    MustOverride Sub ShowDate()
End Class
```

عند استخدامك للكلمة المحجوزة MustOverride فلا تكتب العبارة Sub End أو End Function أسفل الطريقة، ولا التركيب Set … Get أو End Property عند تعريف خاصية.

ملاحظة

لن تتمكن من استخدام الكلمة المحجوزة MustOverride في الفئة إلا عند استخدامك للكلمة المحجوزة MustInherit لحظة تعريف الفئة، والسبب يبدو منطقيا جدا ولست بحاجة إلى توضيحه.

محددات الوصول

ذكرت في الفصول السابقة ثلاث كلمات محجوزة تستخدم لتحديد قابلية الرؤية Visibility هي: Public، Private، و Friend. بالإضافة إلى هذه الكلمات، توجد كلمتين أخريين هما Protected و Protected Friend فضلت تأجيلهما إلى هذا الفصل لانهما تتعلقان بالوراثة. في الفقرات التالية سنرى مدى تأثير هذه الكلمات الجديدة، بالإضافة إلى مراجعة الكلمات السابقة.

قابلية الرؤية للفئات

خمس كلمات محجوزة تستخدم لتحديد قابلية الرؤية للفئة هي:

Private:

الفئات المصرحة بـ Private يمكن الوصول لها من داخل الوعاء الذي عرفت فيه فقط، الوعاء قد يكون –كما ذكرت سابقا- اما وحدة برمجية Module، تركيب من نوع Structure، أو فئة Class، ولن تستطيع الوصول لها في مكان آخر:

```
' غير ممكن
Private Class TestClass
    …
End Class
```

PDF

```
Module Module1
    ' ممكن
    Private Class TestClass
        …
    End Class
    …
End Module
```

بالنسبة للفئات المتداخلة Nested، فلن تتمكن من الوصول لها ان استخدمت Private عند تعريف كائن خارج نطاق وعائها:

```
Class Outer
    Private Class Inner
        …
    End Class
    …
End Class

Module Module1
    Sub Main()
        Dim X As New Outer.Inner() ' لا يمكن
        …
    End Sub
End Module
```

كما ترى، لا يمكن الوصول إلى الفئة Inner إلا من داخل الفئة Outer، أو أي فئة أخرى معرفة داخل Outer:

```
Class Outer
    Private Class Inner
        …
    End Class

    Class Inner2
        Public X As Inner ' ممكن
        …
    End Class
    …
End Class
```

Friend:

استخدامك للكلمة المحجوزة Friend (أو حتى تجاهلها فهي افتراضية) عند تعريف الفئة، يمكنك من استخدام هذه الفئة والوصول لها من مختلف أماكن المشروع الحالي، يمكنك تعريف فئة من النوع Friend من أي مكان ترغبه –حتى وان كانت داخل وعاء لفئة أخرى:

PDF Converter

```
Class Outer ' Friend هنا تعني
    Friend Class Inner
        …
    End Class
End Class

Module Module1
    Sub Main()
        Dim X As New Outer.Inner() ' ممكن جدا
        …
    End Sub
End Module
```

مع ذلك، إن عرفت فئة باستخدام Friend داخل وعاء بمستوى Private فلن تتمكن من الوصول إليها إلا من داخل ذلك الوعاء فقط، فالفئة C التالية:

```
Class A
    Private Class B
        Friend Class C
            …
        End Class
        …
    End Class
    …
End Class
```

لن تتمكن من الوصول لها خارج الفئة B رغم انها على مستوى Friend.

المزيد أيضا، ان عرفت فئة داخل فئة أخرى فلن تتمكن من تعريف كائن منها بكتابة اسم الفئة مباشرة حتى لو كانت Friend، اذ يشترط كتابة اسم الفئة الحاضنة لها أولا، فمثلا الفئة B التالية:

```
Class A
    Class B
        …
        …
    End Class
    …
    …
End Class
```

عليك ذكر اسم الفئة الحاضنة لها لحظة تعريف كائن جديد:

```
Dim obj As New A.B() ' هكذا
Dim obj As New B() ' وليس كذا
```

PDF

Public:

كل ما ذكرته حول Friend ينطبق على Public فهي مثل Friend تماما، الا ان لها ميزة إضافية حيث تمكنك من الوصول إلى الفئة من خارج المشروع الحالي أيضا. قد تستخدم Public في اغلب الأحوال عند بناء مكتبة فئات Class Library تترجم إلى ملفات DLL مثلا.

Protected:

اما Protected فهي مثل Private بشكل عام (فكل ما ذكرته عن Private سابقا يطبق هنا)، إلا انها تكون قابلة للوصول من الفئات المشتقة فقط، فمثلا الفئة Inner التالية:

```
Class BaseClass
    Protected Class Inner
        …
    End Class
    …
End Class
```

لن تستطيع الوصول لها واستخدامها إلا عن طريق فئة مشتقة من BaseClass فقط:

```
Class DerivedClass
    Inherits BaseClass

    Private X As Inner ' ممكن
    …
End Class

Module Module1
    Sub Main()
        Dim X As Inner ' لا تتعب نفسك
        …
    End Sub
End Module
```

في المثال السابق، يتوجب علي اشتقاق BaseClass حتى أتمكن من الوصول إلى الفئة Inner، لان Inner كما أخبرتك Private ولن أستطيع اشتقاقها مباشرة:

```
Class DerivedClass
    Inherits Inner ' بودي ولكن للاسف
    …
    …
End Class
```

PDF

Protected Friend:

يسمح لك محدد الوصول Protected Friend باستخدام الفئة في أي مكان من المشروع الحالي، أي انه مثل Friend تماما، فكل ما ذكرته عن Friend ينطبق هنا على Protected Friend دون أي مشاكل، والفرق الوحيد بين Friend و Protected Friend هو ان Protected Friend تسمح لك باستخدام الفئة ان تم اشتقاقها من مشروع آخر، فالفئة Inner التالية:

```
Class BaseClass
    Friend Class Inner
        …
    End Class

    Protected Friend Class Inner2
        …
    End Class
    …
End Class
```

يمكنك استخدامها من أي مكان في المشروع الحالي فقط، اما الفئة Inner2 فتستطيع استخدامها أيضا إن قمت باشتقاقها وراثيا من مشاريع أخرى.

قابلية الرؤية لأعضاء الفئات

تحدثت في الفقرة السابقة عن تأثير الكلمات المحجوزة عند تعريفها مع الفئة، دعنا نلقي الضوء هنا على أعضاء الفئة وبيان مدى تأثيرها.

Private:

كما هو معلوم، الأعضاء المصرحة بـ Private لا يمكن الوصول لها إلا من داخل الفئة فقط، وبالنسبة للحقول Fields فهي ستكون Private بشكل افتراضي:

```
Class TestClass
    Dim X As Integer ' هنا Private تعني Dim

    Private Sub MyMethod()
        …
    End Sub
    …
End Class
```

وبالنسبة للأحداث Events على المستوى Private، فلن تتمكن من قنصها الا من داخل الفئة فقط:

```
Class TestClass
    Private Event MyEvent()

    Public Sub MySub()
        AddHandler Me.MyEvent, AddressOf Me.EventHandler
        …
    End Sub

    Public Sub EventHandler()
        …
    End Sub
End Class
```

اما الحديث عن الفئات المتداخلة، فيمكنك الوصول إلى أعضاء الفئة الحاضنة من داخل الفئة المحضونة حتى لو كانت على مستوى Private:

```
Class Outer
    Private X As Integer

    Class Inner
        Dim obj As Outer ' عليك إسناد قيمة لهذا المؤشر

        Sub MyMethod()
            …
            obj.X = 10 ' ممكن
            …
        End Sub
    End Class
    …
End Class
```

Friend:

مرة أخرى، جميع الأعضاء المعرفة باستخدام Friend يمكنك الوصول لها من خارج الفئة (عن طريق مؤشر الكائن الذي ستعرفه)، جميع أعضاء الفئة تكون Public وليس Friend بشكل افتراضي –باستثناء الحقول التي تكون Private إن لم تحدد محدد الوصول لها:

```
Class TestClass
    Friend X As Inner

    Friend Sub MyMethod () ' Friend
        …
    End Sub

    Sub MyMethod2 () ' Public
        …
    End Sub
    …
```

PDF

```
    …
End Class
```

عندما تعرف عضو على مستوى Friend فمن غير المنطقي استخدام فئات أو نوع خاص Private (حيث انه خاص بالوعاء الذي عرف فيه فقط ولن تستطيع ايصاله للعالم الخارجي):

```
Class Outer
    Friend X As Inner ' ولا في الاحلام

    Private Class Inner
        …
        …
    End Class
End Class
```

Public:

استخدام Public مع أعضاء الفئات هو مثل استخدام Friend (وجميع مع ذكرته في الفقرة السابقة يطبق أيضا هنا على Public)، ولكن الفرق عند هو قدرة الوصول إلى الأعضاء على مستوى Public من خارج المشروع الحالي، اما Friend فهي محصورة داخل المشروع فقط.

Protected:

أيضا، الأعضاء على مستوى Protected هي أعضاء من النوع Private (وكل ما ذكرته حول Private ينطبق أيضا على Protected أيضا)، ولكنك تستطيع الوصول إليها لحظة اشتقاق الفئة:

```
Class BaseClass
    Protected X As Integer
    …
End Class
```

يمكنك الوصول إلى الحقل X السابق من أي فئة مشتقة من BaseClass:

```
Class DerivedClass
    Inherits BaseClass

    Sub MyMethod()
        Me.X = 10
        …
    End Sub
End Class
```

PDF

قد تعتقد انك تستطيع الوصول إلى المتغير X السابق من خلال مؤشر الكائن الذي تعرفه من الفئة المشتقة، ولكن هذا غير ممكن:

```
Dim obj As New DerivedClass()
obj.x = 10 ' غير ممكن
```

Protected Friend:

الأعضاء على مستوى Protected Friend هي كالأعضاء على مستوى Friend، ولكنها تزيد عنها في إمكانية الوصول لها من فئات مشتقة عرفت خارج المشروع الحالي.

تأثير محددات الوصول على المشيدات

قد يثير اهتمامك معرفة مدى تأثير محددات الوصول السابقة على المشيدات، وان كان لا يثير اهتمامك دعني اريك هذه الفئة:

```
Public Class TestClass
    Private Sub New()
        …
    End Sub
End Class
```

رغم ان الفئة السابقة Public، الا انك لن تستطيع إنشاء نسخة منها، فالسطر التالي سيظهر رسالة خطأ:

```
Dim obj As New TestClass()
```

علي توضيح فرق هام هنا، وهو ان الفئة TestClass يمكن استخدامها في البرنامج دون مشاكل (كأن تجعلها وسيطة لاحد الاجراءات)، ولكنك لن تستطيع إنشاء نسخة كائن منها بنفسك (باستخدام New مثلا). يعرف هذا النوع من الفئات **بالفئات غير قابلة للإنشاء Not Creatable Class**.

المشيدات من النوع Private:

كما ذكرت في الصفحة السابقة، ان كان مشيد الفئة من النوع Private فلن تتمكن من انشاء نسخة كائن جديدة من الفئة مهما كان نوعها.

PDF

المشيدات من النوع Friend:

اما ان كان المشيد من النوع Friend، فيمكن انشاء نسخة كائن جديدة من الفئة فـــي المـــشروع الحالي فقط.

المشيدات من النوع Public:

وهو النوع الافتراضي للمشيدات، بحيث يمكنك انشاء نسخة كائن جديدة من الفئة حتى في المشاريع الأخرى.

المشيدات من النوع Protected:

ان كان المشيد من النوع Protected فيمكنك انشاء نسخة كائن من الفئة من داخل الفئة المـــشتقة فقط، فالفئة الحالية:

```
Public Class BaseClass
    Protected Sub New()
        …
    End Sub
End Class
```

لن تستطيع انشاء نسخة كائن جديدة منها الا من داخل فئة مشتقة فقط:

```
Class DerivedClass
    Inherits BaseClass

    Public x As New BaseClass()
    …
End Class
```

المشيدات من النوع Protected Friend:

مثل تأثير المشيدات من النوع Friend، ولكن ستكون عملية انشاء نسخة كائن جديدة ممكنة أيضا للفئات المشتقة في مشاريع أخرى.

PDF

ملاحظة

في جميع الفقرات السابقة التابعة لهذا القسم من الفصل، ذكرت ان الفرق بين Public و Friend يظهر عند الخروج عن إطار المشروع الحالي، رغم ان العبارة صحيحة تقريبا إلا أنها غير دقيقة، فالفرق يظهر عند الخروج عن إطار المجمع Assembly الحالي، حيث أن المجمع قد يحتوي على أكثر من مشروع كما سترى لاحقا في الفصل الحادي عشر **المجمعات** Assemblies.

قدمت في هذا الفصل كل ما أود ذكره حول الوراثة والاشتقاق الوراثي بين الفئات، لتكون مستوعباً تماما وجاهزاً لاستخدام مكتبة فئات إطار عمل NET. Framework وعليك معرفة ان مبدأ الوراثة في لغات OOP –بصفة عامة- من اعقد المبادئ التي تواجه المبرمجين، وهذا هو كل ما استطعت فعله لتوضيح الوراثة لك. الفصل التالي **الواجهات، التفويض، والمواصفات** سيكون آخر فصل ننهي الجزء الأول **الأساسيات** من هذا الكتاب.

الواجهات، التفويض، والمواصفات

ونحن على مشارف الانتهاء من تعلم أساسيات لغة البرمجة Visual Basic .NET، تبقى لـــدينا مجموعة من المواضيع المتفرقة والتي يتحتم على ذكرها من منطلق الشمولية للإلمام بأساسيات لغة البرمجة Visual Basic .NET.

سأختم معك الجزء الأول من هذا الكتاب بالحديث عــن مواضــيع متعــددة كالواجهــات Interfaces وطريقة تطبيق مبدأ تعدد الواجهات Polymorphism، كما سأتطرق أيــضا إلـــى الإجراءات المفوضة Delegates والتي تعطيك مرونة كبيرة في استدعاء الإجــراءات، واخــتم الفصل بالتلميح إلى موضوع المواصفات Attributes لننهي بذلك مرحلة تعلــم أساســيات لغــة البرمجة Visual Basic .NET.

الواجهات

في البداية دعني أوضح لك ما المقصود بكلمة واجهة في لغات OOP. الواجهة Interface هـــي مجموعة من الطرق والخصائص والأحداث التي تصف فئة معينة، فالفئة التالية:

```
Class Person
    Event Die()

    Public Property Name() As String
        …
        …
    End Property

    Sub Move()
        …
        …
    End Sub
End Class
```

PDF

تحتوي على واجهة متمثلة في حدث باسم Die، خاصية باسم Name، وطريقة باسم ()Move. لاحظ ان كلمة الواجهة تطلق على أسماء الأعضاء فقط وليس وظائفها (أي ليس سلوكها أو الشيفرات التي تحتويها).

من المبادئ الأساسية في لغات OOP هو مبدأ **تعدد الواجهات Polymorphism**، وهو احتواء الفئة الواحدة على أكثر من واجهة تصفها، ليصل الأمر إلى ان تتشارك مجموعة فئات في واجهات موحدة فنقول: أسماء متشابهة لكن إنجازات مختلفة Same names but different implementation.

العديد من الفوائد التي تجنيها من تطبيق واستخدام مبدأ تعدد الواجهات Polymorphism، لعل أبرزها اختصار جمل الشرط في برنامجك، فتخيل ان لدينا فئة تمثل مستند نصي TextFile، وفئة أخرى تمثل مستند منسق RTFFile، وفئة أخرى تمثل مستند صفحة ويب HTMLFile، وفئة رابعة تمثل مستند تنسيق البيانات XMLFile، وفئة أخيرة تمثل صورة PictureFile، ستجد ان برمجة وإنجاز كل فئة من هذه الفئات سيختلف تماما عن الفئة الأخرى، ولكن الواجهات التي تحتويها متشابهة ومتشاركة، فيمكن تعريف خاصية تمثل اسم الملف FileName أو حجم الملف FileSize، وطريقة تؤدي إلى عملية الحفظ ()Save أو الحذف ()Delete، ويمكن تطبيق جميع هذه الواجهات على الفئات المختلفة.

فمثلاً لو قمت بتطوير إجراء يقوم بحفظ الملف النصي فستجعل الوسيطة تستقبل كائن من النوع TextFile:

```
Sub SaveTextFile(ByVal FileObject As TextFile)
    …
    FileObject.Save
    …
End Sub
```

ونفس الشئ ستفعله مع الانواع الاخرى من الملفات:

```
Sub SaveRTFFile(ByVal FileObject As RTFFile)
    …
    FileObject.Save
    …
End Sub

Sub SaveHTMLFile(ByVal FileObject As HTMLFile)
    …
    FileObject.Save
    …
End Sub
```

PDF

ليس هذا فقط، بل أنه عند استدعاء إجراء الحفظ ستجد أنه يتوجب عليك كتابة الجمل الشرطية في كل مرة تود فيها فعل ذلك للتحقق من نوع الملف، وعلى ضوء المقارنة تقوم باستدعاء فئة الملف المعنية لتقوم بالحفظ، كما أن عليك أيضا الإعلان عن جميع الكائنات التي تمثل الفئات المختلفة لإرسالها إلى الإجراء المختص:

```
Dim TextFileObject As New TextFile
Dim RTFFileObject As New RTFFile
Dim HTMLFileObject As New HTMLFile

…
…

Select Case FileType
    Case 1
        TextFileObject.FileName = "xxxx"
        SaveTextFile ( TextFileObject )
        …
    Case 2
        RTFFileObject.FileName = "xxxx"
        RTFRTFFile ( RTFFileObject )
        …
    Case 3
        HTMLFileObject.FileName = "xxxx"
        HTMLTextFile ( HTMLFileObject )
        …
    …
    …
End Sub
```

والآن لنفترض انه ظهرت الحاجة لإضافة فئة جديدة لنوع معين من التسيقات BinaryFile مثلا، فإن ذلك يفرض عليك ان تضيف جميع الاستدعاءات والإجراءات والإعلانات والشروط التي ستتعامل مع هذه الفئة في كل أنحاء البرنامج، وتقوم بتكرار كل ما فعلته في الفئات السابقة.

ولكن مع تعدد الواجهات فإن الأمر مختلف تماماً، فكل ما سنفعله هنا هو القيام بتعريف واجهة نسميها مثلا IFile ونعرف إجراء يستقبل كل أنواع الكائنات شريطة احتوائه على الواجهة IFile:

```
Sub SaveFile(ByVal FileObject As IFile)
    …
    FileObject.Save ()
    …
End Sub
```

PDF

لا يقتصر الإجراء السابق على استقبال الأنواع التي ذكرناها سابقا (TextFile، RTFFile، HMTLFile، ... الخ) بل يمكنه ان يستقبل كل الأنواع الأخرى والتي قد تحتوي يوما من الأيام على الواجهة IFile.

بناء واجهة

بعد ان أوضحت لك الفكرة من الواجهات، لنبدأ ببناء واجهة تحمل الاسم IFile، تحتوي على جميع الأعضاء التي نود من الفئات الأخرى استخدامها. لتعريف واجهة في Visual Basic .NET، استخدم التركيب Interface ... End Interface:

```
Interface IFile
    ' خصائص
    Property FileName() As String
    Property FileSize() As Integer
    …

    ' طرق
    Sub Save()
    Sub Save(ByVal TargetFile As String)
    Sub Delete()
    …
End Interface
```

كما ترى في التركيب السابق، الواجهة لا تحتوي على أية شيفرات برمجية فهي مجرد واصفة لأسماء ووسيطات الأعضاء فقط، فلا تفكر في وضع عبارات كـ End Sub، End Function، أو End Property.

ملاحظة

جرى العرف سابقا عند مبرمجي OOP باستخدام الحرف I قبل اسم الواجهة، وهو نفس الأسلوب الذي تقترحه مستندات NET. مع استخدام PascalCase لكتابة اسم الواجهة.

بالنسبة لقابلية الرؤية Visibility للواجهات، فهي افتراضيا Friend ويمكنك استخدام محددات الوصول الأخرى كما تفعل مع الفئات تماما:

```
' Friend
Interface IFile
    …
End Interface

' Public
Public Interface IView
    …
End Interface

Module Module1
    ' Private
    Private Interface ITool
        …
    End Interface

    Sub Main()
        …
    End Sub
End Module
```

مع ذلك، لا يمكنك استخدام محددات الوصول مع أعضاء الواجهة، فجميع أعضاء الواجهات قابلية رؤيتها Public دائما وأبدا:

```
Interface ITest
    ' لا يمكن استخدام محددات الوصول
    ' مع أعضاء الواجهة
    Public Sub MyMethod ()
    Private Sub YourMethod ()
    …
    …
End Interface
```

المزيد أيضا، يمكن للواجهات ان تكون متداخلة Nested:

```
Interface IView
    Interface IWindow
        …
        …
    End Interface

    Interface IWEB
        …
        …
    End Interface
    …
    …
End Interface
```

PDF

اخيرا، الطرق، الخصائص، والأحداث هي التي يمكن كتابتها في الواجهات، أمـــا الحقـــول فغير ممكنة:

```
Interface IMyInterface
    X As Inetger ' بـودنـا ولـكن لـلاسف
    …
End Interface
```

تضمين الواجهة

بعد تعريفك للواجهة، يمكنك تضمينها في أي فئة باستخدام الكلمة المحجوزة Implement:

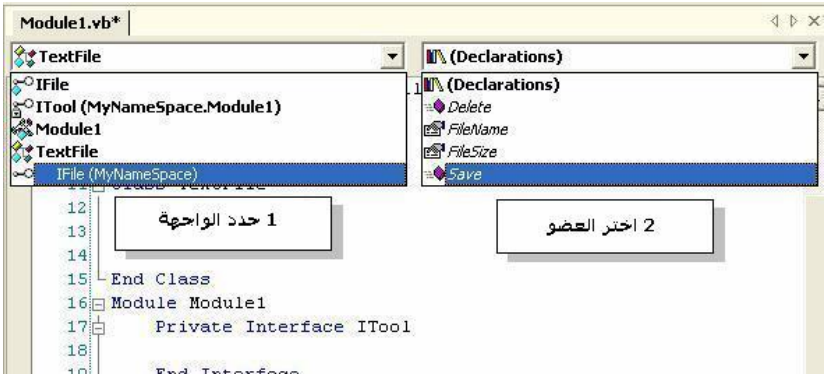
```
Class TextFile
    Implements IFile ' تـضمين الـواجهة
    …
    …
End Class
```

الشرط الأساسي لتضمين الواجهة في الفئة الحالية هو تعريف جميع أعضاء الواجهـــة وإلا ستظهر لك رسالة خطأ:

```
Class TextFile
    Implements IFile

    ' FileName تـضمين الخاصية
    Property FileName() As String Implements IFile.FileName
        Get
            …
        End Get
        Set(ByVal Value As String)
            …
        End Set
    End Property

    ' Delete() تـضمين الـطريـقـة
    Sub Delete() Implements IFile.Delete
        …
    End Sub

    ' عليك اضافة جميع الأعضاء الاخرى
    …
    …
End Class
```

PDF

وبدلا من تعريف أعضاء الواجهات يدويا، يمكنك الاستعانة بمحرر الشيفرات لعمل ذلك نيابة عنك، حدد الواجهة من القائمة العلوية اليسرى ومن ثم اختر العضو المراد تعريفه من القائمة العلوية اليمنى (**شكل 5-1**).



شكل 5-1: الاستعانة بمحرر الشيفرة لكتابة أعضاء الواجهة.

عند تعريف أعضاء الواجهة، فلا بد من توافق وسيطات الطرق، الأحداث، والخصائص مع نفس الوسيطات الموجودة في الواجهة، ولابد من توافق نوع القيم التي تعود بها الدوال Functions. المزيد أيضا، قابلية القراءة والكتابة (اقصد استخدام الكلمات ReadOnly و WriteOnly) للخصائص، لابد وان تكون مطابقة لتلك الموجودة في الواجهة. فالتضمينات الموجودة في الفئة TestClass بالصفحة التالية خاطئة 100%:

```
Interface IMyInterface
    ' خاصية للقرائة فقط
    ReadOnly Property BirthDate() As Date

    ' دالة تعود بقيمة حرفية
    Function MyMethod() As String
End Interface
```

```
Class TestClass
    Implements IMyInterface

    ' قابلية القرائة والكتابة غير متوافقة
    Public WriteOnly Property BirthDate() As Date Implements _
        IMyInterface.BirthDate

            Set(ByVal Value As Date)
                …
                …
            End Set
    End Property

    ' القيمة التي تعود بها الدالة غير متوافقة
    Public Function MyMethod() As Integer Implements _
        IMyInterface.MyMethod

           …
           …
    End Function
End Class
```

بالنسبة للواجهات المتداخلة، فيمكنك تضمينها كما تفعل مع الواجهات العادية، ولكن لا تنسى كتابة اسم الواجهة الحاضنة لها أولا:

```
Interface IView
    Interface IWindow
        …
    End Interface

    Interface IWEB
        …
    End Interface
    …
End Interface

Class TestClass
    Implements IView.IWindow ' لا تنسى كتابة اسم الواجهة الحاضنة اولا

    …
    …
End Class
```

اما إن استخدمت الواجهة الحاضنة:

```
Class TestClass2
    Implements IView
    …
End Class
```

PDF Factory Pro trial version www.fastio.com

فالمؤكد ان الفئة السابقة لن تحتوي إلا على الواجهة IView فقط، ولا تعتقد للحظة ان الواجهات المحضونة بها (IWindow و IWeb) قد تم تضمينها هي الأخرى.

أخيرا، يمكن للفئة الواحدة ان تحتوي على أكثر من واجهة:

```
Class ManyInterfaces
    Implements IFile
    Implements IView
    Implements IView.IWindow
    Implements IView.IWEB

    …
    …
End Class
```

عند تعارض اسماء الأعضاء في اكثر من واجهتين، فلابد من تغيير اسمها لحظة كتابتها في الفئة، فمثلا لو اكتشفت أن الطريقة ()MyMethod موجودة في الواجهتين الـــسابقتين IView و IFile، فيمكنك تمييزهما بوضع رقم:

```
Class ManyInterfaces
    Implements IFile
    Implements IView
    …
    …

    Sub MyMethod() Implements IFile.MyMethod
        …
    End Sub

    ' لابد من تغيير اسم الطريقة
    Sub MyMethod2() Implements IView.MyMethod
        …
    End Sub
    …
    …
End Class
```

من المثال السابق، يتضح لنا ان مترجم Visual Basic .NET لا يهتم باسم العضو بقدر ما يهتم بالاسم الذي يلي الكلمة المحجوزة Implements لحظة تعريفه، لنستنتج من ذلك انه يمكننا تغيير اسم الطرق السابقة:

```
Class ManyInterfaces
    Implements IFile
    Implements IView
    …
    …

    Sub MethodFromFile () Implements IFile.MyMethod
        …
    End Sub


    Sub MethodFromView () Implements IView.MyMethod
        …
    End Sub
    …
    …
End Class
```

الوصول إلى الواجهة

بافتراض ان فئات الملفات السابقة (TextFile، RTFFile، HTMLFile ...الخ) قد تم تضمين الواجهة IFile بها، فان باستطاعتنا تعريف متغير من النوع IFile وإسناد اي قيمة له من النـــوع TextFile، RTFFile، HTMLFile ...الخ:

```
Dim textFileObject As New TextFile()
Dim file As IFile

' ممكن عمل ذلك
file = textFileObject
file.FileName = "MyFile.TXT"
…
…
```

كما يمكن تعريف إجراء يستقبل وسيطة من النوع IFile:

```
Sub DoSave(ByVal fileObject As IFile)
    …
    fileObject.Save ()
    …
End Sub
```

لتتمكن من إرسال اي قيمة له من النوع TextFile، RTFFile، HTMLFile ....الخ:

PDF Factory Pro trial version www.fastio.com

```
Dim htmlFileObject As New HMTLFile()
…
…
DoSave (htmlFileObject)
…
…
```

وراثة الواجهات

عند الحديث عن وراثة الواجهات، يمكننا ان ننظر إليها من منظورين مختلفين، المنظور الأول عند اشتقاق فئة قاعدية تحتوي على واجهة:

```
Interface IMyInterface
    Sub MyMethod()
End Interface

Class BaseClass
    Implements IMyInterface

    Sub MyMethod() Implements IMyInterface.MyMethod
        …
    End Sub
End Class
```

عندما تشتق الفئة BaseClass السابقة، عليك معرفة ان الواجهة IMyInterface المضمونة بها سيتم اشتقاقها أيضا:

```
Class DerivedClass
    Inherits BaseClass

    …
    …
End Class
```

لتتعامل مع الكائن من الفئة DerivedClass كما تتعامل مع الكائن BaseClass:

```
Dim DerivedObject As New DerivedClass()
Dim X As IMyInterface

X = DerivedObject

' واجهة من الفئة القاعدية
X.MyMethod()
```

PDF

اما المنظور الثاني فهو يتعلق بوراثة الواجهات، وهو توارث الواجهات فيما بينها باستخدام الكلمة المحجوزة Inherits أيضا:

```
Interface IBaseInterface
    Sub MyMethodInBase()
End Interface

Interface IDerivedInterface
    Inherits IBaseInterface

    Sub MyMethodInDerived()
End Interface
```

عند تضمين الواجهة IDerivedInterface في اي فئة، فيتوجب عليك تعريف جميع الأعـضاء التابعة للواجهة القاعدية أيضا:

```
Class TestClass
    Implements IDerivedInterface

    ' الواجهة القاعدية
    Sub MyMethodInBase() Implements IDerivedInterface.MyMethodInBase
        …
    End Sub

    ' الواجهة المشتقة
    Sub MyMethodInDerived() Implements
IDerivedInterface.MyMethodInDerived
        …
    End Sub
End Class
```

واجهات من إطار عمل .NET Framework

في الفصل الثالث **الفئات والكائنات** تعرفنا على الواجهة IDisposable، وذكـرت حينهـا انـك تضمنها في الفئة لتعرف الإجراء Dispose() لمحاكاة المهدمات Destructors في الفئة:

```
Class TestClass
    Implements IDisposable


    Public Sub Dispose() Implements System.IDisposable.Dispose
        …
    End Sub
    …
    …
End Class
```

PDF

توجد مئات الواجهات الأخرى والمعرفة في مكتبة فئات إطار عمل .NET Framework. الا أني سأحاول في هذا القسم عرض بضعة واجهات هامة تستخدمها في معظم برامجك ومشاريعك بـ Visual Basic .NET.

الواجهة IComparable

عملية إجراء المقارنة بين عددين تعتمد على قيمة العدد، ويمكنك استخدام معاملات المقارنة (<، >، <= ...الخ)، ويمكن أيضا استخدام علامات المقارنة بين القيم الحرفية Strings، حيث سيعتمد المترجم في هذه الحالة على المقابل الرقمي للحرف في جداول المحارف ASCII أو UNICODE حسب الحالة:

```
Dim X As Integer = 10
Dim Y As String = "تركي"

If X > 5 Then …
If Y < "احمد" Then …
```

ولكن عند الحديث عن أنواع البيانات الأخرى والتي تصممها بفئاتك الخاصة، فلا يمكن تحديد ايهما اكبر وايهما اصغر، فلو عرفت كائنين من فئة Person فأنت تحدد بنفسك ما هي القيم التي تريد الاعتماد عليها لحظة المقارنة، لذلك سنقوم بتضمين الواجهة IComparable في الفئة لتعرف طريقتها الوحيدة CompareTo() والتي تعود بالقيمة -1 اذا كان الكائن الحالي اصغر من الكائن المرسل، 0 في حالة تساوي الكائنين، أو 1 ان كان الكائن الحالي اكبر من الكائن المرسل:

```
Class Person
    Implements IComparable

    Public Name As String

    Function CompareTo(ByVal obj As Object) As Integer Implements _
        System.IComparable.CompareTo

        Dim tempObj As Person = CType(obj, Person)

        ' الكائن الحالي اصغر من الكائن المرسل
        If Me.Name < tempObj.Name Then
            Return -1
        ' الكائن الحالي اكبر من الكائن المرسل
        ElseIf Me.Name > tempObj.Name Then
            Return 1
```

PDF

```
        ' الكائنان متساويان
        Else
            Return 0
        End If
    End Function
End Class
```

في المثال السابق، اعتمدت على خاصية الاسم Name لتحديد ناتج المقارنة بين الكائنين، قد تغير أنت رأيك وتعتمد على خاصية العمر Age مثلا في فئاتك الخاصة:

```
Class Person
    Implements IComparable
    Public Age As Integer

    Function CompareTo(ByVal obj As Object) As Integer Implements _
        System.IComparable.CompareTo

        Dim tempObj As Person = CType(obj, Person)

        ' Age الاعتماد على خاصية العمر
        If Me.Age < tempObj.Age Then
            Return -1
        ElseIf Me.Age > tempObj.Age Then
            Return 1
        Else
            Return 0
        End If
    End Function
End Class
```

والآن فئتنا Person تحتوي على الواجهة IComparable لذلك تستطيع إرسال الكائنات المنشئة من هذه الفئة إلى جميع إجراءات فئات .NET Framework الأخرى (والتي تتطلب الواجهة IComparable طبعا). فمثلا الطريقة Sort() (والتابعة للفئة Array) يمكن ان تستقبل هذا النوع:

```
Dim PersonObject(10) As Person
Dim Counter As Integer

For Counter = 0 To UBound(PersonObject)
    PersonObject(Counter) = New Person()
Next

PersonObject(0).Age = 33
PersonObject(1).Age = 50
PersonObject(2).Age = 14
```

```
…
…

' الطريقة Array.Sort تقبل اي وسيطة تحتوي
' على الواجهة IComparable
Array.Sort(PersonObject)

For Counter = 0 To UBound(PersonObject)
    ArabicConsole.WriteLine(PersonObject(Counter).Age)
Next
```

الواجهة ICloneable

عند التعامل مع المتغيرات المرجعية Reference Type Variables، فقد أخبرتك أكثر من مرة ان عملية إسناد القيم بين متغيرين لا تقوم بإنشاء نسخة من المتغير وإسنادها إلى الآخر، وإنما تقوم بجعل كلا المتغيرين يشيران إلى نفس الكائن:

```
Dim Turki As New Person ()
Dim Ali As Person

…
…

' لا تقوم بإنشاء نسخة جديدة وانما
' جعل كلا المتغرين يشيران إلى نفس الكائن
Ali = Turki
```

معظم الفئات في عالم NET. تقوم بتضمين الواجهة ICloneable لتعريف طريقتها الوحيدة Clone() والتي تقوم بنسخ الكائن وإنشاء نسخة جديدة منه وإسنادها إلى متغير آخر، هذا مثال لتعريف الطريقة في فئة:

```
Class Person
    Implements ICloneable

    Public Name As String
    Public Age As Integer

    Function Clone() As Object Implements ICloneable.Clone
        ' إنشاء كائن مؤقت
        Dim tmpObject As New Person()

        ' نسخ قيم الكائن الحالي إلى الكائن
        ' المؤقت يدويا
        tmpObject.Name = Me.Name
        tmpObject.Age = Me.Age
```

```
        ' العودة بالكائن المنشئ للتو
        Return tmpObject
    End Function
End Class
```

وبدلا من قيامك بإنشاء الكائن ونسخ خصائصه يدويا (حيث انه يمكن أن توجد متغيرات ستاتيكية Static داخل الطرق والتي لن تتمكن من نسخها بنفسك)، فيمكنك استخدام الطريقة MemberwiseClone() وهي مشتقة من الفئة System.Object (أعود للتذكير بان جميع البيانات والأنواع وكل شيء في عالم .NET. مشتقة وراثيا من الفئة System.Object):

```
Class Person
    Implements ICloneable

    Public Name As String
    Public Age As Integer

    Function Clone() As Object Implements ICloneable.Clone
        ' العودة بنسخة جديدة من الكائن الحالي
        Return Me.MemberwiseClone()
    End Function
End Class
```

والآن يمكنك استخدامها بهذا الشكل:

```
Dim Turki As New Person()
Dim Ali As Person

Turki.Name = "تركي"

' نسخ الكائن
Ali = CType(Turki.Clone, Person)

Ali.Name = "علي"

' تحقق ان الكائن الاول لم يتم تغيير قيمته
ArabicConsole.WriteLine(Turki.Name)
```

كما ترى في الشيفرة السابقة، اضطررت إلى استخدام المعامل CType بافتراض ان العبارة Option Strict On مفعلة في الملف الحالي أو على مستوى المشروع، والسبب ان الطريقة Clone() تعود بقيمة من النوع Object. قد يكون استخدام المعامل CType غير محبذ

لمستخدمي فئاتك، لذلك اقترح عليك تغيير اسم الطريقة ()Clone في الفئة واجعلها مخفية، ومن ثم عرف طريقة أخرى لتحل محلها:

```
Class Person
    Implements ICloneable
    …
    …

    Private Function PrivateClone() As Object Implements
ICloneable.Clone
        Return Me.Clone()
    End Function

    Public Function Clone() As Person
        Return CType(Me.MemberwiseClone(), Person)
    End Function
End Class
```

هنا سيتمكن مستخدم الفئة من استدعاء الطريقة ()Clone دون الحاجة لاستخدام المعامل CType:

```
Ali = Turki.Clone
```

وقبل ان اختم فقرة **الواجهة ICloneable**، دعني أحذرك وأنبهك بان عمليات النسخ التي قمنا بها سابقا (باستخدام الطريقة ()MemberwiseClone) توجد بها مشكلة خطيرة جدا جدا، ولم اعرضها هنا لان حلها يتم عن طريق ما يعرف بالتسلسل. لذلك، لا تتبع الأسلوب السابق في نسخ الكائنات بمشاريعك الحالية حتى تصل إلى الفصل التاسع **تسلسل الكائنات Object Serialization**. اللهم إني قد بلغت، اللهم فاشهد.

الواجهتان IEnumerable و IEnumerator

ذكرت في الفصل الثاني **لغة البرمجة** انك تستطيع تطبيق الحلقة For Each … Next على المصفوفات Arrays أو المجموعات Collections، وبما أننا نتحدث الآن بلغة الواجهات Interfaces فدعني أكون أكثر دقةً معك، وأخبرك ان الحلقة For Each يمكن تطبيقها أيضا على جميع أنواع البيانات الاخرى التي تدعم الواجهتين IEnumerable و IEnumerator.

تعمل الواجهتان IEnumerable و IEnumerator جنبا إلى جنب لتمكين المبرمجين من استخدام الحلقة For Each مع فئاتك، وهذا مثال عملي يشرح كيفية استخدام هاتين الواجهتين، وسنبدأ أولاً بتصميم الفئة SplitString والتي تقوم بتقسيم الجملة إلى كلمات، تحتوي هذه الفئة على الخاصية Sentence والتي تمثل الجملة المراد فصل كلماتها:

PDF Factory Pro trial version www.fastio.com

```
Class SplitString
    Private currentPosition As Integer = 0

    ' خاصية تمثل الجملة المراد فصل كلماتها
    Private m_Sentence As String

    Property Sentence() As String
        Get
            Return m_Sentence
        End Get
        Set(ByVal Value As String)
            m_Sentence = Value
            Me.Reset()
            End Set
    End Property
End Class
```

الخطوة التالية هي تضمين الواجهتين IEnumerable و IEnumerator في الفئة السابقة:

```
Class SplitString
    Implements IEnumerable
    Implements IEnumerator
    …
    …
End Class
```

بالنسبة للواجهة IEnumerable، فهي لا تحتوي إلا على طريقة واحـــدة GetEnumerator() سيتم استدعائها بمجرد تنفيذ حلقة For Each، وهي تعود بقيمة لاي كائن يحتوي على الواجهـــة الأخرى IEnumerator. يكفي كتابة الأمر Return Me لتنفيذها:

```
Class SplitString
    …
    …
    Private Function GetEnumerator() As IEnumerator Implements _
        IEnumerable.GetEnumerator

        Return Me
    End Function
End Class
```

اما الواجهة IEnumerator فهي تحتوي على طريقتين وخاصية واحدة، الطريقة الأولـــى تسمى MoveNext() يتم استدعائها في كل دورة من دورات حلقة التكرار For Each، ويفترض ان تعود بالقيمة True إن بقي عنصر تالي و False إن لم يتبقى اي شيء:

```
Class SplitString
    …
    …
    Private Function MoveNext() As Boolean Implements
IEnumerator.MoveNext
        If Me.currentPosition > Me.Sentence.Length - 1 Then
                Me.Reset()
            Return False
        Else
            Return True
        End If
    End Function
End Class
```

والطريقة الثانية هي Reset() والغرض منها إعادة المؤشر الداخلي في الفئة إلى أول عنصر فيها حتى تتمكن من البدء باستخدام الحلقة For Each مرة أخرى:

```
Class SplitString
    …
    …
    Private Sub Reset() Implements IEnumerator.Reset
        Me.currentPosition = 0
    End Sub
End Class
```

اما الخاصية فهي تحمل الاسم Current والغرض منها العودة بالقيمة الحالية في الحلقة، لا تنسى ان هذه الخاصية للقراءة فقط ReadOnly:

```
Class SplitString
    …
    …
    Private ReadOnly Property Current() As Object Implements _
        IEnumerator.Current

        Get
            Dim counter As Integer
            Dim tmpLength As Integer = 0
```

```
            For counter = Me.currentPosition To Me.Sentence.Length - 1
                If Me.Sentence.Chars(counter) = " "c Then
                    Exit For
                Else
                    tmpLength += 1
                End If
            Next
            Current = Me.Sentence.Substring(Me.currentPosition,
tmpLength)
            Me.currentPosition += tmpLength + 1
        End Get
    End Property
End Class
```

والآن يمكنك تطبيق الحلقة For Each على الفئة السابقة SplitString بمرونة كبيرة، لتتمكن من كتابة شيئا مثل:

```
Dim testObject As New SplitString()
Dim x As String

testObject.Sentence = "سيتم فصل كلمات هذه الجملة في سطور مستقلة"

For Each x In testObject
    ArabicConsole.WriteLine(x)
Next
```

مخرجات الشيفرة السابقة ستكون كالتالي:

```
سيتم
فصل
كلمات
هذه
الجملة
في
سطور
مستقلة
```

PDF Converter

ملاحظة

الهدف من إنشاء الفئة SplitString هو تعليمي بحت، أردت أن أوضح من خلاله كيفية تطبيق الواجهتين IEnumerable و IEnumerator، فلا تحاول استخدامها في برامجك الجدية، حيث يوفر لك اطار عمل .NET Framework فئات أفضل وأسرع بكثير لتقسيم النصوص.
نقطة أخرى حول تضمين الواجهتين IEnumerable و IEnumerator، من المفضل ألا تقوم بتضمين هاتين الواجهتين في فئة واحدة، فيفضل تضمين الواجهة IEnumerable في فئة حاضنة، والواجهة IEnumerator في فئة محضونة بها، وذلك لتصميم كائني توجه OOP أفضل.

التفويض

يقصد بكلمة **التفويض Delegates**: بتفويض عملية استدعاء الإجراء إلى متغير يشير إلى ذلك الإجراء، مما يعطيك مرونة كبيرة في اختيار أوقات استدعاء الإجراءات، كما يختصر عليك الكثير من الشيفرات المصدرية والتي تستدعي هذه الإجراءات، فلو كان لدينا الإجراءات التالية:

```
Sub MySub (ByVal X As Integer)
    …
    …
End Sub

Sub YourSub (ByVal X As Integer)
    …
    …
End Sub
```

فإنه بامكاننا استدعائها عن طريق متغير يمثل مؤشر إليها:

```
' سيتم استدعاء الإجراء الذي يشير له المتغير Dlgt
' كما سترسل اليه قيم الوسيطات
Dlgt.Invoke (100)
```

المزيد أيضا، تستطيع إرسال هذه المؤشرات كوسيطة إلى إجراء اخر وتمكن الإجراء من استدعائها:

```
Sub ShowError(ByVal delegatePointer As OneParameter)
    delegatePointer.Invoke("حدث خطأ في العملية")
End Sub
```

PDF

تفيدك الطريقة السابقة كثيرا في اختصار تكرار الشيفرات المتشابهة، حيث ان الإجراء السابق قد يظهر رسالة الخطأ اما على شكل نافذة Message Box، في شاشة Console، عرضها على متصفح Browser (ان كنت تصمم موقعاً)، أو الكتابة في ملف ...الخ، دون الحاجة لاعادة استدعاء الإجراء السابق من جديد. ميزة اخرى اكثر إثارة-سترى تطبيقاً عملياً عليها قريباً- هي إمكانية استدعاء إجراءات الفئات (أعضاء الفئات) من داخل الإجراءات المشتركة Shared دون مشاكل.

ان كنت من مبرمجي لغة C\C++ فقد يخطر ببالك **مؤشرات الدوال Function Pointers** والتي تمكنك من استدعاء الدوال عن طريق مؤشراتها. بشكل مبدئي، مؤشرات الدوال في لغة C\C++ شبيهة بالإجراءات المفوضة في لغة Visual Basic .NET ولكنها تختلف في ميكانيكية عملها وصيغ كتابتها، حيث ان الإجراءات المفوضة في Visual Basic .NET اكثر أماناً من مؤشرات الدوال في لغة C\C++ وذلك لانها تحصر نوعية الإجراءات بحيث تتوافق مع وسيطاتها. نقطة أخرى، الإجراءات المفوضة في Visual Basic .NET يمكن ان تتبع لإجراءات ستاتيكية Static Procedures أو تابعة لفئات Instance Procedures، بينما مؤشرات الإجراءات في لغة C\C++ لا يمكن تطبيقها الا مع الإجراءات الستاتيكية فقط.

الإجراءات الستاتيكية

اقصد بعبارة الإجراءات الستاتيكية Static Procedures الإجراءات (سواء كانت Subs أو Functions) التي تعرفها في الفئات Classes بحيث تكون مشتركة (باستخدام الكلمة المحجوزة Shared)، أو التي تعرفها في الوحدات البرمجية Modules. عند تفويض الإجراء إلى مؤشر، عليك اولا القيام بتعريف نوع مؤشر الإجراء باستخدام الكلمة المحجوزة Delegates:

```
Module Module1
    ' تعريف نوع مؤشر الإجراء لابد ان يكون
    ' على مستوى الوحدة البرمجية
    Delegate Function OneParameter(ByVal X As Integer) As Integer

    Sub Main()
        …
        …
    End Sub
    …
    …
End Module
```

ولنفترض الآن أن لدينا الإجرائين التاليين:

```
Function Abs(ByVal x As Integer) As Integer
    If x < 0 Then
        Return -x
    Else
        Return x
    End If
End Function

Function Square(ByVal x As Integer) As Integer
    Return x * x
End Function
```

ونريد تفويضها إلى اي مؤشر من النوع OneParameter (الذي عرفناه سابقا باستخدام الكلمة المحجوزة Delegate) في اي وقت بالطريقة التالية:

```
Sub Main ()
    ' Abs () تفويض الإجراء
    Dim Dlgt As New OneParameter(AddressOf Abs)
    …
    …
End Sub
```

الان يمكنك استدعاء الإجراء Abs() عن طريق هذا المؤشر باستدعاء الطريقة Invoke() وارسال الوسيطات المطلوبة:

```
Sub Main ()
    Dim Dlgt As New OneParameter(AddressOf Abs)

    ' استدعاء الإجراء
    ArabicConsole.WriteLine( Dlgt.Invoke(-5) ) ' 5
End Sub
```

تستطيع الآن تفويض الإجراء الاخر Square() في اي وقت بإسناد عنوانه (باستخدام الكلمة المحجوزة AddressOf) إلى المؤشر Dlgt السابق، وعملية الاستدعاء ستتم بنفس الطريقة Invoke():

```
Sub Main ()
    …
    …
    ' Square () تفويض الإجراء
    Dlgt = AddressOf Square
    ArabicConsole.WriteLine( Dlgt.Invoke(-5) ) ' 25
End Sub
```

المزيد أيضا، تستطيع تعريف إجراء يستقبل وسيطة من النوع OneParameter لتستدعي الإجراءات التي يشير إليها الكائن المرسل:

```
Sub Main()
    Dim Dlgt As New OneParameter(AddressOf Abs)
    …
    …
    MySub(Dlgt)
End Sub

' OneParameter إجراء يستقبل وسيطة من النوع
Sub MySub(ByVal delegatePointer As OneParameter)
    ArabicConsole.WriteLine(delegatePointer.Invoke(-5))
End Sub
```

دعنا نرى الآن ماذا يحدث خلف الكواليس، لأوضح لك حقيقة استخدام الكلمة المحجوزة Delegates كما في السطر الاول من المثال السابق عندما عرفت النوع OneParameter:

```
Delegate Function OneParameter(ByVal X As Integer) As Integer
```

تقنيا، قام مترجم اللغة بتحويل العبارة السابقة إلى فئة باسم OneParameter مشتقة وراثيا من الفئة System.MulticastDelegate (والتي هي أيضا مشتقة من الفئة System.Delegate)، اي يمكنك افتراض أن الشيفرة السابقة قد تم كتابتها لتعريف فئة OneParameter:

```
' هذا مجرد افتراض، فلا تطبقه في الواقع
Class OneParameter
    Inherits System.MulticastDelegate
    …
    …
End Class
```

لذلك، عندما أردنا إنشاء نسخة جديدة من الكائن اضطررنا لاستخدام الكلمة المحجوزة New ومن ثم إرسال القيم لوسيطات المشيد:

```
Dim Dlgt As New OneParameter(AddressOf Abs)
```

ما أود الوصول إليه هو ان Dlgt مؤشر لكائن يحتوي على خصائص وطرق إضافية (ذكرت إحداها وهي الطريقة Invoke() والتي تستدعي الإجراء المشار اليه)، كما يمكنك التعامل مع هذا الكائن كما تتعامل من كائنات الفئات التي تعرفها بنفسك لتتمكن من كتابة شيئاً مثل:

```
Dim Dlgt As New OneParameter(AddressOf Abs)
Dim X As OneParameter

' إسناد قيم بين كائنين
X = Dlgt

ArabicConsole.WriteLine(X.Invoke(-5)) ' 5
```

إجراءات الفئات

في الفقرة السابقة طبقنا التفويض على الإجراءات الستاتيكية، اما إن رغبت في تطبيق التفويض على إجراءات الفئات فالعملية ستتم بنفس الأسلوب السابق إلا انك بحاجة إلى استخدام اسم الكائن الذي يتبع له الإجراء عند إرسال عنوان الإجراء (باستخدام AddressOf) إلى المؤشر المفوض:

```
Class TestClass
    Sub TestMethod()
        ArabicConsole.WriteLine("طريقة من الكائن")
    End Sub
End Class

Module Module1
    Delegate Sub NoParameter()

    Sub Main()
        Dim testObject As New TestClass()

        ' testObject لاحظ هنا استخدام اسم الكائن
        Dim dlgt As New NoParameter(AddressOf testObject.TestMethod)

        dlgt.Invoke()
    End Sub
End Module
```

باستخدام التفويض، يمكنك تخطي حاجز المنع من استدعاء إجراءات الفئة من داخل الإجراءات المشتركة في الفئة، حيث وكما ذكرت في الفصل الثالث **الفئات والكائنات** أنه لا يمكنك استدعاء طريقة في الفئة من داخل طريقة مشتركة في نفس الفئة:

```
Class TestClass
    Sub InstanceMethod()
        …
        …
    End Sub

    Shared Sub SharedMethod()
        ' لا يمكنك عمل ذلك
        InstanceMethod()
    End Sub
End Class
```

تستطيع تطبيق العملية السابقة باستخدام التفويض بعشرات الأساليب إما باستخدام متغيرات عامة Global Variables أو إرسالها كوسيطات للإجراءات المشتركة:

```
Class TestClass
    Sub InstanceMethod()
        ArabicConsole.WriteLine("تم استدعاء الطريقة")
    End Sub

    Shared Sub SharedMethod(ByVal methodPointer As NoParameter)
        methodPointer.Invoke()
    End Sub
End Class

Module Module1
    Delegate Sub NoParameter()

    Sub Main()
        Dim testObject As New TestClass()
        Dim dlgt As New NoParameter(AddressOf
testObject.InstanceMethod)

        ' InstanceMethod() سيتم استدعاء الطريقة
        ' SharedMethod() من خلال الطريقة المشتركة
        testObject.SharedMethod(dlgt)
    End Sub
End Module
```

محاكاة الأحداث

إن استخدمت المنطق البرمجي قليلا، فسرعان ما ستكتشف ان الأحداث Events التي تعرفها في الفئات تطبق أسلوب التفويض Delegates ولكن بصيغة مختلفة، فالحدث Die التالي:

```
Class Person
    Event Die()

    Sub Kill()
        RaiseEvent Die()
    End Sub
End Class
```

PDF Converter

```
Module Module1
    Sub Main()
        Dim Turki As New Person()

        AddHandler Turki.Die, AddressOf PeronHasDied

        Turki.Kill()
    End Sub

    Sub PeronHasDied()
        ArabicConsole.WriteLine("لقد توفي الشخص")
    End Sub
End Module
```

ما هو إلا تفويض متمثل في الاسم Die، فأنت تقوم بتحديد عنوان الإجراء الذي سيتم تنفيذه باستخدام AddHandler. يمكنك أيضا الاستغناء عن صيغة إطلاق الأحداث السابق وتعريف تفويض (باسم NoParameter مثلا) ومن ثم التصريح عن متغير في الفئة يحمل اسم الحدث Die، وان أردت إطلاق هذا الحدث ستقوم باستدعاء الطريقة Invoke() عوضا عن الأمر RaiseEvent. الشيفرة التالية هي نفس الشيفرة السابقة، ولكني هنا استخدمت التفويض لإطلاق الحدث بدلاً من الصيغة المباشرة:

```
Delegate Sub NoParameter()

Class Person
    Public Die As NoParameter

    Sub Kill()
        Die.invoke()
    End Sub
End Class

Module Module1
    Sub Main()
        Dim Turki As New Person()

        Turki.Die = New NoParameter(AddressOf PeronHasDied)

        Turki.Kill()
    End Sub

    Sub PeronHasDied()
        ArabicConsole.WriteLine("لقد توفي الشخص")
    End Sub
End Module
```

دمج التفويضات

من الأشياء الرائعة جدا في استخدام تقنية التفويض، هو إمكانية دمج أكثر من تفويض في مؤشر واحد بحيث يمكنك تنفيذ أكثر من إجراء باستدعاء واحد لتضرب أكثر من عصفور بحجر واحد. فمثلا، لنفترض ان لدينا الاجرائين التاليين:

```
Sub MySub1()
    ArabicConsole.WriteLine("الإجراء الاول")
End Sub

Sub MySub2()
    ArabicConsole.WriteLine("الإجراء الثاني")
End Sub
```

ونريد استدعاء الاجرائين باستخدام التفويض، فإنه يتوجب علينا استدعاء كل إجراء على حدة - باستخدام الطريقة Invoke():

```
Delegate Sub NoParameter()

sub Main ()
    Dim dlgt As New NoParameter(AddressOf MySub1)
    Dim dlgt2 As New NoParameter(AddressOf MySub2)

    dlgt.Invoke()
    dlgt2.Invoke()
End Sub
```

اما عند الدمج فلست بحاجة لكل هذا، حيث يكفي استخدام الطريقة Combine() لتدمج اكثر من تفويض في مؤشر واحد، لتتمكن من تنفيذ الاجرائين MySub1() و MySub2() باستدعاء واحد فقط للطريقة Invoke(). تستطيع فعل ذلك بإنشاء متغير ثالث للقيام بهذه العملية:

```
Delegate Sub NoParameter()

Sub Main()
    Dim dlgt As New NoParameter(AddressOf MySub1)
    Dim dlgt2 As New NoParameter(AddressOf MySub2)
    Dim dlgt3 As NoParameter

    dlgt3 = dlgt.Combine(dlgt, dlgt2)

    ' MySub2 و MySub1 سيتم تنفيذ كلا الاجرائين
    dlgt3.Invoke()
End Sub
```

PDF

أو استخدام متغير واحد فقط لتنفيذ هذه العملية، بحيث يتم إنشاء الكائنات لحظة إرسال الوسيطات:

```
Delegate Sub NoParameter()

Sub Main()
    Dim dlgt As NoParameter

    dlgt = dlgt.Combine(New NoParameter(AddressOf MySub1), _
           New NoParameter(AddressOf MySub2))

    ' MySub2 و MySub1 سيتم تنفيذ كلا الاجرائين
    dlgt.Invoke()
End Sub
```

الطريقة Combine() طريقة مشتركة Shared Method لذلك تمكنا من استدعائها قبل إنشاء نسخة جديدة من الكائن في المؤشر dlgt السابق. ودعني هنا أنبهك إلى أن هذه الطريقة ستعود بقيمة من النوع System.Delegate لذلك لابد من استخدام المعامل CType ان كانت العبارة Option Strict On مفعلة:

```
dlgt = CType(dlgt.Combine( New NoParameter(AddressOf MySub1), _
       New NoParameter(AddressOf MySub2) ), NoParameter)
```

المزيد أيضا، يمكن دمج إجراء ثالث في المؤشر dlgt السابق، وعند استدعاء الطريقة Invoke() سيتم تنفيذ جميع الإجراءات الثلاثة بالتتالي (الأقدم فالأحدث):

```
' دمج إجراء ثالث
dlgt = CType(dlgt.Combine(dlgt, New NoParameter(AddressOf MySub3)), _
       NoParameter)

' سيتم تنفيذ الإجراءات
' MySub1()
' MySub2()
' MySub3()
' بالتسلسل
dlgt.Invoke()
```

نقطة أخرى حول دمج التفويضات، ففي حالة ما اذا كان نوع الإجراء المفوض Function فعليك معرفة أن الإجراء الأخير هو الذي سيعود بالقيمة:

```
Module Module1
    Delegate Function NoParameterFun() As Boolean

    Sub Main()
        Dim dlgt As New NoParameterFun(AddressOf MyFunction1)
        Dim dlgt2 As New NoParameterFun(AddressOf MyFunction2)

        dlgt = CType(dlgt.Combine(dlgt, dlgt2), NoParameterFun)

        ArabicConsole.WriteLine(dlgt.Invoke()) ' True
    End Sub

    Function MyFunction1() As Boolean
        Return False
    End Function

    Function MyFunction2() As Boolean
        Return True
    End Function
End Module
```

أخيرا، إن رغبت في حذف الإجراء من مؤشر التفويض، فلن تجد أسهل ولا أجمل من الطريقة Remove():

```
dlgt = CType(dlgt.Remove(dlgt, dlgt2), NoParameterFun)

ArabicConsole.WriteLine(dlgt.Invoke()) ' False
```

ملاحظة

الطريقة Remove() السابقة تحذف آخر إجراء تم إضافته للمتغير المفوض. مع ذلك، جميع الإجراءات المدمجة في المتغير تحفظ في الخاصية GetInvocationList وهي مصفوفة. لذلك، الشيفرة التالية تمكنك من حذف الإجراء الاول وليس الأخير:

```
dlgt.Remove(dlgt, dlgt.GetInvocationList(0))
ArabicConsole.WriteLine(dlgt.Invoke()) ' True
```

PDF Factory Pro trial version www.fastio.com

المواصفات

مصطلح **المواصفات Attributes** التي يوفرها لك إطار عمل NET. Framework جديدة – نسبيا- في لغات البرمجة، الهدف منها هو ان بعض محتويات مشاريعك قد لا تعتمد على كتابة الشيفرات المصدرية Source Codes بشكل كامل، وإنما قد تشمل تعاريف ونصوص حرفية Textual وعبارات توجيه للمترجم Compiler.

فمثلا، المعلومات العامة حول الفئات وأعضائها تعتمد على النصوص لتشرح الغرض منها ومتطلباتها أو حتى الوصول إلى الصفحة المناسبة لملف التعليمات الخاص بها. هناك أيضا بعض اوامر التوجيه الخاصة بالمترجم Compiler والتي تمكنك من التحكم في الشيفرات التي ترغب في ترجمتها ان تحققت شروط معينة. المزيد أيضا، قد تفرض وتحدد مواقع معينة لمتغيرات التركيبات من نوع Structure وطريقة تحديد مواقعها بالذاكرة.

بعض هذه الأمور متوفرة في لغات البرمجة ولكن لكل لغة طريقة خاصة ومعينة في تطبيقها، فنجد في لغة C\C++ العبارات pragma# للسيطرة على المترجم، واما في لغة Visual Basic 6 فقد كان يعتمد بشكل كبير على صناديق الحوار التي تنشئ ملفات خاصة بها (كـ ctl. و .idl) لحفظ هذه المواصفات. ولكن مع إطار عمل NET. Framework، فقد توحدت فكرة تطبيق المواصفات Attributes بين لغات NET. الأخرى. ليس هذا فقط، بل أصبحت تمكنك من بناء مواصفات خاصة بك.

صيغة كتابة المواصفات في Visual Basic .NET

في البداية عليك معرفة ان المواصفات ما هي الا فئات Classes ولكن لها صيغة قياسية لا بد من إتباعها لتتمكن من استخدامها في مشاريعك، استخدم علامة الأقواس المعقوفة < و > لكتابة المواصفة:

```
<System.ComponentModel.DefaultProperty ("XXX")>
```

وبدلا من كتابة الاسم الكامل للمواصفة كما في المثال السابق، اقترح عليك باستيراد مجال الأسماء Namespace الذي توجد فيه المواصفة باستخدام Imports، لتختصر على نفسك كتابة الاسم الكامل في كل مرة تود فيها استخدام المواصفة:

```
Imports System.ComponentModel
<DefaultProperty ("XXX")>
```

PDF

السؤال هنا أين ستكتب المواصفة؟ والجواب يعتمد على نوع المواصفة التي ستستخدمها، فبعض المواصفات يمكن كتابتها في اي جزء أو كتلة من شيفراتك المصدرية، والبعض الآخر محصور في أماكن معينة كما سترى لاحقا. فمثلا، المواصفة DebuggerStepThrough لن تستطيع استخدامها إلا عند تعريف الإجراءات فقط:

```
<System.Diagnostics.DebuggerStepThrough()> Sub MySub()
    …
    …
    …
End Sub
```

قياسياً سنستخدم المعامل _ لفصل المواصفة عن الشيفرة المصدرية بحيث تتوزع على سطرين مختلفين لتسهيل قراءة وتتبع الشيفرة:

```
<System.Diagnostics.DebuggerStepThrough()> _
Sub MySub()
    …
    …
    …
End Sub
```

مواصفات من إطار عمل .NET Framework

يوفر لك إطار عمل .NET Framework العشرات من المواصفات والتي يمكن استخدامها في أجزاء متفرقة من البرنامج لأداء وظائف معينه، في الفقرات التالية سنلقي الضوء على بعض هذه المواصفات والموجهة للغة البرمجة Visual Basic .NET، كما سأخصص فقرة كاملة لأعرض لك كيفية بناء مواصفة خاصة بك.

المواصفة Conditional Attribute

عملية توجيه المترجم تقتضي ترجمة سطور معينة من الشيفرة المصدرية لحظة تحقق شرط معين، في معظم لغات البرمجة السابقة يستخدم الموجه #If لهذا الغرض:

```
#If MyCondition Then
    Sub MySub ()
        …
        …
    End Sub
#End If
```

```
Sub Main
    MySub ()
    …
    …
    MySub ()
End Sub
```

المشكلة في هذا الأسلوب تبرز إن أردت إلغاء الإجراء MySub() السابق (عن طريق تغيير قيمة الشرط MyCondition ليكون False) فستظهر مئات الأخطاء في باقي سطور البرمجة –لحظة الترجمة- عند كل استدعاء للإجراء MySub()، احد الحلول هو استخدام الموجه #If عند كل استدعاء:

```
Sub Main
    #If MyCondition Then
        MySub ()
    #End If
    …
    …
    #If MyCondition Then
        MySub ()
    #End If
    …
    …
End Sub
```

صحيح ان الاسلوب السابق غير عملي، لذلك قد تدخل عبارة الشرط للموجه داخل الإجراء MySub() وتسهل عليك المهمة:

```
Sub MySub ()
    #If MyCondition Then
        …
        …
    #End If
End Sub
```

مع ذلك، إن كانت نتيجة الشرط MyCondition تساوي False فإن مئات السطور من الشيفرات المصدرية (والتي تستدعي الإجراء MySub() بالتحديد) أصبحت غير ذات قيمة، والتي ستتسبب في بطء عملية الترجمة –والتنفيذ أيضا- بالإضافة إلى كبر حجم البرنامج.

يوفر لك إطار عمل .NET Framework. المواصفة Conditional Attribute والتي تمكنك من توجيه المترجم عند تحقق شرط معين، وطريقة استخدامها يكون عند تعريف الإجراءات من نوع Sub:

```
<Conditional("MyCondition")> _
Sub MySub()
    …
    …
End Sub

Sub Main
    MySub ()
    …
    …
    MySub ()
    …
    …
End Sub
```

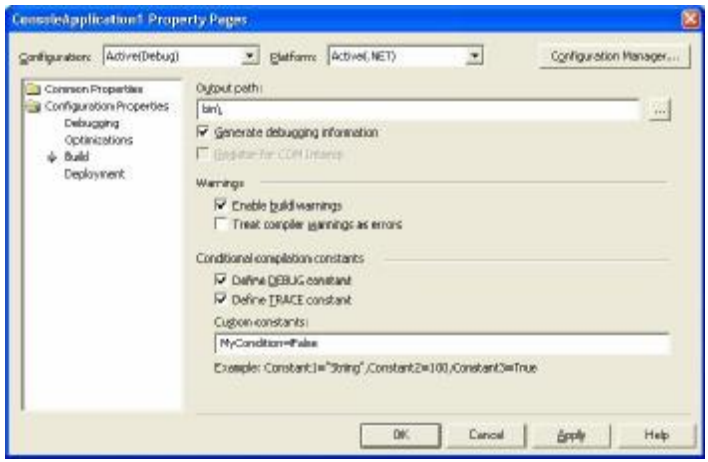
الإجراء MySub() السابق وجميع السطور التي ستستدعيه في كافة ملفات المشروع الأخرى، سيتم تجاهلها لحظة الترجمة ولن يتم ترجمتها وتضمينها في الملف النهائي ان كانت نتيجة الشرط MyCondition تساوي صفر أو False.

ملاحظة

لا يمكنك تطبيق المواصفة Conditional على الإجراءات من نوع Function.

بالنسبة للشرط MyCondition السابق فما هو الا ثابت يمكنك تعريفه وتحديد قيمته من صندوق حوار خصائص المشروع Project Property Pages، والانتقال إلى خانة التبويب Configuration Properties ثم خانة التبويب الفرعية Build ثم كتابة اسم الثابت وقيمته تحت خانة النص Custom Constants **(شكل 5-2 بأعلى الصفحة المقابلة)**.

PDF Factory Pro trial version www.fastio.com



شكل 5-2: تحديد ثوابت الترجمة في صندوق حوار Project Property Pages.

المواصفة DebuggerStepThrough Attributes

في الفصل السابع **اكتشاف الأخطاء** سأتطرق إلى بعض طرق التنقيح Debugging التي توفرها بيئة Visual Studio .NET ومنها التنفيذ سطرا سطرا (لعمل ذلك اختر الأمر Step Into من قائمة Debug أو اضغط على المفتاح [F11]).

في بعض الإجراءات التي تكون متأكدا من صحتها ولست بحاجة إلى التدقيق فيها سطرا تلو الآخر، يمكنك استخدام المواصفة System.Diagnostics.DebuggerStepThrough عند بداية الإجراء ليتم تنفيذه دفعة واحد:

```
<System.Diagnostics.DebuggerStepThrough()> _
Sub MySub()
    …
    …
    …
End Sub
```

المواصفة Obsolete Attribute

الغرض من هذه المواصفة هو إظهار رسائل التنبيه Warning أو الخطأ لحظة ترجمة البرنامج، يمكنك الاستفادة منها مثلا عندما تقوم بإرسال برنامجك إلى آخرين وتود لفت انتباههم إلى جزء معين من الشيفرة أو التحقق منها:

```
<Obsolete("يا عباس، لا تنسى تغيير شيفرة هذا الإجراء قبل التوزيع")> _
Sub MySub()
    …
    …
End Sub
```

عند الترجمة، ستظهر الرسالة السابقة كإنذار Warning في نافذة المخرجات Output (والتي تصل إليها باختيار الأمر View->Other Windows->Output) (**شكل 5-3**).

شكل 5-3: رسالة التنبيه Warning ظهرت أثناء الترجمة.

المزيد أيضا، المواصفة Obsolete تستقبل وسيطة أخرى من النوع Boolean، إن أرسلت لها القيمة True ستتحول الرسالة من رسالة تنبيه إلى رسالة خطأ لتوقف عملية الترجمة ولن يتم تنفيذ البرنامج:

```
<Obsolete("الم اقل لك عدل الإجراء يا عنيد", True)> _
Sub MySub()
    …
    …
    …
End Sub
```

ملاحظة

سواء كانت رسالة تنبيه أو خطأ، فلن يتم عرضها إلا عند السطر الذي يستدعي الإجراء وليس بمجرد استخدام المواصفة Obsolete.

PDF Factory Pro trial version www.fastio.com

المواصفة StructLayout والمواصفة FieldOffset

استخدام المواصفة StructLayout موجه بشكل حصري إلى التركيبات من النـــوع Structure، حيث تمكنك من تحديد طريقـــة ترتيـــب متغيـــرات التركيـــب فـــي الـــذاكرة، أرســـل القيمـــة LayoutKind.Auto للحصول على أفضل وانسب أداء للمعالج:

```
' استيراد مجال الاسماء لاختصار الشيفرات
Imports System.Runtime.InteropServices

<StructLayout(LayoutKind.Auto)> _
Structure RGBValue
    Dim Red As Byte
    Dim Green As Byte
    Dim Blue As Byte
End Structure
```

أما إن أرسلت القيمة LayoutKind.Explicit فعليك تحديد موقع كل متغير من متغيرات التركيب في الذاكرة بنفسك عن طريقة استخدام المواصفة FieldOffset:

```
<StructLayout(LayoutKind.Explicit)> _
Structure RGBValue
    <FieldOffset(0)> Dim Red As Byte
    <FieldOffset(1)> Dim Green As Byte
    <FieldOffset(2)> Dim Blue As Byte
End Structure
```

استخدامك للمواصفة FieldOffset يعطيك مرونة كبيرة للتحكم في مواقع المتغيرات فـــي الذاكرة، فمثلا التركيب التالي لا يحتجز الا بايت واحد تصل اليه من ثلاث متغيرات:

```
<StructLayout(LayoutKind.Explicit)> _
Structure RGBValue
    <FieldOffset(0)> Dim Red As Byte
    <FieldOffset(0)> Dim Green As Byte
    <FieldOffset(0)> Dim Blue As Byte
End Structure
```

ولنتأكد من كلامي:

```
Dim X As RGBValue
X.Blue = 10
ArabicConsole.WriteLine(X.Red) ' 10
```

PDF

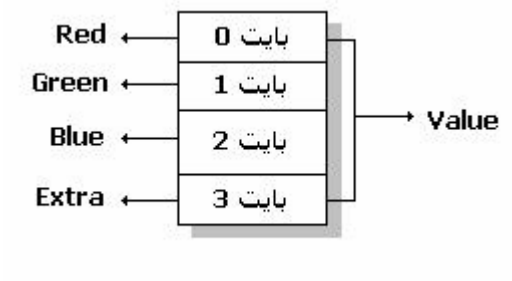
المزيد أيضا، تستطيع الوصول إلى قيمة المتغيرات الثلاثة كلها عن طريق متغيـــر واحـــد، راقب هذا التركيب:

```
<StructLayout(LayoutKind.Explicit)> _
Structure RGBValue
    <FieldOffset(0)> Dim Red As Byte
    <FieldOffset(1)> Dim Green As Byte
    <FieldOffset(2)> Dim Blue As Byte
    <FieldOffset(3)> Dim Extra As Byte
    <FieldOffset(0)> Dim Value As Integer
End Structure
```

لديك الآن الحرية المطلقة في الوصول إلى قيمة التركيب إما عن طريق المتغير Value (وهـــو يشمل المتغيرات الأربعة) أو عن طريق كل متغير على حدة:

```
Dim X As RGBValue

X.Value = 0
ArabicConsole.WriteLine(X.Red)        ' 0
ArabicConsole.WriteLine(X.Green)      ' 0
ArabicConsole.WriteLine(X.Blue)       ' 0

X.Red = 255
X.Green = 255
X.Blue = 255
ArabicConsole.WriteLine(X.Value)      ' 16777215
```

(**الشكل 5-4**) يوضح لك مواقع متغيرات التركيب RGBValue السابق في الذاكرة.



شكل 5-4: مواقع متغيرات التركيب RGBValue في الذاكرة.

بناء مواصفات خاصة

المواصفات الخاصة Custom Attributes ما هي إلا فئات Classes تقليدية تحتوي على جميع الشيفرات التي تود من المواصفة القيام بها، مع ذلك لابد للفئات -التي ترغب في ان تكون مواصفة Attributes- ان تستوفي عدة شروط، سأذكرها تباعاً، وسنطبقها معاً على مراحل ليسهل عليك استيعاب الفكرة.

سأقوم معك ببناء مواصفة CodeInfo تمكنك من الاحتفاظ بمجموعة بيانات تتعلق بالشيفرة المصدرية (كاسم كاتبها، اسم مراجعها، تاريخ كتابتها، ....الخ).

الشرط الأول: يجب أن ينتهي اسم الفئة بالكلمة Attribute:

```
Class CodeInfoAttribute

End Class
```

الشرط الثاني: ان تكون الفئة مشتقة من الفئة System.Attribute:

```
Class CodeInfoAttribute
    Inherits System.Attribute

End Class
```

الشرط الثالث: ضرورة استخدام المواصفة AttributeUsage وتحديد المكان الذي يمكن استخدام المواصفة الحالية في شيفرات البرنامج:

```
<AttributeUsage(AttributeTargets.All) _
Class CodeInfoAttribute
    Inherits System.Attribute

End Class
```

تلاحظ اني ارسلت القيمة AttributeTargets.All وذلك لأجعل المواصفة الحالية قابلة للاستخدام في جميع أماكن الشيفرات المصدرية، يمكنك إرسال قيم أخرى لحصر مجال استخدام هذه المواصفة في موقع معين، الجدول الموجود في الصفحة التالية يوضح لك قيمة المواصفة، ومجال عملها.

PDF

القيمة	المكان الذي ستستخدم فيه المواصفة الحالية
AttributeTargets.All	في كل الاماكن.
AttributeTargets.Assembly	في المجمع الحالي.
AttributeTargets.Class	عند تعريف الفئات.
AttributeTargets.Constructor	مشيدات الفئات.
AttributeTargets.Delegate	الاجرائات المفوضة –باستخدام الكلمة المحجوزة Delegate.
AttributeTargets.Enum	التركيبات من النوع Enum.
AttributeTargets.Event	الأحداث.
AttributeTargets.Field	الحقول.
AttributeTargets.Interface	الواجهات.
AttributeTargets.Method	الطرق.
AttributeTargets.Module	الوحدات البرمجية.
AttributeTargets.Parameter	وسيطات الإجراءات.
AttributeTargets.Property	الخصائص.
AttributeTargets.ReturnValue	الاجرائات التي تعود بقيمة Functions.
AttributeTargets.Struct	التركيبات من النوع Structure.

ملاحظة

يمكنك تحديد أكثر من موقع باستخدام المعامل Or لحظة ارسال القيمة للمواصفة AttributeUsage، فالقيمة التالية تحصر مجال استخدام المواصفة الحالية على الطرق والخصائص فقط:

```
<AttributeUsage(AttributeTargets.Method Or
AttributeTargets.Property)
```

الشرط الرابع: أنواع البيانات المسموح بها في هذا النوع من الفئات هي: Boolean، Byte، Short، Integer، Long، Char، String، Single، Double، Object، التركيب من نوع Enum، المصفوفات أحادية البعد One-Dimensional Array فقط.

سنقوم الآن بتعريف أعضاء الفئة مع الالتزام بالأنواع المحددة في الشرط الرابع:

```
<AttributeUsage(AttributeTargets.All) _
Class CodeInfoAttribute
    Inherits System.Attribute

    Public ProgrammerName As String   ' اسم المبرمج
    Public TesterName As String       ' اسم المراجع
    Public Tested As Boolean          ' تمت المراجعة؟

    Sub New(ByVal programmerName As String)
        Me.ProgrammerName = programmerName
    End Sub
End Class
```

مبروك! المواصفة جاهزة للاستخدام الان، أرسل وسيطة المشيد لحظة استخدامها في اي مكان من البرنامج (لاحظ تجاهل كتابة الكلمة Attribute عند استدعاء المواصفة):

```
<CodeInfo("تركي العسيري")> _
Sub DoSomthing()
    …
    …
End Sub
```

بالنسبة للخصائص الأخرى TesterName و Tested، فيمكنك إسناد القيم لها بهذه الصيغة:

```
<CodeInfo("تركي العسيري")> _
Class Person

<CodeInfo("تركي العسيري", TesterName:="عباس السريع", Tested:=False)> _
    Sub Move ()
        …
        …
    End Sub

<CodeInfo("تركي العسيري", TesterName:="عبود اللوح", Tested:=True)> _
    Sub Kill ()
        …
        …
    End Sub
    …
    …
End Class
```

كنوع من الاقتراح، قد تضيف شيفرات إضافية لحفظ بيانات المواصفة CodeInfo في ملفات نصية خارجية.

بهذا أكون قد انتهيت من تشييد بنية أساسية لتكون مبرمج Visual Basic .NET حقيقي بعدما تطرقت إلى جميع المبادئ والأساسيات التي لابد على كل مبرمج Visual Basic .NET من معرفتها وإتقانها للإبحار في برمجة واستخدام مكتبة فئات إطار عمل .NET Framework Class Library والآن فإن لديك مطلق الحرية في الانتقال إلى أي جزء ترغب به، رغم أنني أنصحك بشدة بالالتزام بالتسلسل الموضوع في هذا الكتاب والبدء من الفصل التالي **الفئات الأساسية** حتى تتعرف على اكبر قدر من الفئات الأولية قبل تطوير التطبيقات المتقدمة

PDF