Лекция № 5

Управление потоком

Вопросы лекции:

1. Операторы условного перехода
2. Операторы циклов
3. Использование оператора With

Управляющие структуры можно разбить на 3 главные группы:

- операторы условного и безусловного перехода;
- циклы;
- операторы With.

Операторы условного перехода определяют, какую из ветвей кода выполнять в зависимости от того, какое значение (True или False) .

Цикл повторяет выполнение некоторого блока кода либо заданное число раз, либо пока некоторое условие не примет значение True или False.

Оператор **With** позволяет выполнить множество действий с одним и тем же объектом **без необходимости каждый раз указывать имя**

Операторы условного перехода

Достаточно часто требуется изменить порядок выполнения команд в зависимости от определенного условия. В VB, как и в других языках программирования, существуют конструкции, которые предназначены для управления порядком выполнения команд. Различают следующие основные типы управляющих конструкций (условных переходов):

If - если определяющее условие может принимать два значения: True/False (истина/ложь);

Select Case - если определяющее условие является выражением, которое может принимать более двух значений;

Try Catch - используется для контроля ошибки и обработки исключений. Позволяет во время работы приложения при возникновении ошибки выполнить набор операторов исключения.

Условные выражения

Основанием для принятия решений в управляющих конструкциях являются условные выражения, в которых используются операторы:

- Равно (=);
- Больше (>);
- Меньше (<);
- Не равно (< >);
- Больше или равно (>=);
- Меньше или равно (<=).

Над условными выражениями можно выполнять действия логической математики: AND (И), OR (ИЛИ), XOR (исключающее ИЛИ), NOT (НЕ) и др.

Оператор условного перехода If…Then

Конструкция If...Then применяется в том случае, когда необходимо выполнить один оператор или группу операторов при определенном условии, т. е. когда значение заданного условия равно True.

Ключевое слово Then располагается в одной строке с If и условным выражением. После окончания работы условия обязательно нужно печатать оператор End If, который должен быть последним в блоке.

Существуют две разновидности данного оператора: однострочный и многострочный.

Однострочный оператор имеет следующий синтаксис:

```
Dim A, B, C As Integer   'при объявлении разделитель - запятая
  A = 12 : B = 5 : C = 7   'при присвоении разделитель - двоеточие
   If A > 10 Then A = A + 1 : B = B + A : C = C + B
   MsgBox("A = " & A & ", B = " & B & " ,C = " & C)
```

Оператор условного перехода If…Then

После оператора Then в строке условия может быть записано несколько операторов, разделяемых двоеточием. Оператор End If в этом случае отсутствует. При многострочной записи код выглядит, следующим образом:

```
If A > 10 Then
A = A + 1
B = B +A
C = C + B
End If
```

Оператор условного перехода If…Then… Else

Конструкция If…Then… Else аналогична конструкции If…Then, но позволяет задать действия, исполняемые как при выполнении, так и при невыполнении условий (оператор Else). При этом синтаксис условного перехода:

If условие Then

(операторы, выполняющиеся, когда условие принимает значение True)

Else

(операторы, выполняющиеся, когда условие принимает значение False)

End If

Оператор условного перехода If…Then… Else

Если нужно проверить, кроме основного, **ещё и другие условия,** используется ключевое слово **ElseIf**. Оператор Else при этом необязателен, но если он присутствует, то должен быть в структуре **последним,** в частности:

```
If x > 10 Then
        Label1.Text = "Значение положительное"
        ElseIf  x = 0 Then
        Label1.Text = "Значение равно нулю"
Else
        Label1.Text = "Значение отрицательное"
End If
```

Структуры If…Then и могут вкладываться друг в друга.

```
If objFilm.ExpDate < Date Then
        MsgBox "Эта пленка никуда не годится"
ElseIf objFilm.Type = "Слайд" Then
        Слайд = Слайд + 1
ElseIf objFilm.Color Then
        ЦветНегатив = ЦветНегатив + 1
Else
        ЧернобелНегатив = ЧернобелНегатив + 1
End If
```

Оператор условного перехода If…Then… Else

Первый оператор проверяет **срок годности** пленки, и если он превышен, то выдается сообщение в окне MsgBox.

Если пленка **годная**, т. е. первое условие False, то выполняется второй оператор, который проверяет свойство – **тип пленки**, слайдовая или нет.

Если плёнка **слайдовая** (True), то выполняется следующий оператор: в количество пленок такого типа **добавляется 1**.

Если же условие не выполняется (False), то выполняется следующая операция ElseIf – проверка **цвета** пленки (цветная или нет).

Если плёнка **цветная** (True), то в счетчик цветных негативных **добавляется 1.**

Если False, то осуществляется переход к последнему оператору – Else и добавление в счетчик **черно-белых** плёнок единицу, после чего операция выбора заканчивается.

Конструкция Select Case

Оператор If…ElseIf хорошо подходит для принятия решений на основе последовательной проверки **уменьшающегося** количества значений по различным условиям. Если же приходится проверять **одно и то же количество значений** в сравнении **с различными условиями,** то более эффективным, простым в записи и удобочитаемым является оператор **Select Case** (Case – ящик).

Синтаксис оператора Select Case:

Select Case значение
Case критерий1
(операторы, выполняемые, когда значение удовлетворяет критерию 1)
Case критерий2
(операторы, выполняемые, когда значение удовлетворяет критерию 2)
дополнительные операторы Case
Case Else
(операторы, выполняемые, когда значение не удовлетворяет ни одному из приведенных критериев)
End Select

Конструкция Select Case

Если значение свойства не будет удовлетворять ни одному из критериев в операторах Case, то управление перейдет к оператору **Case Else**, который всегда находится в самом конце структуры **Select Case**.

Пример оператора условного перехода Select Case:

```
Select Case objFilm.Type
   Case "Слайдовая"
   Слайдовая = Слайдовая +1
   Case "Цветная негативная"
   ЦветнаяНегативная = ЦветнаяНегативная + 1
Case "ЧБ негативная"
   ЧБНегативная = ЧБНегативная + 1
Case Else
   MsgBox "Неизвестный тип пленки"
End Select
```

Операторы циклов

- **For…Next** - заданное число раз;
- **For Each…Next** - для каждого объекта из семейства объектов;
- **Do…Loop** - до тех пор, пока некоторое условие имеет значение True;

Циклы могут быть вложенными: один – внутри другого.

При работе с вложенными циклами существует правило: внутренний цикл должен закончиться раньше, чем начнется внешний.

Цикл For…Next

Если перед выполнением цикла известно, **сколько раз** он должен выполняться, то используется цикл **For…Next**. Это так называемый **арифметический** тип цикла. Число проходов цикла задается значениями **начало** и **конец**, которые могут быть целыми числами, переменными и даже сложными выражениями.

В процессе выполнения переменная **Счетчик** хранит информацию о числе выполненных проходов цикла.

Операторы циклов

Синтаксис цикла:

For Счетчик = начало To конец

(операторы, выполняющиеся при каждом проходе цикла)

Next Счетчик

Пример цикла For…Next

В окне отладки "И**Dim j As Integer**

For j = 1 To 10

Debug.Print "Дубль № " & j 'печать в окне отладки

Next j

В окне отладки "Интерпретация" визуализируется текст:

Дубль № 1, Дубль № 2,… Дубль № 10

Перед каждым проходом цикла Visual Basic сравнивает значения счетчика и аргумента "Конечное значение". Если значение счетчика не превышает установленного конечного значения, то выполняются конструкции тела цикла. В противном случае управление переходит к следующему за Next оператору.

Переменная **Счетчик** должна быть числового типа и допускать операции больше (>), меньше (<) и сложение'(+).

Рекомендуется указывать переменную Счетчик после ключевого слова Next для улучшения читабельности программы, особенно когда несколько циклов вложены один в другой.

Операторы циклов

Пример вложенных циклов For…Next

```
Dim C  As Integer
Randomize  'инициализация генератора случайных чисел
 For A = 1 To 5 'начало внешнего цикла For…Next
C = Rnd( )   ' присвоение переменной C случайного значения
   For B = 1 To 10  'начало внутреннего цикла For…Next,
вычисление '10 случайных чисел с повторением вызова Rnd( ) при
каждом проходе цикла
    MsgBox(C * Rnd( ))  'отображение произведения C * Rnd( )
     Next B 'завершение внутреннего цикла после 10 вычислений
Next A 'завершение внешнего цикла после выполнения 5
вычислений
```

Составляя, например, программу воспроизведения музыкальных произведений можно таким образом сформировать **случайный выбор 10 фрагментов из 5 компакт-дисков.**

Операторы циклов

Операции, которые выполняет программа:

- предварительная подготовка – код начинается с объявления переменной sngC и инициализации генератора случайных чисел;
- начало внешнего цикла For…Next: вызов функции Rnd( ), чтобы присвоить переменной sngC случайное значение;
- начало внутреннего цикла For…Next: этот цикл вычисляет 10 других чисел, повторяя вызов функции Rnd при каждом проходе цикла. Результат отображается в окне Immediate отладки (Debug);
- завершение внутреннего цикла после выполнения им всех 10 вычислений.

После этого, подчиняясь оператору Next A, программа возвращается к началу внешнего цикла, и второй и третий шаги повторяются еще 4 раза.

Оператор For… Next особенно важен при работе с массивами, например, при **заполнении массива** множеством вычисленных значений.

Операторы циклов

Пример работы с одномерным массивом

Объявить одномерный массив, заполнить значениями в соответствии с выбранной формулой и визуализировать его в окне MsgBox():

```
Dim intКвадраты(15) As Integer  'Объявление массива
For a = 0 To 15
intКвадраты(a) = a*a  ' Заполнение массива значениями
MsgBox("Квадрат "& a & " = " & intКвадраты(a))
'визуализация массива
Next a
```

Оператор For…Next очень удобно также использовать при обработке **многомерных** массивов, если организовать вложенные циклы так, чтобы каждый из них соответствовал одному измерению массива.

Операторы циклов

Замечания по поводу циклов For…Next

Для понятности кода следует начинать цикл For… Next **с единицы.**

Исключения могут быть, когда работа проводится с массивом. В этом случае первый номер петли цикла выбирается равным **0**.

В операторе **Next** имя счетчика, завершающем цикл For… Next, имя переменной счетчика в принципе указывать необязательно, т. к. ключевое слово Next автоматически вычислит следующее значение счетчика и отошлет программу в начало структуры.

Но всё-таки имя счетчика необходимо включать в оператор Next, т. к. в этом случае при вложенных циклах For… Next можно сразу опознать, какому циклу принадлежит данный оператор Next.

Следует помнить, что, стоит допустить ошибку со счетчиком, и можно либо пропустить пару важных шагов, либо заставить цикл повторяться бесконечно.

Операторы циклов

Оператор цикла For Each… Next

Ключевым различием в использовании For Each… Next и For … Next является то, что в цикле For Each… Next **не требуется указывать число повторений**.

Такой тип цикла применяется тогда, когда точное количество объектов в группе заранее **неизвестно**.

В операторе For Each (для каждого элемента в группе) с помощью переменной **элемент** определяется тип объекта в семействе, а с помощью аргумента **группа** задается семейство, с которым нужно работать.

Синтаксис цикла For Each… Next:

For Each **элемент** in **группа**
(операторы, выполняемые при каждом проходе цикла)
Next **элемент**

Операторы циклов

Пример использования цикла For Each… Next

Например, группа (коллекция) **Народ** содержит неопределённое количество элементов **Имя**, которые отображаются с использованием цикла For Each… Next.

При выполнении программного кода Имя (тип String) каждого объекта может визуализироваться как в окне сообщений отладки **MsgBox**, так и в окне **Вывод** с помощью оператора Console.WriteLine( )

```
Dim Народ = {"Антон", "Герман", "Сильвия"}
        For Each Имя As String In Народ
            MsgBox("Имя -  " & Имя)
            Console.WriteLine("Имя -  " & Имя)
Next
```

Элементы семейства могут быть данными любого типа. При выполнении цикла For Each… Next порядок прохождения элементов в цикле не определён.

Операторы циклов

Циклы Do…Loop

Все возможные версии оператора Do…Loop предназначены для повторения заданного блока программы до тех пор, пока не будет **выполнено некоторое условие**. Это логический тип циклов.

Синтаксис логического цикла:

Do While *условие*
операторы
Loop

Для того, чтобы решить: продолжать цикл или нет, оператор Do…Loop оценивает выполнение заданного условного перехода типа If…Then.

Примеры применения циклов Do…Loop:

- отображение сообщения об ошибке снова и снова, пока пользователь не введет **верную информацию**;
- чтение данных из файла на диске, пока не будет обнаружен **конец файла**;
- организация холостой работы программы в **течение некоторого времени;**
- выполнение некоторых действий по отношению **ко всем элементам массива;**
- выполнение некоторых действий по отношению ко всем элементам массива или семейства, удовлетворяющих **определенным критериям.**

Операторы циклов

Типы оператора цикла Do…Loop

Операторы **Do…Loop** повторяют выполнение действий, пока некоторый условный оператор внутри цикла не выполнит команду **End** (слово **Loop** означает "петля"):

Do While (условие) **Loop -** цикл выполняется **в случае и до тех пор,** пока заданное условие имеет значение **True** (слово While - значит "пока"), т. е. выполнение кода может **не состояться ни разу**, если заданное условие не является истинным (**логический цикл с предусловием**);

Do … Loop While (условие) – оператор начинает процедуру и выполняет блок кода **один раз**, т. к. условие проверяется после выполнения цикла, а затем **повторяет** выполнение цикла, пока заданное условие имеет значение **True** (**логический цикл с постусловием**);

Do Until (условие) **Loop -** начинает и повторяет выполнение блока кода, **в случае и до тех пор,** пока заданное условие принимает значение **False** (**логический цикл с предусловием**);

Do … Loop Until (условие) - выполняет блок программного кода **один раз**, а затем **повторяет** выполнение, пока заданное условие имеет значение **False** (**логический цикл с постусловием**).

Операторы циклов

Пример оператора Do While (условие) Loop
(цикл с предусловием)

```
Dim ВводимыеИмена As String = " "
Do While ВводимыеИмена <> "Готово"
 ВводимыеИмена = InputBox("Введите Ваше имя или для выхода - слово 'Готово', 'Цикл While с предусловием'")
        MsgBox("Введённое имя " & ВводимыеИмена)
Loop
         End
```

VB трактует код, как "исполнять цикл до тех пор, пока переменная ВводимыеИмена не будет содержать значение "Готово". Если содержанием этой переменной сразу же является текстовая строка "Готово", то цикл Do не выполнится ни разу, и программа выполнит оператор **End**.

После объявления переменной ВводимыеИмена следует придать её нулевое значение, иначе при проверке предусловия может возникнуть неопределённость.

Операторы циклов

Пример оператора Do Loop While (условие)

(цикл с постусловием)

```
Dim Число As Integer
Do
    Число = InputBox("Вводите числа < 10")
    MsgBox("Введённое число " & Число)
Loop While Число < 10
End
```

Такой вариант даёт возможность изменения переменной **Число** до проверки условия, поэтому в случае невыполнения ранее присвоенного переменной значения "Число <10" цикл не будет пропущен, а гарантированно выполнится **один раз**.

Операторы циклов

Использование оператора Do …Until Loop (условие)

```
Dim Отклик As String
Do
  Отклик = InputBox("Нажмите 'Q' и Enter для выхода")
   MsgBox(Отклик)
 Loop Until Отклик = "Q"
 End
```

т. е., в данном примере в смысловом отношении более удобно использовать оператор цикла **Do … Loop Until** (условие), при этом условие записывается более просто: **Отклик = "Q".**

Операторы циклов

Применение оператора While… End While

```
Dim Количество As Integer = 5
 While Количество >= -2
MsgBox("Количество = " & Количество)
        Количество -= 1 'это краткая запись равенства
Количество = Количество - 1
  End While
  End
```

Оператор **While** означает, что цикл работает, пока выполняется условие: "Количество >= -2". Изменение количества от значения 5 до -2 происходит с уменьшением в каждой петле цикла на единицу (Количество -= 1). Такой вариант придаёт гораздо большую гибкость для модификации цикла.

Выход с помощью оператора Exit

Оператор используется в условных выражениях If…Then и Select ...Case, вложенных в основной цикл For…Next для прерывания цикла и перехода к следующим за циклом операторам. Синтаксис этого оператора внутри цикла For…Next имеет вид **Exit For**, а внутри цикла Do … Loop - **Exit Do**.

```
Dim Счётчик, Произведение As Integer
    For Счётчик = 100 To 1 Step -10
      Произведение = Счётчик * 10
MsgBox("Произведение " & Произведение)
Console.WriteLine("Произведение " & Произведение)
If Произведение <= 500 Then Exit For
    Next Счётчик
    MsgBox("Выполнено")
    End
```

При действии цикла визуализация фразы происходит как в окне MsgBox, так и в окне "Интерпретация".

Операторы циклов

Пример с оператором Exit Do

```
Dim y, x As Integer
    y = 10
    x = 1
    Do While y > 1
        MsgBox ("y = " & y)
        If y < 5 Then Exit Do
        y = y - x
    Loop
```

Оператор Exit можно также использовать для выхода из процедур Sub и Function. Синтаксис операторов в этих случаях имеет вид **Exit Sub** и **Exit Function** соответственно.

Эти операторы могут находиться в любом месте тела процедуры и используются в случае, когда процедура выполнила требуемые действия и из неё необходимо выйти.

Операторы циклов

Оператор Continue

Оператор Continue позволяет немедленно перейти к следующей итерации цикла. С помощью его можно осуществлять переход от одной итерации к другой в любом из циклов: For. . . Next, For Each. . .Next, Do. . .Loop.

В качестве примера можно рассмотреть следующий код:

```
Dim i As Integer
        For i = 1 To 5
    If i <= 3 Then Continue For  'позволяет перейти к следующей итерации
'(петле) цикла, т. е будут выполнены тоько циклы с i = 4 и i = 5
            MsgBox(i)
            Console.WriteLine(i)
        Next i
        End
```

После выполнения программы в окне MsgBox и на консоли (в окне "Вывод") будут отображены числа 4 и 5, т. е. петли 1, 2, 3 не проигрываются.

Операторы циклов

Использование оператора Is

Если необходимо выяснить **идентичность двух ссылок** на объекты, то это можно сделать с помощью оператора **Is** (результат будет True или False), например:

```
Dim objObject1 As Object, objObject2 As Object
If objObject1 Is objObject2 Then
MsgBox "Это тот же самый объект!"
Else
MsgBox "Это разные объекты!"
End If
```

Использование оператора With…End With

Если в программе существует последовательность операторов, работающих **с одним и тем же объектом**, можно использовать оператор With…End With с целью **однократного** указания имени объекта для всех последующих операторов. Эта конструкция делает код более компактным и понятным, а его выполнение более быстрым. Например:

```
With objПолнаяОбъективность
.Name =  "Опрос общественного мнения" ' установка свойства Name
.DisplayName ' определение DisplayName
sngRegion = .Area  'присвоение значения свойству sngRegion
intПодтасовка = .Rotate (60)     'определение метода поворота
End With
```

Как видно, конструкция **With…End With** может включать операторы, в которых читаются и устанавливаются свойства, вызываются методы.

Данная конструкция **не задает цикл**: входящие в нее операторы будут выполнены только один раз.

Конструкции, использующие With, можно, как и циклы, **вкладывать одна в другую**. Это бывает удобно, когда необходимо выполнить ряд действий и по отношению к некоторому **объекту**, и по отношению **к одному из содержащихся в нем объектов**.

Объектно-ориентированное программирование

Объектно-ориентированное программирование (ООП) представляет собой методику анализа, проектирования и написания приложений с помощью объектов.

Объектом называется фрагмент программного кода, обладающий свойствами, методами и реагирующий на внешние воздействия – события.

Объекты могут моделировать правила обработки данных, различные ситуации, физические предметы и различные материальные явления материального мира.

В общем случае объект можно представить как **нечто**, состоящее из данных и программного кода, обрабатывающего эти данные.

Для идентификации в программном коде метода или свойство объекта печатают **имя объекта**, ставят **точку**, затем печатают **имя метода или свойства.**

Все объекты объединяются общим термином **тип** платформы Microsoft .NET Framework, который включает в себя в качестве членов: классы (klasses), модули (modules), перечисления (enum) и т. п.

Каждый из классов включают в себя описание однотипных объектов. Таким образом, каждый объект является **экземпляром** определённого класса. Для конкретного класса экземпляр – это не один из объектов

Объектно-ориентированное программирование

Структура класса

Каждый класс содержит члены класса: набор полей, методов, свойств и событий.

- поля – это переменные, принадлежащие классу или экземпляру класса;
- методы – процедуры и функции класса;
- свойство – способ доступа к внутреннему состоянию объекта, имитирующий переменную некоторого типа, позволяющий осуществлять вызов функции;
- событие – сообщение , посылаемое объектом, чтобы сигнализировать о совершении какого-либо действия.

Объявление класса задаёт последовательность расположения полей в памяти и способа вызова функций.

При создании (инициализации) конкретного экземпляра класса (объекта) происходит выделение памяти согласно структуре полей класса.

Объектно-ориентированное программирование

Создание классов

Кроме использования встроенных (стандартных) типов VB, называемых примитивами (primitive types), например, формы, кнопки, метки и т. п., можно создавать и свои собственные. Это делается путём создания класса, а с его помощью и создания экземпляра объекта.

Чтобы создать новый, индивидуально определённый класс, нужно щёлкнуть правой кнопкой мыши по названию проекта, выбрать команду **Добавить - Класс**, назвать класс (вместо имени по умолчанию) именем по своему выбору (при этом к имени класса автоматически присваивается расширение .**vb).**

В окне программного кода появится шаблон, в который добавляется оператор объявления глобальной переменной (члена класса) FirstName:

```
Public Class Сотрудники
    Public Dim FirstName As String
End Class
```

В состав класса можно добавлять новые члены, которыми могут быть событиями (events), полями (fields), методами и свойствами (properties).

Объявление переменной в новом классе:

```
Public Class Сотрудники
Dim Фамилия As New Сотрудники
Фамилия.FirstName = "Пётр"
```

Таким образом, полю FirstName нового члена **Фамилия** класса **Сотрудники** было присвоено строковое значение " Пётр ", т. е. создан новый класс и экземпляр (член) нового класса.

Основными понятиями объектно-ориентированного программирования (ООП) являются инкапсуляция, наследование и полиморфизм.

Объектно-ориентированное программирование

Инкапсуляция

Инкапсуляция представляет собой механизм, который объединяет данные и методы и защищает то и другое как от внешнего вмешательства, так и от неправильного использования. Это означает сокрытие деталей реализации класса внутри него самого.

Например, если есть класс, предоставляющий возможность скачать файл из Интернета, то все операции по соединению с сервером, проведению обмена, все переменные должны быть скрыты внутри этого класса, т. к. нет необходимости пользователям класса видеть все детали реализации процесса. Если всё-таки надо что-либо изменить в классе, расширить его функционал (базовые функции), то в этом помогут наследование и полиморфизм.

Примером реализации инкапсуляции может служить оператор **Private** (локальный). Он обеспечивает закрытость объектов, т. е. запрещает доступ к программным модулям со стороны других форм и модулей.

Объектно-ориентированное программирование

Наследование

Класс может использовать члены другого класса, используя наследование (inheritance).

Наследование - это способность дочернего класса сохранять атрибуты класса-родителя.

Классы редко содержат в себе абсолютно весь функционал. Обычно часть функционала переносится из других классов.

Класс-потомок (child) создается на основе класса-родителя (parent) и может наследовать такие его члены, как, например, поля, методы, наследование которых допускается родительским классом.

В VB2010 абсолютно все классы прямо или косвенно (через цепочку других классов) наследуются от класса **System.Object.**

Для приведённого выше класса **Сотрудники** можно создать класс-потомок **Кассир.**

Объектно-ориентированное программирование

Чтобы создать новый класс-наследник, следует выбрать команду **Добавить – Класс** и присвоить имя новому классу.

Для наследования атрибутов, переменных, свойств, процедур и событий другого класса используется команда Inherits.

Таким образом, в шаблоне кода следует создать команду:

```
Public Class Кассир
    Inherits Сотрудники
End Class
```

При этом класс **Кассир** имеет те же члены, что и класс **Сотрудник.**

Преимущества наследования состоят в том, что позволяет использовать в дочерних классах функционал родительского класса и, при необходимости, дополнять его. При наследовании можно не только добавлять новый функционал, но и изменять существующий. Для этого используется полиморфизм.

Объектно-ориентированное программирование

Полиморфизм

Полиморфизм представляет собой способность к изменению функционала, унаследованного от базового класса.

Как пример, можно привести класс "Фигура", который отображается с помощью метода "Отобразить". В каждом из созданных на его основе классах "Круг". "Квадрат", "Треугольник" можно изменить только функционал по отображению их на экране. Если бы не было такой возможности, пришлось бы заново создавать код, проверяющий тип фигуры, и, в зависимости от него, выбирает метод прорисовки. Полиморфизм позволяет значительно сократить объём кода и повысить его читабельность.

Объектно-ориентированное программирование

Обобщённые классы

Основанием для их появления послужила необходимость создания одного и того же кода для различных типов данных. Примером такого кода могут служить **классы-контейнеры**: списки, массивы, в которых можно хранить объекты разных типов. При разных типах данных и использовании одного и того же кода может нарушиться корректность кода. Концепция обобщённых классов (generic) решает эту проблему, вводя понятие параметра. Каждый класс, структура, интерфейс могут быть параметризованы, при этом конкретный тип параметра может задаваться как на этапе конструирования, так и при выполнении программы.

Например, если необходимо создать список строк, то можно использовать обобщённый класс списков: System.Collections.Generic.List, задав ему в качестве параметра тип данных String:

Dim a As New System.Collections.Generic.List(Of String).

Объектно-ориентированное программирование

Визуальные классы

Создание визуальных классов (форм, кнопок и т. п.) ничем не отличается от создания любого класса объектов.

Элемент управления Windows Forms представляет собой класс, производный от класса **System.Windows.Forms.Control**.

Можно разработать составной элемент управления, объединяя разные элементы управления Windows Forms. Чтобы создать составной элемент управления, нужно сделать его производным от класса **System.Windows.Forms.UserControl**. Базовый класс **UserControl** обеспечивает для дочерних элементов управления работу в группе. Составные элементы управления, производные от **System.Windows.Forms.UserControl**, называются пользовательскими элементами управления.

Они могут расширить функциональные возможности (например, проверять допустимость вводимых пользователем данных, изменять свойства отображения или выполнять другие действия, необходимые разработчику).

Объектно-ориентированное программирование

Объектная модель

Кроме обладания своими собственными свойствами, методами и событиями, объекты высших ступеней иерархии служат в качестве **контейнеров** для одного или целого множества **подчиненных** объектов.

Эти вложенные объекты содержат в свою очередь другие объекты (тоже являясь контейнерами) и т. п. Вся система таких иерархических отношений в VB называется **объектной моделью**, и, соответственно, каждый из объектов имеет свой собственный набор свойств, методов и событий.

Ввиду того, что для VB придется сообщать, какой конкретный объект нужен вам для работы, понимание объектной модели приложения оказывается очень важным для эффективной работы в нем, т. к. тогда очень легко проследить цепочку объектов, которой принадлежит ваш объект.

Для обозначения объекта в объектной модели используется **объектное выражение.** Объектное выражение представляет собой **фрагмент программного кода,** "указывающий" на конкретный объект. С помощью правильно построенного объектного выражения можно изменять свойства объекта, вызывать его методы или присвоить объект переменной.

Объектно-ориентированное программирование

Свойство какого-либо объекта может тоже оказаться объектом. Например, что если объект содержит подчиненные объекты, то любой подчиненный объект можно идентифицировать посредством свойства первого объекта. Выражение, которое используется для указания нужного свойства, является объектным выражением.

Например, имеется объектное выражение, характеризующее свойство:

ObjМузыка.Джаз(5).Воспроизведение

В этом выражении не одна, а две точки. **Воспроизведение** является свойством объекта **Джаз**, который в свою очередь является свойством объекта **Музыка.**

Первая часть указанного выше выражения **Музыка**. **Джаз** – это идентификация конкретного семейства **Джаз**, принадлежащего объекту **Музыка**. После этого можно идентифицировать конкретный член семейства Джаз – **Джаз (5)**, что является ссылкой на 5-й раздел семейства.

Завершающая часть выражения - **Воспроизведение** – это свойство семейства Джаз, а конкретным значением этого свойства является объект **Воспроизведение.**

Таким образом, все выражение обеспечивает **ссылку на этот объект.**

Использование выражения, прокладывающего подобным образом путь к конкретному объекту, называется "**получением объекта**".

Объектно-ориентированное программирование

Объектная переменная

Печатать длинные объектные выражения достаточно трудоемко, даже когда они понятны, поэтому, если в программе один и тот же объект встречается несколько раз, для него **создают переменную**, в которой будет храниться **ссылка на этот объект**. Тогда вместо объектного выражения достаточно просто напечатать имя переменой.

Кроме того, что **объектная переменная** гораздо короче, она имеет еще два преимущества:

- ее использование ускоряет выполнение программного кода, так как VB обращается к объекту напрямую, а не через частокол свойств промежуточных объектов;
- ее можно использовать для сохранения ссылок на различные объекты, и сделать код более гибким, так как появляется возможность выбирать, ссылку на какой объект должна хранить переменная во время выполнения программы.

Процесс создания **объектной переменной** разбивается, как обычно, на **два этапа**:

объявление переменной, которая будет использоваться для ссылки на объект;

```
Dim objФормаОбъекта As Shape   'объект типа Shape
```

присваивание переменной ссылки на объект, с которым в дальнейшем необходимо будет работать, с указанием типа объекта (для этого используется оператор Set).

```
Set objФормаОбъекта = ThisDocument.Pages (1).Shapes (4)
```

Когда доступ к объекту уже не нужен, следует освободить объект **от привязки** к переменной. В результате этого программа может использовать освободившийся участок памяти:

```
Set objФормаОбъекта = Nothing
```