Вступление

В Visual Basic 6 для хранения наборов данных применяются массивы и объекты Collection.

При работе с наборами данных программисты довольно часто сталкиваются с некоторыми проблемами. Например, приходится создавать свои алгоритмы поиска и сортировки элементов, хотя программисты Microsoft могли создать встроенные механизмы. Да и удобством использования массивы и коллекции VB 6 едва ли могут похвастаться.

Но с выходом на сцену новой платформы Microsoft .NET все изменилось к лучшему.

Существенно увеличилась функциональность массивов, вместо единственного класса Collection, доступного в VB 6, в .NET появилась возможность использовать 6 различных видов коллекций, каждая из которых «заточена» под свои цели. Эти 6 классов выделены в отдельное пространство имен System.Collections.

Массивы

Все типы данных в .NET являются объектами. Например, при создании переменной типа Integer, на самом деле создается экземпляр класса System.Integer. Это касается и массивов – любой массив являются экземпляром класса System.Array.

Правила объявления массивов не изменились со времен VB 6. Единственное существенное изменение – теперь нумерация элементов массива по умолчанию начинается с нуля. Впрочем, задать нижнюю границу всё-таки можно. Это можно сделать с помощью метода CreateInstance класса Array. Ниже приведены несколько примеров объявлений массивов.

```
Dim a (11) As Date ' Одномерный массив, содержащий 12 элементов
Dim b (9, 11) As Object ' Двухмерный массив размером 10*12 элементов
Dim c () As Integer ' Динамический массив
```

Другим очень удобным нововведением является возможность инициализации массива при объявлении. После строки объявления нужно поставить оператор присваивания ‘=’, а после него в фигурных скобках через запятую перечислить значения элементов массива.

```
Dim a () As Integer = {1, 2, 3, 4, 5}
```

Таким же образом можно инициализировать и многомерные массивы. Только тогда нужно в объявлении сразу указать количество измерений массива.

```
Dim b(,) As String = {{"аа", "аб"}, {"ба", "бб"}}
```

Каждый класс может иметь свойства и методы. Если массив – это класс, то логично предположить, что у него есть свойства или методы, или и то и другое.

Начнем с методов. Подробно со структурой объектов .NET Framework можно ознакомиться в MSDN. Остановимся лишь на основных методах, предназначенных для сортировки и поиска.

Класс Array предлагает два метода поиска элементов: простой и двоичный. Для простого поиска используются методы IndexOf и LastIndexOf. IndexOf ищет первое вхождение указанного объекта, а LastIndexOf – последнее. Синтаксис у этих двух методов одинаков. Они перегружены. Это значит, что функция может принимать разные комбинации параметров. В Object Browser (открывается по нажатию F2) описаны 3 варианта метода IndexOf:

```
Public Shared Function IndexOf(ByVal array As System.Array, ByVal value
As Object) As Integer
Public Shared Function IndexOf(ByVal array As System.Array, ByVal value
As Object, ByVal startIndex As Integer) As Integer
Public Shared Function IndexOf(ByVal array As System.Array, ByVal value
As Object, ByVal startIndex As Integer, ByVal count As Integer) As
Integer
```

Первый вариант служит для поиска первого появления искомого элемента в целом массиве, второй – для поиска элементов в части массива, начиная с указанного индекса и до конца, третий – для поиска элемента в части массива, который начинается с указанного индекса и содержит в себе указанное число элементов.

При использовании IndexOf и LastIndexOf программа в цикле проверяет все элементы массива и сравнивает их с искомым объектом. Сравнение выполняется с помощью метода Equals искомого объекта. Он сравнивает 2 объекта и если они равны, то возвращает True, а в противном случае – False. Почти все объекты в .NET Framework имеют этот метод.

Метод возвращает индекс найденного элемента, если поиск увенчался успехом или -1, если ничего не найдено. Все сказанное относится и к методу LastIndexOf.

Примеры использования функций поиска:

```
Dim a() As String = {"a", "b", "c", "a", "b", "c"}
Console.WriteLine(System.Array.IndexOf(a, "b"))
Console.WriteLine(System.Array.IndexOf(a, "b", 2))
Console.WriteLine(System.Array.LastIndexOf(a, "b", 3, 3))
```

Двоичный поиск работает несколько иначе. Он значительно быстрее простого поиска, но может работать только с отсортированными массивами

Чтобы найти объект в массиве, BinarySearch сравнивает его с элементом, расположенным в середине массива. Если искомый объект оказывается меньше, значит он находится в первой половине массива, и во второй половине можно его не искать. А если искомый объект больше, то он располагается во второй половине массива. Далее процесс повторяется для той половины массива, в которой находится объект.

Если передать методу BinarySearch неотсортированный массив, то он все равно выполнит все положенные шаги и, скорее всего, сообщит, что элемент не найден, даже если он присутствует в массиве (возможно, элемент и будет найден, но это, естественно, будет случайность). Этот метод не проверяет порядок элементов, предполагая, что они отсортированы.

Сравнение элементов при двоичном поиске. Если в массиве хранятся объекты базовых типов, то строки сортируются в алфавитном порядке, сравнение чисел тоже вопросов не вызывает. А как быть, если в массиве хранятся другие объекты? Для того чтобы сортировка массивов, содержащих пользовательские объекты, была возможна, они должны реализовывать интерфейс IComparer. Нужно создать класс, наследующий интерфейс IComparer и реализовать в нём код сравнения двух объектов. Затем экземпляр этого класса передаётся методу BinarySearch массива.

Двоичный поиск осуществляется с помощью метода BinarySearch класса System.Array. Он перегружен и имеет 4 комбинации параметров. Минимальный набор параметров – массив и искомый объект. Также к этим параметрам можно добавить первый элемент и количество элементов области:

```
Public Shared Function BinarySearch(ByVal array As System.Array, ByVal
index As Integer, ByVal length As Integer, ByVal value As Object) As
Integer
```

Можно также передать методу BinarySearch объект, наследующий интерфейс IComparer, чтобы задать собственные параметры сравнения объектов при поиске.

```
Public Shared Function BinarySearch(ByVal array As System.Array, ByVal
index As Integer, ByVal length As Integer, ByVal value As Object, ByVal
comparer As System.Collections.IComparer) As Integer
Public Shared Function BinarySearch(ByVal array As System.Array, ByVal
value As Object, ByVal comparer As System.Collections.IComparer) As
Integer
```

Если объект найден, то метод BinarySearch вернет индекс элемента массива. А если искомый объект не найден, то метод вернёт отрицательное число. Если дополнить его до -1 (изменить знак и вычесть 1), то получится индекс первого из тех элементов массива, которые больше искомого объекта. Если все элементы массива меньше искомого, то получится индекс большего из них.

Пример использования метода BinarySearch:

```
Dim a() As String = {"a", "b", "c", "e", "f"}
Dim ind As Integer
ind = System.Array.BinarySearch(a, "d")
If ind >= 0 Then
  Console.WriteLine("Элемент найден: " & ind.ToString)
Else
  Console.WriteLine("Найдено ближайшее соответствие: " & (-ind -
1).ToString)
End If
```

Другая полезная возможность, которую предоставляет класс System.Array – сортировка массива. Она производится с помощью метода Sort. Он также перегружен. Самый простой вариант его применения выглядит так:

```
Public Shared Sub Sort(ByVal array As System.Array)
```

Он сортирует весь массив. Если необходимо отсортировать только часть массива, то к параметру array следует добавить индекс начального элемента и длину области, которую необходимо отсортировать.

Sort, в отличие от методов рассмотренных ранее, ничего не возвращает. Ниже приведен пример сортировки части массива.

```
Dim a() As String = {"c", "r", "z", "b", "d", "a"}
System.Array.Sort(a, 2, 4)
```

У метода Sort есть ещё одна интересная разновидность, позволяющая сортировать элементы одного массива в соответствии со значениями элементов другого массива.

```
Public Shared Sub Sort(ByVal keys As System.Array, ByVal items As
System.Array)
```

В первом параметре задаётся массив ключей. На основе элементов этого массива будет отсортирован массив, передаваемый во втором параметре. Допустим, у нас имеется массив со словами (words) и массив, в котором для каждого слова указана частота его употребления в тексте (freq). Используя первую форму метода Sort можно отсортировать слова по алфавиту. А с помощью последней рассмотренной формы метода мы можем отсортировать слова по частоте их использования. Для этого нужно первым параметром указать массив freq, а вторым – words.

```
System.Array.Sort (freq, words)
```

Такая сортировка имеет смысл, если длины массивов равны. Если длиннее будет массив items, то последние элементы, для которых нет соответствий в массиве keys, не будут учитываться при сортировке и останутся после вызова метода Sort на своих местах. Например, если в массиве keys 3 элемента, а в items – 5, то будут отсортированы только первые 3 элемента массива items, а четвёртый и пятый элементы не будут участвовать в сортировке и останутся на своих местах. Ну а если длиннее будет массив keys, то будет сгенерировано исключение System.ArgumentException.

ArrayList

Несмотря на всю полезность и удобство, массивы обладают рядом недостатков. Например, для того, чтобы удалить элемент из середины массива, нужно сдвинуть все элементы после него влево. А для добавления элемента в середину массива нужно сдвинуть элементы после него вправо. Для этого программисту приходится заниматься написанием довольно скучного кода, вместо того, чтобы решать свои основные задачи.

Но не все так плохо. В пространстве имен System.Collections существует класс ArrayList, который позволяет решить эти проблемы.

> *ArrayList – это список, в котором можно хранить нетипизированные данные (то есть, в одном объекте ArrayList могут храниться и строки, и числа, и даты, и прочие объекты).*

При объявлении списка нет необходимости задавать его размер. Кроме того, можное добавлять и удалять элементы из любого места списка, будь то начало, середина или конец.

Для создания нового объекта ArrayList используется следующий код:

```
Dim collname As New ArrayList()
```

После объявления можно приступить к работе. Новые элементы добавляются методом Add. Он добавляет объект в конец коллекции. В качестве параметра передается добавляемый объект. Метод возвращает индекс, присвоенный добавленному элементу.

```
Console.WriteLine (collname.Add ("test"))
```

Для добавления объекта в середину коллекции служит метод Insert. Ему передается индекс, который будет иметь добавляемый элемент и, собственно, сам объект.

```
collname.Insert (1, "abc")
```

Индекс, передаваемый методу Insert должен быть неотрицателен (index>=0) и меньше или равен размеру коллекции (index<=collname.Count). Все элементы, расположенные после добавляемого элемента будут сдвинуты вправо.

Чтобы удалить элемент коллекции, можно использовать метод RemoveAt. Ему передается индекс удаляемого элемента.

```
collname.RemoveAt (0)
```

Для получения элемента по его индексу используется индексированное свойство Item объекта ArrayList. С его помощью можно также устанавливать значения уже существующих элементов. Синтаксис свойства прост:

```
data = collname.Item (index) ' чтение элемента в переменную data
collname.Item (index) = data ' установка значения элемента
```

Получить количество элементов коллекции можно с помощью свойства Count.

```
Console.WriteLine (collname.Count)
```

Механизмы поиска и сортировки элементов ArrayList и массивов практически одинаковы. Стоит отметить только одну деталь: объект ArrayList не может быть отсортирован по значениям другого ArrayList.

Итак, с точки зрения удобства вставки/удаления элементов ArrayList удобнее и гибче массивов. Также не нужно заботиться о размерах коллекции – она словно резиновая! Впрочем, размеры коллекции все же можно контролировать. Объем памяти, отведенной для коллекции, задается свойством Capacity. Оно имеет тип Integer и устанавливает максимальное количество элементов, которое можно уместить в коллекцию. По умолчанию Capacity равно 16. Если количество элементов превышает значение, установленное в Capacity, то оно автоматически удваивается.

Если в ArrayList хранится много данных, то после заполнения коллекции стоит установить свойство Capacity равным количеству элементов для более экономичного использования оперативной памяти. Это можно сделать простым присваиванием свойства Count свойству Capacity. Свойство Capacity не может быть меньше свойства Count! Если установить Capacity меньше количества элементов, находящихся в коллекции, то будет сгенерировано исключение.

```
collname.Capacity = collname.Count
```

Можно воспользоваться специально для этого предназначенным методом TrimToSize. Он не имеет параметров. Действие метода TrimToSizeаналогично вышеуказанной строке кода.

Метод TrimToSize должен освободить некоторое количество оперативной памяти. Однако этого не происходит! А всё потому что приложения .NET по-особому используют оперативную память. В начале работы приложения .NET создаётся куча, состоящая из одного большого блока памяти. Когда сборка запрашивает у CLR объект, память выделяется из свободного блока в верхней части кучи, при этом среда даже не пытается заполнять пустые места, появившиеся на месте ранее освобождённых объектов. Когда весь свободный блок исчерпан, CLR приступает к сборке мусора (garbage collection). При сборке мусора все объекты, на которые нет ссылок, удаляются из памяти. Именно поэтому после вызова метода TrimToSize память не освобождается сразу – неиспользуемые объекты будет удалены из памяти после первой сборки мусора.

Хотя массивы и коллекции ArrayList отличаются по возможностям, обе структуры предназначены для хранения данных. Поэтому неудивительно, что в классе ArrayList предусмотрены механизмы для конвертирования коллекции в массив. Это делается с помощью метода ToArray. Он перегружен. Если объект ArrayList содержит данные одного типа, то вы можете указать в методе ToArray используемый тип данных – параметр типа System.Type. Тогда метод вернет массив указанного типа. Если же вы не укажете тип данных, то будет возвращен массив типа Object.

```
Dim a () As String
```

```
collname.ToArray (System.Type.GetType("System.String"))
```

Быстродействие

Как известно, за все нужно платить. За удобство и функциональность коллекций ArrayList также приходится платить.

По сравнению с массивами коллекции ArrayList работают гораздо медленнее и занимают больше оперативной памяти. Происходит это из-за особенностей коллекций, которые, собственно, и создают удобство их использование (возможность добавлять и удалять элементы из середины коллекции).

При создании массива его размер будет равен количеству элементов массива помноженному на размер каждого элемента. Например, для массива с 10 элементами типа Integer выделяется 10*4=40 байт. Затем для считывания элемента массива вычисляется его положение в памяти на основании номера элемента и размера элементов и затем считывается нужный участок памяти. С коллекциями всё иначе.

Каждый элемент коллекции представлен не одной переменной, содержащей его значение, а структурой данных. Она содержит указатель на данные элемента, ссылки на предыдущий и следующий элемент коллекции. Естественно, такая структура будет занимать больше пространства в оперативной памяти, чем одна единственная переменная. Доступ к данным в коллекции также организуется по-особому. Для получения, например, пятого элемента, коллекция перебирает все элементы, начиная с первого, пока не дойдёт до нужного индекса.

Проведем небольшие испытания для сравнения скорости работы с массивами и коллекциями ArrayList.

```
Private Sub ArrayAdd()
  Dim d1 As Date
  Dim d2 As Date
  Dim a(10000000) As Integer
  Dim i As Integer
  Dim d As New TimeSpan()

  d1 = DateTime.Now
  For i = 1 To 10000000
    a(i) = i
  Next
  d2 = DateTime.Now
  d = d2.Subtract(d1)
  Console.WriteLine(d.TotalMilliseconds)
End Sub

Private Sub ArrayListAdd()
  Dim d1 As Date
  Dim d2 As Date
  Dim a As New ArrayList(10000000)
  Dim i As Integer
  Dim d As New TimeSpan()

  d1 = DateTime.Now
  For i = 1 To 10000000
    a.Add(i)
  Next
  d2 = DateTime.Now
  d = d2.Subtract(d1)
  Console.WriteLine(d.TotalMilliseconds)
End Sub

Private Sub ArraySort()
```

```
  Dim d1 As Date
  Dim d2 As Date
  Dim s As System.Text.StringBuilder
  Dim a(10000) As String
  Dim i As Integer
  Dim j As Integer
  Dim d As New TimeSpan()

  For i = 1 To 10000
    s = New System.Text.StringBuilder(5)
    For j = 1 To 5
      Randomize()
      s = s.Append(Chr(CInt((Rnd() * 25) + 65)))
      a(i) = s.ToString
    Next
  Next
  d1 = DateTime.Now
  Array.Sort(a)
  d2 = DateTime.Now
  d = d2.Subtract(d1)
  Console.WriteLine(d.TotalMilliseconds)
End Sub

Private Sub ArrayListSort()
  Dim d1 As Date
  Dim d2 As Date
  Dim s As System.Text.StringBuilder
  Dim a As New ArrayList(10000)
  Dim i As Integer
  Dim j As Integer
  Dim d As New TimeSpan()

  For i = 1 To 10000
    s = New System.Text.StringBuilder(5)
    For j = 1 To 5
      Randomize()
      s = s.Append(Chr(CInt((Rnd() * 25) + 65)))
      a.Add(s.ToString)
    Next
  Next
  d1 = DateTime.Now
  a.Sort()
  d2 = DateTime.Now
  d = d2.Subtract(d1)
  Console.WriteLine(d.TotalMilliseconds)
End Sub

Private Sub ArrayGet()
  Dim d1 As Date
  Dim d2 As Date
  Dim a(10000000) As Integer
  Dim i As Integer
  Dim b As Integer
  Dim d As New TimeSpan()

  For i = 1 To 10000000
    a(i) = i
  Next
  d1 = DateTime.Now
  For i = 1 To 10000000
    b = a(i)
  Next
  d2 = DateTime.Now
  d = d2.Subtract(d1)
  Console.WriteLine(d.TotalMilliseconds)
End Sub

Private Sub ArrayListGet()
  Dim d1 As Date
```

```
  Dim d2 As Date
  Dim a As New ArrayList(10000000)
  Dim i As Integer
  Dim b As Integer
  Dim d As New TimeSpan()

  For i = 1 To 10000000
    a.Add(i)
  Next
  d1 = DateTime.Now
  For i = 1 To 10000000
    b = a(i - 1)
  Next
  d2 = DateTime.Now
  d = d2.Subtract(d1)
  Console.WriteLine(d.TotalMilliseconds)
End Sub
```

Таблица 1. – результаты тестирования

Тест	Массив	КоллекцияArrayList
Заполнение 10000000 элементов значениями типаInteger, мс.	215	31900
Получение значений 10000000 элементов типа Integer, мс.	340	3500
Сортировка 10000 элементов типа String, мс.	210	1200

Результаты впечатляют... Массивы работают быстрее коллекций ArrayList в десятки раз! Если приложение постоянно работает с большим объемом данных, стоит отказаться от коллекций ArrayList и использовать массивы.

Коллекция HashTable

Коллекцию ArrayList можно назвать усовершенствованной формой массива. К элементам ArrayList можно обращаться только по индексу. А это не всегда приемлемо.

В .NET Framework имеется коллекция HashTable, реализующая, как нетрудно догадаться, хэш-таблицу. Каждый элемент в этой коллекции имеет значение и ключ.

> *Ключ – это уникальный идентификатор, предназначенный для доступа к элементу. Он может быть любого типа, будь то String, Integer, Objectили любой другой объект.*

Класс HashTable очень похож на ArrayList, но имеет ряд серьезных отличий. Стоит отметить отсутствие у класса HashTable свойства Capacity.

Работа с коллекцией начинается с создания нового объекта HashTable. Это делается с помощью оператора New.

```
Dim col As New HashTable ()
```

Для добавления нового элемента в коллекцию используется метод Add. Ему передаются ключ и собственно добавляемый объект.

```
Dim orders As New HashTable ()
orders.Add ("Moscow", 8)
orders.Add ("Tomsk", 5)
```

При добавлении элемента необходимо помнить, что ключи должны быть уникальными. Если попытаться использовать уже имеющийся ключ, то будет сгенерировано исключение.

Узнать, имеется ли в коллекции некоторый ключ, можно с помощью метода ContainsKey. Ему передается проверяемый ключ. Метод возвратит True, если ключ существует и False в обратном случае. Также существует метод ContainsValue. Он позволяет узнать, имеется ли в коллекции значение и используется так же, как и ContainsKey.

```
If col.ContainsKey ("abc") = True Then
  Console.WriteLine ("Ключ 'abc' имеется в коллекции")
End If
```

Удаление элементов из коллекции HashTable осуществляет метод Remove. Ему следует передать ключ удаляемого элемента.

Другие коллекции

Кроме рассмотренных выше классов ArrayList и HashTable в пространстве имен System.Collections имеются еще несколько коллекций. Они не сильно отличаются от рассмотренных выше коллекций.

> *Коллекция SortedList представляет собой нечто вроде гибрида HashTable и ArrayList.*

К элементам коллекции можно обращаться как по индексам, так и по ключам. Более того, эта коллекция всегда отсортирована по ключам. Метода для сортировки коллекции по значениям не существует.

Еще две интересных коллекции – Queue и Stack. Они отличаются лишь способом добавления и удаления элементов. Класс Stack добавляет элементы в конец коллекции и удаляет всегда последний элемент.

> *Коллекция Stack реализует структуру LIFO (last in first out – последний вошел, первый вышел).*

> *Класс Queue добавляет элементы в конец коллекции и удаляет только первый элемент. Это очередь или структура FIFO (first in first out – первый вошел, первый вышел).*

И ещё одна коллекция, которая находится в пространстве имён System.Collections – BitArray.

> *Коллекция BitArray предназначена для хранения битов (значений типа Boolean).*

Стоит подчеркнуть один интересный момент: в этом классе содержатся методы, реализующие поэлементные логические операции XOR, OR,AND и NOT.

Заключение

Были рассмотрены базовые возможности массивов и коллекций. Microsoft .NET предлагает программистам богатые возможности хранения и обработки наборов данных.

Массивы стали намного удобнее и функциональнее. А богатство коллекций .NET ни в какое сравнение не идет с единственным классом Collection в VB6. Практически для любых целей можно подобрать подходящую коллекцию или создать собственную на базе имеющихся.

Абстрактные базовые классы

На стадии проектирования наследственных связей в программе часто выясняется, что многие классы обладают целым рядом сходных черт. Например, внештатные сотрудники не относятся к постоянным работникам, но и те и другие обладают рядом общих атрибутов – именем, адресом, кодом налогоплательщика и т. д.

Было бы логично выделить все общие атрибуты в базовый класс PayableEntity. Этот прием, называемый факторингом, часто используется при проектировании классов и позволяет довести абстракцию до ее логического завершения.

В классах, полученных в результате факторинга, некоторые методы и свойства невозможно реализовать, поскольку они являются общими для всех классов в иерархии наследования. Например, класс PayableEntity, от которого создаются производные классы штатных и внештатных работников, может содержать свойство с именем TaxID.

Обычно в процедуре этого свойства следовало бы организовать проверку кода налогоплательщика, но для некоторых категорий внештатных работников эти коды имеют особый формат. Следовательно, проверка этого свойства должна быть реализована не в базовом классе PayableEntity, а в производных классах, поскольку лишь они знают, как должен выглядеть правильный код.

В таких ситуациях обычно определяется абстрактный базовый класс.

> *Абстрактным называется класс, содержащий хотя бы одну функцию с ключевым словом MustOverride; при этом сам класс помечается ключевым словом Mustlnherit.*

Ниже показано, как может выглядеть абстрактный класс PayableEntity:

```
Public Mustlnherit Class PayableEntity
  Private m_Name As String
  Public Sub New(ByVal itsName As String)
    m_Name = itsName
  End Sub
  Readonly Property TheName()As String
    Get
      Return m_Name
    End Get
  End Property
Public MustOverride Property TaxID()As String
End Class
```

Свойство TaxID, помеченное ключевым словом MustOverride, только объявляется без фактической реализации. Члены классов, помеченные ключевым словом MustOverride, состоят из одних заголовков и не содержат команд End Property, End Sub и End Function. Доступное только для чтения свойство TheName при этом реализовано; из этого следует, что абстрактные классы могут содержать как абстрактные, так и реализованные члены. Ниже приведен пример класса Employee, производного от абстрактного класса PayableEntity:

```
Public Class Employee
  Inherits PayableEntity
  Private m_Salary As Decimal
  Private m_TaxID As String
  Private Const LIMIT As Decimal = 0.1D

  Public Sub New CByVal theName As String, ByVal curSalary As Decimal.
ByVal TaxID As String) MyBase.New(theName)
m_Salary = curSalary
```

```
m_TaxID = TaxID End Sub
Public Overrides Property TaxID() As String Get
Return m_TaxID
End Get
Set(ByVal Value As String)
If Value.Length <> 11 then
' См. главу 7 Else
m_TaxID = Value
End If
End Set
End Property
Readonly Property Salary() As Decimal Get
Return MyClass.m_Salary
End Get
End Property
Public Overridable Overloads Sub RaiseSalary(ByVal Percent As Decimal)
If Percent > LIMIT Then
' Операция запрещена - необходим пароль
Console.WriteLineC'NEED PASSWORD TO RAISE SALARY MORE " & _
"THAN LIMIT!!!!") Else
m_Salary =(1D + Percent) * m_Salary
End If
End Sub
Public Overridable Overloads Sub RaiseSalary(ByVal Percent As
Decimal. ByVal Password As String) If Password ="special" Then
m_Salary MID + Percent) * m_Salary
End If
End Sub
End Class
```

Первая ключевая строка расположена внутри конструктора, который теперь должен вызывать конструктор абстрактного базового класса для того, чтобы правильно задать имя.

Во втором выделенном фрагменте определяется элементарная реализация для свойства Taxld, объявленного с ключевым словом MustOverride (в приведенном примере новое значение свойства не проверяется, как следовало бы сделать в практическом примере).

Ниже приведена процедура Sub Main, предназначенная для тестирования этой программы:

```
Sub Main()
Dim tom As New Employee("Tom". 50000. "111-11-1234")
Dim sally As New Programmed "Sally", 150000. "111-11-2234".)
Console.Wri teLi ne(sa1ly.TheName)
Dim ourEmployees(l) As Employee
ourEmployees(0) = tom
ourEmployees(l) = sally
Dim anEmployee As Employee
For Each anEmployee In ourEmployees anEmployee.RaiseSalary(0.lD)
Console.WriteLine(anEmployee.TheName & "has tax id " & _
anEmployee.TaxID & ".salary now is " & anEmployee.Salary())
Next
Consol e.ReadLine() End Sub
```

В программе невозможно создать экземпляр класса, объявленного с ключевым словом Mustlnherit. Например, при попытке выполнения следующей команды:

```
Dim NoGood As New PayableEntity("can't do")
```

компилятор выводит сообщение об ошибке:

```
Class 'PayableEntity' is not creatable because it contains at least one
member marked as 'MustOverride' that hasn't been overridden.
```

Тем не менее объект производного класса можно присвоить переменной или контейнеру абстрактного базового класса, что дает возможность использовать в программе полиморфные вызовы:

```
Dim torn As New Employee("Tom". 50000, "123-45-6789")
Dim whoToPay(13) As PayableEntity whoToPay(0) = tom
```

Теоретически класс Mustlnherit может не содержать ни одного члена с ключевым словом MustOverride (хотя это будет выглядеть несколько странно).

CollectionBase

При использовании классов коллекций .NET Framework (таких, как ArrayList и HashTable) возникает неожиданная проблема: эти классы предназначены для хранения обобщенного типа Object, поэтому прочитанные из них объекты всегда приходится преобразовывать к исходному типу функцией CType.

Также возникает опасность того, что кто-нибудь сохранит в контейнере объект другого типа и попытка вызова CType завершится неудачей.

Проблема решается использованием коллекций с сильной типизацией — контейнеров, позволяющих хранить объекты конкретного типа и типов, производных от него.

Хорошим примером абстрактного базового класса .NET Framework является класс CollectionBase. Классы, производные от CollectionBase, используются для построения коллекций с сильной типизацией (прежде чем создавать собственные классы коллекций, производные от CollectionBase, убедитесь в том, что нужные классы отсутствуют в пространстве имен System.Collections.Specialized).

Коллекции, безопасные по отношению к типам, строятся на основе абстрактного базового класса System.Collections.CollectionBase; требуется лишь реализовать методы Add и Remove, а также свойство Item. Хранение данных во внутреннем списке реализовано на уровне класса System.Collections.CollectionBase, который и выполняет все остальные операции.

Рассмотрим пример создания специализированных коллекций (предполагается, что проект содержит класс Employee или ссылку на него):

```
1 Public Class Employees
2 Inherits System.Collections.CollectionBase
3 ' Метод Add включает в коллекцию только объекты класса Employee.
4 ' Вызов перепоручается методу Add внутреннего объекта List.
5 Public Sub AddtByVal aEmployee As Employee)
6 List.Add(aEmployee)
7 End Sub
8 Public Sub Remove(ByVal index As Integer)
9 If index > Count-1 Or index < 0 Then
10 ' Индекс за границами интервала, инициировать исключение (глава 7)
11 MsgBox("Can't add this item")' MsgBox условно заменяет исключение
12 Else
13 List.RemoveAt(index)
14 End If
15 End Sub
16
17  Default  Public  Readonly  Property  Item(ByVal  index  As  Integer)As
Employee
18 Get
19 Return CType(List.Item(index). Employee)
20 End Get
21 End Property
22 End Class
```

В строках 5-7 абстрактный метод Add базового класса реализуется передачей вызова внутреннему объекту List; метод принимает для включения в коллекцию только объекты Employee. В строках 8-10 реализован метод Remove. На этот раз мы также используем свойство Count внутреннего объекта List, чтобы убедиться в том, что удаляемый объект не находится перед началом или после конца списка. Наконец, свойство Item реализуется в строках 17-21. Оно объявляется свойством по умолчанию, поскольку пользователи обычно ожидают от коллекций именно такого поведения. Свойство объявляется доступным только для чтения, чтобы добавление новых элементов в коллекцию могло осуществляться только методом Add. Конечно, свойство можно было объявить и

доступным для чтения/записи, но тогда потребовался бы дополнительный код для проверки индекса добавляемого элемента.

Следующий фрагмент проверяет работу специализированной коллекции; недопустимая операция включения нового элемента закомментирована:

```
Sub Main()
Dim torn As New Employee("Tom", 50000)
Dim sally As New Employee("Sally", 60000)
Dim myEmployees As New Employees()
myEmployees.Add(tom)
myEmployees.Add(sally)
' myEmployees.Add("Tom")
Dim aEmployee As Employee
For Each aEmployee In myEmployees
Console.WriteLine(aEmployee.TheName)
Next
Console. ReadLine()
End Sub
```

Если убрать комментарий из строки myEmployees.Add("Tom"). Программа перестанет компилироваться, и возникнет следующее сообщение об ошибке:

```
C:\book to comp \chapter 5\EmployeesClass\EmployeesClass\Modulel.vb(9):
A value of type 'String'cannot be converted to
'EmployeesClass.Employee'.
```

Перед вами замечательный пример того, какими преимуществами VB .NET обладает перед включением в прежних версиях VB. Конечно, мы продолжаем перепоручать вызовы внутреннему объекту, чтобы избавиться от дополнительной работы, но возможность перебора элементов в цикле For-Each появляется автоматически, поскольку наш класс является производным от класса с поддержкой For-Each!

Сериализация. Десериализация

> *Сериализация (в программировании) – процесс перевода какой-либо структуры данных в последовательность битов. Обратной к операции сериализации является операция десериализации (структуризации) – восстановление начального состояния структуры данных из битовой последовательности.*

Сериализация используется для передачи объектов по сети и для сохранения их в файлы. Например, нужно создать распределённое приложение, разные части которого должны обмениваться данными со сложной структурой. В таком случае для типов данных, которые предполагается передавать, пишется код, который осуществляет сериализацию и десериализацию. Объект заполняется нужными данными, затем вызывается код сериализации, в результате получается, например, XML-документ. Результат сериализации передаётся принимающей стороне, например, по электронной почте или HTTP. Приложение-получатель создаёт объект того же типа и вызывает код десериализации, в результате получая объект с теми же данными, что были в объекте приложения-отправителя. По такой схеме работает, например, сериализация объектов через SOAP в Microsoft .NET.

Пространство имен System.Runtime.Serialization содержит классы, которые можно использовать для сериализации и десериализации объектов.

> *Сериализация – это процесс преобразования объекта или графа объектов в линейную последовательность байтов для сохранения или передачи в другое расположение.*

> *Десериализация — это процесс получения сохраненных данных и восстановления из них объектов.*

Интерфейс ISerializable предоставляет классам возможность управлять своим поведением при сериализации. Классы в пространстве имен System.Runtime.Serialization.Formatters управляют фактическим форматированием различных типов данных, инкапсулированных в сериализованные объекты.

Три кита ООП (Объектно-ориентированное программирование)

Пролог

На заре компьютерной эры, когда польза от их использования обусловливалась прежде всего промышленными интересами, вдруг назрела потребность в "очеловечивании" и в то же время укреплении языка, с помощью которого человеку суждено было общаться с машиной. Языков программирования уже тогда было более чем предостаточно, однако такое же несметное количество не годилось для решения критических задач, в которых не было места грубым ошибкам и недоработкам. В ту пору уже доминировал Си. Вскоре на смену ему пришел Си++, - более изощренный и мощный, нежели первичное детище Карнигана и Ритчи. Наряду со многими нововведениями Си программисту в руки было вложено новое средство - Объект. Это резко упростило некоторые вещи (стоп!, не некоторые) - в тот период все перевернулось с ног на голову, - программист, оперирующий аморфными понятиями, теоретическими "субъектами" и прочими персонажами из сказки о clsMyObj, решал те же задачи, что и программист, не решившийся бросить процедурный (последовательно-алгоритмический - устаревш. информ.) язык, выполнял свою работу качественней, быстрей: и, несомненно, с бОльшим удовольствием и с меньшими услиями - один из попутных "козырьков" ООП - использование готового кода.

Все так, однако бытует легенда, будто Объекты зародились еще задолго до появления первого языка программирования. Ну: не будем ломать голову над тем, что в данной статье как раз и не принципиально, (язык Си взят как пример по причине безоговорочного господства на поприще низкоуровневых языков и по сей день) - оставим сии измышления археологам, и признаем лишь тот факт, что постепенно все, даже самые молодые и "вторичные", детские (Basic) и студенческие (Pascal) языки программирования стали обзаводиться своей объектной стратегией. И хотя некоторые только имитировали ООП, тенденция все же дошла до наших дней. До сих пор на пике популярности книги по Объектно-Ориентированному программированию, причем не применительно к какому-либо конкретному языку, а освещающие принципы, идеологию и образ "объектного мышления" Сами ОО-специалисты называют это "ОО-философией". По своей природе и идее ООП не привязывается к конкретному языку - это принцип построения программы, ее структурная схема. Это - глубже, чем можно описать в рамках сей публикации, потому как затрагивает не только лишь период проектирования модели приложения, его концепции и схемы, - образа, если хотите: Это - поистине образ мышления: * * *

ООП подразумевает в первую очередь наличие классов. Классы - это как-будто самый мелкий элемент той структуры, которая и формирует Объектно-Ориентированную концепцию в целом. Каждый класс имеет свои свойства, свои методы свои события. Непременным свойством истинного ОО-языка является инкапсуляция, что означает "закупоренность" механизма того или иного явления в Объектах.

Вообще, инкапсуляция основана на области видимости (Scope) внутренних переменных Класса. Таким образом программист зачастую использует объекты, созданные другими программистами, и абсолютно не задумывается, как все это устроено, он просто доверяет "производителю объекта" и всецело занят лишь решением задачи, поставленной конкретно ему. Яркий пример такого подхода к созданию приложений при помощи инкапсулированных механизмов - элементы управления (ActiveX-компоненты), которые, к стати, можно использовать во всех поддерживающих ActiveX-технологию языках программирования. Одним словом, вы добавляете в проект текстовое поле, не задумываясь о том, как реализована, например, прорисовка его текста.

"Однажды созданные, они живут своей жизнью, размножаются:" - из книги дотошного ОО-следопыта и исследователя производительности ПО.

Говорят, ООП держится на трех "китах". В таком случае первым является инкапсуляция, поскольку оказалась бы нецелесообразной ОО-система вообще, не будь инкапсулированных методов, свойств, спрятанных за ненадобностью имитаций событий: Однако все это отнюдь не значит, что программист не в силах вникнуть в систему того или иного объекта - если речь идет не о готовых коммерческих компонентах, скомпилированных и защищенных от декомпиляции, а об Объектах и Классах, созданных или включенных в проект в качестве равноправного кода, программист имеет к нему такой же доступ, как и к любому другому модулю.

Второй "кит" - наследование. Что это значит? Это значит, что Объекты (Классы, Коллекции) могут перенимать некоторые свойства у своих прародителей. Как - это зависит от того языка, на котором пишется программа. Однако в любом случае картина та же: это приводит к повторному использованию уже написанного однажды кода. Наследование-то по определению заставляет что-то у кого-то наследовать, значит, можно создать свое текстовое поле на основе уже существующего класса TextBox (в Visual Basic), причем новый Класс (назовем его EnhancedBox) наделен всем тем, чем располагает его стандартный родитель, плюс новыми свойствами, определяемыми его создателем. Никуда не денутся свойства Font, Alignment, Multiline, - если их специально не "ампутировать". На основе наследования, - пусть даже искусственного, - в Visual Basic выросла и техника Субклассинга (Subclassing), при которой компоненты наделяются новыми свойствами. Чаще термин употребляется применительно к элементам управления. На рисунке показан фрагмент работы с ClassBuilder. В текстовое поле необходимо ввести имя класса-родителя, или же оставить Based on: (New class). Я выбрал первичный класс cLink, поэтому новый Класс cLoadedLink унаследовал все его "аксессуары". О ClassBuilder читайте ниже.

Третий "кит" ООП - полиморфизм. Вообще, это уже из области искусственного интеллекта J. В данном случае речь идет о той роскоши, за которую стоит выдерживать и немногие издержки ООП (однако же выигрыш очевиден и бесспорен!). Объекты, располагающие

одноименными методами или свойствами, могут с легкостью управляться в ходе программы, независимо от того, что эти одноименные свойства и методы выполняют абсолютно разные действия, в отношении абсолютно разных классов, да и устроены по-разному. Например, возьмем свойство Font, широко распространенное во многих компонентах:

```
Private Sub Command1_Click()
   Dim Ctl As Control
   For Each Ctl In Me.Controls
       Ctl.FontName = "Courier"
   Next
End Sub
```

Таким образом, мы избежали тщательного учета каждого элемента управления на нашей форме, и упомянули сразу всех при помощи системы For Each:In : Next (Для Каждого: В_Коллекции: Стоп). Конечно, ничто не мешает поступить по-иному. Это - проще: Но при условии, что на форму динамически не добавляются компоненты.

Впрочем, указанная конструкция - явный атрибут ООП, поскольку For Each работает лишь с коллекциями. Коллекции - это своеобразные классы, являющиеся учетными наборами других классов. Чем удобны коллекции? Прежде всего методами Add, Remove, свойствами Item. Используя наряду с другими стандартными для классов методами Add и Remove, программист имеет возможность пополнять коллекцию, удалять из нее определенные с помощью Item(Index) элементы. Любая коллекция имеет метод Clear.

Допустим, мы создали собственный Интернет-загрузчик. Это повлечет за собой работу со списками гиперссылок, их хранением и управлением.

Для того, чтобы программа оказалась эффективной и более-менее компактной, необходимо избавить ее от излишней конкретики, во всяком случае, там, где это возможно. Например, гиперссылка может вести как к графическому ресурсу, так и к обычной гипертекстовой странице: а может это запрос к удаленной базе данных?. Поэтому целесообразно было бы создать одноименные методы для обоих случаев - DownLoad, Save и так далее. Тогда великолепен код:

```
For Each cLink In cLinks
     cLink.DownLoad
     cLink.Save
Next
cLinks.Clear
```

Теперь можно забыть создание Классов cLink и cLinks как кошмарный сон. Впереди -лишь удовольствие от работы с "запечатанными" методами/свойствами /событиями:

Это и есть полиморфизм в примитиве. Почитатели ООП уже не мыслят себе процесса написания программы без Классов, свойств, методов и событий. Бытует также понятие Объектно-Ориентированного Дизайна (ООД).

Природа Свойств, Методов и Событий. Перечислимые типы

Механизм Классов в Бейсике реализован весьма просто и не должен вызывать особых затруднений. В модуле класса для каждого свойства объявляется локальная (для Класса) переменная-посредник в схеме Класс--пользователь. Присвоение свойству Объекта нового значения или считывание его - не более чем общение с этой переменной.

Не все так сложно, уважаемый читатель. Как-то в цикле "Мышление в стиле Visual Basic" была затронута тема окна сообщения - проверенного фундамента, на котором нетрудно построить общие представления неподготовленного читателя о функциях, процедурах, аргументах. Там же была рассмотрена способность оболочки Бейсика к автозавершению кода. Действительно, вследствие того, что код Бейсика компилируется еще на стадии набора его с клавиатуры (!), те или иные лексемы могут быть распознаны VB, или наоборот - ждешь всплывающих "тултипов" - минуту, две (шутка), а их все нет и нет. Помните: Бейсик - для "непрограммистов". Visual C++ - наоборот, поэтому там подсказок ждать не приходится.

Перед тем, как сесть и написать первую строку кода, ООП-почитатель тщательно продумывает структуру приложения, причем даже сидя с карандашом и бумагой, - не за клавиатурой. Когда он, наконец, примет решение, что такие-то методы должны быть одноименными (из вышеперечисленных соображений), такие-то объекты должны входить в такие-то коллекции/наборы, а такие коллекции - в другие коллекции, и когда придет к выводу, что система совершенна, - начнет проект!

Объектно-Ориентированный подход немыслим без UDT - User Defined Types, - пользовательского типа данных. Объявив такой тип в своей программе, можно объявлять и переменные этого типа. Процедура Type "лепит" из встроенных типов Бейсика то, что нужно пользователю для создания другого, нового типа. Так, естественен код в ОО-приложении:

```
Public Type HyperLink
    Address As String
    Port As Long
    ContentType As String
End Type
Dim MyLink As HyperLink
```

При этом MyLink уже имеет необходимые нам свойства веб-ссылки: адрес, порт, тип данных, и так далее.

Итак, если в проекте Visual Basic правильно объявлены все Типы, Классы, Коллекции, IDE VB предлагает свои варианты кода в виде выпадающих вариаций. Очевидно, что такие действия со стороны среды разработки заметно упрощают программирование: в большинстве случаев можно не знать досконально тот или иной компонент и действовать интуитивно.

Visual Basic располагает также таким средством как перечислимые типы данных. К ним можно отнести, например, кнопки в окне сообщения:

vbOkOnly, vbYesNoCancel. Каждый раз при указании процедуры или функции MsgBox на экране появляется соответствующий список возможных значений. Кроме того, Классы, Коллекции, их свойства, методы и события, а также константы, объявленные в Классе проекта наглядно представлены в Броузере Объектов.

Подключение к проекту библиотек типов (*.tlb) и динамически подключаемых библиотек (*.dll) позволяет "достучаться" к Объектам и типам других приложений. Они-то и выполняют большую часть черной работы по объявлениям и типам. Отыскав в диалоговом окне References (кнопка Browse) библиотеку типов, пользователь пополняет содержимое Броузера Объектов - чьи-то Классы полностью готовы к употреблению.

Поистине к глобальным последствиям привело появление OLE, COM - Component Object Model, DCOM - Distributed Component Object Model, позволяющие, грубо говоря, относительно свободно обмениваться информацией между приложениями. Например, двусторонне обмениваться компонентами с Internet Explorer, MS Word, и другими серверами. Информация о доступных Классах находится в Системном Реестре в ключе HK_CLASSES_ROOT\CLSID.

Создание Класса или Коллекции в среде Visual Basic может осуществляться как непосредственно через написание всего необходимого кода, так и с использованием надстроек, например, ClassBuilder`a.

В открытом проекте:

Меню Project--Add Class Module

Выбор элемента списка ClassBuilder приведет к появлению Построителя Классов. С помощью этого мастера в визуальном режиме можно за пятнадцать минут создать обширнейшую иерархию классов, наборов (коллекций), создать для каждого класса свойства, события и методы, однако ClassBuilder именует переменные и присваивает им тип данных не совсем так, как хотелось бы, поэтому ручная работа неизбежна. Впрочем, не только по этой причине J.

Однако при всех удобствах Построителя Классов это чудесное средство все же рекомендуется в большей степени начинающим ООП-исследователям Visual Basic 5, 6. Так называемые гуру ООП предпочитают порождать Объекты "руцями": компактно, лаконично, строго. Седьмая версия многострадального Васика, судя по анонсам Microsoft, "заставит пересмотреть познания этого языка".

В среде разработки Visual Basic имеется еще великолепное творение, предназначенное помочь в "разборках" с Объектами, Коллекциями, Константами, объявленных в этих Классах и всем, что с ними связано - Броузер Объектов.

Нажав F2, программист может воочию убедиться в правильности созданной им иерархии Классов. Сопутствующие подсказки и система элементарного поиска - вот что нужно тому, кто здесь впервые. Однако

ObjectBrowser используется отнюдь не только в целях изучения собственных творений: создав Reference ("зависимость") к чужим распределенным (Distributed) Объектам, можно изучать чужие Классы, наблюдать, как они устроены, а затем грамотно использовать.

Рисунок (***) изображает подключенную к обычному проекту Объектную модель Microsoft Word 8 (меню Project--References, в появившемся диалоге выбираем Microsoft Word 8 Type Library). Таким образом, мой проект имеет полную власть над создаваемым документом чужого программного продукта, причем я не имею ни малейшего понятия, как происходит добавления стиля в Normal.dot, я также не знаком с форматами файлов doc и dot, однако это не мешает мне и другим программистам читать и создавать "доки". Вспомните, например, экспорт в формат Microsoft Word из FineReader компании ABBYY (хотя здесь и основная нагрузка ложится на технологию OLE, программист все же видит внешнее приложение как "Объект", наделенный своими свойствами, событиями и методами):

Для одних языков программирования ООП - родная среда, в то же время другие, - такие как Visual Basic, - лишь имитируют ООП, в частности, такое понятие, как полиморфизм. Что касается будущего Бейсика - ждем следующей версии (в составе Visual Studio .NET), где специалистам-Бейсиковцам придется заново проходить курс молодого бойца - осваивать VB .NET (7), так как изменений хоть не так много, однако они есть, причем радикальные. Итак, по данным Microsoft, VB .NET - "нормальный Объектно-Ориентированный язык программирования".

Извечная классовая борьба Си++ и Визуального Бейсика

Начну, пожалуй, с того, что никогда ни Бейсик, ни Visual Basic ни с кем не соперничали, - сам по себе первозданный Basic - мягко говоря, не такое уж мощное средство. Однако, облаченный в оболочку (IDE), с API через плечо, с ActiveX-совместимостью - через другое и достаточно "закупоренный" для посредственного круга пользователей, позволяет смело соперничать с Delphi (из той же оперы: сначала повзрослел Паскаль до статуса Объектного, затем - признание), Visual C++ - молниеносно создавать сложнейшие программы для Windows. Runtime-файлы msvbvm*0.dll содержат стандартный ассортимент для UI, - в виде экземпляров классов, присутствующих сейчас в системе. Поэтому голова разработчика на VB болит гораздо меньше и реже по поводу реализации пользовательского интерфейса, нежели у программиста на C++. (Будем говорить так: вообще не болит). И пусть программист на C++ бьет себя в грудь: мол, его язык хоть сложнее, зато мощнее, быстрее, компактнее и: ОС Windows - продукт C++, поэтому творить для Окон нужно на Си++. Однако, уважаемый читатель, следует запомнить: скорость в работе Си-программ почти не отличается от скорости современного (v.5, 6) Бейсик-софта (для того, чтобы заметить разницу, необходимо писать специальные счетчики-секундомеры).

Говорят, на Си пишут автономные приложения (это те, которые вмещаются на 3,5"-диски и работают без DLL), однако теперь это редкость - подавляющее большинство ПО, написанного на солидных языках все равно обращаются к библиотекам. Потому что программирование для графической среды - а именно Windows - без готовых классов, стандартных графических компонентов, без обращений к Реестру - настоящая преисподня. Кроме того, единство стиля - часть успеха. Скажу больше: VB .NET восполнит прорехи в этой области - теперь создание консольного приложения на Бейсике больше не "трюк" и выполняется примерно в три-четыре строки кода.

Итак, Microsoft Visual C++ позволяет создавать приложения, используя готовые продукты. Среди них не последнее место по "употребляемости" занимает библиотека Microsoft Foundation Classes (MFC).

MFC, по сути, является аналогом готовых компонентов, огромнейшим количеством которых, например, обладает Delphi, Borland C++ благодаря VCL (Visual Component Library) - различия могут состоять лишь в визуализации тех или иных классов, но факт остается фактом: и те и другие библиотеки являются полноценным законченным продуктом. Если уважаемый читатель не вздумал сочинять PictureBox`ы, Кнопки и Списки с нуля - самое время использовать чьи-то классы в полный рост.

Не стану рассказывать о роли классов в C++, - вне сомнения, что C++ является сугубо Объектно-Ориентированным языком программирования, и это уже знают сегодняшние школьники. С другой стороны, Visual Basic 5 и выше - имитатор ООП. (Помнится, ранее это был даже не компилятор - другими словами, "имитатор исполняемого кода"). Да, Бейсик до сих пор БЫЛ почти ОО-языком. Почти? Дело во втором немного странном "ките" Бейсика девяностых: при создании Классов в VB 5, 6, якобы наследующих признаки родителей, на самом деле по наследству передаются лишь родительские интерфейсы. Однако все здесь, опять-таки, весьма относительно: поскольку ООП является лишь тем ментальным аспектом процесса проектирования, так сказать условностью, абсолютно не влияющей на компилируемый код, за исключением лишь компактности и удобства для программиста (можно даже допустить - синтаксическим и структурно-лингвистическим расширением), а также практически не ведет к явному ускорению кода, стоит ли грешить на Бейсик как на лже-Объектный инструмент?

Эпилог

Определенный вклад в развитие ООП внес, конечно же, каждый из существующих и успешно применяемых сегодня языков; каждый по-своему популяризовал ОО-схему проектирования программного обеспечения, показал, насколько упрощаются некоторые задачи, а иногда - просто становятся выполнимыми. Однако техника ОО до сих пор на стадии изучения - существуют целые институты, исследующие концепцию в целом, существуют и отдельные деятели движения. Кроме того, идея ООП и ООД порождает новые идеи и течения.

Более подробно о принципах ООП в среде Visual Basic 5, 6 читайте в цикле "Мышление в стиле Visual Basic".

Классы - это просто

Михаил Эскин, VBline

Модули классов хранятся в файлах с расширением CLS и похожи на модули форм за исключением того, что у них нет видимого интерфейса пользователя. Их можно использовать для создания собственных объектов, включая программный код для методов и свойств. *(Microsoft Press)*

На основе этого проекта вы познакомитесь созданием коллекций и классов, а также сможете грамотно их использовать в своих программах.

Шаг 1. Создадим проект Standard EXE. Та форма, которая создается по умолчанию, нам пригодится в будущем, как тестировочная. Поэтому внесем в нее маленькие исправления: переименуем ее в frmTest, свойство AutoRedraw = True (так как мы будем выводить результаты методом Print на саму форму). Изменим также шрифт, установим его равным "Courier New Cyr" и кегль = 10. Внизу формы расположим кнопку:

```
Name= cmdPrint
Caption="Print"
```

Шаг 2. Теперь займемся непосредственно созданием нашего класса. Откройте Class Builder Utility. Сделать это можно несколькими способами: Меню Project/Add Class Module и в диалоговом окне выберите VB Class Builder; или Меню Add-Ins/Add-In Manager, выберите VB6 Class Builder Utility, нажмите OK и еще раз Меню Add-Ins/Class Builder Utility. Представленный Add-In, значительно облегчит вам жизнь при построении классов и коллекций, избавив вас от рутинной работы - написания однотипных кодов. Прежде чем мы начнем строить, давайте определимся со структурой нашего класса. Во-первых, мы должны иметь возможность добавлять и удалять записи, знать их количество в коллекции и иметь возможность обращаться к каждой конкретной записи для чтения или изменения ее. Для этого в коллекции существуют стандартные свойства и методы, которые будут сгенерированы нам автоматически. Каждая запись (в нашем примере) будет состоять из трех полей: Player (строковый тип), CurDate (тип дата/время), Result (тип длинное целое).

NB! В дальнейшем Вы, поняв правила построения классов, можете добавить или убрать дополнительные поля, или изменить их типы.

Шаг 3. Итак, строим. В визарде выберем меню File/New/Collection и в диалоговом окне в поле Name изменим имя, данное по умолчанию на Resultats. Справа определим, на каком классе будет базироваться данная коллекция, выберем New Class и в поле переназовем его на Resultat. Нажмем кнопку OK. Если мы сейчас выделим коллекцию, то увидим, что автоматически созданы следующие методы и свойства: Add, Remove, Item, Count, NewEnum. В окне слева щелкнем на крестике, раскрывая коллекцию - там будет находиться класс, на котором данная коллекция базируется, то есть Resultat. Добавим в него наши свойства (Player,

CurDate, Result), с вышеуказанными типами. Закроем данное приложение, не забыв подтвердить Update нашего проекта.

Шаг 4. Займемся исследованием и исправлением полученного с помощью визарда. В классе, помимо заказанных нами свойств, автоматически создается открытая переменная Key. Толку от открытых переменных, как мы знаем, не много, кроме того, в данной ситуации мы это свойство и не заказывали. Поэтому просто-напросто удалим данную строку. Каждое свойство, созданное визардом по нашей просьбе, состоит из двух частей: Let (на запись) и Get (на чтение). Этим самым обуславливается закрытость доступа к нашим свойствам.

NB! Если бы мы использовали в качестве одного из свойств объект, то вместо процедуры Let необходимо было бы использовать процедуру Set.

Переходим в коллекцию. Открытая переменная Key там завязана только в одном методе, а именно методе Add. Удалим ее из объявления в заглавии метода. Кроме того, удаляем строку objNewMember.Key=Key, и вместо цикла If Then : Else запишем всего одну строку: mCol.Add objNewMember. Вчерне мы сделали вполне работоспособную коллекцию для работы с данными.

Шаг 5. Для того чтобы отследить, как все это работает, в коллекцию добавим метод PrintCls. Добавить его можно, написав код вручную или с помощью того же Class Builder Utility.

Лирическое отступление 1. Данный метод мы будем использовать на тестировочной форме для получения и обработки результатов. В дальнейшем для своих приложений вы можете написать форму с другими параметрами и исправить данный метод (или вообще его убрать) "под себя".

В данном методе мы вначале очистим подложку, затем соберем в виде строк все значения нашей коллекции и, наконец, методом Print напечатаем. Именно поэтому мы объявляем подложку как Object (т.е. на печать мы можем выводить как на саму форму, так, например, и на PictureBox).

obj.Cls For i = 1 To Me.Count str = str & Me.Item(i).Player & " | " & Me.Item(i).CurDate & " | " & Me.Item(i).Result & vbCrLf Next obj.Print str

NB! В цикле For ... Next мы проверяем наши компоненты, начиная с номера 1. Так как коллекции такого типа всегда базируются на 1, а не на 0.

Шаг 6. Переходим в форму. Вначале, в разделе деклараций, объявляем переменную R (или с другим именем, как вам захочется) с типом нашей коллекции. Затем в процедуре Form_Load озвучим ее, создав новый образец.

```
Set R = New Resultats
```

И здесь же, как говорится, "не отходя от кассы", сделаем уничтожение нашего класса, тем самым освобождая память. В Form_Unload присвоим нашему объекту состояние Nothing.

```
Set R = Nothing
```

NB! То же самое мы могли сделать в одну строку в разделе деклараций:

```
Private R As New Resultats
```

Принципиально они ничем не различаются, однако я предпочитаю первый вариант, т.к. он позволяет мне конкретно знать, когда начинает работать мой класс.

Лирическое отступление 2. Каждый раз, создавая новый объект, не забывайте уничтожать его по мере окончания его работы, чаще всего это делается при выгрузке формы или завершении приложения вообще.

После того, как мы озвучили нашу переменную как новый класс, добавим несколько записей в него, согласно запрашиваемой структуре. А в обработку нажатия кнопки заносим метод PrintCls. Вот теперь можно запустить приложение и, нажав кнопку Print, посмотреть, как работает вновь созданный нами класс.

Шаг 7. Когда пользователь достигает какого-либо результата, то это должно заноситься в массив уже существующих значений. Для реализации этого в разделе деклараций коллекции объявим переменную NewResult с типом Result. А в методе Add в самом конце озвучим ее, приравняв ее значение к последнему значению коллекции.

```
Set NewResult = Me.Item(Me.Count)
```

В данной ситуации каждое новое значение добавляется в конец коллекции. Поэтому, чтобы расположить полученный нами результат на должном уровне в зависимости от его значения, напишем небольшую внутреннюю процедуру Sort, запускать которую мы будем сразу же после объявления переменной NewResult.

```
Sort NewResult.Player, NewResult.CurDate, NewResult.Result
```

Она будет состоять из цикла For : Next, внутри которого последовательно будет сверяться соответствие уровня последнего значения с каждым из имеющихся и устанавливаться на соответствующую ему позицию, отодвигая вниз меньшие (худшие) результаты.

```
For i = 1 To Me.Count
    If NewResult.Result > Me.Item(i).Result Then
        For j = (Me.Count - 1) To i Step -1
            Me.Item(j + 1).Player = Me.Item(j).Player
            Me.Item(j + 1).CurDate = Me.Item(j).CurDate
            Me.Item(j + 1).Result = Me.Item(j).Result
        Next
        Me.Item(i).Player = Pl
        Me.Item(i).CurDate = Dat
        Me.Item(i).Result = Res
        Exit For
    End If
Next
```

На тестируемую форму добавим 3 текстовых поля, кнопку cmdAdd, которой присвоим метод Add, считывающий значения из этих 3

текстовых полей. Запустим программу на выполнение. Каждый раз, нажав кнопку Add, мы добавляем новую запись в коллекцию, обновление которой можно посмотреть, нажав кнопку Print.

Шаг 8. Тестируя программу на данном уровне, вы, наверно заметили, что наращивание нашей коллекции может происходить достаточно долго, в чем нет необходимости для конкретного приложения. Обычно в играх ограничиваются 10-20 лучшими результатами. Поэтому давайте создадим свойство Max, отвечающее за максимальное количество записей в коллекции. Данное свойство проще всего создать с помощью все той же Class Builder Utility. Добавив в коллекцию это свойство, необходимо задействовать его в процедуре Sort для отсечения лишних записей.

```
If Me.Count > Me.Max Then
    Me.Remove Me.Count
End If
```

А в форме объявить его сразу же после создания нашего класса. Ограничимся 10 записями.

```
R.Max = 10
```

Шаг 9. Добавим в коллекцию метод Clear, очищающий всю коллекцию от записей, последовательно удаляя каждую из них. (Пользователь нажал "Очистить все результаты").

```
For i = Me.Count To 1 Step -1
    Me.Remove i
Next
```

NB! Здесь мы используем цикл For : Next с отрицательным шагом для избежания выдачи сообщения об ошибке ("отсутствие необходимого количества элементов").

Шаг 10. Зайдем в класс Resultat и сделаем исправления в свойстве Player для ограничения вводимого количества знаков. Допустим, ограничим данную строку 10 знаками. В случае большего значения - обрезаем, в случае меньшего - наращиваем пробелами.

```
Select Case Len(vData)
    Case Is > 10
        vData = Left(vData, 10)
    Case Is < 10
        Do Until Len(vData) = 10
            vData = vData & " "
        Loop
End Select
```

Запустим программу на исполнение и увидим выровненные колонки результатов.

Лирическое отступление 3. Мы закончили создание класса, управляющего записями результатов игры. Данная статья не предусматривала вопросы считывания и записи значений. Вариантов для этого превеликое множество (ini-файлы, реестр, различные базы данных или просто текстовый файл). Я думаю, что это лучше реализовать отдельным классом (коллекцией). Имея перед глазами эту статью, вы сможете это сделать самостоятельно.

NB! Если вы внимательно просмотрели код, то заметили, что, находясь внутри класса (коллекции) я обращаюсь к их свойствам и методам, используя ключевое слово Me. Делать это совсем не обязательно, так как класс сам "знает", что в него входит. Здесь я использовал это ключевое слово умышленно (откуда что берется), для облегчения восприятия, читающих эту статью.

Объекты Visual Basic

Хочу сразу предупредить - то что вы будете читать здесь - ни в коем случае не является академическим описанием объектно ориентированного программирования. Более того - это нельзя назвать даже нормальной учебной статьей. То о чем я буду писать вероятнее всего уже вам знакомо, или вы обо всем этом догадывались. В то же время так, не совсем традиционно изложенная информация может в какой-то мере прояснить вам картину видения обьектов, сквозь призму Visual Basic. Так что - Вперед!

Как вы уже несомненно слышали, версии Бэйсика начиная от пятой и выше поддержвают создание почти полноценных обьектов. Почему почти? Потому что полноценнные обьекты должны иметь возможность наследоваться, а также поддерживать инкапсуляцию и полиморфизм. Не поленились посмотреть что я там в словарике написал? Ну, ну. Запомните это - это основа, которую все равно надо знать. Так вот, из трех указанных в Бэйсике, реализовано только два последних. Как-же. как-же закричат любители правды - мы можем сослаться внутри обьекта на методы и свойства другого класса и в результате получить эти самые методы и свойства к использованию, как унаследованные. Правильно, но вы не наследуете обьект - вы наследуете его интерфейсы. И сейчас, давайте лучше плюнем на это - что дано, тем и пользуемся.А начнем мы с самых азов. Что такое класс и как написать свой собственнный первый объект.

Класс (а в Бэйсике он реализуеся добавлением в проект class module) это просто шаблон. Некая коробка, на которой большими буквами написано - то что лежит внутри это не просто код, это ОБЪЕКТ! Испугались? Вот и Windows такой надписи пугается, да так сильно, что все, что вы обьявили с волшебным словом Public превращается в свойства или методы объекта. Итак, добавили в проект класс модуль, в свойствах этого класс модуля изменили имя с бестолкового Class1 на MyFirstClass, и смело пишем

```
Public A as integer
Public Sub AA ()
End Sub
Public Function AAA() as integer
End function
```

Возвращаемся в форму, и на Form_load создаем обьект типа MyFirstClass:

```
dim obj as new MyFirstClass
obj.
```

И тут справа от точки выпадает список свойств и методов нашего новоявленного объекта. То, что обозначенно пиктограмкой летящего зеленого кирпича - это методы, а листик с перстом указующим - свойства. Запоминаем эти обозначения, с ними вы встретитесь еще не раз, и не только в этом месте. Итак, листик содержит A, AA, AAA Чем между собой отличаются 2 последних ? А ничем, точнее почти ничем - функция может вернуть значение прямо в своем имени, а процедура только в одном или нескольких аргументах. Снаружи они выглядят

одинаково, только немного по разному вызываются. Как мы можем использовать класс? Очень просто. Выбирайте один из методов или свойств обьекта obj.aa

При этом выполнится код, содержашийся в процедуре AA. Любой объект проходит один раз при рождении через процедуру class_initialaize а при смерти - class_terminate Когда же происходят рождение и смерть Инстанса класса? Рождение (и инициализация) класса происходит при первом вызове метода или свойства объекта. Смерть класса наступает при уничтожении переменной, являющейся референсом (ссылкой на данный обьект - В нашем случае эта переменная obj). Последнее верно до тех пор, пока референс только один

Мы дошли до интересного момента - это переменные ссылки на обьект. Вы можете спросить что же в них интересного? А интересного в них - поведение. Вы наверное слышали в детстве от мамы, что присвоить рефреренс объекту просто, надо сказать Set референс_name = Class_Object_name Ключевое слово SET обязательно. Декларация референс_name как обьекта, тоже обязательна. При этом происходит присваивание ссылки объекту (но не создание обьекта). Ссылка на обьект является полноценным "хозяином" этого обьекта, его именем. Зная это имя вы можете манипулировать свойствами, методами, участвовать в эвентах. А что будет если присвоить одной ссылке другую?

```
Set референс2_name = референс_name
```

Вы получите Две одинаковые и равноправные ссылки на обьект - Если убить одну, с самим обьектом ничего не случится. Вот если прибить и вторую...

Да, я так вольно толкую о времени жизни этих ссылок. Естественно время жизни переменной оределяется тем, где эта самая преременная обьявленна. Если на General формы - то и умрет вместе с формой, если в процедуре - то по окончанию процедуры. Естественно существует способ убивать обьектные переменные - Set референс_name = Nothing

Если референс на обьект(единственный референс) будет перенаправлен на другой обьект - смерти первому обьекту не миновать. Поэкспериментируйте с эвентами Initialize & Terminate класса, поместив туда мессадж боксы.

Теперь про волшебное слово NEW - оно меняет смысл указанных выше присвоений

Set референс_name = New Class_Object_name

При этом вы создаете не ссылку на существующий обьект , а ссылку на Новую Инстанс этого обьекта. Сам этот несуществующий обьект еще не родится, (родится он при первом же обращении к свойствам или методам) но ссылка уже будет подготовлена. Так же как и раньше вы можете иметь несколько ссылок на один обьект (родившийся или не родившийся), но использование слова NEW создаст вам еще одну копию

(инстранс). Ну и напоследок - ссылка это все-таки не класс из которого рожден обьект, поэтому конструкция типа

Set референс2_name = NEW референс_name - вызовет ошибку времени выполнения.

Обещанное продолжение:Начну , пожалуй с того , что открою вам тайну. Даже две. Первая, наиболее страшная, это то, что обьекты внутри состоят из обычного кода, который вы лихо лепите тысячами строк. Т.е. не надо ожидать увидить внутри классов что-то, чего вы не видите в обычных формах . Кстати, как дополнение к этой страшной тайне - формы, такие привычные и любимые нами формы - это тоже инстансы классов. Предопределенные заранее, уложенные в коллекцию форм, но в то же время просто объекты.

Думаю к последнему факту мы еще вернемся.

Вторая тайна, неким неуловимым образом тоже связанная с первой, состоит в том, что изнутри класса - обьект (инстанс этого класса), как таковой - не виден. Его там (внутри класс модуля) не существует и существовать не может. Зато ооттуда вы легко можете получить доступ ко всем свойствам и методам, так как это, не более чем переменные, процедуры и функции. Из того, что не встречалось вам в обычном программировании можно отметить специальные процедуры - Property Let, Property Get, Property Set.

Интуитивно ясно, для чего предназначенны эти процедуры. Let - установить значение свойства , Get - получить у обьекта установленное значение, Set - то же что и первое, только для свойства типа объект.

Предвижу вопросы типа "А зачем это вообще нужно? Мы легко создаем свойства объявляя в классе Public переменные." Да. Вы действительно можете, и обязательно будете это делать - только такие свойства - статические. Максимум на что они годятся - это хранить значения. Чаще же изменение значения свойства влечет за собой какое либо действие. Давайте рассмотрим тривиальный пример - наша форма [Еще помните про первую тайну :-) ?] уже имеет свойство StartUpPosition, однако это такое "одноразовое" свойство. Создадим нашему обьекту-форме еще одно свойство - Center. Это простое свойство будет иметь всего 2 значения - Да и Нет (boolean). В случае установки этого свойства в True Форма должна изменить свое положение, и переместится в центр экрана. В случае установки этого свойства в False Форма должна "вспомнить" свое прошлое положение.Для приготовления этого блюда вам понадобится кастрюля, свежее мясо... Впрочем отвлекся я куда-то в сторону. Так вот нам понадобится вспомнить про вторую тайну. Т.е. форма изнутри - совсем не обьект, и для обращения с ней, как с обьектом вам нужна вторая форма, чтобы "смотреть" на первую "снаружи". А внутри формы нужно добавить две переменные для хранения координат формы (мы ведь собираемся из восстанавливать по False), и Property Let & Property Get Последние две проще всего добавить так - откройте окно кода, затем выберите в меню Tools Бэйсика "Add

Procedure", надписать Center, пометить Property и Public. После Ok, вы увидите в коде две новые процедуры.

```
Public Property Get Center() As Variant
End Property
Public Property Let Center(ByVal vNewValue As Variant)
End Property
```

Что такое vNewValue? Собственно, это переменная, которая будет принимать значение передаваемое в свойство. Эта переменная Должна быть того же типа что и сам аргумент принимаемый и возвращаемый свойством. Естественно и Property Get Center должно быть того же типа. Так что меняем оба Variant на Boolean

Далее нам необходимы две Private переменные MyLeft и MyTop (long), и еще одна , чтобы помнить какое же значение этого свойства установлено сейчас - blnCenter (boolean) Последняя нужна так же для передачи этого значения между Let и Get. Вот что получится:

```
Option Explicit
Dim MyTop As Long
Dim MyLeft As Long
Dim blnCenter As Boolean


Public Property Get Center() As Boolean
     Center = blnCenter
End Property

Public Property Let Center(ByVal vNewValue As Boolean)
      blnCenter = vNewValue
      If blnCenter Then
      MyTop = Me.Top
     MyLeft = Me.Left
     Me.Top = (Screen.Height - Me.Height) / 2
     Me.Left = (Screen.Width - Me.Width) / 2
Else
     Me.Top = MyTop
     Me.Left = MyLeft
     End If
End Property

Добавьте вторую форму, на нее 2 кнопки, и код
Option Explicit
Dim obj As New Form1

Private Sub cmdCenter_Click()
     obj.Center = True
End Sub

Private Sub cmdNonCenter_Click()
     obj.Center = False
End Sub

Private Sub Form_Load()
     obj.Show
     obj.Left = 500
     obj.Top = 1000
End Sub
```

Да, не забудьте в свойствах проекта указать Startup Form - Form2Рекомендую поставить точки останова , и посмотреть как это точно работает.

Да, в списке свойств Fom1 появилось только одно свойство Center, хотя процедур 2. Также не видны и три наши переменные. Теперь вы легко догадаетесь, как сделать свойство "только для чтения" или "только для записи"- правильно - стереть ненужную процедуру. И, помните событие class_initialaize? Это самый удобный момент присвить переменной blnCenter значение по умолчанию (соответсвенно и свойство будет иметь такое же значение. Ну, а в качестве домашнего задания сделате метод SlowMove, которые медленно и плавно двигает форму в указанную точку, и свойство Proporcional - несущее отношение высоты формы к ширине. Что то , связанное со свойствам осталось непонятным? Пишите вопросы- буду дописыватьСобственно, появилось дополнение, и чтобы не вводить вас в заблуждение, я должен о нем сказать - внутри обьекта вы можете обратиться к самому обьекту использую псевдоним Me.

Поставленная после Me точка подсветит вам все существующие свойства и методы обьекта.

Обещанное продолжение:Обьекты изнутри.

Тайну эту я вам уже выболтал в прошлом выпуске. Там внутри просто код. Ну и соотвественно все , что мы можем делать в обычном коде - можем и там. Какие возможности открывает перед нами запихивание кода в класс? А разные! Вот например такая задачка. Всем хороши коллекции. Но нередко необходимо ограничить количество элементов. Можем мы это сделать в оригинальной коллекции? Нет, не можем. Мы не в состоянии влезть внутрь реализованных встроенных свойств и методов. Зато мы можем написать собственный класс. Вот и давайте быстренько это сделаем.

- Добавить класс модуль
- Назвать его MyCol
- Создать Private переменную типа New Collection
- Тут стоп. Почему Private? Потому что нам не надо , чтобы коллекцию было видно снаружи. Почему New Collection? Потому что иным способом обьявить коллекцию нельзя. Можно создать ссылку на существующую коллекцию, но не более. А нам нужна новая.
- Создать методы Remove, ADD, Item, Count - первая как процедура, последние 3 как функции. Начнем с простой - Count :

```
Public Function Count() As Long
     Count = Col.Count
End Function
```

Аналогично создаются и остальные - просто повторяете матоды коллекции, принимая их аргументы. Остановимся на ADD:

```
Public Function Add(Item As Variant, Key As String) As Boolean
     if Col.Count <=5 Then
          Col.Add Item, Key
          Add = true
     Else
```

```
            Add = False
        end if
End Function
```

Итак, обратите внимание - функция возвращает True только если добавить удалось. Если не удалось - ложь. Второе - первый аргумент обьявлен как variant - очень удобно, можно передать и строку, и число, и даже обьект. И это не всегда плюс - тут у вас в руках еще один механизм ограниченииий - измените тип переменной, и получите коллекцию кнопок или строк, или...

Да, вы можете спросить как на счет необязательных параметров Before и After в нашем этом методе. Хороший Вопрос. Почитайте ответ в хелпе на IsMissing и Optional Arguments

В этом примере количество элементов, допустимых к добавлению - зашито в коде - а это как то не к лицу нашему универсальному классу. Пэтому давайте добавим Public свойство NumberOfItem, и на всякий случай зададим ему значение по умолчанию, и сделаем это в событии class_initialaize: NumberOfItem = 32000 (Число - чисто к примеру, и ничего не означает) Еще необходимо осветить один момент, Что делать с ошибками? Конечно, если кода так мало - как в нашем примере - то все можно предусмотреть, и ошибок времени выполнения успешно избежать. Если же код большой, то и возможность появления этих ошибок растет. Необходимы обработчики ошибок . Эти обработчики прекрасно делают свое дело - если вы пишете просто классы, которые встраиваются в ваш проект, а не оформляются в виде отдельных модулей exe (серверов)

Если вам надо вернуть ошибку в вызывающую программу , то могут возникнуть сложности - в этом случае нужно использовать Rise метод Err обьекта. Рекомендую вам попробовать это ручками.

Правда до этого вам надо почитать по поводу создания и регистрации внешних модулей. Кое что я писал об этом в "DCOM или Компоненты типа Клиент-Сервер", скорее всего я размещю аналогичный материал и здесь. Но это тема для нашего следующего "урока".

Инстансы Обьектов.

Мы говорили уже о разных способах создания обьектов. Вы уже обращали внимание на то, что в разных случаях (тип библлиотеки класса exe или dll, значение свойства public) свойство Instansing имеет различные доступные значения. Это естественно, так как это свойство определяет какой конретно тип доступа к экземплярам класса будет назначен.

Для VB6 это свойство будет иметь 6 разных значений:

1 - Private : Экземпляры этого класса не могут быть созданны извне.Более того они извне не видны Вы можете создавать и использовать такие обьекты только изнутри проекта, в которм "сидит" этот класс модуль.

2 - PublicNotCreatable : Экземпляры этого класса тоже не могут быть созданны извне , однако они могут быть использованы если уже созданы "изнутри".

3 - SingleUse : Вы можете использовать CreateObject function или ключевое слово New для создания экземпляров этого класса. При этом каждый новый элемент класса будет стартовать в отдельном "рабочем пространстве" Это свойство не появляется, если вы создаете ActiveX DLL. Обратите внимание вы таки можете насоздавать хоть сотню экземпляров этого класса.

4 -GlobalSingleUse : Префикс Global означает , что использовать этот тип класса можно без дополнительный обьявлений. В остальном это то же самое что и 3

5 - MultiUse : Да, именно, вы можете создавать и использовать экземпляры этого класса отовсюду, при этом допустимо, что одна физическая копия обьекта в памяти будет обслуживать все созданные экземпляры. Некоторые проблемы, связанные с использованием этого свойства(отказ в корректной работе при определенных настройках DCOM) я рассматривал в статье о настройке DCOM компонентов

6 - GlobalMultiUse : Префикс Global означает , что использовать этот тип класса можно без дополнительный обьявлений, т.е класс создается автоматически, вы можете сразу использовать его свойства и методы как глобальные функции. . В остальном это то же самое что и 5

Создание эвентов класса.

PS. Я ничего не пишу здесь об UserControls. Это тема для другого большого разговора. И, к тому же, 9/10 всех моих попыток написать свой собственный юзерконтрол приводили к тому, что я задумавшись над предстоящей работой понимал - лучше этого не делать. Много труда придется вложить, а реальная польза от написанного не всегда настолько очевидна. К тому же нередко такой контрол за тебя уже написан. Я не навязываю этот свой опыт - Юзерконтролы модный инструмент, но однако - Думайте перед написанием кода. Делайте это всегда :-) Удачи!

Словарик терминов

Наследование - Означает то, что один обьект может быть построен на базе другого. при этом могут быть унаследованы свойства, методы и события. К примеру класс магнитофон - это нечто умеющее проигрывать музыку с аудио кассет. На его основе можно построить класс плеер - спрятать свойство "запись" и минимизировать свойства "размер" и "вес" (шучу - но идеологически правильно)

Инкапсуляция - Инкапсуляцию можно тремя словами описать так обьект это "вещь в себе". Т.е. информация об обьекте - его свойства и методы содержатся в описании этого обьекта. Если вернуться к нашему

магнитофону - то в описание магнитофона входит , например "двухкассетная дека", с тюнером и приемником FM. Методами в данном случае будут: перемотать ленту, воспроизводить, записывать.

Полиморфизм - Полиморфизмом называют способность многих обьектов использовать один и тот же метод. При этом производимые действия будут зависить от того, какого типа обьект их инициировал. С магнитофоном тут пример не пойдет :-) А вот из Бэйсика - пожалуйста. Почти все контролы в бэйсике имеют метод SetFocus , но передаваться он будет именно тому контролу, который его вызвал. И в случае с кнопкой - это вызовет появление рамки, а в случае с текстбоксом - перенос курсора.

Методы - Методы это в принципе то, как ваш обьект реагирует на события.. Возвращаясь к уже надоевшему магнитофону - вы просто нажимете кнопку Play (это событие) далее магнитофон начинает прокручивать пленку, усиливать сигнал ... Вам уже не надо обьяснять ему что делать дальше.

Интерфейс - Интерфейсом для обьекта можно назвать набор связанных свойств и методов

Инстанс класса - Инстансом называется одна копия обьекта загруженная в память. Одновременно в памяти может существовать много инстансов одного обьекта.

Коллекции - Коллекции - как ясно из названия, это набор элементов, собранный в кучу. Тип элементов не существенен. Доступ к элементам осуществляется по текстовому уникальному ключу. Этот ключ присваевается элементу при добавлении в коллекцию. Так же до любого элемента коллекции можно добраться последовательным перебором.

Потоки в Visual Basic

www.desaware.com

Код примера для этой статьи можно загрузить отсюда.

Содержание

- Функция API CreateThread
- Обратно к функции API CreateThread
- Заключение

Только потому, что Вы должны ,
что-то делать не всегда означает,
что у Вас получится...

С появлением оператора AddressOf, часть индустрии ПО стала ориентироваться на авторов, показывающих как с использованием Visual Basic решать ранее невозможные задачи. Другая часть быстро охватила консультантов, помогающих пользователям, имеющим проблемы при решении таких задач.

Проблема не в Visual Basic или в технологии. Проблема в том, что большинство авторов применяют одно и тоже правило к AddressOf методикам, что большинство компаний по разработке ПО считают, что если Вы должны что-то сделать, то Вы сможете. Идея о том, что применение самой новой и последней технологии должно, по определению, быть самым лучшим решением проблемы, широко распространена в индустрии ПО. Эта идея неверна. Развертывание технологии должно управляться прежде всего проблемой, которую необходимо решить решить, а не технологией, которую кто-то пробует Вам впарить;).

Очень плохо, что из-за того, что компании часто пренебрегают упоминанием об ограничениях и недостатках их инструментальных средств, авторы иногда бывают не в состоянии обратить внимание читателей на следствия некоторых методик, которые они описывают. И журналы и книги иногда пренебрегают своей ответственностью, чтобы удостовериться, что практика программирования, которую они описывают, является приемлемой.

Программисту очень важно выбрать необходимый инструмент для своей работы. Это - ваша задача, чтобы разработать код, который работает теперь не только на одной специфической платформе, но также работает на разных платформах и системных конфигурациях. Ваш код должен быть хорошо документирован и поддержан другими программистами, участвующими в проекте. Ваш код должен следовать правилам, продиктованными операционной системой или стандартами, которые Вы используете. Отказ так делать может привести к проблемам в будущем, поскольку системы и программное обеспечение постоянно совершенствуются.

Недавние статьи в Microsoft Systems Journal и Visual Basic Programmer's Journal представили программистам на Visual Basic возможность использования функции API CreateThread, чтобы непосредственно поддерживать многопоточный режим под Visual Basic. После этого, один читатель пожаловался, что моя книга Visual Basic Programmer's Guide to the Win32 API является неполной, потому что я не описал в ней эту

функцию и не продемонстрировал эту технологию. Эта статья - частично является ответом этому читателю, и частично - ответом на другие статьи, написанными на эту тему. Эта статья также является дополнением к главе 14 моей книги "Разработка ActiveX компонент на Visual Basic 5.0" относительно новых возможностей, обеспечиваемых Visual Basic 5.0 Service Pack 2.

Быстрый обзор Многопоточности

Если Вы уже хорошо разбираетесь в технологии многопоточного режима, то Вы можете пропустить этот раздел и продолжать чтение с раздела, названного "Что нового в *Service Pack 2."*

Каждый, кто использует Windows, знает, что Windows способно делать больше чем одну вещь одновременно. Может одновременно выполнять несколько программ, при одновременном проигрывании компакт-диска, посылке факса и пересылке файлов. Каждый программист знает (или должен знать) что ЦЕНТРАЛЬНЫЙ ПРОЦЕССОР компьютера может только выполнять одну команду одновременно (проигнорируем существование многопроцессорных машин). Как единственный ЦЕНТРАЛЬНЫЙ ПРОЦЕССОР может выполнять множество задач?

Это делается быстрым переключением между многими задачами. Операционная система содержит в памяти все программы, которые запущены в настоящий момент. Это позволяет ЦЕНТРАЛЬНОМУ ПРОЦЕССОРУ выполнять программы по очереди. Каждый раз происходит переключение между программами, при этом меняется содержимое внутренних регистров, включая указатель команды и указатель вершины стека. Каждая из таких "задач" называется потоком выполнения (thread of execution).

В простой многозадачной системе, каждая программа имеет емеет единственный поток. Это означает, что ЦЕНТРАЛЬНЫЙ ПРОЦЕССОР начинает выполнение команд в начале программы и продолжает следуя инструкциям в последовательности, определенной программой до тех пор, пока программа не завершается.

Скажем, программа имеет пять команд: B C D и E, которые выполняются последовательно (никаких переходов нет в этом примере). Когда приложение имеет один поток, команды будут всегда выполнять в точно том же самом порядке: A, B, C, D и E. Действительно, ЦЕНТРАЛЬНЫЙ ПРОЦЕССОР может потребовать времени для выполнения других команд в других программах, но они не будут влиять на это приложение, если не имеется конфликт над общими ресурсами системы, но это уже отдельная тема для разговора.

Продвинутая многопоточная операционная система типа Windows позволяет приложению выполнять больше чем один поток одновременно. Скажем, команда D в нашем типовом приложении могла создать новый поток, который стартовал командой B и далее выполнял последовательность команд C и E. Первый поток был бы все еще A, B, C,

D, E, но когда команда D выполнится, возникнет новый поток, который выполнит команды бы B, C, E (здесь команды D уже не будет, иначе мы получим еще один поток).

В каком порядке будут следовать команды в этом приложении?

Это могло бы быть:

Thread 1	**A**	**B**	**C**	**D**		**E**		
Thread 2					**B**		**C**	**E**

Или так:

Thread 1	**A**	**B**	**C**	**D**			**E**	
Thread 2					**B**	**C**		**E**

Или этак:

Thread 1	**A**	**B**	**C**	**D**				**E**
Thread 2					**B**	**C**	**E**	

Другими словами, когда Вы начинаете новый поток выполнения в приложении, Вы никогда не можете знать точный порядок, в котором команды в двух потоках выполнятся относительно друг друга. Два потока полностью независимы.

Почему - это проблема?

Имитатор Многопоточности

Рассмотрим проект MTDemo:

Проект содержит один модуль кода, в котором содержится две глобальных переменных:

```
' MTDemo - Multithreading Demo program
' Copyright © 1997 by Desaware Inc. All Rights Reserved
Option Explicit
Public GenericGlobalCounter As Long
Public TotalIncrements As Long
' Этот проект содержит одну форму - frmMTDemo1, которая содержит
' следующий код:
' MTDemo - Multithreading Demo program
' Copyright © 1997 by Desaware Inc. All Rights Reserved
Option Explicit
Dim State As Integer
        ' State = 0 - Idle
        ' State = 1 - Loading existing value
        ' State = 2 - Adding 1 to existing value
        ' State = 3 - Storing existing value
        ' State = 4 - Extra delay
Dim Accumulator As Long
Const OtherCodeDelay = 10
Private Sub Command1_Click()
```

```
        Dim f As New frmMTDemo1
        f.Show
End Sub
Private Sub Form_Load()
        Timer1.Interval = 750 + Rnd * 500
End Sub
Private Sub Timer1_Timer()
        Static otherdelay&
        Select Case State
                Case 0
                        lblOperation = "Idle"
                        State = 1
                Case 1
                        lblOperation = "Loading Acc"
                        Accumulator = GenericGlobalCounter
                        State = 2
                Case 2
                        lblOperation = "Incrementing"
                        Accumulator = Accumulator + 1
                        State = 3
                Case 3
                        lblOperation = "Storing"
                        GenericGlobalCounter = Accumulator
                        TotalIncrements = TotalIncrements + 1
                        State = 4
                Case 4
                        lblOperation = "Generic Code"
                        If otherdelay >= OtherCodeDelay Then
                                State = 0
                                otherdelay = 0
                        Else
                                otherdelay = otherdelay + 1
                        End If
                End Select
        UpdateDisplay
End Sub
Public Sub UpdateDisplay()
        lblGlobalCounter = Str$(GenericGlobalCounter)
        lblAccumulator = Str$(Accumulator)
        lblVerification = Str$(TotalIncrements)
End Sub
```

Эта программа для моделирования многопоточного режима использует таймер и простой конечный автомат. Переменная State описывает пять команд, которые эта программа выполняет. State = 0 - неактивное состояние. State = 1 загружает локальную переменную глобальной переменной GenericGlobalCounter. State = 2 увеличивает на единицу локальную переменную. State = 3 запоминает результат в переменной GenericGlobalCounter и увеличивает переменную TotalIncrements (которая считает количество приращений переменной GenericGlobalCounter). State = 3 добавляет дополнительную задержку, представляющую собой время, затраченное на выполнение других команд в программе.

Функция UpdateDisplay обновляет три метки на форме, которые показывают текущее значение переменной GenericGlobalCounter, локального сумматора, и общего количества приращений.

Каждый сигнал таймера моделирует цикл ЦЕНТРАЛЬНОГО ПРОЦЕССОРА в текущем потоке. Если Вы запустите программу, то увидете, что значение переменной GenericGlobalCounter будет всегда точно равно переменной

TotalIncrements, потому что переменная TotalIncrements показывает количество увеличений счетчика GenericGlobalCounter потоком.

Но что случится, когда Вы нажимаете кнопку Command1 и запустите второй экземпляр формы? Эта новая форма смоделирует второй поток.

Время от времени, команды выстроятся в линию таким образом, что обе формы загрузят одинаковое значение GenericGlobalCounter, увеличат и сохранят его. В результате, значение счетчика увеличится только на единицу, даже при том, что каждый поток полагал, что он независимо увеличивает значение счетчика. Другими словами, переменная была увеличена дважды, но значение увеличилось только на единицу. Если Вы запускаете несколько форм, то сразу заметите, что число приращений, представляемой переменной TotalIncrements, растет намного быстрее, чем счетчик GenericGlobalCounter.

Что, если переменная представляет объектный счет блокировки - который следит, когда объект должен быть освобожден? Что, если она представляет собой сигнал, который указывает, что ресурс находится в использовании?

Такая проблема может привести к появлению ресурсов, постоянно недоступных в системе, к объекту, блокируемому в памяти, или преждевременно освобожденному. Это может привести к сбоям приложения.

Этот пример был разработан, чтобы достаточно просто увидеть проблему, но попробуйте поэкспериментировать со значением переменной OtherCodeDelay. Когда опасный код относительно небольшой по сравнению со всей программой, проблемы появятся менее часто. Хотя это и звучит обнадеживающе, но истина состоит в следующем. Проблемы Многопоточного режима могут быть чрезвычайно неустойчивы и их трудно обнаружить. Это означает, что многопоточный режим требует осторожного подхода к проектированию приложения.

Решение проблем Многопоточности

Имеются два относительно простых способа избежать проблем многопоточного режима.

Избегайте всеобщего использования глобальных переменных.

Добавьте код синхронизации везде, где используются глобальные переменные.

Первый подход используется в основном в Visual Basic. Когда Вы включаете многопоточный режим в Visual Basic приложения, все глобальные переменные станут локальными для специфического потока. Это свойственно способу, с которым Visual Basic выполняет apartment model threading - подробнее об этом позднее.

Первоначальный выпуск Visual Basic 5.0 позволял использовать многопоточность только в компонентах, которые не имели никаких

элементов пользовательского интерфейса. Так было потому что они не имели безопасного потока управления формами. Например: когда Вы создаете форму в Visual Basic, VB дает ей имя глобальной переменной (таким образом, если Вы имеете форму, именованную Form1, Вы можете непосредственно обращаться к ее методам, используя Form1.метод вместо того, чтобы объявить отдельную переменную формы). Этот тип глобальной переменной может вызывать проблемы многопоточного режима, которые Вы видели ранее. Имелись несомненно другие проблемы внутри управления формами.

С service pack 2, управление формами Visual Basic было сделано безопасным потоком. Это говорит о том, что каждый поток имеет собственную глобальную переменную для каждой формы, определенной в проекте.

Что нового в Service Pack 2

Сделав поток управления формами безопасным, Service pack 2 предоставил возможность с помощью Visual Basic создавать клиентские приложения, использующие многопоточный режим.

Приложение должно быть определено как программа ActiveX Exe с установкой запуска из Sub Main:

```
' MTDemo2 - Multithreading demo program
' Copyright © 1997 by Desaware Inc. All Rights Reserved
Option Explicit
Declare Function FindWindow Lib "user32" Alias "FindWindowA" _
(ByVal lpClassName As String, ByVal lpWindowName As String) _
                                                    As Long
Sub Main()
        Dim f As frmMTDemo2
        ' We need this because Main is called on each new thread
        Dim hwnd As Long
        hwnd = FindWindow(vbNullString, "Multithreading Demo2")
        If hwnd = 0 Then
                Set f = New frmMTDemo2
                f.Show
                Set f = Nothing
        End If
End Sub
```

Первый раз программа загружает и отображает основную форму приложения. Подпрограмма Main должна выяснить, является ли это первым потоком приложения, поэтому этот код выполняется при старте каждого потока. Вы не можете использовать глобальную переменную, чтобы это выяснить, потому что Visual Basic apartment model хранит глобальные переменные специфическими для одиночного потока. В этом примере используется функция API FindWindow, чтобы проверить, была ли загружена основная форма примера. Имеются другие способы выяснить, является ли это основным потоком, включая использование объектов синхронизации системы - но это отдельная тема для разговора.

Многопоточный режим реализуется созданием объекта в новом потоке. Объект должен быть определен, используя модуль класса. В этом случае, простой модуль класса определяется следующим образом:

```
' MTDemo2 - Multithreading demo program
```

```
' Copyright © 1997 by Desaware Inc. All Rights Reserved
Option Explicit
Private Sub Class_Initialize()
        Dim f As New frmMTDemo2
        f.Show
        Set f = Nothing
End Sub
```

Мы можем установить переменную формы как nothing после того, как она создана, потому что после отображения формы она будет сохранена.

```
' MTDemo2 - Multithreading demo program
' Copyright © 1997 by Desaware Inc. All Rights Reserved
Option Explicit
Private Sub cmdLaunch1_Click()
        Dim c As New clsMTDemo2
        c.DisplayObjPtr Nothing
End Sub
Private Sub cmdLaunch2_Click()
        Dim c As clsMTDemo2
        Set c = CreateObject("MTDemo2.clsMTDemo2")
End Sub
Private Sub Form_Load()
        lblThread.Caption = Str$(App.ThreadID)
End Sub
```

Форма отображает идентификатор потока в метке на форме. Форма содержит две командные кнопки, одна из которых использует оператор New, другая -использует оператор CreateObject.

Если Вы запустите программу внутри среды Visual Basic, то увидите, что формы всегда создаются в одном и том же потоке. Это происходит, потому что среда Visual Basic поддерживает только одиночный поток. Если Вы скомпилируете и запустите программу, то увидите, что подход, использующий CreateObject создает и clsMTDemo2 и ее форму в новом потоке.

Почему многопоточность

Откуда вся суета относительно многопоточного режима, если он включает так много потенциальной опасности? Потому что, в некоторых ситуациях, многопоточный режим может значительно улучшать эффективность приложения. В некоторых случаях это может улучшать эффективность некоторых операций синхронизации типа ожидания завершения приложения. Это позволяет сделать архитектуру приложения более гибкой. Например, операция Add a long в форме MTDEMO2 со следующим кодом:

```
Private Sub cmdLongOp_Click()
Dim l&
Dim s$
For l = 1 To 1000000
    s = Chr$(l And &H7F)
Next l
End Sub
```

Запустите несколько экземпляров формы, используя кнопку cmdLaunch1. Когда Вы нажимаете на кнопку cmdLongOp на любой из форм, то увидите, что это действие замораживает операции на всех других формах. Так происходит, потому что все формы выполняются в одиночном потоке - и этот поток занят выполнением длинного цикла.

Если Вы запустите несколько экземпляров формы кнопкой cmdLaunch2 и нажимете кнопку cmdLongOp на форму, то только эта форма будет заморожена - другие формы будут активными. Они выполняются в собственных потоках, и длинный цикл будет выполняться только в собственном потоке. Конечно, в любом случае, Вы вероятно не должны размещать длительные операции такого типа в ваших формах.

Дальше краткое резюме, когда важен многопоточный режим:

Сервер ActiveX EXE – без общих ресурсов.

Когда Вы имеете ActiveX EXE сервер, который Вы собираетесь совместно использовать среди нескольких приложений, многопоточный режим предотвращает приложения от нежелательных взаимодействий с друг другом. Если одно приложение выполняет длинную операцию на объекте в однопоточном сервере, другие приложения будут вытеснены, т.е. будут ждать, когда освободится сервер. Многопоточный режим рещает эту проблему. Однако, имеются случаи, где Вы можете хотеть использовать ActiveX EXE сервер, чтобы регулировать доступ к общедоступнному ресурсу (shared resource). Например, сервер stock quote, описанный в моей книге Developing ActiveX Components. В этом случае сервер stock quote выполняется в одиночном потоке и который доступен для всех приложений, использующих сервер по очереди.

Многопоточный клиент – выполняемый как ActiveX EXE сервер

Простая форма этого подхода продемонстрирована в приложении MTDEMO2. Этот подход используется, когда приложение поддерживает множественные окна, которые должны исходить из одного приложения, но работать полностью независимо. Интернет-браузер - хороший пример такого многопоточного клиента, где каждое окно выполняется в собственном потоке. Здесь следует обратить внимание на то, что многопоточный режим не должен использоваться как замена для хорошего событийно управляемого проекта.

Многопоточные серверы DLL или EXE

В архитектуре клиент-сервер, многопоточный режим может увеличить эффективность, если Вы имеете смесь длинных и коротких клиентских запросов. Будьте внимательным, хотя - если все ваши клиентские запросы имеют подобную длину, многопоточный режим может фактически замедлять среднее время ответа сервера! Никогда не принимайте на веру тот факт, что если ваш сервер является многопоточным, то обязательно его эффективность увеличится.

Соглашение о потоках

Верите или нет, но все это было введением. Часть этого материала является обзором материала, который описан в моей книге Developing ActiveX Components, другая часть материала описывает новую информацию для service pack 2.

Теперь, позволите задавать вопрос, который имеет отношение к многопоточному режиму, использующему COM (модель многокомпонентных объектов, на которой основаны не только все Visual Basic объекты, но и другие windows приложения, использующие технологии OLE).

Дано:

Многопоточный режим является потенциально опасным вообще, и особенно попытки многопоточного кодирования приложений, которые не разработаны для поддержки многопоточного режима, скорее всего приведут к фатальным ошибкам и сбоям системы.

Вопрос:

Как это возможно, что Visual Basic позволяет Вам создавать объекты и использовать их с одиночными и многопоточными средами безотносительно к тому, разработаны ли они для одиночного или многопоточного использования?

Другими словами - Как многопоточные Visual Basic приложения могут использовать объекты, которые не разработаны для безопасного выполнения в многопоточной среде? Как могут другие многопоточные приложения использовать однопоточные объекты Visual Basic?

Коротко: как COM поддерживает потоки?

Если Вы знаете COM, то Вы знаете, что COM определяет структуру соглашения. Объект COM соглашается следовать некоторым правилам так, чтобы этим можно было успешно пользоваться из любого приложения или объекта, который поддерживает COM.

Большинство людей сначала думает о интерфейсной части соглашения - о методах и свойствах, которые предоставляет объект.

Но Вы не можете не знать того, что COM также определяет поточность как часть соглашения. И подобно любой части соглашения COM - если Вы нарушаете эти условия, то будете иметь проблемы. Visual Basic, естественно, скрывает от Вас большинство механизмов COM, но чтобы понять как использовать многопоточность в Visual Basic, Вы должны разобраться COM модели потоков.

Модель одиночного потока:

Однопоточный сервер - самый простой тип реализации сервера. И самый простой для понимания. В этом случае EXE сервер выполняется в одиночном потоке. Все объекты создаются в этом потоке. Все вызовы методов каждого объекта, поддерживаемого сервером должны прибыть в этот поток.

Но что будет, если клиент выполняется в другом потоке? В том случае, для объекта сервера должен быть создан промежуточный объект (proxy

object). Этот промежуточный объект выполняется в потоке клиента и отражает методы и свойства фактического объекта. Когда вызывается метод промежуточного объекта, он выполняет операции, необходимые для подключению к потоку объекта, а затем вызывает метод фактического объекта, используя параметры, переданные к промежуточному объекту. Естественно, что этот подход требует значительного времени на выполнение задачи, однако он позволяет выполнить все соглашения. Этот процесс переключения потоков и пересылки данных от промежуточного объекта к фактическому объекту и обратно называется marshalling. Эта тема обсуждается в главе 6 моей книги Developing ActiveX Components.

В случае DLL серверов, одиночная потоковая модель требует, чтобы все объекты в сервере создавались и вызывались в том же самом потоке что и первый объект, созданный сервером.

Модель Apartment Threading

Обратите внимание, что модель Apartment Threading как определено COM не требует, чтобы каждый поток имел собственный набор глобальных переменных. Visual Basic таким образом реализует модель Apartment Threading. Модель Apartment Threading декларирует, что каждый объект может быть создан в собственном потоке, однако, как только объект создан, его методы и свойства могут вызываться только тем же самым потоком, которая создал объект. Если объект другого потока захочет иметь доступ к методам этого объекта, то он должен действовать через промежуточный объект.

Такая модель относительно проста для реализации. Если Вы устраняете глобальные переменные (как делает Visual Basic), модель Apartment Threading автоматически гарантирует безопасность потока - так как каждый объект действительно выполняется в собственном потоке, и благодаря отсутствию глобальных переменных, объекты в разных потоках не взаимодействуют друг с другом.

Модель свободных потоков

Модель свободных потоков (Free Threading Model) заключается в следующем.. Любой объект может быть создан в любом потоке. Все методы и свойства любого объекта могут быть вызываны в любое время из любого потока. Объект принимает на себя всю ответственность за обработку любой необходимой синхронизации.

Это самая трудная в реализации модель, так как требуется, чтобы всю синхронизацию обрабатывал программист. Фактически до недавнего времени, технология OLE непосредственно не поддерживала эту модель! Однако, с тех пор marshalling никогда не требуется и это наиболее эффективная модель потоков.

Какую модель поддерживает ваш сервер?

Как приложение или сама Windows узнает, которую модель потоков использует сервер? Эта информация включена в системный реестр

(registry). Когда Visual Basic создает объект, он проверяет системный реестр, чтобы определить, в каких случаях требуется использовать промежуточный объект (proxy object) и в каких - marshalling.

Эта проверка является обязанностью клиента и необходима для строгой поддержки требований многопоточности для каждого объекта, которого он создает.

Функция API CreateThread

Теперь давайте посмотрим, как с Visual Basic может использоваться функция API CreateThread. Скажем, Вы имеете класс, что Вы хотите выполненять в другом потоке, например, чтобы выполнить некоторую фоновую операцию. Характерный класс такого типа мог бы иметь следующий код (из примера MTDemo 3):

```
' Class clsBackground
' MTDemo 3 - Multithreading example
' Copyright © 1997 by Desaware Inc. All Rights Reserved
Option Explicit
Event DoneCounting()
Dim l As Long
Public Function DoTheCount(ByVal finalval&) As Boolean
Dim s As String
        If l = 0 Then
                s$ = "In Thread " & App.threadid
                Call MessageBox(0, s$, "", 0)
        End If
        l = l + 1
        If l >= finalval Then
                l = 0
                DoTheCount = True
                Call MessageBox(0, "Done with counting", "", 0)
                RaiseEvent DoneCounting
        End If
End Function
```

Класс разработан так, чтобы функция DoTheCount могла неоднократно вызываться из непрерывного цикла в фоновом потоке. Мы могли бы поместить цикл непосредственно в сам объект, но вы вскоре увидите, что были веские причины для проектирования объекта как показано в примере. При первом вызове функции DoTheCount появляется MessageBox, в котором показан идентификатор потока, по которому мы можем определить поток, в котором выполняется код. Вместо VB команды MessageBox используется MessageBox API, потому что функция API, как известно, поддерживает безопасное выполнение потоков. Второй MessageBox появляется после того, как закончен подсчет и сгенерировано событие, которое указывает, что операция закончена.

Фоновый поток запускается при помощи следующего кода в форме frmMTDemo3:

```
Private Sub cmdCreateFree_Click()
        Set c = New clsBackground
        StartBackgroundThreadFree c
End Sub
```

Функция StartBackgroundThreadFree определена в модуле modMTBack следующим образом:

```
Declare Function CreateThread Lib "kernel32" _
```

```
        (ByVal lpSecurityAttributes As Long, ByVal _
        dwStackSize As Long, ByVal lpStartAddress As Long, _
        ByVal lpParameter As Long, ByVal dwCreationFlags _
        As Long, lpThreadId As Long) As Long
Declare Function CloseHandle Lib "kernel32" _
        (ByVal hObject As Long) As Long
' Start the background thread for this object
' using the invalid free threading approach.
Public Function StartBackgroundThreadFree _
                        (ByVal qobj As clsBackground)
        Dim threadid As Long
        Dim hnd&
        Dim threadparam As Long
        ' Free threaded approach
        threadparam = ObjPtr(qobj)
        hnd = CreateThread(0, 2000, AddressOf _
                BackgroundFuncFree, threadparam, 0, threadid)
        If hnd = 0 Then
                ' Return with zero (error)
                Exit Function
        End If
        ' We don't need the thread handle
        CloseHandle hnd
        StartBackgroundThreadFree = threadid
End Function
```

Функция CreateThread имеет шесть параметров:

- lpSecurityAttributes - обычно устанавливается в нуль, чтобы использовать заданные по умолчанию атрибуты защиты.
- dwStackSize - размер стека. Каждый поток имеет собственный стек.
- lpStartAddress - адрес памяти, где стартует поток. Он должен быть равен адресу функции в стандартном модуле, полученном при использовании оператора AddressOf.
- lpParameter - long 32 разрядный параметр, который передается функции, запускающей новый поток.
- dwCreationFlags - 32 бит переменная флагов, которая позволяет Вам управлять запуском потока (активный, приостановленный и т.д.). Подробнее об этих флагах можно почитать в Microsoft's online 32 bit reference.
- lpThreadId - переменная, в которую загружается уникальный идентификатором нового потока.

Функция возвращает дескриптор потока.

В этом случае мы передаем указатель на объект clsBackground, который мы будем использовать в новом потоке. ObjPtr восстанавливает значение указателя интерфейса в переменную qobj. После создания потока закрывается дескриптор при помощи функции CloseHandle. Это действие не завершает поток, - поток продолжает выполняться до выхода из функции BackgroundFuncFree. Однако, если мы не закрыли дескриптор, то объект потока будет существовать даже после выхода из функции

BackgroundFuncFree. Все дескрипторы потока должны быть закрыты и при завершении потока система освобождает занятые потоком ресурсы.

Функция BackgroundFuncFree имеет следующий код:

' A free threaded callback.

```
' A free threaded callback.
' This is an invalid approach, though it works
' in this case.
Public Function BackgroundFuncFree(ByVal param As _
                                        IUnknown) As Long
        Dim qobj As clsBackground
        Dim res&
        ' Free threaded approach
        Set qobj = param
        Do While Not qobj.DoTheCount(100000)
        Loop
        ' qobj.ShowAForm ' Crashes!
        ' Thread ends on return
End Function
```

Параметром этой функции является- указатель на интерфейс (ByVal param As IUnknown). При этом мы можем избежать неприятностей, потому что под COM каждый интерфейс основывается на IUnknown, так что такой тип параметра допустим независимо от типа интерфейса, передаваемого функции. Мы, однако, должны немедленно определить param как тип объекта, чтобы затем его использовать. В этом случае qobj установливается как объект clsBackground, который был передан к объекту StartBackgroundThreadFree.

Функция затем выполняет бесконечный цикл, в течение которого может выполняться любая требуемая операция, в этом случае повторный счет. Подобный подход мог бы использоваться здесь, чтобы выполнить операцию ожидания, которая приостанавливает поток пока не произойдкт системное событие (типа завершения процесса). Поток затем мог бы вызвать метод класса, чтобы сообщить приложению, что событие произошло.

Доступ к объекту qobj чрезвычайно быстр из-за использования подхода свободного потока (free threading) - никакая переадресация (marshalling) при этом не используется.

Обратите внимание на то, что если Вы попробуете использовать объект clsBackground, который показывает форму, то это приведет к сбоям приложения. Обратите также внимание на то, что событие завершения никогда не происходит в клиентской форме. Действительно, даже Microsoft Systems Journal, который описывает этот подход, содержит очень много предупреждений о том, что при использовании этого подхода есть некоторые вещи, которые не работают.

Некоторые разработчики, кто пробовали развертывать приложения, применяющие этот тип многопоточности, обнаружили, что их приложения вызывают сбои после обновления к VB5 service pack 2.

Является ли это дефектом Visual Basic?
Означает ли это, что Microsoft не обеспечила совместимость?

Ответ на оба вопроса: Нет
Проблема не в Microsoft или Visual Basic.
Проблема состоит в том, что вышеупомянутый код является мусором.

Проблема проста - Visual Basic поддерживает объекты и в модели одиночного потока и в apartment model. Позвольте мне перефразировать это: объекты Visual Basic являются COM объектами и они,согласно COM соглашению, будут правильно работать как в модели одиночного потока так и в apartment model. Это означает, что каждый объект ожидает, что любые вызовы методов будут происходить в том же самом потоке, который создал объект.

Пример, показанный выше, нарушает это правило.
Это нарушает соглашение COM.

Что это означает?

- Это означает, что поведение объекта подчиненно изменениям, так как Visual Basic постоянно модифицируется.
- Это означает, что любая попытка объекта обратиться к другим объектам или формам может потерпеть неудачу и что причины отказов могут изменяться, поскольку эти объекты модифицируются.
- Это означает, что даже код, который сейчас работает, может внезапно вызвыть сбой, поскольку другие объекты добавляются, удаляются или изменяются.
- Это означает, что невозможно характеризовать поведение приложения или предсказать, будет ли оно работать или может ли работать в любой данной среде.
- Это означает, что невозможно предсказать, будет ли код работать на любой данной системе, и что его поведение может зависеть от используемой операционной, числа используемых процессоров и других проблем конфигурации системы.

Вы видите, что как только Вы нарушаете соглашение COM, Вы больше не защищены теми возможностями COM, которые позволяют объектам успешно взаимодействовать друг с другом и с клиентами.

Этот подход является программной алхимией. Это безответственно и ни один программист не должен когда-либо использовать это. Точка.

Обратно к функции API CreateThread

Теперь, когда я показал Вам, почему подход к использованию CreateThread API, показанный в некоторых статьях, является мусором, я покажу Вам, как можно использовать эту API функцию безопасно. Прием прост - Вы должны просто твердо придержаться соглашения COM о потоках. Это займет немного больше времени и усилий, но практика показала, что получаются очень надежные результаты.

Пример MTDEMO3 демонстрирует этот подход в форме frmMTDemo3, имеющей код, который запускает класс фона в apartment model следующим образом:

```
Private Sub cmdCreateApt_Click()
        Set c = New clsBackground
        StartBackgroundThreadApt c
End Sub
```

Пока это выглядит очень похоже на подход свободных потоков. Вы создаете экземпляр класса и передаете его функции, которая запускает фоновый поток. В модуле modMTBack появляется следующий код:

```
' Structure to hold IDispatch GUID
Type GUID
        Data1 As Long
        Data2 As Integer
        Data3 As Integer
        Data4(7) As Byte
End Type
Public IID_IDispatch As GUID
Declare Function CoMarshalInterThreadInterfaceInStream Lib _
        "ole32.dll" (riid As GUID, ByVal pUnk As IUnknown, _
        ppStm As Long) As Long
Declare Function CoGetInterfaceAndReleaseStream Lib _
        "ole32.dll" (ByVal pStm As Long, riid As GUID, _
        pUnk As IUnknown) As Long
Declare Function CoInitialize Lib "ole32.dll" (ByVal _
        pvReserved As Long) As Long
Declare Sub CoUninitialize Lib "ole32.dll" ()
' Start the background thread for this object
' using the apartment model
' Returns zero on error
Public Function StartBackgroundThreadApt(ByVal qobj _
                                        As clsBackground)
        Dim threadid As Long
        Dim hnd&, res&
        Dim threadparam As Long
        Dim tobj As Object
        Set tobj = qobj
        ' Proper marshaled approach
        InitializeIID
        res = CoMarshalInterThreadInterfaceInStream _
                        (IID_IDispatch, qobj, threadparam)
        If res <> 0 Then
                StartBackgroundThreadApt = 0
                Exit Function
        End If
        hnd = CreateThread(0, 2000, AddressOf _
                BackgroundFuncApt, threadparam, 0, threadid)
        If hnd = 0 Then
                ' Return with zero (error)
                Exit Function
        End If
        ' We don't need the thread handle
        CloseHandle hnd
        StartBackgroundThreadApt = threadid
End Function
```

Функция StartBackgroundThreadApt немного более сложна чем ее эквивалент при применении подхода свободных потоков. Первая новая функция называется InitializeIID. Она имеет следующий код:

```
' Initialize the GUID structure
Private Sub InitializeIID()
        Static Initialized As Boolean
        If Initialized Then Exit Sub
```

```
        With IID_IDispatch
                .Data1 = &H20400
                .Data2 = 0
                .Data3 = 0
                .Data4(0) = &HC0
                .Data4(7) = &H46
        End With
        Initialized = True
End Sub
```

Вы видите, нам необходим идентификатор интерфейса - 16 байтовая структура, которая уникально определяет интерфейс. В частности нам необходим идентификатор интерфейса для интерфейса IDispatch (подробная информация относительно IDispatch может быть найдена в моей книге Developing ActiveX Components). Функция InitializeIID просто инициализирует структуру IID_IDISPATCH к корректным значениям для идентификатора интерфейса IDispatch. Значение Это значение получается с помощью использования утилиты просмотра системного реестра.

Почему нам необходим этот идентификатор?

Потому что, чтобы твердо придерживаться соглашения COM о потоках, мы должны создать промежуточный объект (proxy object) для объекта clsBackground. Промежуточный объект должен быть передан новому потоку вместо первоначального объекта. Обращения к новому потоку на промежуточном объекте будут переадресованы (marshaled) в текущий поток.

CoMarshalInterThreadInterfaceInStream выполняет интересную задачу. Она собирает всю информацию, необходимую при создании промежуточного объекта, для определенного интерфейса и загружает ее в объект потока (stream object). В этом примере мы используем интерфейс IDispatch, потому что мы знаем, что каждый класс Visual Basic поддерживает IDispatch и мы знаем, что поддержка переадресации (marshalling) IDispatch встроена в Windows - так что этот код будет работать всегда. Затем мы передаем объект потока (stream object) новому потоку. Этот объект разработан Windows, чтобы быть передаваемым между потоками одинаковым способом, так что мы можем безопасно передавать его функции CreateThread. Остальная часть функции StartBackgroundThreadApt идентична функции StartBackgroundThreadFree.

Функция BackgroundFuncApt также сложнее чем ее эквивалент при использовании модели свободных потоков и показана ниже:

```
' A correctly marshaled apartment model callback.
' This is the correct approach, though slower.
Public Function BackgroundFuncApt(ByVal param As Long) As Long
        Dim qobj As Object
        Dim qobj2 As clsBackground
        Dim res&
        ' This new thread is a new apartment, we must
        ' initialize OLE for this apartment
        ' (VB doesn't seem to do it)
        res = CoInitialize(0)
        ' Proper apartment modeled approach
        res = CoGetInterfaceAndReleaseStream(param, _
```

```
                                        IID_IDispatch, qobj)
        Set qobj2 = qobj
        Do While Not qobj2.DoTheCount(10000)
        Loop
        qobj2.ShowAForm
        ' Alternatively, you can put a wait function here,
        ' then call the qobj function when the wait is satisfied
        ' All calls to CoInitialize must be balanced
        CoUninitialize
End Function
```

Первый шаг должен инициализировать подсистему OLE для нового потока. Это необходимо для переадресации (marshalling) кода, чтобы работать корректно. CoGetInterfaceAndReleaseStream создает промежуточный объект для объекта clsBackground и реализует объект потока (stream object), используемый для передачи данных из другого потока. Интерфейс IDispatch для нового объекта загружается в переменную qobj. Теперь возможно получить другие интерфейсы - промежуточный объект будет корректно переадресовывать данные для каждого интерфейса, который может поддерживать.

Теперь Вы можете видеть, почему цикл помещен в эту функцию вместо того, чтобы находиться непосредственно в объекте. Когда Вы впервые вызовите функцию qobj2.DoTheCount, то увидите, что код выполняется в начальном потоке! Каждый раз, когда Вы вызываете метод объекта, Вы фактически вызываете метод промежуточного объекта. Ваш текущий поток приостанавливается, запрос метода переадресовывается первоначальному потоку и вызывается метод первоначального объекта в той же самом потоке, который создал объект. Если бы цикл был в объекте, то Вы бы заморозили первоначальный поток.

Хорошим результатом применения этого подхода является то, что все работает правильно. Объект clsBackground может безопасно показывать формы и генерировать события. Недостатком этого подхода является, конечно, его более медленное исполнение. Переключение потоков и переадресация (marshalling) - относительно медленные операции. Вы фактически никогда не захотите выполнять фоновую операцию как показано здесь.

Но этот подход может чрезвычайно хорошо работать, если Вы можете помещать фоновую операцию непосредственно в функцию BackgroundFuncApt! Например: Вы могли бы иметь фоновый поток, выполняющий фоновые вычисления или операцию ожидания системы. Когда они будут завершены, вы можете вызывать метод объекта, который сгенерирует событие в клиенте. Храня количество вызовов метода, небольшое относительно количества работы, выполняемой в фоновой функции, Вы можете достигать очень эффективных результатов.

Что, если Вы хотите выполнить фоновую операцию, которая не должна использовать объект? Очевидно, проблемы с соглашением COM о потоках исчезают. Но появляются другие проблемы. Как фоновый поток сообщит о своем завершении приоритетному потоку? Как они обмениваются данными? Как два потока будут синхронизированы? Все

это возможно выполнить с помощью соответствующих вызовов API. В моей книге Visual Basic 5.0 Programmer's Guide to the Win32 API имеется информации относительно объектов синхронизации типа Событий, Mutexes, Семафоров и Waitable Таймеров.

Эта книга также включает примеры файлов отображаемых в память, которые могут быть полезны при обмене данных между процессами. Вы сможете использовать глобальные переменные, чтобы обмениваться данные, но надо знать, что такое поведение не гарантируется Visual Basic(другими словами, даже если это сейчас работает, не имеется никаких гарантий, что это будет работать в будущем). В этом случае я мог бы предложить Вам использовать для обмена данными методики, основанные на API. Однако, преимуществом показанного здесь подхода, основанного на объектах, является то, что этот подход делает проблему обмена данными между потоками тривиальной, просто делайте это через объект.

Заключение

Я однажды услышал от опытного программиста под Windows, что OLE является самой трудной технологией, которой он когда-либо обучался. Я с этим согласен. Это очень обширная тема, и некоторые части этой технологии очень трудно понять. Visual Basic, как всегда, скрывает от Вас много сложностей.

Имеется сильное искушение, чтобы пользоваться преимуществом продвинутых методов типа многопоточного режима, используя подход "tips and techniques". Это искушение поощрено некоторыми статьями, которые иногда представляют специфическое решение, приглашая Вас вырезать и вставить (cut and past) их методики в ваши собственные приложения.

Когда я писал книгу Visual Basic Programmer's Guide to the Windows API, я выступал против такого подхода к программированию. Я чувствовал, что вообще безответственно включать в приложение код, который Вы не понимаете, и что реальное знание, которое так тяжело получить, стоит затраченных усилий.

Таким образом мои книги по API были разработаны, чтобы обеспечить не быстрые ответы и простые решения, а чтобы обучить использованию API к такой степени, что программисты могли бы интеллектуально правильно применять даже наиболее продвинутые методы. Я применил это тот же самый подход к моей книге Developing ActiveX Components, которая требует много времени для обсуждения принципов ActiveX, COM и объектно-ориентированного программирования перед описанием подробностей реализации этой технологии.

Многое из моей карьеры на ниве Visual Basic и многое из деятельности в фирме Desaware, основано на обучении Visual Basic программистов продвинутым методам. Читатель, кто вдохновил меня на написание этой

статьи, критикуя меня за сдерживание технологии многопоточности, пропустил точку.

Да, я обучаю и демонстрирую продвинутые методы программирования - но я пытаюсь никогда не пропустить большую картинку. Продвинутые методы, которым я обучаю, должны быть непротиворечивы с правилами и спецификациями Windows. Они должны быть такими безопасными, насколько это возможно. Они должны быть поддерживаемыми в конечном счете. Они не должны разрушаться, когда изменяются Windows или Visual Basic.

Я могу требовать только частичного успеха, так как Microsoft может изменить правила игры всякий раз, когда им покажется, что это необходимо. Но я всегда помню об этом и пробую предупреждать людей, когда я думаю, что могу протолкнуть ограничение.

Я надеюсь, что приведенное здесь обсуждение многопоточного режима показывает опасности применения "простых методов" без хорошего понимания основной технологии.

Я не могу обещать, что использование apartment model версии CreateThread является абсолютно корректным, но мое понимание проблемы и опыт показывают, что это безопасно.

Могут иметься другие факторы, которые я пропустил. OLE - действительно сложная вещь и модули OLE DLL и сам Visual Basic подвержены изменениям. Я только могу утверждать, что лучшее из моего знания - код, который я здесь показал, удовлетворяет правилам COM и что эмпирическое доказательство показывает, что Visual Basic runtime 5 0's является достаточно безопасным для выполнения фонового кода потока в стандартном модуле.

Ссылки:

"Dan Appleman's Developing ActiveX Components with Visual Basic 5.0: A Guide to the Perplexed" published by Ziff-Davis Press, ISBN 1-56276-510-8.

"Dan Appleman's Visual Basic 5.0 Programmer's Guide to the Win32 API" published by Ziff-Davis Press, ISBN 1-56276-446-2.

Краткие описания основных функций и команд VB (для начинающих)

Автор: Shura Mirzaev (2:5020/1572.3)
Оригинал: www.vbrussian.com

Это краткий справочник по операторам VB.

- **Формат справочника:** "оператор" - "зачем нужен".
- **Цель создания:** на первых порах (да и потом тоже ;-)) не знаешь или не помнишь название оператора или функции, которая делает то, что тебе требуется. То есть в хелп рад бы заглянуть, да не знаешь, что искать.
- **Использование:** узнал, как называется нужная функция или процедура, дальше спокойно лезешь в хелп.
- **Примечания:** ничего специфического данное творение не содержит - практически ничего, связанного с базами данных, SQL, API там нет, поскольку предназначено это все для начинающих.
- **Источник:** содрано с оглавления от книжки по VB3.0-VB4.0 - автор Х. Арушанов. Все применимо и для VB5.0 и 6.0.

- **Abs (функция)** - возвращает абсолютное значение числа
- **And (операция)** - логическое И
- **AppActivate (оператор)** - активизирует окно приложения
- **Array (функция)** - создает массив из параметров и возвращает его какпеременную типа Variant
- **Asc (функция)** - возвращает числовой код первого символа строки аргумента
- **Atn (функция)** - возвращает арктангенс числа в радианах
- **Beep (оператор)** - проигрывает звуковой сигнал через динамик компьютера
- **Call (оператор)** - передает управление процедуре модуля (Sub), функциимодуля (Function) или подпрограмме DLL
- **CBool (функция)** - приводит выражение к типу Boolean
- **CByte (функция)** - преобразует выражение к типу Byte
- **CCur (функция)** - преобразование выражения к типу Currency
- **CDate (функция)** - преобразование выражения к типу Date
- **CDbl (функция)** - преобразование к типу Double

- **ChDir (оператор)** - изменяет текущий каталог на устройстве
- **ChDrive (оператор)** - изменяет текущее устройство
- **Choose (функция)** - возвращает значение из списка аргументов с определеннымпорядковым номером
- **Chr (функция)** - возвращает символ, связанный с определенным числовымкодом
- **CInt (функция)** - преобразование выражения к типу Integer
- **CLng (функция)** - преобразование выражения к типу Long
- **Close (оператор)** - закрывает файл, открытый оператором Open
- **Command (функция)** - возвращает командную строку, используемую длязапуска Visual Basic или приложения на Visual Basic
- **Const (оператор)** - объявления констант
- **Cos (функция)** - возвращает косинус числа
- **Create Object (функция)** - создать OLE Automation объект
- **CSng (функция)** - преобразование выражения к типу Single
- **CStr (функция)** - преобразование выражения к типу String
- **CurDir (функция)** - возвращает текущий каталог логического устройства
- **CVar (функция)** - преобразование выражения к типу Variant
- **CVErr (функция)** - возвращает подтип ошибки, для определенного пользователемномера ошибки
- **Date (оператор)** - устанавливает значение системной даты
- **Date (функция)** - возвращает значение системной даты
- **DateAdd (функция)** - возвращает переменную типа Variant, содержащуюдату, отличающуюся от заданной на определенный интервал времени
- **DateDiff (функция)** - возвращает число временных интервалов между двумядатами
- **DatePart (функция)** - возвращает определенную часть заданной даты
- **DateSerial (функция)** - возвращает дату для заданного года, месяцаи дня
- **DateValue (функция)** - возвращает дату

- **Day (функция)** - возвращает число от 1 до 31, соответствующее текущемудню месяца
- **DDB (функция)** - возвращает значение амортизационных потерь за определенныйпериод
- **Declare (оператор)** - на уровне модуля объявляет ссылки ко внешним подпрограммамв DLL
- **Deftype (операторы)** - устанавливает тип данных по умолчанию на уровнемодуля для переменных, параметров подпрограмм, а также возвращаемыхзначений для функций и операторов Property Get, начинающихся с определенныхсимволов
- **Dim (оператор)** - объявляет переменные и выделяет память под них
- **Dir (функция)** - возвращает имя файла или каталог, подходящий дляданного шаблона или атрибута файла, или метку тома устройства
- **DoEvents (функция)** - прерывает выполнение приложения
- **Do... Loop (оператор)** - повторяет блок команд до тех пор, пока условиеверно или до тех пор, пока условие не станет верным
- **End (оператор)** - заканчивает подпрограмму или блок команд
- **Environ (функция)** - возвращает строку, связанную с переменной окруженияоперационной системы
- **EOF (функция)** - возвращает значение, указывающее, достигнут ликонец файла
- **Eqv (оператор)** - проверяет логическое равенство двух выражений
- **Erase (оператор)** - повторно инициализирует элементы массивов фиксированногоразмера и перераспределяет память под динамические массивы
- **Error (оператор)** - эмулирует возникновение ошибки
- **Error (функция)** - возвращает текст сообщения данного номера ошибки
- **Exit (операторы)** - осуществляет выход из циклов Do ... Loop, For... Next, функции и процедур
- **Exp (функция)** - возвращает экспоненту числа
- **FileAttr (функция)** - возвращает режим открытия или номер (handle) файла
- **FileCopy(операмор)** - копирует файл

- **FileDateTime (функция)** - возвращает дату и время создания или последнеймодификации файла
- **FileLen (функция)** - возвращает длину файла в байтах
- **Fix (функция)** - возвращает целую часть числа
- **For Each...Next (оператор)** - повторяет одну и ту же последовательностькоманд для каждого элемента массива или коллекции
- **For...Next (оператор)** - повторяет последовательность команд определенноечисло раз
- **Format (функция)** - форматирует выражение в соответствии с заданнымформатом
- **FreeFile (функция)** - возвращает следующий не занятый номер файла дляиспользования в операторе Open
- **Function (оператор)** - объявляет имя, аргументы и код подпрограммы,возвращающей значение (функции)
- **FV (функция)** - возвращает значение ренты, основываясь на периодическихвзносах и постоянной норме капиталовложений
- **Get (оператор)** - читает данные из открытого файла в переменную
- **GetAttr (функция)** - возвращает атрибуты файла, каталога или меткитома
- **GetObject (функция)** - возвращает OLE Automation объект для файла сданным расширением
- **GoSub... Return (оператор)** - выполняет подпрограмму
- **GoTo (оператор)** - передает управление определенной строке подпрограммыбез возврата контроля
- **Hex (функция)** - возвращает строку, представляющую шестнадцатеричноезначение числа
- **Hour (функция)** - возвращает целое число в диапазоне 0 - 23 включительно,представляющее определенный час дня
- **If...Then... Else (оператор)** - выполнение групп команд в зависимостиот значения выражения
- **Iff (функция)** - возвращает одно из двух значении в зависимости от значениявыражения
- **Imp (операция)** - импликация двух выражений

- **Input (функция)** - возвращает символы из файла, открытого для последовательногодоступа или как двоичный файл
- **Input # (оператор)** - считывает данные из открытого файла в переменные
- **InputBox (функция)** - показывает диалоговое окно ввода, ожидает вводатекста и возвращает содержимое введенного текста, после закрытия окна
- **InStr (функция)** - возвращает позицию первой найденной подстроки в строке
- **Int (функция)** - возвращает целую часть числа
- **Is (операция)** - сравнение двух ссылок на объекты
- **IsArray (функция)** - возвращает булево значение, указывающее, являетсяли данная переменная массивом
- **IsDate (функция)** - возвращает булево значение, указывающее, можетли выражение быть преобразовано к типу Date
- **IsEmpty (функция)** - возвращает булево значение, указывающее, инициализированоли значение данной переменной
- **IsError (функция)** - возвращает булево значение, указывающее, являетсяли выражение значением кода ошибки
- **IsMissing (функция)** - возвращает булево значение, указывающее,был ли передан данный необязательный параметр в подпрограмму
- **IsNull (функция)** - возвращает булево значение, указывающее, несодержит ли выражение недопустимое (Null) значение
- **IsNumeric (функция)** - возвращает булево значение, указывающее,может ли данное выражение рассматриваться как число
- **IsObject (функция)** - возвращает булево значение, указывающее, являетсяли выражение объектом OLE Automation
- **Kill (оператор)** - удаляет файл
- **LBound (функция)** - возвращает значение нижней границы индекса массива
- **LCase (функция)** - возвращает строку в нижнем регистре
- **Left (функция)** - возвращает определенное число символов с начала строки
- **Len (функция)** - возвращает число символов строки или число байт,необходимых для хранения переменной

- **Let (оператор)** - присваивает значение выражения переменной или свойству
- **Like (операция)** - сравнение двух строк
- **Line Input # (оператор)** - считывает строку из файла в переменную
- **Load (оператор)** - загружает в память форму или элемент управления
- **LoadPicture (функция)** - загружает графический образ в объекты: Form,
- **Loc (функция)** - возвращает текущую позицию чтения/записи в открытомфайле
- **Lock (оператор)** - контролирует доступ других процессов ко всему или части открытого файла
- **LOF (функция)** - возвращает размер в байтах открытого файла
- **Log (функция)** - возвращает натуральный логарифм числа
- **LSet (оператор)** - копирует строку в строковую переменную, а такжекопирует значение переменной одного специализированного типа в переменнуюдругого специализированного типа
- **LTrim (функция)** - возвращает копию строки без лидирующих пробелов
- **Mid (оператор)** - замещает определенное число символов в строке на символыиз другой строки
- **Mid (функция)** - возвращает определенное число символов с определеннойпозиции строки
- **Minute (функция)** - возвращает целое число в диапазоне 0 - 59, представляющееминуту часа
- **MkDir (оператор)** - создает новый каталог
- **Mod (операция)** - возвращает остаток от деления двух чисел
- **Month (функция)** - возвращает целое число в диапазоне 1 - 12, представляющеенномер месяца
- **MsgBox (функция)** - показывает сообщение в диалоговом окне, ожидаетвыбор одной из кнопок пользователем и возвращает значение, указывающее,какая кнопка была выбрана
- **Name (оператор)** - переименовывает файл или каталог
- **Not (операция)** - логическое отрицание

- **Now (функция)** - возвращает текущие значения даты и времени
- **Oct (функция)** - возвращает строку, представляющую восьмеричное представлениечисла
- **On Error (оператор)** - устанавливает обработчик ошибок и задает местоположениеподпрограммы обработки; используется также для отмены обработки ошибкоподпрограммой обработчика
- **On..GoSub, On...GoTo (операторы)** - передача управления на одну изнескольких определенных строк (меток), в зависимости от значения выражения
- **Open (оператор)** - скрывает файл для ввода/вывода
- **Option Base (оператор)** - используется для объявления значения нижнейграницы размерности индексов массивов по умолчанию
- **Option Compare (оператор)** - используется на уровне модуля для объявленияметода сравнения по умолчанию при сравнении строк
- **Option Explicit (оператор)** - используется на уровне модуля для установкипроверки наличия объявлений для всех переменных в данном модуле
- **Option Private (оператор)** - используется на уровне модуля для указания,что весь модуль является Private
- **Or (операция)** - логическое ИЛИ
- **Partition (функция)** - возвращает строку, указывающую, сколько развстретились числа из заданного диапазона
- **Print # (оператор)** - записывает форматированные данные в файл
- **Private (оператор)** - используется на уровне модуля для объявления Privateпеременных и выделяет место в памяти для их хранения
- **Property Get (оператор)** - объявляет имя, аргументы и код подпрограммыполучения значения свойства
- **Property Let (оператор)** - объявляет имя, аргументы и код процедурыустановки значения свойства
- **Property Set (оператор)** - объявляет имя, аргументы и код процедурыустановки ссылки на объект
- **Public (оператор)** - используется на уровне модуля для объявления Publicпеременных и выделяет место в памяти для их хранения
- **Put (оператор)** - записывает переменную в файл

- **QBColor (функция)** - возвращает RGB код, соответствующий номеру цвета
- **Randomize (оператор)** - инициализирует генератор случайных чисел
- **RGB (функция)** - возвращает целое число, представляющее значение RGBкода
- **ReDim (оператор)** - используется на уровне подпрограммы для переопределенияразмера динамических массивов и выделения под них места в памяти
- **Rem (оператор)** - вставка комментариев в программу
- **Reset (оператор)** - закрывает все открытые программой файлы
- **Resume (оператор)** - продолжает выполнение программы после завершенияпроцедуры обработчика ошибок
- **Right (функция)** - возвращает определенное число символов с правой стороныстроки
- **RmDir (оператор)** - удаляет каталог
- **Rnd (функция)** - возвращает случайное число
- **RSet (оператор)** - копирует правую часть строки в строковую переменную
- **RTrim (функция)** - возвращает копию строки без конечных пробелов
- **SavePicture (оператор)** - сохраняет в файл графический образ объектаForm, элементов управления Picture Box или Image
- **Second (функция)** - возвращает целое значение в диапазоне 0 - 59,представляющее секунду в минуте
- **Seek (оператор)** - устанавливает позицию для следующей операции чтенияили записи в открытый файл
- **Seek (функция)** - возвращает текущую позицию чтения/записи открытогофайла
- **Select Case (оператор)** - выполняет одну или несколько команд, в зависимостиот значения выражения
- **SendKeys (оператор)** - посылает одно или несколько нажатий клавиш активномуокну, как если бы они были введены пользователем с клавиатуры
- **Set (оператор)** - связывает ссылку на объект с переменной или свойством

- **SetAttr (оператор)** - устанавливает атрибуты файла
- **Sgn (функция)** - возвращает знак числа
- **Shell (функция)** - запускает внешнюю программу на выполнение
- **Sin (функция)** - возвращает значение синуса угла
- **Space (функция)** - возвращает строку, содержащую определенное числопробелов
- **Spc (функция)** - позиционирование в строке вывода
- **Sqr (функция)** - подсчет значения квадратного корня числа
- **Static (оператор)** - используется на уровне модуля для объявления переменныхи выделяет место в памяти для их хранения. Переменные сохраняют значениядо завершения программы
- **Stop (оператор)** - приостанавливает выполнение программы
- **Str (функция)** - возвращает строковое представление числа
- **StrComp (функция)** - возвращает результат сравнения строк
- **StrConv (функция)** - возвращает преобразованную строку
- **String (функция)** - возвращает строку заданной длины из одинаковых символов
- **Sub (оператор)** - объявляет имя, параметры и тело процедуры
- **Switch (функция)** - подсчитывает значения списка выражении и возвращаетзначение или выражение, связанное с выражением из списка, значениекоторого равно True
- **Tab (функция)** - позиционирование в строке вывода
- **Tan (функция)** - возвращает значение тангенса угла
- **Time (оператор)** - устанавливает значение системных часов
- **Time (функция)** - возвращает значение типа Date, указывающее текущеесистемное время
- **Timer (функция)** - возвращает число секунд, прошедших после полуночи
- **TimeSerial (функция)** - возвращает значение типа Date, содержащееевремя для заданного часа, минуты и секунды
- **Time Value (функция)** - возвращает значение типа Date, содержащееевремя суток

- **Trim (функция)** - возвращает копию строки без начальных и конечных пробелов
- **Type (оператор)** - объявляет на уровне модуля специализированный типданных
- **TypeName (функция)** - возвращает строку информации о заданной переменной
- **UBound (функция)** - возвращает значение наибольшего индекса для даннойразмерности массива
- **UCase (функция)** - возвращает строку, преобразованную в верхний регистр
- **Unload (оператор)** - выгружает форму или элемент управления из памяти
- **Unlock (оператор)** - контролирует доступ других процессов ко всему иличасти открытого файла
- **Val (функция)** - возвращает числовое представление строки
- **VarType (функция)** - возвращает значение, указывающее тип переменной
- **Weekday (функция)** - возвращает целое число, представляющее день недели
- **While...Wend (оператор)** - выполняет в цикле последовательность команддо тех пор, пока верно условие
- **Width # (оператор)** - назначает ширину строки вывода для операции записив открытый файл
- **With (оператор)** - выполняет последовательность команд для конкретногообъекта или переменной специализированного типа
- **Write # (оператор)** - записывает данные в файл
- **Xor (операция)** - исключающее ИЛИ
- **Year (функция)** - возвращает целое число, представляющее год

Базы данных

> *Ба́за да́нных – представленная в объективной форме совокупность самостоятельных материалов (статей, расчётов, нормативных актов, судебных решений и иных подобных материалов), систематизированных таким образом, чтобы эти материалы могли быть найдены и обработаны с помощью электронной вычислительной машины (ЭВМ).*

Многие специалисты указывают на распространённую ошибку, состоящую в некорректном использовании термина «база данных» вместо термина «система управления базами данных», и указывают на необходимость различения этих понятий.

В литературе предлагается множество определений понятия «база данных», отражающих скорее субъективное мнение тех или иных авторов, однако общепризнанная единая формулировка отсутствует.

Определения из международных стандартов:

По ГОСТ Р ИСО МЭК ТО 10032-2007: Эталонная модель управления данными (идентичен ISO/IEC TR 10032:2003 Information technology – Reference model of data management)

> *База данных – совокупность данных, хранимых в соответствии со схемой данных, манипулирование которыми выполняют в соответствии с правилами средств моделирования данных.*

По ISO/IEC 2382-1:1993. Information technology – Vocabulary – Part 1: Fundamental terms

> *База данных – совокупность данных, организованных в соответствии с концептуальной структурой, описывающей характеристики этих данных и взаимоотношения между ними, причём такое собрание данных, которое поддерживает одну или более областей применения.*

Определения из авторитетных монографий:

Когаловский М. Р. Энциклопедия технологий баз данных. – М.: Финансы и статистика, 2002. – 800 с. – ISBN 5-279-02276-4

> *База данных – организованная в соответствии с определёнными правилами и поддерживаемая в памяти компьютера совокупность данных, характеризующая актуальное состояние некоторой предметной области и используемая для удовлетворения информационных потребностей пользователей.*

Дейт К. Дж. Введение в системы баз данных Introduction to Database Systems. – 8-е изд. – М.: Вильямс, 2005. – 1328 с. – ISBN 5-8459-0788-8 (рус.) 0-321-19784-4 (англ.)

> *База данных – некоторый набор перманентных (постоянно хранимых) данных, используемых прикладными программными системами какого-либо предприятия.*

Коннолли Т., Бегг К. Базы данных. Проектирование, реализация и сопровождение. Теория и практика Database Systems: A Practical Approach to Design, Implementation, and Management. – 3-е изд. – М.: Вильямс, 2003. – 1436 с. – ISBN 0-201-70857-4

> *База данных – совместно используемый набор логически связанных данных (и описание этих данных), предназначенный для удовлетворения информационных потребностей организации.*

В определениях наиболее часто (явно или неявно) присутствуют следующие отличительные признаки:

БД хранится и обрабатывается в вычислительной системе. Таким образом, любые внекомпьютерные хранилища информации (архивы, библиотеки, картотеки и т. п.) базами данных не являются.

Данные в БД логически структурированы (систематизированы) с целью обеспечения возможности их эффективного поиска и обработки в вычислительной системе. Структурированность подразумевает явное выделение составных частей (элементов), связей между ними, а также типизацию элементов и связей, при которой с типом элемента (связи) соотносится определённая семантика и допустимые операции.

БД включает схему, или метаданные, описывающие логическую структуру БД в формальном виде (в соответствии с некоторой метамоделью). В соответствии с ГОСТ Р ИСО МЭК ТО 10032-2007, «постоянные данные в среде базы данных включают в себя схему и базу данных. Схема включает в себя описания содержания, структуры и ограничений целостности, используемые для создания и поддержки базы данных. База данных включает в себя набор постоянных данных, определённых с помощью схемы. Система управления данными использует определения данных в схеме для обеспечения доступа и управления доступом к данным в базе данных».

Из перечисленных признаков только первый является строгим, а другие допускают различные трактовки и различные степени оценки. Можно лишь установить некоторую степень соответствия требованиям к БД.

В такой ситуации не последнюю роль играет общепринятая практика. В соответствии с ней, например, не называют базами данных файловые архивы, Интернет-порталы или электронные таблицы, несмотря на то, что они в некоторой степени обладают признаками БД. Принято считать, что эта степень в большинстве случаев недостаточна (хотя могут быть исключения).

История возникновения и развития технологий баз данных может рассматриваться как в широком, так и в узком аспекте.

В широком смысле понятие истории баз данных обобщается до истории любых средств, с помощью которых человечество хранило и обрабатывало данные. В таком контексте упоминаются, например, средства учёта царской казны и налогов в древнем Шумере (4000 г. до н. э.), узелковая письменность инков – кипу, клинописи, содержащие документы Ассирийского царства и т.п. Следует помнить, что недостатком этого подхода является размывание понятия «база данных» и фактическое его слияние с понятиями «архив» и даже «письменность».

История баз данных в узком смысле рассматривает базы данных в традиционном (современном) понимании. Эта история начинается с 1955 года, когда появилось программируемое оборудование обработки записей. Программное обеспечение этого времени поддерживало модель обработки записей на основе файлов. Для хранения данных использовались перфокарты.

Оперативные сетевые базы данных появились в середине 1960-х. Операции над оперативными базами данных обрабатывались в интерактивном режиме с помощью терминалов. Простые индексно-последовательные организации записей быстро развились

к более мощной модели записей, ориентированной на наборы. За руководство работой Data Base Task Group (DBTG), разработавшей стандартный язык описания данных и манипулирования данными, Чарльз Бахман получил Тьюринговскую премию.

В это же время в сообществе баз данных COBOL была проработана концепция схем баз данных и концепция независимости данных.

Следующий важный этап связан с появлением в начале 1970-х реляционной модели данных, благодаря работам Эдгара Ф. Кодда. Работы Кодда открыли путь к тесной связи прикладной технологии баз данных с математикой и логикой. За свой вклад в теорию и практику Эдгар Ф. Кодд также получил премию Тьюринга.

Сам термин база данных (англ. database) появился в начале 1960-х годов, и был введён в употребление на симпозиумах, организованных фирмой SDC (System Development Corporation) в 1964 и 1965 годах, хотя понимался сначала в довольно узком смысле, в контексте систем искусственного интеллекта. В широкое употребление в современном понимании термин вошёл лишь в 1970-е годы.

Виды баз данных.

Существует огромное количество разновидностей баз данных, отличающихся по различным критериям. Например, в «Энциклопедии технологий баз данных» определяются свыше 50 видов БД.

Основные классификации приведены ниже.

Классификация по модели данных

- Иерархическая
- Объектная и объектно-ориентированная
- Объектно-реляционная
- Реляционная
- Сетевая
- Функциональная.

Классификация по среде постоянного хранения

- Во вторичной памяти, или традиционная (англ. conventional database): средой постоянного хранения является периферийная энергонезависимая память (вторичная память) – как правило жёсткий диск. В оперативную память СУБД помещает лишь кеш и данные для текущей обработки.
- В оперативной памяти (англ. in-memory database, memory-resident database, main memory database): все данные на стадии исполнения находятся в оперативной памяти.
- В третичной памяти (англ. tertiary database): средой постоянного хранения является отсоединяемое от сервера устройство массового хранения (третичная память), как правило на основе магнитных лент или оптических дисков. Во вторичной памяти сервера хранится лишь каталог данных третичной памяти, файловый кеш и данные для текущей обработки; загрузка же самих данных требует специальной процедуры.

Классификация по содержимому

- Географическая
- Историческая
- Научная
- Мультимедийная.

Классификация по степени распределённости

- Централизованная, или сосредоточенная (англ. centralized database): БД, полностью поддерживаемая на одном компьютере.

- Распределённая (англ. distributed database): БД, составные части которой размещаются в различных узлах компьютерной сети в соответствии с каким-либо критерием.
- Неоднородная (англ. heterogeneous distributed database): фрагменты распределённой БД в разных узлах сети поддерживаются средствами более одной СУБД
- Однородная (англ. homogeneous distributed database): фрагменты распределённой БД в разных узлах сети поддерживаются средствами одной и той же СУБД.
- Фрагментированная, или секционированная (англ. partitioned database): методом распределения данных является фрагментирование (партиционирование,секционирование), вертикальное или горизонтальное.
- Тиражированная (англ. replicated database): методом распределения данных является тиражирование (репликация).

Другие виды БД

- Пространственная (англ. spatial database): БД, в которой поддерживаются пространственные свойства сущностей предметной области. Такие БД широко используются в геоинформационных системах.
- Временная, или темпоральная (англ. temporal database): БД, в которой поддерживается какой-либо аспект времени, не считая времени, определяемого пользователем.
- Пространственно-временная (англ. spatial-temporal database) БД: БД, в которой одновременно поддерживается одно или более измерений в аспектах как пространства, так и времени.
- Циклическая (англ. round-robin database): БД, объём хранимых данных которой не меняется со временем, поскольку в процессе сохранения данных одни и те же записи используются циклически.

Сверхбольшие базы данных

> *Сверхбольшая база данных (англ. Very Large Database, VLDB) – это база данных, которая занимает чрезвычайно большой объём на устройстве физического хранения.*

Термин подразумевает максимально возможные объёмы БД, которые определяются последними достижениями в технологиях физического хранения данных и в технологиях программного оперирования данными.

Количественное определение понятия «чрезвычайно большой объём» меняется во времени; в настоящее время считается, что это объём, измеряемый по меньшей мерепетабайтами (10^{15} или 2^{50} байт). Для сравнения, в 2005 г. самыми крупными в мире считались базы данных с объёмом хранилища порядка 100 терабайт.

Специалисты отмечают необходимость особых подходов к проектированию сверхбольших БД. Для их создания нередко выполняются специальные проекты с целью поиска таких системотехнических решений, которые позволили бы хоть как-то работать с такими большими объёмами данных. Как правило, необходимы специальные решения для дисковой подсистемы, специальные версии операционной среды и специальные механизмы обращения СУБД к данным.

Исследования в области хранения и обработки сверхбольших баз данных VLDB всегда находятся на острие теории и практики баз данных. В частности, с 1975 года проходит ежегодная конференция International Conference on Very Large Data Bases («Международная конференция по сверхбольшим базам данных»). Большинство

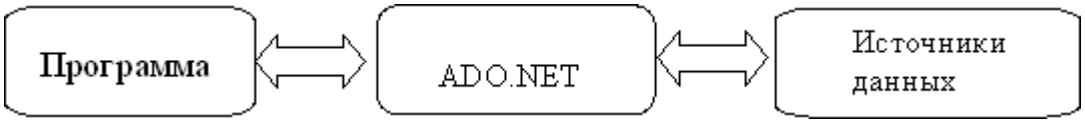
исследований проводится под эгидой некоммерческой организации VLDB Endowment (Фонд целевого капитала «VLDB»), которая обеспечивает продвижение научных работ и обмен информацией в области сверхбольших БД и смежных областях.

ADO .NET

> *ADO.NET – часть фреймфорка .NET, предоставляющая доступ к данным для приложений основанных на Microsoft .NET. Является не развитием более ранней технологии ADO, а самостоятельной технологией. Позволяет приложению легко управлять и взаимодействовать со своим файловым или серверным хранилищем данных.*

Все проектировщики информационных систем подвержены одной большой проблеме: сложность выбора СУБД и дальнейшая реализация взаимодействия с ней.

В NET Framework библиотеки ADO.NET находится в пространстве имени System.Data. Эти библиотеки обеспечивают подключение к источникам данных, выполнении команд, а также хранилище, обработку и выборку данных.



ADO.NET отличается от предыдущих технологий доступа к данным тем, что она позволяет взаимодействовать с базой данных автономно, с помощью кеша данных.

Автономный доступ к данным необходим, когда невозможно удерживать открытое физическое подключение к базе данных каждого отдельного пользователя или объекта.

Важным элементом автономного доступа к данным является контейнер для табличных данных, который не знает о СУБД. Такой незнающий о СУБД автономный контейнер для табличных данных представлен в библиотеках ADO.NET классом DataSet или DataTable.

Однако важно помнить, что ADO.NET наследует предыдущую технологию доступа к данным, разработанную Microsoft, которая называется классической ADO или просто ADO. Хотя ADO.NET и ADO - это полностью различные архитектуры доступа к данным.

Объекты ADO.NET

Как и любая другая технология, ADO.NET состоит из нескольких важных компонентов. Все классы .NET группируются в пространства имен. Все функции, относящиеся к ADO.NET находятся в пространстве имен System.Data. Кроме того, как и любые другие компоненты .NET, ADO.NET работает, не изолировано и может взаимодействовать с различными другими компонентами .NET.

Архитектуру ADO.Net можно разделить на две фундаментальные части: подключаемую и автономную. Все классы в ADO.NET можно поделить по этому критерию. Единственное исключение – класс DataAdapter, который является посредником между подключенной и автономной частями ADO.Net.

> *DataAdapter – посредник между подключенной и автономной частями ADO.Net.*

Поставщики данных .NET

Подключаемая часть ADO.Net представляет собой набор объектов подключений.

Объекты подключений разделяются в ADO.NET по конкретным реализациям для различных СУБД. То есть для подключения к базе данных SQL SERVER имеется специальных класс SqlConnection.

Эти отдельные реализации для конкретных СУБД называются поставщиками данных .NET

При наличии широкого выбора доступных источников данных ADO.NET должна иметь возможность поддерживать множество источников данных. Каждый такой источник данных может иметь свои особенности или набор возможностей.

Поэтому ADO.NET поддерживает модель поставщиков.

> *Поставщики для конкретного источника данных можно определить как совокупность классов в одном пространстве имен созданных специально для данного источника данных.*

Эта особенность несколько размыта для источников данных OleDb, ODBC, так как они по своей сути созданы для работы с любой базой данных совместимых с OLE и ODBC.

Подключаемые классы и объекты.

В подключаемой части ADO.NET имеются следующие основные классы:

Connection. Этот класс, позволяющий устанавливать подключение к источнику данных. (OleDbConnection, SqlConnection, OracleConnection)

Transaction. Объект транзакций (OleDbTransaction, SqlTransaction, OracleTransaction. В ADO.NET имеется пространство имен System.Transaction)

DataAdapter. Это своеобразный шлюз между автономными и подключенными аспектами ADO.NET. Он устанавливает подключение, и если подключение уже установлено, содержит достаточно информации, чтобы воспринимать данные автономных объектов и взаимодействовать с базой данных. (DataAdapter – SqlDataAdapter, OracleDataAdapter)

Command. Это класс представляющий исполняемую команду в базовом источнике данных.

Parameter. Объект параметр команды.

DataReader. Это эквивалент конвейерного курсора с возможностью только чтения данных в прямом направлении.

Поведение объектов подключения

Необходимость подключения к источнику данных очень важна для любой архитектуры доступа к данным. ADO.NET позволяет организовать подключение к источнику данных с помощью подходящего объекта подключения, который входит в состав соответствующего поставщика данных.

Чтобы открыть подключение необходимо указать, какая информация необходима – например, имя сервера, идентификатор пользователя, пароль и т.д. Поскольку каждому целевому источнику подключения может понадобиться особый набор информации, позволяющий ADO.NET подключится к источнику данных, выбран гибкий механизм указания всех параметров через строку подключения.

Строка подключения содержит элементы с минимальной информацией, необходимой для установления подключений, в виде последовательности пар ключей – значений. Различные пары ключей – значений в строке подключений могу определять, некоторые конфигурируемы параметры, определяющие поведение подключения.

Сам объект подключения источника данных наследуется от класса DbConnection и получает уже готовую логику, реализованную в базовых классах.

Приложение должно разделять дорогостоящий ресурс – открытое подключение – и совместно использовать его с другим пользователем. Именно для этих целей и был введен пул подключений.

> *Пул – набор инициализированных и готовых к использованию объектов. Когда системе требуется объект, он не создаётся, а берётся из пула. Когда объект больше не нужен, он не уничтожается, а возвращается в пул.*

По умолчанию пул подключений включен. При запросе ADO.NET неявно проверяет, имеется ли доступное неиспользуемое физическое подключение к базе данных. Если такое подключение имеется, то она использует его. Для принятия решения имеется ли такое физическое подключение или нет ADO.Net учитывает загрузку приложения, и если поступает слишком много одновременных запросов, ADO.Net может удерживать одновременно открытыми несколько физических подключений, то есть увеличивать при необходимости количество подключений. Если детально рассмотреть картину, то ситуация такова:

Под классом DbConnection имеется брокер, управляющий пулом открытых подключений. Он отвечает за увеличение или уменьшение реального количества открытых подключений в зависимости от потребностей приложения. Для класса брокера каждое запрошенное подключение уникально идентифицируется соответствующей строкой подключения. Поэтому когда любое приложение на одной и той же запрашивает открытое подключение, оно в начале просматривает внутренний Кеш пула подключений, и если в нем есть доступное подключение, то используется оно. При отсутствии допустимого подключения создается новое, которое и предается приложению.

Вторым наиболее ресурсоемким объектом в ADO.NET являются транзакции, отвечающие за корректность изменений в БД.

Транзакции

> *Транзакции – это набор операций, которые для обеспечения целостности и корректного поведения системы должным быть выполнены успешно или неудачно только все вместе. Это набор операций, переводящий систему из одного непротиворечивого состояния в другое.*

Обычно транзакции следуют определенным правилам, Известным как свойства ACID: неделимой (Atomic), согласованной (Consistent), изолированной (Isolated) и долговечной (Durable).

Иногда при разработке ПО возможны ситуации, когда нужно изменить некоторые характеристики. В частности, можно изменить принцип изолированности. А управлять неделимостью транзакции можно с помощью таких конструкций, как вложенные транзакции. Но отступать от этих признаков следует только после тщательного анализа.

В ADO.NET реализован мощный механизм поддержки транзакций БД. Сама ADO.NET поддерживает транзакции одиночной БД, которые отслеживаются на основании подключений. Но она может задействовать пространство имен System.Transactions для выполнения транзакций с несколькими БД или транзакций с несколькими диспетчерами ресурсов.

В ADO.NET класс подключений используется просто для начала транзакции.

Все управляемые .NET поставщики, доступные в .NET Framework OleDb, SqlClient, OracleClient, ODBC имеют свои собственные реализации класса транзакций.

Все эти классы реализуют интерфейс IDbTransaction из пространства имен System.Data.

Если при работе с транзакцией в момент возникновении ошибки не будет произведен откат, то сама среда вызовет метод Rollback, но произведет она это лишь при закрытии или повторном использовании соответствующего физического подключения.

Подключение к БД необходимо производить перед вызовом метода BeginTransaction класса Connection. Так как в данном случае ресурс физического подключения используется меньше, что уменьшает нагрузку на СУБД.

Основное преимущество транзакций – производительность. При одиночных или коротких операциях транзакции могут, выполняются медленнее, но для больших наборов данных они быстрее.

Для гибкого управления поведением транзакций используется уровни изоляции.

> *Уровни изоляции – это степень видимости внутри транзакции изменений, выполненных за пределами этой транзакции. Он определяют, насколько чувствительна транзакция к изменениям, выполненным другими транзакциями.*

Например, если в SQL Server две транзакции запущенны, не зависимо одна от другой, то по умолчанию записи, вставленные одной транзакцией, не видны в другой, пока первая транзакция не будет зафиксирована.

Концепция уровней изоляции тесно связана с концепцией блокировок: определив уровень изоляции транзакции – получаем информацию о том, как долго эта транзакция будет блокировать другие ресурсы, чтобы защищать их от изменений, которые могут быть сделаны другими операциями.

При откате транзакции происходит аннулирование эффекта каждой операции транзакции. В некоторых случаях не требуется отменять все операции – значит, нужен механизм отката только части транзакцию. Это можно реализовать с помощью "точек сохранения".

Точки отката – это маркеры, реализующие роль закладок. Во время выполнения транзакции, можно поместить какую-либо точку, и затем выполнить откат к этой точки вместо полного отката всей транзакции. Для этих целей предусмотрен метод Save в объекте транзакции. Но надо отметить, что данный метод доступен лишь для подключений к SQL SERVER. Поставщик Oracle не поддерживает точки сохранения, но их всегда можно реализовать через прямые запросы или используя хранимые процедуры.

Подведя итог, получается, что точки отката позволяют организовать откат транзакции в виде последовательности действий, для каждого из которых можно производить индивидуальных откат. А вложенные транзакции дают возможность организовывать иерархию таких действий. В случае вложенных транзакций одна транзакция включает в себя одну или несколько других транзакций. Но вложенные транзакции доступны лишь объекта подключения типа OleDb.

Автономные классы и объекты

Одни лишь подключенные приложения не удовлетворяют всем требованиям, предъявляемым к современным распределенным приложениям. В автономных приложениях, созданных с помощью ADO.NET, используются другой подход.

Для обеспечения автономности используются объекты DataAdapter. Они осуществляют выполнение запросов, используя для этого объекты подключения. А результаты выполнения, то есть данные, передает автономным объектам.

Благодаря такому принципу автономные объекты не знают о существовании объектов подключения, так как на прямую не работают с ними. То есть реализация объекта, хранящего данные, не должна зависеть от конкретного поставщика данных, а именно от СУБД. Поскольку конкретная реализация адаптера данных зависит от

соответствующего источника данных, конкретные адаптеры данных реализованы в составе конкретных поставщиков.

Автономные приложения обычно подключаются к базе как можно позже и отключаются как можно раньше. Важным элементом в такой схеме подключения и предоставления автономного доступа к данным является контейнер для табличных данных, который не знает о СУБД. Такой незнающий о СУБД автономный контейнер для табличных данных представлен в библиотеках ADO.NET классом DataSet или DataTable.

При работе в автономном режиме ADO.NET ведет пул реальных физических подключений для различных запросов, за счет которого достигается максимальная эффективность использования ресурсов подключения.

Можно выделить несколько основных классов автономной модели ADO.NET: DataSet, DataTable, DataView, DataRelation.

> *DataSet представляет собой отображение используемой базы данных, перенесенное на машину пользователя. При этом нет необходимости постоянно подключаться к серверу базы данных для модификации данных.*

Класс DataSet является ядром автономного режима доступа к данным в ADO.NET. Лучше всего рассматривать, как будто в нем есть своя маленькая СУБД, полностью находящаяся в памяти.

> *Объект типа DataTable представляет собой таблицу базы данных.*

Такой объект может быть создан программно или путем запроса к базе данных. Объект DataTable состоит из строк и столбцов. Строки представляют собой отдельные записи таблицы, столбцы – соответствующие поля. Для получения совокупности столбцов объект DataSet имеет свойство Columns, возвращающее DataColumnCollection, которое в свою очередь состоит из объектов типа DataColumn. Каждый объект DataColumn представляет собой отдельный столбец таблицы, из которого можно получить любую запись. Больше всего этот класс похож на таблицу БД. Он состоит из объектов DataColumn, DataRow, представляющих из себя строки и столбцы.

> *DataView – это объект представлений базы данных.*

> *DataRelation – это класс позволяющий задавать отношения между различными таблицами, с помощью которых можно проверять соответствие данных из различных таблиц.*

Кроме набора таблиц DataSet имеет свойство Relations, которое возвращает объект типа Data.Relation.Collection, состоящий из объектов DataRelation.

Каждый DataRelation объект хранит данные о связях между двумя таблицами посредством объектов DataColumn. Например, в базе данных таблица имеет связь с другой таблицей посредством столбца CustomerlD. Такое отношение называется на языке баз данных один ко многим (one-to-many). Для любого заказа может быть только один заказчик, но один заказчик может иметь сколько угодно заказов.

Провайдеры данных

В ADO.NET используются так называемые провайдеры данных (Data Providers) .NET. Они обеспечивают доступ к соответствующим источникам данных и содержат четыре ключевых объекта (Connection, Command, DataReader и DataAdapter). В настоящее время с ADO.NET поставляются два провайдера:

- SQL Server .NET Data Provider. Предназначен для работы с базами данных Microsoft SQL Server 7.0 и более поздних версий. Оптимизирован для доступа к SQL Server и взаимодействует с ним напрямую по «родному» протоколу передачи данных SQL Server.
- OLE DB .NET Data Provider. Управляемый провайдер для источников данных OLE DB. Немного уступает по эффективности SQL Server .NET Data Provider, так как взаимодействует с базой данных через уровень OLE DB.
- ODBC .NET Data Provider. В данный момент доступна для загрузки первая бета-версия. Этот провайдер обеспечивает «родной» доступ к ODBC-драйверам так же, как и OLE DB .NET Data Provider к «родным» провайдерам OLE DB.
- Управляемый провайдер для считывания XML из SQL Server 2000. XML for SQL Server Web update 2 (в настоящий момент проходит бета-тестирование) включает, помимо всего прочего, управляемый провайдер, предназначенный специально для считывания XML из SQL Server 2000.

Заключение

Технология ADO.NET в полной мере способна предоставить механизм для доступа к любому источнику данных, тем самым, предоставляя разработчику мощный механизм взаимодействия с базами данных способный в полной мере реализовать все потребности, возникающие при проектировании ИС.

Абстрактный класс

> *Абстрактный класс в объектно-ориентированном программировании – это базовый класс, который не предполагает создания экземпляров.*

Абстрактные классы реализуют на практике один из принципов ООП –полиморфизм. Абстрактный класс может содержать (и не содержать) абстрактные методы и свойства. Абстрактный метод не реализуется для класса, в котором объявлен, однако должен быть реализован для его неабстрактных потомков.

Абстрактные классы представляют собой наиболее общие абстракции, то есть имеющие наибольший объём и наименьшее содержание.

В одних языках создавать экземпляры абстрактных классов запрещено, в других это допускается (например, Delphi), но обращение к абстрактному методу объекта этого класса в процессе выполнения программы приведёт к ошибке. Во многих языках допустимо объявить любой класс абстрактным, даже если в нём нет абстрактных методов (например, Java), именно для запрещения создания экземпляров.

Абстрактный класс можно рассматривать в качестве интерфейса к семейству классов, порождённому им, но, в отличие от классического интерфейса, абстрактный класс может иметь определённые методы, а также свойства.

Абстрактные методы часто являются и виртуальными, в связи с чем понятия «абстрактный» и «виртуальный» иногда путают.

На стадии проектирования наследственных связей в программе часто выясняется, что многие классы обладают целым рядом сходных черт. Например, внештатные сотрудники не относятся к постоянным работникам, но и те и другие обладают рядом общих атрибутов – именем, адресом, кодом налогоплательщика и т. д. Было бы логично выделить все общие атрибуты в базовый класс.

Этот прием, называемый факторингом, часто используется при проектировании классов и позволяет довести абстракцию до ее логического завершения.

> *Факторинг – выделение общих атрибутов в базовый класс.*

В классах, полученных в результате факторинга, некоторые методы и свойства невозможно реализовать, поскольку они являются общими для всех классов в иерархии наследования.

Например, класс Payabl eEntity, от которого создаются производные классы штатных и внештатных работников, может содержать свойство с именем TaxID. Обычно в процедуре этого свойства следовало бы организовать проверку кода налогоплательщика, но для некоторых категорий внештатных работников эти коды имеют особый формат. Следовательно, проверка этого свойства должна быть реализована не в базовом классе Payabl eEntity, а в производных классах, поскольку лишь они знают, как должен выглядеть правильный код.

В таких ситуациях обычно определяется абстрактный базовый класс.

> *Абстрактным называется класс, содержащий хотя бы одну функцию с ключевым словом MustOverride; при этом сам класс помечается ключевым словом Mustlnherit.*

Ниже показано, как может выглядеть абстрактный класс Payabl eEntity:

```
Public Mustlnherit Class PayableEntity
  Private m_Name As String
  Public Sub New(ByVal aName As String)
```

```
    mName = aName
  End Sub
  Readonly Property Name()As String
    Get
      Return mName
    End Get
  End Property
  Public MustOverride Property TaxID()As String
End Class
```

Обратите внимание: свойство TaxID, помеченное ключевым словом MustOverride, только объявляется без фактической реализации. Члены классов, помеченные ключевым словом MustOverride, состоят из одних заголовков и не содержат команд End Property, End Sub и End Function. Доступное только для чтения свойство Name при этом реализовано; из этого следует, что абстрактные классы могут содержать как абстрактные, так и реализованные члены. Ниже приведен пример класса Employee, производного от абстрактного класса PayableEntity:

```
Public Class Employee
  Inherits PayableEntity

  Private mSalary As Decimal
  Private mTaxID As String
  Private Const LIMIT As Decimal = 0.1D

  Public Sub NewCByVal theName As String, ByVal curSalary As Decimal,
                                  ByVal TaxID As String)
    MyBase.New(theName)
    m_Salary = curSalary
    m_TaxID = TaxID
  End Sub

  Public Overrides Property TaxID() As String
    Get
      Return m_TaxID
    End Get
    Set(ByVal Value As String)
      m_TaxID = Value
    End Set
  End Property

  Readonly Property Salary() As Decimal
    Get
      Return MyClass.m_Salary
    End Get
  End Property

  Public Overridable Overloads Sub RaiseSalary(ByVal Percent As Decimal)
    If Percent > LIMIT Then
      ' Операция запрещена - необходим пароль
      Console.WriteLine("NEED PASSWORD TO RAISE SALARY MORE " &
                                              "THAN LIMIT!!!!")
    Else
      m_Salary =(1D + Percent) * m_Salary
    End If
  End Sub

  Public Overridable Overloads Sub RaiseSalary(ByVal Percent As
                                  Decimal, ByVal Password As String)
    If Password ="special" Then
      m_Salary MID + Percent * m_Salary
    End If
  End Sub
End Class
```

Первая ключевая строка расположена внутри конструктора, который теперь должен вызывать конструктор абстрактного базового класса для того, чтобы правильно задать имя.

Во втором выделенном фрагменте определяется элементарная реализация для свойства Taxld, объявленного с ключевым словом MustOverride (в приведенном примере новое значение свойства не проверяется, как следовало бы сделать в практическом примере).

Ниже приведена процедура Sub Main, предназначенная для тестирования этой программы:

```
Sub Main()
  Dim tom As New Employee("Tom". 50000. "111-11-1234")
  Dim sally As New Programmed "Sally", 150000. "111-11-2234".)

  Console.WriteLine(sa1ly.TheName)

  Dim ourEmployees(l) As Employee
  ourEmployees(0) = tom
  ourEmployees(l) = sally

  Dim anEmployee As Employee
  For Each anEmployee In ourEmployees
    anEmployee.RaiseSalary(0.lD)
    Console.WriteLine(anEmployee.TheName & "has tax id " & _
              anEmployee.TaxID & ".salary now is " & anEmployee.Salary())
  Next
  Console.ReadLine()
End Sub
```

В программе невозможно создать экземпляр класса, объявленного с ключевым словом Mustlnherit. Например, при попытке выполнения следующей команды:

```
Dim NoGood As New PayableEntity("can't do")
```

компилятор выводит сообщение об ошибке:

```
Class 'PayableEntity' is not creatable because it contains at least one
member marked as 'MustOverride' that hasn't been overridden.
```

Тем не менее объект производного класса можно присвоить переменной или контейнеру абстрактного базового класса, что дает возможность использовать в программе полиморфные вызовы:

```
Dim torn As New Employee("Tom". 50000, "123-45-6789")
Dim whoToPay(13) As PayableEntity whoToPay(0) = tom
```

Теоретически класс Mustlnherit может не содержать ни одного члена с ключевым сло-вом MustOverride (хотя это будет выглядеть несколько странно).

Пример: класс CollectionBase

При использовании классов коллекций .NET Framework (таких, как ArrayList и HashTable) возникает неожиданная проблема: эти классы предназначены для хранения обобщенного типа Object, поэтому прочитанные из них объекты всегда приходится преобразовывать к исходному типу функцией CType. Также возникает опасность того, что кто-нибудь сохранит в контейнере объект другого типа и попытка вызова CType завершится неудачей. Проблема решается использованием коллекций с сильной типизацией – контейнеров, позволяющих хранить объекты конкретного типа и типов, производных от него.

Хорошим примером абстрактного базового класса .NET Framework является класс CollectionBase. Классы, производные от CollectionBase, используются для построения коллекций с сильной типизацией (прежде чем создавать собственные классы коллекций, производные от CollectionBase, убедитесь в том, что нужные классы отсутствуют в пространстве имен System.Collections.Specialized). Коллекции, безопасные по отношению к типам, строятся на основе абстрактного базового класса

System.Collections.CollectionBase; от вас лишь требуется реализовать методы Add и Remove, а также свойство Item. Хранение данных во внутреннем списке реализовано на уровне класса System.Collections.CollectionBase, который и выполняет все остальные операции.

Рассмотрим пример создания специализированных коллекций (предполагается, что проект содержит класс Employee или ссылку на него):

```
1 Public Class Employees
2 Inherits System.Col lections.CollectionBase
3 ' Метод Add включает в коллекцию только объекты класса Employee.
4 ' Вызов перепоручается методу Add внутреннего объекта List.
5 Public Sub Addt (ByVal aEmployee As Employee)
6   List.Add(aEmployee)
7 End Sub
8 Public Sub Remove(ByVal index As Integer)
9   If index > Count-1 Or index < 0 Then
10    ' Индекс за границами интервала, инициировать исключение (глава 7)
11    MsgBox("Can't add this item")' MsgBox условно заменяет исключение
12  Else
13    List.RemoveAt(index)
14  End If
15 End Sub
16
17  Default  Public  Readonly  Property  Item(ByVal  index  As  Integer)As
Employee
18    Get
19      Return CType(List.Item(index). Employee)
20    End Get
21  End Property
22 End Class
```

В строках 5-7 абстрактный метод Add базового класса реализуется передачей вызова внутреннему объекту List; метод принимает для включения в коллекцию только объекты Employee. В строках 8-10 реализован метод Remove. На этот раз мы также используем свойство Count внутреннего объекта List, чтобы убедиться в том, что удаляемый объект не находится перед началом или после конца списка. Наконец, свойство Item реализуется в строках 17-21. Оно объявляется свойством по умолчанию, поскольку пользователи обычно ожидают от коллекций именно такого поведения.

Свойство объявляется доступным только для чтения, чтобы добавление новых элементов в коллекцию могло осуществляться только методом Add. Конечно, свойство можно было объявить и доступным для чтения/записи, но тогда потребовался бы дополнительный код для проверки индекса добавляемого элемента.

Следующий фрагмент проверяет работу специализированной коллекции; недопустимая операция включения нового элемента (в строке, выделенной жирным шрифтом) закомментирована:

```
Sub Main()
  Dim torn As New Employee("Tom", 50000)
  Dim sally As New Employee("Sally", 60000)
  Dim myEmployees As New Employees()
  myEmployees.Add(tom)
  myEmployees.Add(sally)
  ' myEmployees.Add("Tom")
  Dim aEmployee As Employee
  For Each aEmployee In myEmployees
    Console.WriteLine(aEmployee.TheName)
  Next
  Console.ReadLine()
End Sub
```

Попробуйте убрать комментарий из строки myEmployees. Add("Tom"). Программа перестанет компилироваться, и вы получите следующее сообщение об ошибке:

A value of type 'String'cannot be converted to 'EmployeesClass.Employee'.

Это замечательный пример того, какими преимуществами VB .NET обладает перед включением в прежних версиях VB. Конечно, мы продолжаем перепоручать вызовы внутреннему объекту, чтобы избавиться от дополнительной работы, но возможность перебора элементов в цикле For-Each появляется автоматически, поскольку наш класс является производным от класса с поддержкой For-Each!

Полимофизм

> *В традиционной трактовке термин «полиморфизм» (от греческого «много форм») означает, что объекты производных классов выбирают используемую версию метода в зависимости от своего положения в иерархии наследования.*

Полиморфизм подобен инкапсуляции, но в то же время имеет некоторые существенные отличия. Инкапсуляция выражается в сокрытии внутренней реализации объекта. Полиморфизм, в свою очередь, проявляется в том, что различные классы могут иметь один и тот же интерфейс.

Например, и в базовом классе Employee, и в производном классе Manager присутствует метод для повышения зарплаты работника. Тем не менее метод RaiseSalary для объектов класса Manager работает не так, как одноименный метод базового объекта Employee.

Классическое проявление полиморфизма при работе с классом Manager, производным от Employee, заключается в том, что при вызове метода по ссылке на Employee будет автоматически выбрана нужная версия метода (базового или производного класса). Допустим, в программе метод RaiseSalary вызывается по ссылке на Employee.

Если ссылка на Employee в действительности относится к объекту Manager, будет вызван метод RalseSalary класса Manager.

В противном случае вызывается стандартный метод RaiseSalary базового класса.

В VB5 и VB6 смысл термина «полиморфизм» был расширен, и к традиционному полиморфизму на базе наследования добавился полиморфизм на базе интерфейсов

> *Полиморфизм на базе интерфейсов – объект, реализующий интерфейс, вызывает метод интерфейса вместо другого метода с тем же именем*

Объект, реализующий интерфейс Manager, правильно выберет метод RaiseSalary в зависимости от контекста использования.

В обоих случаях объект выбирает метод в зависимости от полученного сообщения. При отправке сообщения не нужно знать, к какому классу фактически принадлежит объект; достаточно разослать сообщение всем объектам Employee и поручить выбор полиморфного метода компилятору.

Следующий пример показывает, почему полиморфизму придается такое большое значение. Одному из авторов доводилось консультировать компанию, занимавшуюся компьютерной обработкой медицинских анализов. Каждый раз, когда в процесс тестирования включался новый реактив, программистам приходилось просматривать многие тысячи строк кода, искать команды Select Case и добавлять в них секции Case для нового реактива. Стоило пропустить хотя бы одну... и нам бы не хотелось, чтобы этот реактив испытывался на наших анализах крови. Конечно, исправление многих команд Select Case превращало сопровождение в настоящий кошмар, требующий долгих часов тестирования.

Полиморфизм позволяет написать программу, которая в подобной ситуации ограничивается единственным изменением. Все, что от вас потребуется, – определить для нового реактива новый класс и правильно запрограммировать в нем переопределяемые или добавленные методы.

Почему? Потому что в главной программе можно будет использовать конструкции следующего вида:

```
For Each reagent in Reagents
  reagent.Method
Next
```

Показанный цикл будет автоматически работать с новым реактивом, а необходимость в долгих поисках Select Case отпадет.

```
Select Case reagent Case iodine
' Действия с йодом Case benzene
' Действия с бензолом
' И т. д. для 100 разных случаев в 100 местах
```

В приведенном выше фрагменте цикл For Each перебирает все возможные реактивы, и благодаря волшебному свойству полиморфизма компилятор найдет метод, который должен вызываться для каждого конкретного реактива. Правильное использование полиморфизма избавит вас от громоздких команд Select Case, выбирающих нужное действие в зависимости от типа объекта.

Задача «ArrayList»

Постановка задачи

Дан стандартный класс ArrayList

Задания:

- объявить набор
- заполнить набор строками
- отобразить число элементов набора
- отобразить элементы набора
- добавить еще один элемент в набор на третье место
- отобразить число элементов набора
- отобразить элементы набора
- отсортировать элементы набора
- копировать набор в массив
- отобразить массив
- очистить набор
- отобразить число элементов набора

Решение

Объектный анализ

Пусть объектный анализ системы приводит к следующей ОМ:

Приложение

—Модуль

Модель потока событий

Нажатие F5 – выполнение программы

Объектное проектирование

Воспользуемся стандартным классом ArrayList для построения потока

Спецификация классов

Управляющий модуль Main

Свойства:

Item- возвращает элемент по индексу из набора

Count- возвращает количество элементов набора

Объект:

ArlA- набор как экземпляр стандартного класса ArrayList

Методы:

Add-добавляет элемент в набор

Sort- сортирует

Insert- вставляет элемент на заданную позицию

CopyTo- копирует набор в массив

Clear- очищает набор

Исходный код

```
Module Module1
  Sub Main()
    Dim a() As String
    Dim n, i As Byte
    n = InputBox("Введите количество элементов", "окно ввода", 2)
    'Дан стандартный класс ArrayList. Требуется:
    '- объявить набор
    Dim arlA As New ArrayList()
    '- заполнить набор строками
    For i = 0 To n - 1
      arlA.Add(InputBox("a(" & i & ") =  ", "Ввод элементов ArrayList",
"qqq"))
    Next i
    '- отобразить число элементов набора
    MsgBox("число элементов равно " & arlA.Count)
```

```
        '- отобразить элементы набора
        For i = 0 To n - 1
           MsgBox(i & " -й элемент  " & arlA.Item(i), , "Элементы ArrayList")
        Next i
        '- добавить еще один элемент в набор на третье место
        arlA.Insert(2, InputBox("a(" & i & ") = ", "Добавление элемента",
"PPP"))
        '- отобразить число элементов набора
        MsgBox("число элементов равно " & arlA.Count)
        '- отобразить элементы набора
        For i = 0 To n
           MsgBox(i & " -й элемент  " & arlA.Item(i), , "Элементы ArrayList")
        Next i
        '- отсортировать элементы набора
        arlA.Sort()
        '- копировать набор в массив
        ReDim a(arlA.Count)
        arlA.CopyTo(a)
        '- отобразить массив
        For i = 0 To arlA.Count
           MsgBox(i & " -й элемент  " & a(i), , " Отсортированные элементы
массива")
        Next i
        '- очистить набор
        arlA.Clear()
        '- отобразить число элементов набора
        MsgBox("Массив очищен. Число элементов равно " & arlA.Count,
MsgBoxStyle.Information)
```

```
    End Sub
End Module
```

HashTable

Теоретические сведения

Интерфейсы коллекций

В VB .NET под коллекцией понимается группа объектов.

Пространство имен System.Collections содержит множество интерфейсов и классов, которые определяют и реализуют коллекции различных типов. Все эти коллекции разработаны на основе набора четко определенных интерфейсов. Ряд встроенных реализаций интерфейсов в таких коллекциях как ArrayList, Hashtable, Stack и Queue, можно использовать "как есть".

У каждого программиста есть возможность реализовать собственную коллекцию, но в большинстве случаев достаточно встроенных.

Среда .NET Framework поддерживает три основных типа коллекций:

• общего назначения,

• специализированные,

• ориентированные на побитовую организацию данных.

Коллекции общего назначения реализуют ряд основных структур данных, включая динамический массив, стек и очередь. Сюда также относятся словари, предназначенные для хранения пар ключ/значение.

Коллекции общего назначения работают с данными типа object, поэтому их можно использовать для хранения данных любого типа.

Мощь коллекций состоит в том, что они могут хранить не только встроенные типы, но и объекты любого типа, включая объекты классов, создаваемых программистами.

Коллекции специального назначения ориентированы на обработку данных конкретного типа или на обработку уникальным способом. Например, существуют специализированные коллекции, предназначенные только для обработки строк или однонаправленного списка.

Классы коллекций, ориентированных на побитовую организацию данных, служат для хранения групп битов. Коллекции этой категории поддерживают такой набор операций, который не характерен для коллекций других типов. Например, в биториентированной коллекции BitArray определены такие побитовые операции, как И и исключающее ИЛИ.

Интерфейсы коллекций

Интерфейс	**Описание**
IEnumerable	Определяет метод GetEnumerator(), который поддерживает перечислитель для любого класса коллекции
IEnumerator	Содержит методы, которые позволяют поэлементно получать содержимое коллекции
ICollection	Определяет элементы, которые должны иметь все коллекции
IList	Определяет коллекцию, к которой можно получить доступ посредством индексатора
IDictionary	Определяет коллекцию, которая состоит из пар ключ/значение
IDictionaryEnumerator	Определяет перечислитель для коллекции, которая реализует интерфейс IDictionary
IComparer	Определяет метод compare(), который

	выполняет сравнение объектов, хранимых в коллекции
IHashCodeProvider	Определяет хеш-функцию

Интерфейсы IEnumerable, IEnumerator и IDictionaryEnumerator

Основополагающим для всех коллекций является реализация перечислителя (нумератора), который поддерживается интерфейсами IEnumerator и IEnumerable.

Перечислитель обеспечивает стандартизованный способ поэлементного доступа к содержимому коллекции. Поскольку каждая коллекция должна реализовать интерфейс IEnumerable, к элементам любого класса коллекции можно получить доступ с помощью методов, определенных в интерфейсе IEnumerator.

Следовательно, после внесения небольших изменений код, который позволяет циклически опрашивать коллекцию одного типа, можно успешно использовать для циклического опроса коллекции другого типа (например, в цикле for each).

Класс Hashtable.

Предназначен для создания коллекции, в которой для хранения объектов используется хеш-таблица. В хеш-таблице для хранения информации используется механизм, именуемый хешированием (hashing), Суть хеширования состоит в том, что для определения уникального значения, которое, называется хэш-кодом, используется информационное содержимое соответствующего ему ключа. Хэш-код затем используется в качестве индекса, по которому в таблице отыскиваются данные, соответствующие этому ключу.

Преимущество хеширования — в том, что оно позволяет сохранять постоянным время выполнения таких операций, как поиск, считывание и запись данных, даже для больших объемов информации.

Hashtable-коллекция не гарантирует сохранения порядка элементов.

Класс Hashtable реализует интерфейсы:

- IDictionary,
- ICollection,
- IEnumerable,
- ISerializable,
- IDeserializationCallback,
- ICloneable.

В классе Hashtable определено множество конструкторов, но чаще всего используется public Hashtable()

В классе Hashtable, помимо свойств, определенных в реализованных им интерфейсах, также определены два собственных public-свойства. Используя следующие свойства, можно из Hashtable-коллекции получить коллекцию ключей или значений public Keys и public Values.

Итак, класс System.Collections.HashTable реализует хэш-таблицу – эффективную разновидность ассоциативных коллекций, обеспечивающую быстрый доступ к любому объекту.

Условие задачи

```
Дан стандартный класс HashTable.
Задания:
- объявить набор
- заполнить набор ключами равными номеру элемента и значениями - строками
- отобразить число элементов набора
- отобразить элементы набора
- добавить еще один элемент в набор
```

- отобразить число элементов набора
- отобразить элементы набора оператором For-Next
- отобразить элементы набора, используя интерфейс IDictionaryEnumerator и метод Get Enumerator. Итерации по набору выполнить оператором While - End While
- удалить элемента набора с заданным ключом
- отобразить элементы набора
- копировать набор в массив
- отобразить массив
- очистить набор
- отобразить число элементов набора

Пример решения задачи

```
Module Module_HashTable
  Sub Main()
    Dim i As Byte
    Dim n As Byte
    Dim str As String
N1:   Try
      n = InputBox("Введите число элементов набора:", "Окно ввода", "")
    Catch ex As Exception
      MsgBox("Неверно введено число элементов набора. Повторите ввод.", MsgBoxStyle.Critical)
      GoTo N1
    End Try
    Dim myHashTable As New System.Collections.Hashtable()
    For i = 0 To n - 1
      myHashTable.Add("Key" & i, InputBox("Добавляемая в набор строка - ", "", "Об_" & i))
    Next i
    MsgBox("Число элементов набора после начального ввода - " & myHashTable.Count)
    For i = 0 To n - 1
      str = str & myHashTable.Item("Key" & i) & Chr(13)
    Next i
    MsgBox("Набор после начального ввода:" & Chr(13) & str)
    str = ""
    Dim c, d As String
    c = InputBox("Введите ключ нового " & n + 1 & "-го элемента:", "Окно ввода", "Key" & n)
    d = "Новый_Об"
    myHashTable.Add(c, d)
    MsgBox("Число элементов набора после добавления элемента - " & myHashTable.Count)
    For i = 0 To n
      str = str & myHashTable.Item("Key" & i) & Chr(13)
    Next i
    MsgBox("Набор после добавления элемента:" & Chr(13) & str)
    str = ""
    Dim myEnumerator As IDictionaryEnumerator = myHashTable.GetEnumerator
    While myEnumerator.MoveNext()
      str = str & "Имя ключа - " & myEnumerator.Key & "   Объект - " & myEnumerator.Value & Chr(13)
    End While
    MsgBox("Отображение элементов набора с использованием интерфейса IDictionaryEnumerator и метода GetEnumerator():"
& Chr(13) & str)
    str = ""
    c = ""
    d = ""
M1:   c = InputBox("Введите имя ключа элемента набора, который необходимо удалить:", "Окно ввода", "Key")
    If c.Length > 3 Then
      Try
        d = CDbl(Mid(c, 4))
      Catch ex As Exception
        MsgBox("Элемента с таким ключем нет. Повторите ввод.", MsgBoxStyle.Critical)
        GoTo M1
      End Try
    Else
      MsgBox("Элемента с таким ключем нет. Повторите ввод.", MsgBoxStyle.Critical)
      GoTo M1
    End If
    If Mid(c, 1, 3) = "Key" And IsNumeric(d) And CDbl(d) >= 0 And CDbl(d) <= n Then
      myHashTable.Remove(c)
    Else
      MsgBox("Элемента с таким ключем нет. Повторите ввод.", MsgBoxStyle.Critical)
      GoTo M1
    End If
    MsgBox("Число элементов набора после удаления элемента - " & myHashTable.Count)
    For i = 0 To n
      str = str & myHashTable.Item("Key" & i) & Chr(13)
```

```
        Next i
        MsgBox("Набор после удаления элемента:" & Chr(13) & str)
        str = ""
        Dim myArray As Array = Array.CreateInstance(GetType(String), n)
        myHashTable.Values.CopyTo(myArray, 0)
        For i = 0 To n - 1
            str = str & myArray(i) & Chr(13)
        Next i
        MsgBox("Элементы массива: " & Chr(13) & str)
        str = ""
        myHashTable.Clear()
        MsgBox("Число элементов набора  после очистки - " & myHashTable.Count)
    End Sub
End Module
```

```
Imports System.IO
    Imports System.Collections
    Imports System.Runtime.Serialization
    Imports System.Runtime.Serialization.Formatters

    Module Module_SOAP
      Sub Main()
        Dim nameProblem, AnswerProblem As String
        nameProblem = "1. Ежемесячный отчет в налоговой службе"
        AnswerProblem = "1 Определение списка проверки задолженностей - «Бегунка»"
& vbCr & "2 Проверка «Бегунка» на долг по «Оплате единого налога»" _
        & vbCr & "3 Проверка «Бегунка» на долг по «Оплате подоходного налога»" _
        & vbCr & "4 Проверка «Бегунка» на долг по «Погашению штрафных санкций»"
_
        & vbCr & "5 Изготовление формы 8ДР" _
        & vbCr & "6 Проверка Информационной карты" _
        & vbCr & "7 Проверка формы 8ДР" _
        & vbCr & "8 Проверка формы 1ПП" _
        & vbCr & "9 Проверка формы Расчет Подоходного налога"""
        Dim oProblem1 As New Problem(nameProblem, AnswerProblem)
        nameProblem = "2. Ежемесячный отчет в налоговой службе, если есть долг по
оплате единого налога"
        AnswerProblem = "1 Определение списка проверки задолженностей - «Бегунка»"
_
        & vbCr & "2 Проверка «Бегунка» на долг по «Оплате единого налога»" _
        & vbCr & "3 Устранение долга по «Оплате единого налога»»" _
        & vbCr & "4 Проверка «Бегунка» на долг по «Оплате подоходного налога»»" _
        & vbCr & "5 Изготовление формы 8ДР" _
        & vbCr & "6 Проверка Информационной карты" _
        & vbCr & "7 Проверка формы 8ДР" _
        & vbCr & "8 Проверка формы 1ПП" _
        & vbCr & "9 Проверка формы Расчет Подоходного налога"

        Dim oProblem2 As New Problem(nameProblem, AnswerProblem)
        nameProblem = "3. Ежемесячный отчет в налоговой службе, если есть долг по
оплате подоходного налога"
        AnswerProblem = "1 Определение списка проверки задолженностей - «Бегунка»"
_
        & vbCr & "2 Проверка «Бегунка» на долг по «Оплате единого налога»" _
        & vbCr & "3 Проверка «Бегунка» на долг по «Оплате подоходного налога»" _
        & vbCr & "4 Устранение долга по «Оплате подоходного налога»" _
        & vbCr & "5 Проверка «Бегунка» на долг по «Погашению штрафных санкций»"
_
        & vbCr & "6 Изготовление формы 8ДР" _
        & vbCr & "7 Проверка Информационной карты" _
        & vbCr & "8 Проверка формы 8ДР" _
        & vbCr & "9 Проверка формы 1ПП" _
        & vbCr & "10 Проверка формы Расчет Подоходного налога"

        Dim oProblem3 As New Problem(nameProblem, AnswerProblem)
```

```
        nameProblem = "4. Ежемесячный отчет в налоговой службе, если есть долг по
оплате штрафных санкций"
        AnswerProblem = "1 Определение списка проверки задолженностей - «Бегунка»"
& vbCr & "2 Проверка «Бегунка» на долг по «Оплате единого налога»" _
        & vbCr & "3 Проверка «Бегунка» на долг по «Оплате подоходного налога»" _
        & vbCr & "4 Проверка «Бегунка» на долг по «Погашению штрафных санкций»"
_
        & vbCr & "5 Устранение долга по «Погашению штрафных санкций»" _
        & vbCr & "6 Изготовление формы 8ДР" _
        & vbCr & "7 Проверка Информационной карты" _
        & vbCr & "8 Проверка формы 8ДР" _
        & vbCr & "9 Проверка формы 1ПП" _
        & vbCr & "10 Проверка формы Расчет Подоходного налога"

        Dim oProblem4 As New Problem(nameProblem, AnswerProblem)
        Dim myProblems As New Problems()
        MsgBox("Число элементов в наборе1=" & myProblems.Count)
        myProblems.Add(oProblem1)
        myProblems.Add(oProblem2)
        myProblems.Add(oProblem3)
        myProblems.Add(oProblem4)
        MsgBox("Число элементов в наборе2=" & myProblems.Count)
        Dim path As String
        path = Application.StartupPath
        path = path & "\CB_SOAP.xxx"
        Dim enam As Problem
        For Each enam In myProblems
            MsgBox("Имя     " & enam.Name & vbCrLf & "Решение    " & enam.Answer)
        Next enam
        SerializeToSOAP(myProblems, path)
        myProblems.RemoveAt(1)
        For Each enam In myProblems
            MsgBox("Имя     " & enam.Name & vbCrLf & "Решение    " & enam.Answer)
        Next enam
        myProblems.Clear()
        MsgBox("Обнуляем нашу коллекцию")
        MsgBox("Число элементов в наборе1 - " & myProblems.Count)
        myProblems = DeSerealizeFromSOAP(path)
        MsgBox("выполняем десериализацию")
        For Each enam In myProblems
            MsgBox("Имя     " & enam.Name & vbCrLf & "Решение    " & enam.Answer)
        Next enam
    End Sub

    Sub SerializeToSOAP(ByVal aCollection As Problems, ByVal fName As String)
        Dim fStream As FileStream
        Dim mySOAPFormatters As New Formatters.Soap.SoapFormatter()
        Try
            fStream = New FileStream(fName, FileMode.Create, FileAccess.Write)
            mySOAPFormatters.Serialize(fStream, aCollection)
        Catch e As Exception
```

```
        Throw e
    Finally
        If Not (fStream Is Nothing) Then fStream.Close()
    End Try
End Sub

Function DeSerealizeFromSOAP(ByVal fName As String) As Problems
    Dim FStream As New FileStream(fName, FileMode.Open, FileAccess.Read)
    Dim mySOAPFormatters As New Formatters.Soap.SoapFormatter()
    Try
        Return CType(mySOAPFormatters.Deserialize(FStream), Problems)
    Catch e As Exception
        Throw e
    Finally
        If Not (FStream Is Nothing) Then FStream.Close()
    End Try
End Function

<Serializable()> Public Class Problems
    Inherits System.Collections.CollectionBase
    Public Sub Add(ByVal aProblem As Problem)
        List.Add(aProblem)
    End Sub

    Default ReadOnly Property Item(ByVal Index As Integer) As Problem
        Get
            Return CType(List.Item(Index), Problem)
        End Get
    End Property
End Class

<Serializable()> Public Class Problem
    Private mName As String
    Private mAnswer As String
    Public Sub New(ByVal aName As String, ByVal aAnswer As String)
        mName = aName
        mAnswer = aAnswer
    End Sub
    Public Property Name() As String
        Get
            Return mName
        End Get
        Set(ByVal Value As String)
            mName = Value
        End Set
    End Property
    Public Property Answer() As String
        Get
            Return mAnswer
        End Get
        Set(ByVal Value As String)
```

```
            mAnswer = Value
        End Set
    End Property
End Class
End Module
```

Лекция DynamicEvent

Схема динамической организации и обработки событий

Используется для реализации механизма обработки событий на основе обратных вызовов.

Где-то в классе, генерирующем событие, а GD объявляем событие

```
...
Public Event ИмяСоб(
  ByVal Sender as ИмяКлГдеГенСоб,
  ByVal e      as ИмяКлОписСоб)
```

Где-то в классе, в процедуре, генерирующей событие, генерируем событие

```
...
RaiseEvent ИмяСоб(ИмяКлГдеГенСоб, New ИмяКлОписСоб())
```

В процедуре, обрабатывающей событие, GD объявляем перехватчик события

```
...
Private WithEvents ИмяПерехвСоб as ИмяКлГдеГенСоб
...
```

и пишем процедуру обработки события

```
Public Sub ИмяПерехвСоб_ИмяСоб(
  ByVal Sender as ИмяРешения.ИмяКлПоляБитвы. ИмяКлГдеГенСоб,
  ByVal e      as ИмяКлОписСоб)
  Handles ИмяПерехвСоб.ИмяСоб
...
код обработки события
...
End Sub
```

Где-то в Меин модуля создаем экземпляр класса, в котором генерируется событие

```
Dim ИмяОб as New ИмяКлГдеГенСоб
```

Динамически связываем источник события ИмяОб.ИмяСоб с процедурой обработки события ИмяПерехвСоб_ИмяСоб

```
AddHandler ИмяОб.ИмяСоб, AddressOf ИмяПерехвСоб_ИмяСоб
```

Вызываем метод класса, в котором генерируется событие

```
ИмяОб.ИмяМеинКлГдеГенСоб()
```

Удаляем связь источника события с процедурой обработки события

```
RemoveHandler ИмяОб.ИмяСоб, AddressOf ИмяПерехвСоб_ИмяСоб
```