Использование классов при разработке приложений. Примеры.

Объектно–ориентированное программирование – это методология разработки программ, основанная на представлении программы в виде совокупности объектов, каждый из которых является реализацией определенного класса. В объектно–ориентированном программировании объект представляет собой элемент приложения, например лист, ячейку, диаграмму, форму или отчет. Программный код и данные структурируются так, чтобы имитировалось поведение фактически существующих объектов. В объектно–ориентированном программировании важную роль играют четыре понятия. При *абстрагировании* реальные процессы ограничиваются их функциями, существенными для программирования. Внутреннее содержимое объекта защищается от внешнего мира посредством *инкапсуляции.* Благодаря *наследованию* уже запрограммированные функциональные возможности можно использовать и для других объектов. *Полиморфизм* позволяет использовать различные объекты и по-разному реализуемые функции под одним именем.

Вы уже работали с объектами, возможно, и не замечая этого. Объекты являются программным представлением физических и/или логических сущностей реального мира. Они необходимы для моделирования поведения физических или логических объектов, которые они представляют. Например, элемент управления TextBox служит для отображения или редактирования текста в форме. Для добавления элемента управления TextBox в форму не нужно писать ни строчки – можно просто воспользоваться готовым объектом TextBox из панели элементов. Для изменения поведения и состояния элементов управления используются их свойства и методы.

Классы

На панели элементов Visual Basic находятся, строго говоря, не объекты, а классы. Различие между классами и объектами состоит в том, что объекты в Visual Basic существуют только во время выполнения. Классы же используются для задания структуры объектов. Во время выполнения создаются копии классов. Благодаря этому для данного класса можно создавать произвольное количество объектов. С другой стороны, классы образуют группы одноименных объектов.

Абстрагирование.

Человек использует абстрагирование для того, чтобы проще описывать сложные объекты окружающей среды. Представьте себе, что в обычном языке не существует объектов (понятий), абстрагирующих окружающий мир. Тогда для того, чтобы сообщить другому человеку о каком–то предмете, например автомобиле, пришлось бы его подробно описывать. Трудно вообразить, сколько времени потребовалось бы для описания объекта. Причем описание должно быть очень точным, чтобы не возникло ошибочное представление о другом объекте. То же относится и к программированию. Например, для использования текстового окна (TextBox) не нужно разрабатывать специальный драйвер графической карты. Вы просто используете объект класса TextBox. Этот абстрагированный объект содержит все, что нужно для редактирования текста в Windows.

Если вы создаете собственное приложение, то большую помощь при этом окажут собственные объекты. В исходном коде вы используете программное представление таких объектов, как фирмы, служащие, товары, счета и многое другое. Такой способ рассмотрения больше соответствует реальности, чем при чисто процедурной разработке приложений.

Описывается применение классов внутри проекта (Standard EXE). Для использования классов в элементах управления ActiveX существуют дополнительные возможности, на которых мы подробнее остановимся в следующих главах. В Visual Basic классы используются в качестве шаблонов объектов. С их помощью во время выполнения можно создавать любое количество объектов одного класса. Внутренняя структура класса передается объекту посредством модуля класса. Таким образом, класс описывает семейство объектов, а каждый экземпляр класса является уникальным представителем этого семейства. При написании программ, для имитации поведения реальных объектов, в объектах объединяются как принципы действия, так и данные.

Модуль класса

Для создания собственного объекта нужен шаблон. Эти шаблоны в Visual

Basic представляют модули классов. С помощью команды Add Class Module меню Project или соответствующей кнопки панели инструментов Visual Basic такой модуль класса можно добавить в проект. После добавления модуля класса появляется пустое окно кода, в котором можно реализовать класс. В проектах типа Standard.EXE классы имеют только одно свойство – имя класса. В среде разработки Visual Basic все модули класса сохраняются как отдельные файлы с расширением CLS.

Форма

В качестве классов могут также применяться и формы, в частности если необходимо, чтобы создаваемый объект функционировал как окно. Все

описанные здесь действия можно выполнять как с модулем класса, так и с формой. После добавления модуля класса или формы в проект можно создавать свойства и методы.

Мастер классов

Visual Basic предоставляет и другую возможность создания классов. Если на

вкладке Environment диалогового окна Tools\0ptions в группе Show Templates For: активизировать опцию Class Modules, то при добавлении модуля класса отобразится диалоговое окно Add Class Module. С е го помощью можно либо создать пустой модуль класса, либо запустить мастер классов Visual Basic (утилита Class Builder). Для редактирования уже существующих в проекте классов эту утилиту можно вызвать также посредством команды Class Builder Utility меню Add\_Ins из окна кода Visual Basic. При помощи команды New\CIass меню File данного мастера можно добавить новый модуль класса. В открывающемся диалоговом окне Class Module Builder следует задать имя класса и, при необходимости, указать ряд других параметров.

Свойства

Чтобы можно было использовать объекты класса (class или Form), необходимо создать соответствующие свойства и методы. Свойства в общем случае предназначены для изменения значений переменных состояния объектов. Например, свойство Text текстового окна позволяет задавать либо считывать значение отображаемой строки. При этом обычно объект не выполняет никаких действий. Свойства, не соответствующие этому правилу (например, CommonDialogI. Action = 1), следует заменять методами, реализующими аналогичные функции (CornmonDialogi. showOpen). Есть две возможности реализации свойств в классах: использование процедур свойств, позволяющих контролировать значения свойств, или объявление в классе переменных общего доступа (Public). Однако переменные, объявленные как Public, не обладают всеми возможностями процедур свойств.

Процедуры свойств

Кроме известных процедур Sub и Function, в Visual Basic есть и третья категория – процедуры Property, с помощью которых можно определять свойства класса. Чтобы процедуры были видимыми вне собственного контейнера, их следует объявлять как Public. Поскольку значения свойств можно как считывать, так и устанавливать, то для одного свойства могут потребоваться две процедуры с одним и тем же именем: одна для чтения, другая для присвоения значения свойства. Чтобы различить их, в заголовке процедуры используется дополнительное ключевое слово (Let или Get).

*Оператор Property Let* Явное использование необязательного ключевого слова Let зависит от стиля программирования, поэтому при присваивании значения переменной оно обычно не указывается. Однако в процедуре присваивания значения свойству использование ключевого слова Let обязательно, так как в этом случае оно указывает, что данная процедура является процедурой задания значения свойства.

Public Property Let Property\_Name (ByVal vNewValue As Variant)

mVariable = vNewValue

End Property

Переменная vNewValue содержит значение, присваиваемое свойству. При

этом тип данных этой переменной должен соответствовать типу данных

свойства.

*Оператор Property Get* Процедура считывания значения свойства объявляется с использованием ключевого слова Get.

Public Property Get Property\_Name () As Variant

Property\_Name = mVariable

End Property

Здесь, как и в процедуре Function, возвращаемое значение присваивается имени процедуры. При создании процедуры Property Get следует обратить внимание на то, что она использует тот же тип данных, что и одноименная процедура Property Let. Обычно при определении свойства необходимо задавать обе процедуры. Воспользовавшись командой Add Procedure меню Tools, можно сэкономить время, поскольку эта команда позволяет создать одновременно обе процедуры свойств (Property Get И Property Let).

*Оператор Property Set* Существует и третий тип процедур свойств — процедура Property Set. Процедура Set используется аналогично Let, но передает не значение свойства, а ссылку на объект.

Public Property Set Property\_Name (ByVal objNewValue As Object)

Set Property\_Name = objNewValue

End Property

В этой связи важную роль играет свойство по умолчанию. Каждый элемент управления имеет такое свойство, которое всегда используется Visual Basic, если имя свойства в коде явно не указано.

Textl = "Hello" '(равнозначно Textl.Text = "Hello") vVariable = Textl

В приведенном примере свойству Text текстового окна присваивается определенное значение. Во второй строке это значение считывается. Но если переменной объекта требуется передать сам объект Textl, то строка кода должна выглядеть иначе – при присваивании объектных переменных в Visual Basic вместо неявного ключевого слова Let следует явно использовать ключевое слово Set.

vVariable = Textl "Содержимое: "Hello" Set vVariable = Textl

'Содержимое: копия Textl

Дорога с односторонним движением

Разделение процедур свойств на процедуры присваивания и процедуры считывания позволяет легко создавать свойства, доступные только для чтения или только для изменения. Допустим, вам нужно ограничить доступ к свойству и сделать его доступным только для чтения. Для этого достаточно создать только одну процедуру – в этом окне задается имя, тип данных и доступность свойства. При необходимости можно назначить это свойство свойством по умолчанию. Если при создании свойства в окне Property Builder установить опцию Public Property или Friend Property, мастер добавит в модуль класса три процедуры Property

(Let, Get и Set), а также внутреннюю переменную класса для хранения значения свойства.

Private mvarProperty\_Name As Variant

Public Property Let Property\_Name

(ByVal vData As Variant)

mvarProperty\_Name =.vData

End Property

Public Property Set Property\_Name (ByVal vData As Object)

Set mvarproperty Name = vData

End Property

Public Property Get Property\_Name () As Variant

 If IsObject(Property\_Name) Then

 Set Property\_Name ° mvarProperty\_Name

Else

Property\_Mame = mvarProperty\_Name

 End If

End Property

*Использование*

Обычно в процедурах свойств объем кода не больше, чем в приведенном примере. Дополнительный код может понадобиться лишь для проверки правильности значения, присваиваемого свойству. При этом не следует забывать, что свойства не должны вызывать никаких действий объекта^Ото реализуется только посредством его методов.

Методы

Для использования объектов класса (Class или Form) нужны соответствующие свойства и методы. Методы используются для расширения функциональных возможностей объекта. Существует два типа процедур: с возвращаемым значением (синтаксис функции) и без возвращаемого значения (синтаксис оператора). Это также относится и к методам. Например, метод Clipboard. SetText использует синтаксис оператора, а метод Clipboard. GetText – синтаксис функции. При создании методов действуют те же правила, за исключением того, что процедуры Sub и Function следует объявлять как Public.

*Синтаксис оператора*

Для создания метода без возвращаемого значения в модуль класса необходимо добавить процедуру public Sub, имя которой будет именем метода.

Public Sub Method\_Name()

vVarName = vVarName \* 5

End Sub

Обращение к такому методу аналогично вызову обычной процедуры. При необходимости процедуре могут передаваться аргументы.

Public Sub Method\_Name(argi, Optional arg2 As Integer)

vVarName = argi \* arg2

End Sub

Посредством аргументов более точно определяется характер выполняемого действия или передаются обрабатываемые элементы.

Мастер классов

Создать метод очень просто с помощью мастера классов.

Для добавления метода следует в левой части окна мастера классов выбрать

соответствующий класс и при помощи команды меню File\New\Method или соответствующей кнопки панели инструментов окна мастера открыть диалоговое окно Method Builder.

В поле Name следует указать имя метода, в поле Return Data Type – тип данных возвращаемого значения (если метод должен возвращать данные). Если поле Return Data Type оставить пустым, будет создана процедура Sub. Если указать тип данных, будет создана процедура Function. В список Arguments посредством кнопки со знаком + можно добавить аргумент метода с указанием его имени и типа данных. Если при этом установить флажок ByVal, аргумент будет передаваться как значение (ByVal). В Visual Basic 6.0 аргумент может быть массивом (Array), массивом параметров (ParamArray, который применяется, если количество передаваемых процедуре аргументов заранее неизвестно), параметром по умолчанию (Default), необязательным параметром (Optional), а также передаваться как значение (ByVal). Кроме того, при использовании аргумента поддерживаются перечисления (Enums).

***Использование***

Обычно методы содержат намного больше кода, чем процедуры свойств, так как в них выполняется непосредственная обработка данных. Сложные задачи можно разбить на несколько процедур (Sub или Function), которые объявляются как Private, поскольку могут использоваться только внутри класса. Инкапсуляция внутренних данных является большим преимуществом ООП.

События

События – это реакции объекта на действия системы, пользователя или других приложений. Анализ и обработка событий позволяет объекту реагировать на внешние воздействия. Каждый класс имеет набор собственных событий. По умолчанию все классы содержат два события, которые происходят

после загрузки и перед выгрузкой объекта. Эти события – Initialize и Terminate.

События, определяемые пользователем

Объекты, определяемые пользователем, могут генерировать собственные события (возможность, впервые реализованная в Visual Basic 5.0). События создаются в два этапа. Сначала в секции объявлении класса объявляется

событие (при необходимости с аргументами), а затем в процедуре внутри класса происходит вызов этого события.

'Общие объявления (Class или Form)

Public Event Event\_Name()

Public Sub CallEventO

RaiseEvent Event\_Name

***Оператор Event***

Событие, определяемое пользователем, объявляется внутри класса с помощью оператора Event. Аргументы, передаваемые процедуре обработки события, должны указываться в строке объявления события.

***Оператор RaiseEvent***

Для генерации события внутри класса предназначен оператор RaiseEvent, которому в качестве аргумента передается имя события.

***Ключевое слово WithEvents***

Контейнер, в котором используется объект, должен содержать процедуру обработки события. До создания процедуры обработки события нужно объявить объект, используя ключевое слово WithEvents. Общие объявления Dim WithEvents object

As classi

Private Sub object имя события()

End Sub

После такого объявления указанный объект будет добавлен в список (Object) окна кода, в результате чего из списка (Procedure) можно будет выбирать события объекта.

Раннее и позднее связывание

Скорость выполнения приложения может зависеть от того, каким образом Visual Basic осуществляет связывание переменных с их объектами. Связывание может быть ранним (Early Binding) или поздним (Late Binding).

***Раннее связывание***

Раннее связывание осуществляется, если при объявлении переменной

указывается конкретный тип данных:

Dim objVarl As TextBox

Dim objVar2 As Classi

Dim objVar3 As Word.Application

В приведенном примере при объявлении переменных их типы данных

указываются точно. Раннее связывание выполняется быстрее, чем позднее,

поэтому рекомендуется использовать его всегда.

***Позднее связывание***

Позднее связывание выполняется, если при объявлении переменной

определенный тип объекта не указывается.

Dim objVarl As Object Dim

ob]Var2 As Variant Dim

objVar3 As Control

В данном примере типы переменных определяются только при

присваивании им конкретного объекта, а при объявлении переменной

указывается только универсальный тип. При таком объявлении при каждом

обращении к объекту анализируются все допустимые объекты, а это требует

значительных затрат времени.

Позднее связывание следует использовать в тех случаях, если при

разработке невозможно заранее определить, какие типы объектов будут

использоваться.

***События Initialize и Terminate***

Кроме событий, определяемых пользователем, классы всегда содержат события Initialize и.Тег1minate, которые являются аналогами событий формы Load и Unload.

***Событие Initialize***

Событие Initialize класса наступает при первом использовании объекта. В процедуре обработки этого события следует выполнять код, необходимый для инициализации объекта. Это может быть, например, объявление собственных объектов или подключение к серверу базы данных.

***Событие Terminate***

Как только уничтожаются все ссылки на объект, Windows удаляет этот

объект из памяти. В этот момент наступает событие Terminate.

Private Sub Commandl\_Click()

Set object = Nothing 'наступает Terminate

End Sub

Это событие используется для того, чтобы объект мог корректно освободить память, т.е. удалить все внутренние объекты или отсоединиться от внешних источников данных. При создании собственных классов можно легко контролировать наступление событий Initialize и Terminate. Для этого в модуль класса следует поместить следующий код:

Private Sub Class\_Initialize()

MsgBox "Инициализация" End Sub

Private Sub Class\_Terminate()

MsgBox "Удаление"

End Sub