1 ЛАБОРАТОРНЫЙ ПРАКТИКУМ №1

1.1 Операции над числами. Получение цифр числа

Выражения состоят из операций и операндов. Большинство операций в языке Visual Basic являются бинарными, то есть содержат два операнда. Остальные операции являются унарными и содержат только один операнд. В бинарных операциях используется обычное алгебраическое представление, например: a+b. В унарных операциях операция всегда предшествует операнду, например: -b.

Значение выражение i\j представляет собой математическое частное от i/j, округленное в меньшую сторону до значения целого типа. Если j равно 0, результат будет ошибочным.

13,6 \ 3,21 = 4

- сначала операнды округляются – будет 14 и 3

- потом обычное деление

$$\frac{14}{3} = 4\frac{2}{3}$$

- наконец "целая часть" 4 берется как ответ.

Операция mod возвращает остаток, полученный путем деления двух ее операндов, то есть:

$$i \bmod j = i - (i \operatorname{div} j) * j$$

Пример:

13,6 mod 3,21 = 2

- сначала операнды округляются – будет 14 и 3

- потом обычное деление

$$\frac{14}{3} = 4\frac{2}{3}$$

- потом "не разделившаяся" 2 берется как ответ.

Знак результата операции mod будет тем же, что и знак i. Если j равно нулю, то результатом будет ошибка.

Функция Int отбрасывает дробную часть числа и возвращает целое значение. Для отрицательного значения аргумента число функция Int возвращает ближайшее отрицательное целое число, меньшее либо

равное указанному. Например, функция Int преобразует -8.4 в -9

Функция Int возвращает значение типа, совпадающего с типом аргумента, которое содержит целую часть числа.

Int(12.3) = 12

Int(12.8) = 12

Int(-12.3) = -13

Int(-12.7) = -13

1.2 Работа с символьными выражениями

> ***Символьные выражения*** – *это последовательная запись символьных констант, переменных, функций, регламентированная скобками и знаками операций.*

Значением символьного выражения является строка символов. Знаки символьных операций: & или +

конкатенация (сцепление). Позволяют соединять символы в одну символьную строку.

Символьные функции *– это одна из встроенных функций.*

Некоторые встроенные символьные функции:

1. Asc(Строка)

Возвращает значение типа Integer, представляющее код первого символа строки. В результате действия функции возвращается Asc - код ANSI первого символа строки типа Integer.

Строка – обязательный аргумент может представлять любое строковое выражение. Если строка не содержит символов, возникает ошибка выполнения

2. Chr(кодСимвола)

Возвращает значение типа String, содержащее символ, соответствующий указанному коду символа.

Функция Chr возвращают значение субтипа String типа Variant, содержащее символ, соответствующий указанному коду символа ANSI или

Unicode. Использование в параметре CharCode значения больше, чем 255, генерирует ошибки стадии выполнения 5: Invalid procedure call or argument или 6: Overflow.

КодСимвола – обязательный аргумент является значением типа Long, определяющим символ. Обычно, функция Chr применяется при вставке в текстовые строки непечатных символов(возврат каретки, перевод строки, табулятор и т.д.). Коды 0-31 соответствуют стандартным управляющим символам ASCII. Например, Chr(10) возвращает символ перевода строки.

3. Len(Строка)

Возвращает значение типа Long, содержащее число символов в строке.

Функция Len вычисляет число символов в строке или размер заданной переменной. Из двух возможных аргументов должен быть указан только один (и только один). Для определяемых пользователем типов Len возвращает размер, который требуется для записи

переменной в файл. Обязательный аргумент -любое допустимое строковое выражение.

4. Left(Строка, m)

Возвращает значение типа Variant (String), содержащее m левых символов Строки. Функция Left служит для усечения исходной строки до заданной длины. Для определения числа символов в строке следует использовать функцию Len.

Возвращает значение типа *Variant (String),* содержащее указанное число первых символов исходной строки.

Строка – обязательный аргумент, из которого извлекаются символы. Если выражение имеет значение *Null*,то возвращается *Null.*

m – обязательный аргумент, значение типа Variant (Long). Числовое выражение,указывающее число возвращаемых символов. Если равно 0,то возвращается пустая строка (" ").Если значение Length больше либо равняется числу символов в исходной

строке, то возвращается вся строка.

5. Mid(Строка,k,m)

Возвращает значение типа Variant (String), содержащее m символов Строки, начиная с позиции k.

Функция Mid используется для считывания заданного числа символов или байт подряд от заданной позиции в строке слева направо. Нумерация символов в строке всегда начинается с единицы. Для определения числа символов в строке следует использовать функцию Len. Возвращает значение типа Variant (String), содержащее указанное число символов строки.

Строка – обязательный аргумент, строка, из которой извлекаются символы. Если аргумент имеет значение Null, возвращается Null.

k – обязательный аргумент-значение типа Long. Позиция символа в строке String, с которого начинается нужная подстрока. Если Start больше числа символов в строке string, функция Mid возвращает пустую строку ("").

m – необязательный аргумент, значение типа Variant (Long). Число возвращаемых символов. Если этот аргумент опущен или превышает число символов, расположенных справа от позиции Start, то возвращаются все символы от позиции Start до конца строки.

6. Right (Строка,m)

Возвращает значение типа Variant (String), содержащее m правых символов Строки. Для определения числа символов в строке следует использовать функцию Len.

Строка – обязательный аргументиз которого извлекаются символы. Если выражение имеет значение *Null*, то возвращается *Null.*

m – обязательный аргумент, значение типа *Variant (Long)*. Числовое выражение, указывающее число возвращаемых символов. Если равно 0, то возвращается пустая строка (" "). Если значение Length больше либо равняется числу символов в исходной строке, то возвращается вся строка

7. Str(Число)

Возвращает значение типа Variant (String), являющееся строковым представлением Числа.

При преобразовании в начале строки возвращаемого значения резервируется место для знака числа. Если число положительно, то в этом месте будет пробел, если число отрицательно, то выводится знак минус.

В качестве десятичного разделителя дроби функция Str воспринимает только точку. При использовании других десятичных разделителей(например, запятой) следует использовать функцию CStr.

Пример символьного выражения

Пусть

```
Dim a As String
Dim b As String
Dim c As String
Dim z As String
```

a = "Искра", b = "Мэр", c = "Телетайп"

тогда

z = Left(a, 3) + Mid(b, 2, 1) + Right(c,2) будет z = Искэйп

Пример «Преобразование цепочки символов в массив»

Дана цепочка символов $#/,.[[[fdsa5f6. Сформировать одномерный массив, каждый элемент которого есть отдельный символ цепочки.

Обсуждение алгоритма

Следующий код иллюстрирует решение

```
Dim n As Integer
Dim i As Integer
Dim arrSim() As String
Dim Cep As String
n=Len(Cep)
ReDim arrSim(1 to n)
For i= 1 to n
arrSim(i)=Mid(Cep,i,1)
Next i
```

«Пачка символов» *– это два и более, подряд идущих одинаковых символов.*

«Слово» *- это один или несколько символов «в пробелах».*

1.3 Перестановка элементов массива

Массивы в алгоритмических языках служат для работы с упорядоченными наборами элементов одного типа. Каждый элемент массива имеет свой номер, называемый индексом. Если массив многомерный, то по каждому измерению указывается свой номер, например, если массив двумерный, индекс состоит из двух чисел и т. д. Количество измерений массива называется его рангом (rank), или размерностью.

В Visual Basic существует возможность переопределять границу ранее определенного массива (для многомерного массива можно переопределять только границу по самому последнему измерению). Для этого существует оператор ReDim.

Среди множества алгоритмов сортировки рассмотрим алгоритм сортировки элементов массива по возрастанию – «Метод пузырька»

Дан массив символов a = {3, 6, 4, 1, 7, 8, 5}

Получим массив, отсортированный по возрастанию a={1,3,4,5,6,7,8}

Будем сравнивать два соседних элемента. Если предыдущий элемент больше следующего, то их переставим. Будем так действовать n-1 раз. Возможно, после уже нескольких проходов массив окажется отсортированным, тогда введем переменную Flag, которую будем устанавливать в 1 при каждой перестановке, а перед началом просмотра массива устанавливать в 0. После окончания просмотра массива будем проверять Flag на 0. Если Flag =0, то сортировку завершим, не дожидаясь n-1 перебора.

1.4 Конечные и бесконечные суммы слагаемых

Последовательности *– это конечные или бесконечные суммы слагаемых.*

Если удается найти формулу, позволяющую по номеру i найти любое слагаемое U_i, то нахождение

суммы N первых членов последовательности не составляет труда:

```
S=0
For I=1 to N
U=...
S=S+U
Next I
```

Если надо найти сумму бесконечного числа слагаемых с некоторой точностью ε, то дополнительно требуется оценить сумму отбрасываемых слагаемых (их, кстати, тоже бесконечно много).

Если последовательность – это бесконечная сумма чисел с чередующимся знаком и члены последовательности убывают по модулю, то, как доказал Лейбниц, сумма отброшенных членов не превосходит модуля первого из них. Это позволяет для последовательностей такого вида построить простой алгоритм вычисления суммы с заданной точностью.

```
S=0: U=...: k=1
Do While Abs(U)>Eps
S=S+U
U=...
k=k+1
Loop
```

Для вычисления U удобно иметь рекуррентную формулу.

Для ее вывода надо выполнить в общем виде деление следующего члена последовательности на предыдущий:

$$\alpha = \frac{U_{k+1}}{U_k}$$

и тогда получаем формулу

$$U_{k+1} = \alpha U_k$$

Правда, рекуррентная формула не позволяет найти первый член последовательности. Он должен быть найден независимо от рекуррентной формулы.

1.5 «Метод дихотомии», «Метод Золотого сечения», «Метод Фибоначчи»

Унимодальная на отрезке функция, значит, содержащая там один экстремум. Каждый из методов: «Метод дихотомии», «Метод Золотого сечения», «Метод Фибоначчи» предполагают получение на

отрезке [a,b] двух точек λ и μ, которые используются в дальнейшем для сужения интервала неопределенности.

В методе Дихотомии левую точку λ получают, отступив от середины отрезка влево на некоторую величину L/2, а правую точку - отступив от середины отрезка вправо на некоторую величину L/2, при этом стремятся, чтобы L было как можно меньше. Но не стоит забывать, что на концах отрезка длины L значения функции должны быть "компьютерно" различимы.

В методе Золотого сечения левую точку λ получают, прибавив к левому концу отрезка $(1-\alpha)$ единиц от длины отрезка, а правую точку μ - прибавив к левому концу отрезка α единиц от длины отрезка.

Величину α получают в соответствии с правилом Золотого сечения - "отношение большей части ко всему отрезку равно отношению меньшей части к большей части отрезка".

В методе Фибоначчи левую точку λ получают, прибавив к левому концу отрезка отношение

"предпредпоследнего" числа Фибоначчи к последнему числу, умноженное на длину отрезка. Правую точку μ получают, прибавив к левому концу отрезка отношение предпоследнего числа Фибоначчи к последнему числу, умноженное на длину отрезка. Числа Фибоначчи заготавливают заранее. Сколько их надо? Столько, что последнее из них больше того целого числа, которое наименьшее большее, чем величина (b-a)/ε.

1.6 Одномерные массивы

Массив *– это именованный набор объектов одного типа.*

Элемент массива определяется своим местоположением – индексом. Например a_i – a(i) – i-тый элемент массива a, C_{i3} – C(i,3) – элемент i-той строки и 3-го столбца массива C.

Индексация элементов массива По умолчанию начинается с нуля. При желании начать индексацию с

единицы надо использовать оператор Option Base 1 в разделе General проекта. Одномерные массивы бывают статическими и динамическими.

Статические массивы – количество ячеек памяти, занимаемое массивом, не изменяется при выполнении алгоритма. Объявить статический массив можно двумя способами:

1. Имя_массива (Верхн_Знач_Индекса)

Пример:

```
Dim a(5)
```

В памяти будут зарезервированы ячейки: a(0), a(1), a(2), a(3), a(4), a(5)

Пример:

```
Option Base 1
Dim a(5)
```

В памяти будут зарезервированы ячейки: a(1), a(2), a(3), a(4), a(5)

2. Имя_массива (Нижн_Знач_Индекса to Верхн_Знач_Индекса)

Пример:

```
Option Base 1
```

```
Dim a(2 to 5)
```

В памяти будут зарезервированы ячейки: a(2), a(3), a(4), a(5)

Динамические массивы – количество ячеек памяти, занимаемое массивом не известно до выполнения алгоритма. Для объявления динамического массива необходимо выполнить следующие действия:

Шаг 1:

```
Dim имя_Массива() As Тип
```

Шаг 2:

```
ReDim имя_Массива(Нижн_Знач_индекса to
Верх_Знач_индекса)
```

Пример:

```
Option Base 1
Dim a() As Single
n=InputBox(“Введите число элементов n”,”Окно
ввода n”,6)
ReDim a(1 to n)
...
```

Ввод одномерного массива:

```
...
Dim a() As Double
n = InputBox("Введите n", "Окно ввода n", 5)
ReDim a(1 To n)
For i = 1 To n
  a(i) = InputBox("Введите элемент массива", "",
1)
Next i
...
```

Вывод одномерного массива:

```
...
For i = 1 To n
  Print Spc(1); a(i);
Next i: Print
...
```

1.7 Двумерные массивы

Двумерный массив *– это именованный набор объектов одного типа, индексируемый по строкам и столбцам.*

Главной диагональю двумерного массива называется совокупность диагональных элементов матрицы A(m-1, n-1), берущей своё начало с элемента a(0,0). Побочной же диагональю двумерного массива называется совокупность диагональных элементов матрицы A(m-1, n-1), берущей своё начало с элемента a(0, n-1).

Объявить статический массив можно следующим образом:

```
Option Base 1
Dim C(3,4) As single
```

В памяти будут зарезервированы ячейки: C(1,1),C(1,2),C(1,3),C(1,4),

C(2,1),C(2,2),C(2,3),C(2,4),

C(3,1),C(3,2),C(3,3),C(3,4)

```
Option Base 1
Dim C(1 to 3,3 to 4) As single
```

В памяти будут зарезервированы ячейки: C(1,3),C(1,4),

C(2,3),C(2,4),

C(3,3),C(3,4)

```
Option Base 1
Dim C(3,3 to 4) As single
```

В памяти будут зарезервированы ячейки: C(1,3),C(1,4),

C(2,3),C(2,4),

C(3,3),C(3,4)

Динамический массив можно объявить следующим образом:

```
...
Option Base 1
Dim C() As Single
m=InputBox("Введите число строк m","Окно ввода
m",3)
n=InputBox("Введите число столбцов n","Окно ввода
n",4)
```

```
ReDim C(1 to m, 1 to n)
...
```

Ввод двумерного массива:

```
...
For i = 1 To m
For j = 1 To n
a(i, j) = InputBox("Введите элемент массива", "",
1)
Next j
Next i
...
```

Вывод двумерного массива:

```
...
For i = 1 To m
For j = 1 To n
Print Spc(1); a(i, j);
Next j: Print
Next I
...
```

2 ЛАБОРАТОРНЫЙ ПРАКТИКУМ №2

2.1 Понятия класса. Организация свойств классов

С одной стороны, класс можно рассматривать как еще один из типов данных ссылочного типа и определять переменные привычным образом, например: Dim a as EmptyClass. С другой стороны, классы представляют собой особый тип данных. Переменные этого типа называют экземплярами класса или объектами.

Одной из особенностей классов является наличие процедуры инициализации экземпляра класса (Sub New), называемой конструктором. Для хранения внутренних данных в классе можно описывать переменные. Фрагмент кода:

```
Class MyClass
Public a As Integer
Private b As Integer
Public Function GetBO As Integer
Return b
End Function
End Class
```

Открытые переменные используются крайне редко: дело в том, что они приводят к потенциальным

ошибкам. Предположим, описан класс «автомобиль». Если его переменная, в которой хранится количество колес, объявлена открытой, то любой пользователь класса сможет изменить количество колес в автомобиле, как ему заблагорассудится. Хороший класс не даст возможности «испортить» его внутреннее состояние. Класс «автомобиль», к примеру, должен позволить задать количество колес лишь один раз при создании нового объекта и проследить, чтобы это количество не было отрицательным или слишком большим.

Свойства *– это то, что определяет состояние объекта.*

Свойства *– это поля класса или процедуры свойств.*

Открытое поле *– это Public переменная класса.*

Закрытое поле *– это Private переменная класса.*

Свойства определяют атрибуты объекта. Свойства реализуют механизм доступа для чтения или изменения данных в полях объекта.

Вернувшись к нашему примеру со столом, можно сделать следующее сравнение: поле "количество ножек" объекта "стол" будет хранить целое число, обозначающее количество ножек, а свойство "ножки" объекта "стол" может изменять это значение. С помощью данного свойства можно задать "количество ножек" равным трем, четырем и т. д. Свойства позволяют изменять атрибуты объекта, используя в том числе и вычисляемые значения.

Свойства — это особый механизм, который позволяет создавать более элегантные реализации классов. Рассмотрим общий вид организации процедуры свойства:

Объявляется закрытая переменная класса

dim ИмяЗакрытойПеременной as ТипСвойства

```
      Public Property ИмяСвойства(Параметры) as
ТипСвойства
      get
      Return ИмяЗакрытойПеременной
```

```
      end get
      set (ByVal арг as ТипСвойства) as ТипСвойства
      ИмяЗакрытойПеременной = арг
      end set
      end Property
```

Так организованное свойство позволяет себя устанавливать и читать.

Свойство только для чтения

dim ИмяЗакрытойПеременной as ТипСвойства

```
      Public ReadOnly Property ИмяСвойства(Параметры)
as ТипСвойства
      get
      Return ИмяЗакрытойПеременной
      end get
      end Property
```

Свойство только для записи

```
      dim ИмяЗакрытойПеременной as ТипСвойства
      Public WriteOnly Property ИмяСвойства(Параметры)
as ТипСвойства
      set (ByVal арг as ТипСвойства) as ТипСвойства
      ИмяЗакрытойПеременной = арг
      end set
      end Property
```

Свойство только для Одной записи и многократного чтения

```
      Public Property ИмяСвойства(Параметры) as
ТипСвойства
      get
      Return ИмяЗакрытойПеременной
      end get
      set (ByVal арг as ТипСвойства) as ТипСвойства
      if NOT PropertyAllreadySet then
      ИмяЗакрытойПеременной = арг
      PropertyAllreadySet=true
      else
```

```
msgbox("PropertyAllreadySet")
end if
end set
end Property
```

2.2 Организация методов классов

Метод *– это то, что «умеет» объект.*

Метод *– это Sub или Function процедура класса.*

Методы классов используются для программирования различных действий. Имя для метода выбирается в первую очередь именно с целью обозначить действие, выполняемое методом (по крайней мере, этой рекомендации стоит придерживаться).

Нередки случаи, когда действия, различные с точки зрения реализации, очень сходны по своей сути. Например, метод объекта экран, который печатает на экране целое число, отличается от метода, печатающего число вещественное.

При печати вещественного числа необходимо учитывать определенные соглашения о формате вывода числа (обычная или экспоненциальная форма; точность вывода, то есть количество знаков после запятой; стиль самой запятой — в виде запятой или точки и т. п.). Однако с точки зрения логики эти действия одинаковы — вывод на экран числа.

К счастью, Visual Basic позволяет определять не один, а несколько методов, имеющих одно и то же имя, при условии, что набор аргументов у них будет различаться. При попытке вызова метода будет выбрана версия, набор аргументов которой совпадет с набором фактически переданных значений. В нашем случае можно определить два метода, печатающих числа:

```
Class Screen
Public Sub PrintValue(ByVal N As Integer)
End Sub
Public Sub PrintValue(ByVal N As Float)
End Sub
End Class
```

Описанный механизм называется перегрузкой методов. Если объект некоторого класса может быть создан на основе различных данных, то перегрузка

метода New() (конструктора класса) — уже не удобство, а необходимость. Разреженный массив (объект типа SparseArray) может быть создан не только на основе целого числа (максимально возможного количества элементов), как это сделано в примере, но и на основе другого объекта типа SparseArray (такая потребность возникает, если пользователь захочет скопировать содержимое одного разреженного массива в другой).

Выбор перегруженных методов определяется числом параметров, а при их равенстве – их типом. В перегрузке методов реализуется принцип полиморфизма.

2.3 Организация конструкторов классов

Конструктор *– это Sub процедура класса с именем New и параметрами, используемыми для*

инициализации полей класса в момент создания объекта.

При создании каждого нового экземпляра класса вызывается конструктор (метод New()). В качестве аргумента конструктор принимает целое число — максимальное количество ячеек, которое разреженный массив может хранить. В теле конструктора, во-первых, выделяется необходимый объем памяти под массивы IndexQ и Value(); во-вторых, инициализируется переменная MaxCount (массивы пусты, следовательно, MaxCount = 0). Ибо созданный объект без инициализированных членов потенциально опасен при его использовании. Где-то при создании объекта пишем:

Dim ИмяОб as ИмяКласса(арг1, арг2,...)

Где-то в классе объявляем Поля

Dim ИмяПоля1 as тип

Dim ИмяПоля2 as тип

и пишем конструктор

Public Sub New (арг1 as тип, арг2 as тип,...)

ИмяПоля1=арг1

ИмяПоля2=арг2

...

End Sub

Перегрузка конструкторов позволяет порождать от класса объекты, отличающиеся инициализированными полями.

В перегрузке конструкторов реализуется принцип полиморфизма.

Где-то при создании объекта пишем

Dim ИмяОб1 as ИмяКласса(арг1, арг2,...)

Dim ИмяОб2 as ИмяКласса(арг1, арг2,арг3,...)

Где-то в классе объявляем Поля

Dim ИмяПоля1 as тип

Dim ИмяПоля2 as тип

Dim ИмяПоля3 as тип

и пишем конструктор 1

Public Sub New (арг1 as тип, арг2 as тип,...)

ИмяПоля1=арг1

ИмяПоля2=арг2

...

End Sub

и конструктор 2

Public Sub New (арг1 as тип, арг2 as тип, арг3 as тип,...)

ИмяПоля1=арг1

ИмяПоля2=арг2

ИмяПоля3=арг3

...

End Sub

Бывает удобно писать так

Public Sub New (арг1 as тип, арг2 as тип, арг3 as тип,...)

MyClass.New(арг1,арг2)

ИмяПоля3=арг3

...

```
End Sub
```

Выбор перегруженных конструктов определяется числом параметров, а при их равенстве – их типом.

2.4 Организация наследования

При помощи агрегации и наследования реализуются отношения между различными объектами. Агрегация соответствует отношению содержит (has-a), а наследование — отношению является (is-a).

В соответствии с философией ОА и ОП, когда виртуальная модель системы есть совокупность взаимосвязанных объектов, когда один объект есть частный случай другого, наследование играет существенную роль.

Наследование — один из ключевых аспектов ООП. Ни один язык программирования не может называться объектно-ориентированным, если он не включает поддержку наследования. Пожалуй, в

современном программировании наследование используется чаще всего для создания классов на основе уже существующих (библиотечных). В этом уроке мы разработали класс SparseArray. Допустим, он широко используется в наших проектах, его интерфейс закреплен на бумаге, и менять его реализацию очень нежелательно.

Базовый класс, тот, которого наследуют.

Производный класс, тот, кто наследует.

В производных классах необходимо уделить большое внимание конструкторам. Дело в том, что производный класс состоит из двух частей — части, определенной в самом производном классе, и части, унаследованной от базового. Часть, унаследованная от базового класса, сама по себе является полноценным классом, инициализируемым одним из конструкторов, определенных в базовом классе.

Таким образом, в конструкторе любого производного класса следует выполнить два действия: сначала вызвать один из конструкторов базового

класса для конструирования унаследованной части, а уже после этого инициализировать переменные, объявленные в производном классе. Вызов одного из конструкторов базового класса в первой же строке конструктора производного класса — действие обязательное; в противном случае описание класса будет некорректным.

Общий вид наследования

Inherits ИмяБазового класса

Глубина наследования не ограничена.

Словом MyBase в производном классе может быть «взят» член базового класса в случае его переопределения в производном классе.

MyBase.ИмяЧленаБазовогоКласса

Для использования в производном классе членов базового класса более высокого уровня, нежели один, требуется, чтобы используемый член был отмечен словом Protected в базовом классе

Например, Protected ReadOnly Rroperty ИмяСвойства as ТипСвйства

И тогда в производном классе можно воспользоваться Protected членом без образования экземпляра базового класса. В классах же, не входящих в иерархию классов, воспользоваться Protected членом нельзя.

2.5 Обработка событий

По терминологии корпорации Microsoft,

> ***событие*** *— это сообщение[1], посылаемое некоторым объектом (который может как принадлежать приложению, так и нет), чтобы оповестить о некотором действии.*

События в системе могут происходить по самым разнообразным причинам:

-Пользователь нажимает клавиши на клавиатуре, перемещает мышь и нажимает кнопки мыши, изменяет размеры окон, загружает и выгружает приложения.

-Система Windows сама генерирует различные события, например, определяет, что некоторое окно было ранее закрыто другим, а теперь стало активным и его содержимое необходимо перерисовать.

-Событие может возникнуть внутри приложения, например, по истечении заданного интервала времени после его запуска.

В соответствии с философией ОА и ОП, когда виртуальная модель системы есть совокупность взаимодействующих объектов, события, которые генерируются и обрабатываются объектами, играют заметную роль.

Модель обработки событий в VB .NET

Источник генерирует событие, но на него откликаются лишь те ОбПол, которые на него «подписались». ОбПол заранее «подписавшиеся» на событие, которое они собираются обрабатывать, зарегистрированы ОбИст.

При возникновении события, ОбИст вызывает определенный метод ОбПр. Поэтому ОбПр и их методы, обрабатывающие события, должны быть заранее зарегистрированы в ОбИст – оповещение путем обратного вызова – Call Back Notification

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 Main 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.ИмяMainКлГдеГенСоб()

Удаляем связь источника события с процедурой обработки события:

RemoveHandler ИмяОб.ИмяСоб, AddressOf ИмяПерехвСоб_ИмяСоб

Можно сказать, что событие проявляет себя в шести точках:

1 Точка объявления события

2 Тока генерации

3 Точка перехвата события

4 Точка обработки события – обработчик

5 Точка связывания события с обработчиком

6 Точка удаления связи события с обработчиком

Организация событий на основе классов событий

Всякое событие создается по умолчанию на основе стандартного класса System.EventArgs.

Чтобы создать событие на основе своего класса события, надо создать класс события, наследовав стандартный класс EventArgs и включить в него необходимые члены, которые станут доступны обработчику события при его обработке.

Схема организации собственного класса события:

```
Public Class ИмяСвоегоКлассаСобытия
Inherits System.EventArgs
...
End Class
```

Далее, в точках, где требуется ИмяКлОписСоб, используем ИмяСвоегоКлассаСобытия.

3 ЛАБОРАТОРНЫЙ ПРАКТИКУМ №3

3.1 Абстрактные классы

Значение ключевого слова _abstract (абстрактный) очень похоже на значение ключевого слова abstract (абстрактный) в языке Java. Оно также напоминает о сложившейся традиции рассматривать класс C++, содержащий хотя бы одну чистую (pure) виртуальную функцию, как абстрактный. Ключевое слово _abstract (абстрактный) делает это объявление явным. Как и в случае ключевого слова _interface (интерфейс), ключевое слово _abstract (абстрактный) используется для обозначения того, что класс определяет некоторые общие обязательные соглашения между кодом, реализующим методы этого абстрактного класса, и кодом клиентов, вызывающих эти методы. Обратите внимание, что, если абстрактный класс определяется как управляемый, в его описании следует использовать также и ключевое слово _gc (сборщик мусора).

Класс Number, описанный в примере ниже, не имеет самостоятельного применения, а используется только в качестве базового для других классов. Классы, подобные ему, называются абстрактными. Для описания абстрактных классов в Visual Basic имеются специальные средства. Описание класса Number как абстрактного выглядит так:

```
Public Mustlnherit Class Number
Public Value As Integer
Public Sub NewO
Value = 0
End Sub
Public MustOverride Function AsStringO As String
End Class
```

Абстрактный класс подобен интерфейсу в том, что он является лишь средством проявления полиморфизма, а создать экземпляр такого класса непосредственно нельзя. Однако, в отличие от интерфейса, абстрактный класс может содержать реализации нескольких, или даже всех своих методов. Абстрактный класс может быть использован как базовый для других классов, экземпляры которых можно инициализировать, причем переменная абстрактного ссылочного типа (т.е. ссылка или указатель, но не тип значения) может использоваться

для обращения к экземплярам классов, производных от абстрактного класса.

Обратите внимание на то, что использование ключевого слова _abstract (абстрактный) вместе с _interface (интерфейс) (это слово не является расширением управляемости) является избыточным, так как интерфейсы по определению являются абстрактными. Ключевое слово _abstract (абстрактный) нельзя использовать в комбинации с ключевыми словами _value (значение) или _sealed (конечный). Ключевое слово _value (значение) указывает на то, что объект содержит непосредственно данные, а не ссылки на объекты в динамически распределяемой области памяти. Это значит, что можно создавать экземпляры такого класса, а следовательно, он не может быть абстрактным. Ключевое слово _sealed (конечный) означает, что класс не может быть базовым для других классов, что, очевидно, противоречит концепции абстрактного класса. В следующем фрагменте приведен пример типичного использования ключевого слова _abstract (абстрактный).

Абстрактный класс не может иметь объектов, так как в нем не определены операции над объектами; объекты должны принадлежать конкретным подклассам абстрактного класса. Абстрактные классы используются для спецификации интерфейсов операций (методы, реализующие эти операции впоследствии определяются в подклассах абстрактного класса). Абстрактные классы удобны на фазе анализа требований к системе, так как они позволяют выявить аналогию в различных, на первый взгляд, операциях, определенных в анализируемой системе.

Ключевое слово Mustlnherit в заголовке класса означает, что данный класс предназначен только для создания производных классов; объекты же самого класса Number создавать запрещено (Visual Basic просто не позволит создать объект абстрактного класса).

Атрибут MustOverride используется для декларирования функции, реализация которой предоставляется производному классу. Если класс содержит хотя бы одно описание с атрибутом

MustOverride, то сам класс обязательно должен быть описанс атрибутом Mustlnherit.

3.2 Полиморфизм

Полиморфизм – едва ли не самый впечатляющий инструмент в арсенале программиста, специализирующегося в ООП. Полиморфизм означает возможность создания различных классов, содержащих одинаково именованные методы или свойства, таким образом, чтобы их можно было заменять друг на друга незаметно для участков кода, эти классы использующих.

Рассмотрим простую иерархию классов, изображенную на рис. 3.1.

Класс Number содержит одно-единственное целое число Public Value As Integer и декларирует операцию Function AsStringO As String перевода этого числа в строковую форму (то есть в классе операция не определена; указано лишь, что она существует). Класс

RomanNumber – прямой потомок класса Number. В классе RomanNumber определена операция перевода числа в строковую форму, при этом полученная строка будет содержать запись числа в римской системе исчисления (число 19 будет переведено в строку XIX). Класс ArabianNumber отличается от класса RomanNumber только тем, что строка, возвращаемая функцией перевода числа в строку, содержит запись числа арабскими цифрами (число 19 переводится в строку 19).

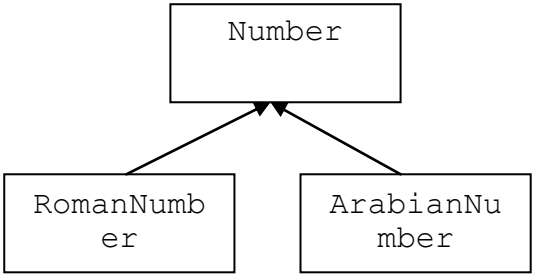

Рис 3.1 – Демонстрационная иерархия классов

В проекте описана процедура, которая выводит на экран число, содержащееся в классе Number:

```
Sub ShowNumber(ByVal N As Number)
MsgBox(N.AsStringO)
```

```
End Sub
```

Заметим, что наследование соответствует отношению «является». В нашем случае можно сказать, что RomanNumber и ArabianNumber являются также объектами типа Number (или, по-русски, римские и арабские числа — это числа). Для программиста этот факт означает, что в качестве аргумента процедуре ShowNumberQ можно передавать не только объекты типа Number, по и объекты типа RomanNumber или ArabianNumber. Соответственно, каждый раз вызов функции N.AsString() будет иметь разный смысл: если фактический тип N есть RomanNumber, то произойдет вызов функции RomanNumber.AsString(), а если тип N есть ArabianNumber, то будет вызвана функция ArabianNumber.AsString():

```
      Dim RN As New RomanNumberO
      Dim AN As New ArabianNumberO
      AN.Value = 1998
      RN.Value = 1998
      ShowNumber(AN) 'вызывается
ArabianNumber.AsStringO: на экране - 1998
      ShowNumber(RN) 'вызывается RomanNumber.AsStringO:
на экране - MCMXCVIII
```

Такое поведение и называется полиморфным (то есть многоформенным, многовариантным). Раньше

описания функций давали нам полную информацию о типах передаваемых параметров: если в заголовке функции указано, что аргумент idx имеет тип Integer, значит, действительно будет передано значение типа Integer (в противном случае программа попросту не откомпилируется). Теперь мы видим, что тип формального аргумента может отличаться от типа фактически переданного аргумента, если эти типы составляют иерархию.

С одной стороны, полиморфизм – мощное средство для создания процедур, оперирующих целым семейством самых разнообразных объектов, при условии, что с каждым переданным объектом можно разговаривать на одном и том же языке, то есть путем использования одних и тех же методов и свойств. На практике нет нужды писать отдельные процедуры для сортировки массива объектов «учетная карточка» и массива объектов «автомобиль». Достаточно в каждом из этих классов определить методы, позволяющие сравнивать между собой два объекта класса и менять местами объекты в массиве. После этого можно

написать общую процедуру, которая с одинаковым успехом будет сортировать как автомобили, так и учетные карточки.

С другой стороны, использование полиморфизма может существенно облегчить процесс проектирования приложения. Верхние уровни иерархии классов можно рассматривать как определения интерфейсов, которые будут использоваться на всех последующих уровнях. Уже на этапе проектирования можно договориться, какой «язык» будет использоваться для «общения» с объектами, и реализовать упрощенные классы-прототипы. В процессе разработки классы-прототипы будут заменяться последующими версиями без особых проблем, поскольку при условии неизменности интерфейса процедуры, работающие с классами, переписывать не потребуется.

Полиморфизм организуется при помощи двух ключевых слов — Overridable и Overrides. Атрибут Overridable используется на уровне базового класса, чтобы указать, что данная функция может переопределяться в производных классах. В нашем

примере атрибут Overridable применен при описании метода AsStringQ в классе Number. Атрибут Overrides используется на уровне производного класса, чтобы указать, что данная функция переопределяет соответствующую функцию базового класса. В нашем примере атрибут Overrides применен при описании метода AsString() каждого из производных классов (RomanNumber и ArabianNumber).

Такой синтаксис имеет довольно неприятное следствие: поскольку возможность переопределения необходимо указать на уровне базового класса (при помощи слова Overridable), то вы должны заранее, уже во время разработки базового класса определить, какие из методов могут переопределяться в производных классах, а какие нет. Не стоит, однако, определять все методы как Overridable только на случай того, что в будущем неожиданно может возникнуть желание какой-то из них переопределить. Дело в том, что полиморфное поведение не дается даром: вызов полиморфных методов происходит медленнее, а объекты классов, использующих полиморфизм,

расходуют больше памяти. Отметим еще, что переопределяемые методы иначе называют виртуальными.