

С этим руководством вы приступите к программированию на языке VB.NET настолько быстро, насколько это возможно!



# Visual Basic® .NET

## ДЛЯ "ЧАЙНИКОВ"™

**Для  
сомневающих**

**Экскурс в объектно-  
ориентированное  
программирование и  
другие возможности  
VB .NET**

**Уоллес Вонг**

Автор книги  
Основы  
программирования  
для "чайников"



*Visual Basic .NET*

ДЛЯ  
"ЧАЙНИКОВ"™

# *Visual Basic .NET* FOR DUMMIES

by ~~Walter Wing~~



Hungry Minds™  
HUNGRY MINDS, INC.

Best-Selling Books • ~~On Demand~~ • e-Books • Answer Networks • e-Newsletters • ~~Audio~~ • ~~Video~~ • e-Learning  
New York, NY • ~~Cleveland~~ • ~~Indianapolis~~ • ~~N~~

*Visual Basic .NET*

ДЛЯ  
"ЧАЙНИКОВ"

Уоллес Вонг



ДИАЛЕКТИКА

Москва • Санкт-Петербург • Киев  
2002

ББК 32.973.26-018.2.75

В73

УДК 681.3.07

Компьютерное издательство “Диалектика”

Зав. редакцией *В. В. Александров*

Перевод с английского и редакция *П. А. Минько*

По общим вопросам обращайтесь в издательство “Диалектика”  
по адресу: [info@dialektika.com](mailto:info@dialektika.com), <http://www.dialektika.com>

Вонг, Уоллес.

В73 Visual Basic .NET для "чайников". : Пер. с англ. — М. : Издательский дом “Вильямс”, 2002. — 336 с. : ил. — Парал. тит. англ.

ISBN 5-8459-0288-6 (рус.)

Добро пожаловать в мир компьютерного программирования с использованием Visual Basic. NET — разработанного компанией Microsoft языка программирования, который поможет вам легко и быстро писать собственные программы. Если вас давно волнует идея создания своих программ, но в то же время вы опасаетесь, что вам это не по зубам, откиньте сомнения прочь. Если вы в состоянии написать краткую пошаговую инструкцию о том, как пройти к вашему дому, значит, сможете освоить и процесс написания программ на языке Visual Basic. NET. Чтобы помочь вам в постижении азов программирования на Visual Basic. NET, материал этой книги изложен простым, доступным языком, а основной акцент сделан на тех функциональных возможностях, которые наверняка пригодятся вам при написании собственных программных продуктов.

Книга предназначена для начинающих пользователей.

ББК 32.973.26-018.2.75

Все названия программных продуктов являются зарегистрированными торговыми марками соответствующих фирм. Никакая часть настоящей издания ни в каких целях не может быть воспроизведена в какой бы то ни было форме и какими бы то ни было средствами, будь то электронные или механические, включая фотокопирование и запись на магнитный носитель, если на это нет письменного разрешения издательства Hungry Minds.

Copyright © 2002 by Dialektika Computer Publishing.

Original English language edition copyright © 2001 by Hungry Minds, Inc.

All rights reserved including the right of reproduction in whole or in part in any form.

This edition published by arrangement with the original publisher. Hungry Minds, Inc.

For [Dummies and Dummies Man are trademarks under exclusive license to Hungry Minds, inc. Used by permission.

ISBN 5-8459-0288-6 (рус.)

ISBN 0-7645-0867-9 (англ.)

© Компьютерное изд-во "Диалектика". 2002

© Hungry Minds, Inc., 2002

# Оглавление

Введение	19
Глава 1. Как работает Visual Basic .NET	25
Глава 2. Пользовательский интерфейс приложения Visual Basic .NET	31
Глава 3. Создаем пользовательский интерфейс	39
Глава 4. Приступаем к написанию кодов BASIC	47
Глава 5. Создание пользовательского интерфейса: разберемся в деталях	61
Глава 6. Разработка форм	79
Глава 7. Элементы как средство предоставления пользователю возможности выбора	91
Глава 8. Использование текстовых полей и надписей	97
Глава 9. Использование списков и полей со списком	105
Глава 10. Настройка отображения пользовательского интерфейса	113
Глава 11. Разработка раскрывающихся меню	121
Глава 12. Подменю, расширяемые меню и выпадающие меню	131
Глава 13. Диалоговые окна	139
Глава 14. Написание процедур обработки событий	155
Глава 15. Использование переменных	167
Глава 16. Получение данных от пользователя	177
Глава 17. Займемся математикой	183
Глава 18. Обработка текстовой информации	195
Глава 19. Определение констант и использование комментариев	205
Глава 20. Создание структур данных	213
Глава 21. Борьба с ошибками	223
Глава 22. Условные операторы If-Then	235
Глава 23. Оператор выбора Select Case	243
Глава 24. Создание циклов	249
Глава 25. Циклы, которые умеют считать	255
Глава 26. Вложенные циклы	261
Глава 27. Общие процедуры	267
Глава 28. Использование аргументов	275
Глава 29. Создание функций	281
Глава 30. Так что же это такое — объектно-ориентированное программирование?	291
Глава 31. Объектно-ориентированное программирование на практике	297
Глава 32. Наследование и перегрузка	305
Глава 33. Десятка полезнейших советов, которые вы вряд ли найдете в каком-нибудь другом месте	315
Глава 34. Советы по использованию интерфейса Visual Basic .NET	321
Предметный указатель	325

# Содержание

<b>Введение</b>	<b>19</b>
<b>Часть I. Создание программ на Visual Basic .NET</b>	<b>23</b>
<b>Глава 1. Как работает Visual Basic .NET</b>	<b>25</b>
Написание программ на Visual Basic .NET	25
Создание пользовательского интерфейса	26
Определение свойств элементов интерфейса	27
Написание команд на языке BASIC	27
Что означает приставка .NET	28
Недостатки концепции .NET	29
<b>Глава 2. Пользовательский интерфейс приложения Visual Basic .NET</b>	<b>31</b>
Запуск Visual Basic .NET	31
Приступаем к созданию нового проекта	32
Открытие существующих проектов	34
Добро пожаловать в пользовательский интерфейс Visual Basic .NET	34
Манипулирование окнами	36
Как сделать окно плавающим	37
Как сделать окно закрепленным	37
Скрытие окон	38
Заккрытие окна	38
Выход из Visual Basic .NET	38
<b>Глава 3. Создаем пользовательский интерфейс</b>	<b>39</b>
Основные компоненты пользовательского интерфейса	39
Использование панели Toolbox для рисования объектов	39
Создание вашего первого пользовательского интерфейса	40
Определение свойств элементов интерфейса	42
Зачем нужны свойства	43
Внесение изменений в свойства объектов	43
Изменение свойств объектов на стадии разработки интерфейса	44
Определение свойств для интерфейса вашей первой программы	45
<b>Глава 4. Приступаем к написанию кодов BASIC</b>	<b>47</b>
Что представляют собой коды BASIC	47
Написание процедур обработки событий	49
Быстрый способ создания процедуры обработки событий	49
Обычный способ создания процедуры обработки событий	50
Что могут делать коды BASIC	51
Как работают процедуры обработки событий	52
Написание кодов BASIC для своей первой программы	53
<b>Часть II. Создание пользовательского интерфейса</b>	<b>59</b>

## **Глава 5. Создание пользовательского интерфейса: разберемся в деталях 61**

До того как приступить к созданию интерфейса	61
Вспомните, для кого создается программа	61
Не дайте пользователю заблудиться	62
Сделайте навигацию очевидной	62
Будьте снисходительны	62
Не забывайте о простоте и удобстве	63
Приступаем к созданию интерфейса	63
Создание форм	64
Рисование объектов в окне формы	64
Изменение свойств объектов	66
Переименование объектов	67
Отображение текста на объекте	68
Настройка размеров объектов	70
Перемещение объектов на экране	70
Прикрепление объектов к сторонам формы	71
Закрепление объектов	72
Копирование созданного объекта	73
Удаление объектов	73
Выделение сразу нескольких объектов для перемещения, копирования или удаления	74
Определение для объектов свойства TabIndex	74
Выделение объектов серым цветом	76
Как сделать объект невидимым	77
Изменение текста, отображаемого на объекте	77

## **Глава 6. Разработка форм 79**

Создание форм	79
Переименование форм	79
Отображение нескольких форм	80
Изменение внешнего вида форм	81
Итак, раскрасим вашу форму	81
Создание фоновых рисунков	82
Виды границ	82
Сворачивание и разворачивание окна формы	84
Размещение формы на экране	86
Удаление и добавление форм	87
Выбор формы, которая будет отображаться первой	88
Открытие, скрытие и закрытие форм	88
Открытие форм	89
Скрытие (и восстановление) форм	89
Закрытие форм	90

## **Глава 7. Элементы как средство предоставления пользователю возможности выбора 91**

Нажми на кнопку - получишь результат...	91
Создание флажков и переключателей	92



Выравнивание флажков и переключателей	93
Группировка флажков и переключателей	94
Отображение текста на кнопках, флажках, переключателях и рамках групп	96
<b>Глава 8. Использование текстовых полей и надписей</b>	<b>97</b>
Создание надписей и текстовых полей	97
Изменение внешнего вида отображаемого текста	98
Раскрашивание текста надписей и текстовых полей	99
Настройка отображения границ	99
Выравнивание текста относительно границ объекта	100
Настройка текстовых полей	101
Прокрутка текста	101
Создание полей для ввода паролей	102
Ограничение длины текста	103
<b>Глава 9. Использование списков и полей со списком</b>	<b>105</b>
Создание списков и полей со списком	105
Как нарисовать объекты в окне формы	106
Настройка параметров поля со списком	106
Наполнение списка и поля со списком элементами для выбора	108
Сортировка элементов списков и полей со списком	110
Удаление элементов из списка	110
Сделайте список удобным	111
<b>Глава 10. Настройка отображения пользовательского интерфейса</b>	<b>113</b>
Установка одинаковых размеров для нескольких объектов	113
Выравнивание объектов	114
Определение расстояния между объектами	115
Выравнивание по центру	116
Фиксация положения объектов	116
Фиксирование положения всех объектов в окне формы	117
Фиксирование положения отдельных объектов	117
<b>Часть III. Создание меню</b>	<b>119</b>
<b>Глава 11. Разработка раскрывающихся меню</b>	<b>121</b>
Основные компоненты строки меню	121
Создание меню для интерфейса вашей программы	122
Добавление и удаление заголовков меню и команд меню	123
Перемещение заголовков и команд меню	124
Присвоение меню имен	124
Настройка меню	125
Добавление в меню разделительных линий	125
Назначение комбинаций клавиш	126
Отображение флажков рядом с командами меню	128
Выделение серым команд меню	129
Скрытие команд меню	130

<b>Глава 12. Подменю, расширяемые меню и выпадающие меню</b>	<b>131</b>
Создание подменю	131
Изменение команд меню в процессе выполнения программы	132
Создание динамически расширяемых меню	133
Создание расширяемых меню в режиме конструктора	133
Добавление новых элементов меню в процессе выполнения программы	134
Создание выпадающих меню	136
Создание команд для контекстных меню	136
Как сделать контекстное меню выпадающим	137
Копирование команд в контекстное меню	138
<b>Глава 13. Диалоговые окна</b>	<b>139</b>
Создание простых диалоговых окон	139
Добавление пиктограмм	140
Добавление командных кнопок	141
Как определить, на какой кнопке щелкнул пользователь	142
Использование стандартных диалоговых окон	143
Создание окна OpenFile	144
Как определить, какой файл выбран пользователем в диалоговом окне OpenFile	146
Создание окна Save File	147
Как определить, какой файл выбран пользователем в окне SaveFile	147
Создание окна Color	147
Как определить, какой цвет был выбран пользователем	148
Создание окна Font	148
Как определить, какие опции были выбраны пользователем в окне Font	149
Создание окна Print	150
Создание окна PageSetup	151
<b>Часть IV. Основы создания кодов</b>	<b>153</b>
<b>Глава 14. Написание процедур обработки событий</b>	<b>155</b>
Работа с редактором кодов	155
Разворачивание и сворачивание кодов BASIC	156
Виды событий	157
Создание процедур обработки событий	157
Из каких частей состоит процедура обработки событий	158
Разделение окна редактора кодов на две части	159
Использование редактора кодов	160
Просмотр процедур обработки событий	162
Написание процедур обработки событий	164
Получение данных от пользователя	164
Вычисление результата	164
Отображение полученных результатов на экране	164
Процедура, которую должна иметь любая программа	164
<b>Глава 15. Использование переменных</b>	<b>167</b>
Чтение данных	167

Переменные и их значения	168
Использование переменных	168
Объявление переменных	168
Присвоение имен переменным	171
Присвоение переменным числовых значений	172
Присвоение переменным текстовых значений	172
Присвоение переменным значений других переменных	173
Присвоение переменным значений свойств объектов	174
Область видимости переменных	174
Переменные, видимые в пределах блока	174
Переменные, доступные в пределах процедуры	175
Переменные, доступные в пределах модуля	175
Глобальные переменные	176
Использование переменных для представления объектов	176
<b>Глава 16. Получение данных от пользователя</b>	<b>177</b>
Использование свойства Text для получения текстовой информации	177
Получение логической информации	178
Получение числовых данных	179
Выбор из списка сразу нескольких элементов	180
Сколько элементов выбрал пользователь?	180
Определение выбранных пользователем элементов	180
<b>Глава 17. Займемся математикой</b>	<b>183</b>
Арифметические операторы	183
Сложение двух чисел	184
Вычитание чисел	184
Получение отрицательных чисел	184
Умножение чисел	184
Операция деления	185
Применение оператора Mod	186
Возведение в степень	186
Конкатенация двух строк	187
Изменение типов переменных	187
Логические операторы	188
Применение оператора Not	189
Применение оператора And	189
Применение оператора Or	189
Применение оператора Xor	190
Операторы сравнения	190
Сравнение числовых и текстовых значений	190
Использование для сравнения строк операторов = и <>	191
Использование для сравнения строк операторов >, >=, < и <=	192
Приоритет операторов	193
<b>Глава 18. Обработка текстовой информации</b>	<b>195</b>
Определение длины строки	195

Изменение регистра	195
Как сделать ПРОПИСНЫЕ буквы строчными	196
Как сделать строчные буквы ПРОПИСНЫМИ	196
Как сделать Первые Буквы Всех Слов Прописными	196
Удаление лишних пробелов	196
Удаление пробелов в начале строки	197
Удаление пробелов в конце строки	197
Удаление пробелов в начале и в конце строки	197
Возвращение символов строки	198
Возвращение первых символов строки	198
Возвращение последних символов строки	198
Возвращение символов, находящихся внутри строки	198
Поиск и замена отдельных слов	199
Поиск одной строки внутри другой	199
Поиск текста по шаблону	199
Применение группового символа "?"	200
Групповой символ "звездочка"	200
Применение группового символа "#"	201
Применение диапазонов	201
Замена части строки другой строкой	201
Преобразование строк и чисел	202
Преобразование строк в числа	203
Преобразование чисел в строки	203
Преобразование строк в коды ASCII	204
Преобразование кодов ANSI в строки	204
<b>Глава 19. Определение констант и использование комментариев</b>	<b>205</b>
Наименование констант	205
Объявление констант	206
Вычисление значений констант	207
Использование констант	207
Определение области видимости констант	207
Локальные константы	207
Константы модуля	208
Глобальные константы	208
Использование комментариев	209
Создание комментариев	209
Комментарии как объяснения	210
Комментарии как средство улучшения читаемости кодов	210
Комментарии как временная дезактивация кодов	211
<b>Глава 20. Создание структур данных</b>	<b>213</b>
Создание массивов	213
Присвоение значений элементам массива	214
Создание многомерных массивов	215
Изменение размеров массива	216
Создание структур	217

Структуры и переменные	217
Сохранение данных	218
Комбинирование структур и массивов	219
Коллекции данных	220
Добавление информации в коллекцию данных	220
Определение количества элементов коллекции	220
Чтение информации, сохраненной в коллекции	221
Удаление данных из коллекции	221
<b>Глава 21. Борьба со ошибками</b>	<b>223</b>
Классификация ошибок	223
Синтаксические ошибки	224
Рабочие ошибки	224
Логические ошибки	224
Стратегия охоты за ошибками	225
А есть ли в программе ошибки?	225
Поиск ошибок	225
Источник возникновения ошибки	226
Ликвидация ошибок	226
Ловушки для ошибок	226
Средства Visual Basic .NET для отслеживания и удаления ошибок	228
Пошаговое выполнение программы	228
Определение точек останова	229
Просмотр значений переменных	230
<b>Часть V. Создание разветвлений и циклов</b>	<b>233</b>
<b>Глава 22. Условные операторы If-Then</b>	<b>235</b>
Логические значения	235
Присвоение логических значений переменным	235
Логические значения выражений	236
Условный оператор If-Then	237
Оператор If-Then-End If	237
Оператор If-Then-Else	238
Оператор If-Then-Elseif	239
Если вариантов должно быть много	240
Если должен быть выбран хотя бы один вариант	240
Использование вложенных операторов If-Then	241
<b>Глава 23. Оператор выбора Select Case</b>	<b>243</b>
Использование оператора выбора Select Case	243
Использование оператора Select Case с операторами сравнения	244
Если хотя бы один вариант должен быть выбран	244
Использование вложенных условных операторов	245
<b>Глава 24. Создание циклов</b>	<b>249</b>
Циклы, которые могут не выполняться	249

Циклы Do-While	249
Циклы Do-Until	250
Циклы, выполняющие не менее одной итерации	251
Циклы Do-Loop Until	251
Циклы Do-Loop While	252
Какой цикл лучше?	253
<b>Глава 25. Циклы, которые умеют считать</b>	<b>255</b>
Как работает цикл For-Next	255
Прямой и обратный порядок отсчета	257
Переменная-счетчик и ее использование	258
Когда For-Next лучше, чем другие циклы	259
<b>Глава 26. Вложенные циклы</b>	<b>261</b>
Как работают вложенные циклы	261
Советы по использованию вложенных циклов	262
Быстрое завершение циклов	263
<b>Часть VI. Создание подпрограмм</b>	<b>265</b>
<b>Глава 27. Общие процедуры</b>	<b>267</b>
Что такое файлы модулей	267
Создание общих процедур	269
Сохранение общих процедур в файле формы	270
Создание и сохранение общих процедур в файлах модулей	271
Присвоение имен общим процедурам	272
Вызов общих процедур	272
<b>Глава 28. Использование аргументов</b>	<b>275</b>
Для чего нужны аргументы	275
Передача процедуре значений аргументов	276
Получение значений аргументов	277
Аргументы-значения и аргументы-ссылки	277
Объявление сразу нескольких аргументов	278
Возможные ошибки при передаче данных процедуре	279
Неверное количество переменных	279
Неправильный тип данных	279
Немедленный выход из процедуры	279
<b>Глава 29. Создание функций</b>	<b>281</b>
Создание функций	282
Создание функции в файле формы	282
Создание и сохранение функции в файле модуля	282
Возвращаемый результат	283
Вызов функций	284
Тип данных принимаемых и возвращаемых значений	285

Тип принимаемых данных	286
Ошибки при определении значений аргументов	287
Передача неверного числа аргументов.	287
Несоответствие типов данных	287
Быстрое завершение работы функции	287
<b>Часть VII. Объектно-ориентированное программирование</b>	<b>289</b>
<b>Глава 30. Так что же это такое –объектно-ориентированное программирование?</b>	<b>291</b>
Принципы структурного программирования	291
Разделение большой программы на множество подпрограмм	292
Объявление переменных	292
Использование последовательных команд, команд ветвления и команд циклов	293
Объектно-ориентированное программирование	293
Инкапсуляция: разделение данных и команд	294
Наследование повторно используемых кодов	294
Перегрузка существующих кодов	295
<b>Глава 31. Объектно-ориентированное программирование на практике</b>	<b>297</b>
Использование приемов ООП в Visual Basic .NET	297
Создание файла классов	298
Определение объекта	298
Объявление переменных	299
Объявление свойств объектов	300
Создание методов	301
Прежде чем приступить к написанию кодов	301
Использование модулей классов в программах Visual Basic .NET	302
Создание объектов	302
Использование объектов	302
Как это выглядит на практике	302
<b>Глава 32. Наследование и перегрузка</b>	<b>305</b>
Что есть хорошего в наследовании кодов?	305
Наследование кодов для создания новых форм	306
Наследование рабочих кодов программы	307
Перегрузка свойств и методов	308
Практическое применение приемов наследования и перегрузки	310
<b>Часть VIII. Горячие десятки</b>	<b>313</b>
<b>Глава 33. Десятка полезнейших советов, которые вы вряд ли найдете в каком-нибудь другом месте</b>	<b>315</b>
Читайте специализированные периодические издания	315
Подпишитесь на информационные бюллетени Visual Basic Developer	316
Ищите информацию в Internet	316
Посещайте технические конференции	316

Покупайте программные продукты через посредников	317
Комбинируйте Visual Basic .NET с C# и другими языками программирования	317
Принимайте участие в разработке открытых проектов Visual Basic .NET	318
Приобретите программу для создания справочной системы	318
Создавайте инсталляционные программы	319
Пишите Basic-программы для Macintosh, Linux, Palm OS и PocketPC	319
<b>Глава 34. Советы по использованию интерфейса Visual Basic .NET</b>	<b>321</b>
Использование окна Properties	321
Окно Solution Explorer	321
Настройка панели Toolbox	322
Создание новой вкладки на панели Toolbox	322
Добавление объектов в новую вкладку панели Toolbox	322
Удаление вкладок панели Toolbox	323
Использование окна Class View	323
<b>Предметный указатель</b>	<b>325</b>



## Об авторе

Для более чем 99% населения нашей планеты имя автора данной книги ни о чем не говорит. Эта неумолимая статистика помогает сохранить ему скромность и не впадать в звездную болезнь, даже несмотря на те небольшие дивиденды в виде известности среди определенного круга программистов и некоторого дохода от продаж настоящего издания. Тем более что эти скромные дивиденды будут существовать лишь в течение сравнительно короткого жизненного цикла книги, ограниченного моментом выпуска компанией Microsoft новой версии Visual Basic.

В перерывах между работой над книгами автор развлекается тем, что летает среди облаков на дельтаплане, взбирается по скалам, погружается с аквалангом на дно океанов, работает наемником в горячих точках планеты, охотится за снежным человеком, контактирует с инопланетянами, добывает минералы на втором спутнике Сатурна и придумывает множество других историй и небылиц, чтобы как-то скрасить свой досуг и не умереть от скуки.

# Посвящения

Настоящая книга посвящается людям, которые не имели никакого отношения к ее написанию, но которым все равно приятно будет увидеть свои имена на ее страницах:

Стиву Ширипу (Steve Schirripa) и Дону Лернеду (Don Learned)— за яркие впечатления, полученные мною в Лас-Вегасе в казино "Ривьера" ([www.theriviera.com](http://www.theriviera.com)). В следующий раз, когда будете в Лас-Вегасе, загляните в это казино, расслабьтесь, посмотрите шоу и спустите немного денег на игровых автоматах. Кто знает, может вам повезет и вы даже выиграете достаточную сумму, чтобы купить себе новые компьютеры.

Патрику Дегиру (Patrick DeGuire)— за помощь при создании нашей компании Top Bananas Entertainment ([www.topbananas.com](http://www.topbananas.com)), а также Крису Клоберу (Chris Clobber, продавцу зоомагазина), Бобу Зэну (Bob Zane), Тони Висич (Tony Vicich), Кипу Эдатта (Kip Addotta), Рону Персону (Ron Pearson), Вили Фарель (Willie Farrell) и Лео Фонтейну (Leo Fontaine). Никто из них не принимал участия в создании этой книги, но они помогли мне расслабиться после тяжелой работы, поэтому также заслуживают моего особого признания.

И наконец, эта книга посвящается моим родственникам, которые попросили, чтобы я назвал их имена: моим родителям Руте и Герберту, брату Вэйну и сестре Гейл. Я, правда, не совсем понимаю, зачем им нужно видеть свои имена в книге, которую они никогда в жизни читать не будут, но если это доставит им радость, то почему бы мне это не сделать?

# Благодарности

Никто не может написать и издать книгу без **помощи** других людей. Не является исключением и то творение, которое вы сейчас держите в руках. Я просто не могу не поблагодарить двух людей, внесших значительный вклад в создание настоящей книги. Это Мэт Вагнер (Matt Wagner) и Билл Глэдстоун (Bill Gladstone) из Waterside Productions. Спасибо, ребята. Я бы даже заплатил вам больше полагающихся 15 процентов, но боюсь, что в таком случае у меня не останется даже на покупку рождественских подарков.

Двух других людей, которые также заслуживают благодарности и признательности, зовут **Колин** Эстэрлайн (Colleen Esterline, друг редактора) и Ален Вьят (**Allen** Wyatt, друг технического редактора, сделавший все для того, чтобы мы могли утверждать: абсолютно все примеры, рассмотренные в этой книге, действительно работают так, как здесь написано) из Discover)' Computing, Inc.

Особая благодарность моей жене Кассандре, сыну Джордану, котам Бобу, Скрапсу, Таше и Ниту — за их терпение и понимание в течение долгих часов, когда я, вместо того чтобы заниматься какими-нибудь домашними делами, был приклеен к своему компьютеру.

Конечно же, я должен поблагодарить все деревья, которые отдали свои жизни, т.е. были растерзаны до состояния древесины, а затем превращены в бумагу, на которой напечатана эта книга. Не забудем также всех лошадок, чьи тела послужили материалом для изготовления клея, которым склеены ее страницы.

Наконец, я должен поблагодарить всех людей, которые гнули свои спины, доставляя коробки с книгами в разные уголки страны, аккуратно расставляли их по полкам, с тем чтобы вы могли, стоя в уютном магазинчике, не спеша листать страницы и просматривать содержание. Поддержите же этих людей, к примеру, покупкой сразу трех экземпляров данной книги.

# Введение

Добро пожаловать в мир компьютерного программирования с использованием Visual Basic. NET — разработанного компанией Microsoft языка программирования, который поможет вам легко и быстро писать собственные программы. Если вас давно волнует идея создания своих программ, но в то же время вы опасаетесь, что вам это не по зубам, откиньте сомнения прочь. Если вы в состоянии написать краткую пошаговую инструкцию о том, как пройти к вашему дому, значит, сможете освоить и процесс написания программ на языке Visual Basic. NET. (Не сомневайтесь.)

Чтобы помочь вам в постижении азов программирования на Visual Basic. NET, материал этой книги изложен простым, доступным языком, а основной акцент сделан на тех функциональных возможностях, которые наверняка пригодятся вам при написании собственных программных продуктов.

Вопреки всеобщему мнению, программирование вовсе не обязано быть скучным и сложным — наоборот, вы даже можете получать от него удовольствие. И пускай стиль изложения, принятый в этой книге, послужит тому доказательством. В конце концов, для чего большинство людей покупают компьютеры? Я думаю, для получения удовольствия. (Не могут же они, в самом деле, приобретать таковые лишь для того, чтобы потратить деньги.)

## ОЭтой книге

Относитесь к этой книге просто как к дружескому справочному пособию и проводнику в мир Visual Basic. NET. Среди затронутых здесь вопросов будут такие:

- ✓ сохранение программ;
- ✓ создание пользовательского интерфейса;
- ✓ создание контекстных меню;
- ✓ поиск и устранение мелких ошибок;
- ✓ написание больших программ.

И хотя на постижение всех нюансов языка Visual Basic. NET могут уйти годы, изучить его понятия и приемы в объеме, достаточном для создания собственных программ, можно поразительно быстро. Цель настоящего издания — научить вас всему, что необходимо для написания программ на Visual Basic. NET, с тем чтобы вы начали программировать настолько быстро, насколько это возможно.

## Как использовать книгу

В книге даны пошаговые инструкции по созданию жизнеспособных работающих программ Windows с использованием языка программирования Visual Basic. NET. Для представления определенного рода информации применяется ряд условных обозначений.

Все коды (тексты программ) набраны другим шрифтом, как это показано ниже:

```
Printer.DrawWidth = Value
```

Если строка программы слишком длинная и не помещается на одной строке страницы, часть ее переносится на другую строку. Поэтому иногда в конце книжной строки вы будете видеть символ подчеркивания, обозначающий, что в действительности на экране компьютера этот код программы отображен одной сплошной строкой:

```
Private Sub Button_Click(ByVal sender As System.Object, _  
    ByVal e As System.EventArgs) Handles Button.Click
```

Для Visual Basic .NET не имеет значения, набрано что-либо ПРОПИСНЫМИ БУКВАМИ, строчными буквами или KaK-To ИначЕ. Однако, чтобы получить точное соответствие между тем, что появится на вашем экране, и тем, что изображено на рисунках в книге, придерживайтесь последовательности прописных и строчных букв, указанной в приводимых здесь примерах.

Приступая к изучению методов и приемов создания пользовательского интерфейса, вы можете столкнуться с таблицами, подобными приведенной ниже.

Объект	Свойство	Значение
Форма	Name	frmHello
	Text	Всем привет!
Кнопка	Name	btnClickMe
	Visible	True

Такие таблицы помогут вам понять, как будет выглядеть создаваемый вами пользовательский интерфейс. В столбце *Объект* определен тип элемента, подлежащего модификации (например, кнопка или форма), в столбце *Свойство* указана изменяемая характеристика элемента, а в столбце *Значение* приведено само значение, которое нужно установить для соответствующего свойства.

Поскольку некоторые люди после изучения материала не могут расслабиться, пока кто-нибудь их не протестирует (на предмет усвоения только что прочитанного), довольно часто в книге вам будут встречаться короткие тесты. Но они не должны удручать вас, а наоборот, призваны повысить вашу самооценку. К каждому вопросу дается на выбор несколько ответов, причем только один из них будет серьезным, и вам не составит труда понять, какой из них правильный. Таким образом, тесты не только помогут вам закрепить свои знания, но и просто поднимут ваше настроение.

## *Избавьтесь от предрассудков*

Предположим, вы уже научились включать и выключать компьютер, пользоваться мышью и клавиатурой, а возможно, у вас даже появилось желание написать несколько программ — для собственного удовольствия, для заработка или для работы (работа, к сожалению, не всегда связана с удовольствием и заработком). Значит, вам пора приступать к изучению Visual Basic .NET.

Да, кстати, на компьютере еще должно быть установлено приложение Visual Basic .NET. Если же вы все еще сомневаетесь в своей способности осилить такую “сложную” вещь, как программирование, вспомните Альберта Эйнштейна (это известный физик, который создал теорию относительности). У маленького Альберта был школьный учитель по физике, который считал, что мальчик, наверное, умственно отсталый — настолько плохо он воспринимал школьную программу.

Говорят, Альберт Эйнштейн в свое время сказал: “Воображение имеет намного большее значение, чем образование”. (Некоторые утверждают, что Эйнштейну также принадлежат слова “Если мой школьный учитель такой умный, то где же его Нобелевская премия?” Однако, имела ли место вторая реплика, точно не известно.)

Итак, если у вас есть воображение, персональный компьютер и копия приложения Visual Basic .NET, значит, вы готовы приступить к написанию программ на Visual Basic .NET.

## *Как организована книга*

Книга состоит из восьми больших частей. Каждая часть содержит по несколько глав, которые, в свою очередь, разбиты на подразделы. Каждый раз, когда вам нужна помощь по какому-либо вопросу, достаточно найти нужный раздел и начать читать. Ниже дано краткое описание всех восьми частей.

## Часть I: Создание программ на Visual Basic .NET

Первая часть содержит краткий обзор принципов использования Visual Basic .NET и написания программ на Visual Basic .NET. Прочитав ее, вы узнаете, насколько легким, простым и приятным может быть процесс программирования.

## Часть II: Создание пользовательского интерфейса

Очень интересная часть, из которой вы узнаете, как изменять внешний вид своего пользовательского интерфейса — от безобразного и отталкивающего до великолепного и привлекательного. В двух словах, что такое пользовательский интерфейс: это способы и приемы отображения информации на экране, с одной стороны, и диалога пользователей с вашей программой — с другой.

## Часть III: Создание меню

Большинство программ имеют в своем интерфейсе раскрывающиеся меню, в которых компактно размещены группы команд. Если вы хотите, чтобы ваша программа оставила глубокий след в памяти ваших друзей и впечатлила незнакомых людей, добавьте к ней такие же меню. Как это сделать, вы узнаете из этой части.

## Часть IV: Основы создания кодов

В этой части рассказывается, как написать понятные компьютеру команды, с тем чтобы объяснить, что ему делать дальше. Конечно, вы наверняка и так можете рассказать компьютеру, что ему делать дальше (используя язык жестов и нецензурную речь), но из данной части вы узнаете, как это сделать с использованием языка программирования BASIC (это будет более прагматично с вашей стороны).

## Часть V: Создание разветвлений и циклов

Из этой части вы узнаете о том, как пишутся не только программы, работающие с различными типами данных, но и программы, выполняющие определенные действия нужное количество раз (это называется созданием циклов). Циклы в сочетании с командами выбора придадут вашим программам (в отдельных местах) гибкость и “понятливость”.

## Часть VI: Создание подпрограмм

Написание коротких программ — занятие довольно простое. Написание больших программ — дело трудоемкое, изнуряющее, способное напугать даже опытных программистов. В этой части книги вы найдете описание ряда приемов, позволяющих разбить большую программу на несколько маленьких, называемых подпрограммами. Теоретически, написав несколько небольших работающих программ, вы можете связать их и получить одну большую программу, которая также будет работать (опять-таки, теоретически).

## Часть VII: Объектно-ориентированное программирование

Одной из новейших техник для написания и разработки больших программ является нечто, называемое объектно-ориентированным программированием. Если вы не знаете, что это такое и зачем вообще все это нужно, седьмая часть ответит на данные вопросы, и вы сможете решить для себя, использовать ли приемы объектно-ориентированного программирования при создании своих будущих программ на Visual Basic .NET.

## Часть VIII: Горячие десятки

Эта часть книги содержит самую разнообразную информацию. Здесь вы найдете много полезных и интересных сведений, и в частности советы относительно того, как использовать надстройки для Visual Basic .NET и где найти дополнительную информацию о программировании на Visual Basic .NET.

### Используемые пиктограммы



Такой пиктограммой сопровождается описание различных технических подробностей, которые информативны (и даже интересны), но вовсе не обязательны. Если вы не хотите их читать, не читайте.



Так отмечена полезная информация или просто советы, которые могут сделать программирование более приятным и менее проблематичным.



Это дружеское напоминание о важности данной информации, помогающее закрепить пройденный материал и сделать программирование на Visual Basic .NET занятием более простым и приятным.



Будьте внимательны к предупреждениям, отмеченным таким значком, поскольку они пытаются предостеречь вас от действий, которые могут серьезно подпортить вам жизнь.



Таким значком отмечены пошаговые инструкции, подробно описывающие работу определенных команд Visual Basic .NET.

### Что делать дальше

Если вы дошли до этого **места**, переворачивайте и данную страницу и продолжайте чтение. Если вы действительно хотите освоить Visual Basic .NET, возьмите за правило каждый день уделять время (хотя бы 15 минут) изучению этого предмета. Просто сядьте за компьютер, положите рядом книгу, запустите Visual Basic .NET и хотя бы немного поэкспериментируйте.



Читайте книгу небольшими порциями и сразу же пытайтесь применить на практике полученные знания. Это лучше, чем пройти за один раз несколько глав и сразу же забыть 90 процентов из только что прочитанного. Помните, чем больше вы будете общаться с Visual Basic .NET, тем увереннее будете себя чувствовать и многие нюансы, характерные для Visual Basic .NET, запомнятся сами по себе.

Как и в любом другом деле, на первых порах информация, скорее всего, будет усваиваться, не так быстро, как вам того хотелось бы. Однако если вы примете наш совет, приведенный выше, то очень скоро сможете убедиться, что объем ваших знаний увеличивается все заметнее и быстрее, а процесс обучения с каждым днем становится все проще и интереснее. Главное, не делайте больших перерывов, придерживайтесь выбранного вами темпа и старайтесь почаще применять полученные знания. Не забывайте об этом, и вы станете мастером по созданию собственных программ на Visual Basic .NET.

## Часть I

# Создание программ на Visual Basic .NET





### *В этой части...*

Написание **программ** — дело довольно простое. Если вас давно привлекает перспектива программирования на компьютере, но достижению цели препятствуют трудно читаемые книги, неприветливое программное обеспечение или путанные и непонятные объяснения “экспертов”, то настоящая **книга** — для **вас**.

Мы не хотим вас удивить математическими доказательствами и теоретическими выкладками по компьютерному программированию. Мы просто поможем вам освоить Visual Basic .NET, с тем чтобы вы смогли создавать **собственные** программы и без посторонней **помощи**. В этой главе вы узнаете, как начать писать **программы** на Visual Basic .NET, как самому разработать пользовательский интерфейс и как использовать коды BASIC, чтобы программы начали делать что-нибудь полезное.

Так что перекусите чем-нибудь, выпейте минеральной **водички**, устройтесь поудобнее и приступайте к освоению программирования. Рано или поздно ваш компьютер наверняка сможет делать то, что вы от него хотите.

# Как работает Visual Basic .NET

*В этой главе...*

- Написание программ — туман рассеивается
- Что такое Visual Basic .NET
- Ограничения, касающиеся приставки .NET

**О**бщая цель программирования — это написание команд для компьютера, которые смогут заставить его сделать что-нибудь полезное, например напечатать отчет, составить бюджет или запустить ядерную ракету в огород вашего соседа.

Так повелось, что программирование в основном используется для написания команд двух видов. Команды первого вида предназначены для отображения информации на экране (называемой пользовательским интерфейсом), а команды второго вида — для манипулирования данными, например для суммирования или умножения чисел.

К сожалению, для большинства людей программирование составляет определенные трудности. Первая и основная трудность связана с тем, что языки программирования наподобие C или C++ слишком формализованы и сложны для понимания. Попытка изучить C или C++ часто сродни попытке человека, владеющего русским или английским, научиться писать и читать на арабском или японском. Вторая трудность состоит в том, что пока вы напишете команды для создания пользовательского интерфейса, у вас уже не останется сил и времени на создание команд, которые делали бы что-нибудь действительно полезное.

Чтобы устранить обе проблемы, компания Microsoft разработала язык Visual Basic .NET, который объединил в себе язык программирования BASIC и возможности быстрого создания пользовательского интерфейса простыми методами. В отличие от многих других языков программирования, BASIC был специально разработан для обучения основам программирования начинающих пользователей. Если вы уже владеете каким-либо языком программирования, то наверняка оцените простоту и доступность BASIC.

Что касается визуальных возможностей Visual Basic .NET, то здесь вы можете создавать свой пользовательский интерфейс без написания отдельных команд. Другими словами, вы можете потратить время на написание работающих программ, а затем, уже без лишних усилий, создать для них пользовательский интерфейс.

## Написание программ на Visual Basic .NET

Приступая к написанию любой программы, первым делом нужно решить, что эта программа должна делать. Когда вы будете точно знать, что хотите получить, вы без труда сможете определить, какие команды нужно дать компьютеру, чтобы он в точности выполнил ваши пожелания.



Не существует единого, "правильного" способа написания программы. Теоретически существуют тысячи способов написания одной и той же программы. Два человека могут написать программы, которые будут работать абсолютно одинаково, но при этом они будут состоять из совершенно разных команд. Поэтому не важно, как написана программа. Важно лишь то, что она работает так, как вам нужно.

Обычно, набору команд на компьютере предшествуют три шага.

1. Нужно решить, что программа должна делать.
2. Необходимо определить основные действия, которые должен произвести компьютер, чтобы выполнить поставленную задачу. (Если вы хотите, чтобы программа посчитала стоимость всех канцелярских штучек, которые вы стащили с работы, нужно написать команды, объясняющие компьютеру, что именно необходимо посчитать, как определить стоимость каждой принадлежности и что в конечном счете все это нужно просуммировать.)
3. Нужно решить, как программа должна выглядеть на экране монитора.



Второй и третий шаг совершенно не зависят друг от друга. Вы можете вначале придумать, как программа будет выглядеть на экране, а затем определить основные действия, которые она должна выполнить.

Когда вы будете точно знать, что программа должна делать, какие шаги должны быть предприняты для достижения цели и что при этом должно отображаться на экране, вы можете садиться за компьютер и приступать к созданию своей программы.

На самом деле принятие решения о том, что именно нужно получить от компьютера, пожалуй, является самым трудным и наиболее важным шагом в деле создания программ. Ведь программирование — это всего лишь написание правильных команд с целью заставить компьютер делать то, что вы от него хотите.

## Создание пользовательского интерфейса

Visual Basic воплощает в жизнь идею быстрого создания пользовательского интерфейса простым методом. С его помощью "нарисовать" интерфейс так же легко, как изобразить изогнутую линию, окружность или улыбающуюся рожицу в программе для рисования, например в Microsoft Paint.

Пользовательский интерфейс служит двум целям: отображает информацию на экране и принимает от пользователя команды и дополнительные данные. В Visual Basic .NET все элементы интерфейса делятся на две группы; формы и объекты.

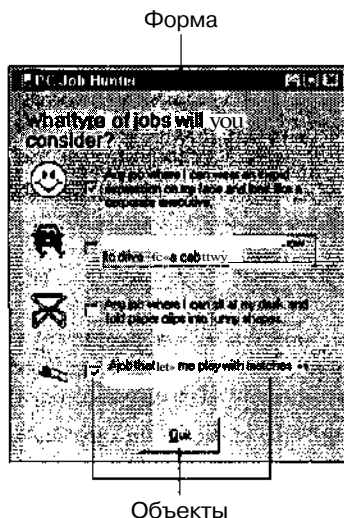


Рис. 1.1. Форма и объекты, содержащиеся в форме

Форма — это всего лишь прямоугольное окно, отображаемое на экране. Объекты представляют собой элементы, которые содержатся в форме и служат для отображения или получения информации от пользователя. Объектом может быть текст, кнопка, флажок опции и т.д. (рис. 1.1).



Не всем программам нужен пользовательский интерфейс. Но если вы собираетесь написать программу, которая предполагает получение некоторой информации от пользователя, то хоть какой-то интерфейс ей все-таки нужен.

## Определение свойств элементов интерфейса

После того как вы создали форму и разместили в ней нужные объекты, эту форму и ее объекты нужно подогнать под конкретную программу. С точки зрения Visual Basic .NET, каждый элемент интерфейса имеет набор собственных *свойств*, который определяет, как этот объект выглядит и что делает. В частности, свойствами определяются такие атрибуты объекта, как размер, размещение на экране, цвет и т.д.

Разные элементы имеют разные свойства. Каждый раз, когда вы рисуете форму или объект, Visual Basic .NET назначает им свойства, установленные по умолчанию. Но если вы не хотите создать безликую программу, то вам просто необходимо изменить некоторые (не обязательно все) свойства для каждого нарисованного вами объекта или формы.

## Написание команд на языке BASIC

В отличие от традиционных языков программирования (в том числе от уже упоминаемых C и C++), Visual Basic .NET избавляет вас от необходимости использовать команды (программисты называют их также *кодами*) для создания пользовательского интерфейса, но эти команды все же необходимы для того, чтобы ваша программа заработала. В мире Visual Basic .NET команды служат двум целям: делают пользовательский интерфейс рабочим, а также обрабатывают различные данные, с тем чтобы вернуть пользователю полезный результат.

Даже если вы с применением Visual Basic .NET создали какой-то пользовательский интерфейс, он не будет работать до тех пор, пока вы не напишете команду, объясняющую компьютеру, что нужно делать, если пользователь, скажем, щелкнул на кнопке. Каждый раз, когда пользователь перемещает курсор, нажимает клавишу или щелкает кнопкой мыши, то есть выполняет действие, таковое рассматривается компьютером как *событие*. Когда такое событие происходит, компьютер обращается к командам BASIC, с тем чтобы они объяснили, как на это нужно реагировать (отобразить диалоговое окно, издать какой-нибудь звук или, предположим, сохранить в памяти данные, показанные на экране).

Помимо команд BASIC, делающих пользовательский интерфейс рабочим, вам также нужно написать команды, выполняющие какую-нибудь полезную работу, например такую, как вычисление общей суммы и отображение на экране итогового значения.

Разделяя программу на несколько отдельных частей, Visual Basic .NET позволяет вам сосредоточить свое внимание на работе с каждой из них, не беспокоясь о том, что одна часть программы после внесения в нее изменений может войти в противоречие с другой частью. Таким образом, Visual Basic .NET поможет вам создавать даже очень сложные программы быстрее и более простым способом, чем любой другой существующий на сегодняшний день язык программирования.

# Что означает приставка .NET

Visual Basic .NET — это самая последняя версия Visual Basic. Чтобы понять суть изменений, внесенных Microsoft в язык Visual Basic, в первую очередь нужно понять, что скрывается за приставкой .NET.

Как известно, все языки программирования создаются для решения самых разнообразных задач. Однако большинство программ пишется с использованием только одного языка, например, С или BASIC. И тот факт, что языки программирования не приспособлены для совместной работы, является большой проблемой.

Даже языки программирования, разработанные одной компанией (например, Microsoft), вряд ли смогут функционировать совместно. Попытка написать программу с использованием более ранних версий языков Visual C++ и Visual Basic будет сопряжена со многими трудностями, поскольку оба языка сохраняют данные, в частности строки и числа, по-разному. Поскольку изучение и запоминание способов сохранения и манипулирования данными, применяемых в разных языках программирования — дело весьма трудоемкое и сомнительное, большинство программистов в своей работе используют какой-то один язык, даже если для решения определенной задачи более подходящим является какой-либо другой язык программирования.

Итак, компания Microsoft разработала нечто, называемое средой .NET и функционирующее в качестве посредника между операционной системой и всеми написанными вами программами. Это позволило решить сразу две очень важные проблемы.

Первая решенная проблема — это возможность совместной работы программ, написанных на разных языках программирования. Вместо того чтобы предоставлять каждой программе непосредственный доступ к операционной системе (где программа будет сохранять данные своим особым способом), среда .NET выступает посредником между операционной системой и различными программами (которые написаны на языках, имеющих приставку .NET, например Visual Basic .NET) и следит за тем, чтобы все данные были сохранены одинаково. Таким образом, вы можете на разных языках программирования писать программы, которые будут иметь доступ к одним и тем же переменным, и не сомневаться, что данные, сохраненные одной программой, будут восприняты другой.

Вторая решенная проблема касается распространения программ. Сейчас большинство пользователей запускает программы, сохраненные на жестких дисках. В свою очередь, среда .NET позволяет запускать программы через Internet и локальные сети. Другими словами, вы можете сохранить копию программы на одном компьютере, а затем, используя Internet или локальную компьютерную сеть (LAN), запустить эту программу на других компьютерах. Теперь, если нужно, скажем, обновить программное обеспечение, нет необходимости делать это отдельно для каждого компьютера, стоящего в вашем офисе, — достаточно обновить программу только на том компьютере, на котором она установлена. И самое приятное: используя языки программирования с приставкой .NET, можно создавать программы, которые будут работать в Internet и иметь при этом пользовательский интерфейс ничуть не хуже, чем у настольных прикладных программ.



Программа, созданная с использованием языка программирования с приставкой .NET, теоретически будет работать на любом компьютере, где поддерживается среда .NET. К моменту написания этой книги единственной операционной системой, поддерживающей указанную среду, была Microsoft Windows. Но если компания Microsoft (или кто-то еще) адаптирует среду .NET к другим операционным системам (предположим, к Linux или Macintosh), вы сможете писать программы на языке Visual Basic .NET и запускать их потом в разных операционных системах.

# Недостатки концепции .NET

Вы наверняка уже научены горьким опытом и знаете, что любая программа может рано или поздно зависнуть и что новые, более сложные технологии далеко не всегда могут помочь в решении наболевших проблем. Поэтому, хотя компания Microsoft пророчит радужные перспективы для среды .NET, не стоит приукрашивать действительность.

Среда .NET сама по себе является программным обеспечением, а это означает, что она имеет предрасположенность к возникновению различных ошибок и сбоев, которые наверняка не замедлят себя проявить. А что еще хуже, так это то, что кроме Windows XP среда .NET не поддерживается никакой другой версией Windows. Если вы захотите написать программу, которая должна работать, скажем, с операционной системой Windows 95 (или даже не с Windows, а с Macintosh или Linux), вы не сможете применить язык программирования с приставкой .NET, которым, в частности, является Visual Basic .NET. (Чтобы написать программу для предыдущих версий Windows, нужно будет использовать более ранние версии Visual Basic, например Visual Basic 6.0.)

Другая проблема заключается в том, что не все языки программирования могут работать со средой .NET. Microsoft предлагает для этого языки Visual C++, Visual Basic и C# (обновленная версия C++). Что же касается других языков, то здесь приходится только надеяться, что разработчики в будущем приспособят их для работы со средой .NET. Если вы изучили язык программирования, который не работает со средой .NET, то эта среда для вас будет недоступной.

Для тех программистов, которые уже в совершенстве овладели Visual Basic, наибольшей неприятностью будет то, что Microsoft внесла в него много изменений, с тем чтобы Visual Basic .NET обрабатывал числа и строки так же, как это делается в C++. Это значит, что программы, написанные на Visual Basic 6.0, могут не запуститься в среде .NET и потребуют внесения значительных изменений.

Итак, если вы собираетесь использовать Visual Basic .NET, взвесьте все "за" и "против". В обмен на совместимость с другими языками программирования, такими как Visual C++, вы теряете совместимость с более ранними версиями Visual Basic. Но несмотря на эти проблемы, Visual Basic .NET по-прежнему остается самым быстрым и наиболее легким средством создания качественных программ.



# Пользовательский интерфейс приложения Visual Basic .NET

*В этой главе...*

- Запуск Visual Basic .NET
- > Знакомство с пользовательским интерфейсом
- > Открытие, закрытие и перемещение окон
- > Выход из Visual Basic .NET

**П**режде чем приступить к созданию собственных программ, необходимо познакомиться с пользовательским интерфейсом самого приложения Visual Basic .NET, состоящего из открывающихся меню и их команд, специальных окон, отображающих различную информацию о программе, и панелей инструментов с расположенными на них кнопками наиболее часто используемых команд.

В мире Visual Basic .NET написание программ на языке Visual Basic .NET называется созданием *проекта*. Проект состоит из одного или нескольких файлов, сохраненных в одной папке. Каждый раз, когда вы приступаете к разработке нового проекта, Visual Basic .NET создает новую папку и сохраняет в ней все файлы. Таким образом, вы всегда можете быть уверены, что файлы, относящиеся к одному проекту, случайным образом не будут перемешаны с файлами другого проекта.

Чтобы пользоваться приложением Visual Basic .NET, вы должны знать три вещи:

- ✓ как запустить Visual Basic .NET;
- ✓ как использовать Visual Basic .NET для написания программ;
- ✓ как выйти из Visual Basic .NET.

## *Запуск Visual Basic .NET*

Чтобы запустить Visual Basic .NET, нужно всего лишь сделать следующее.

1. Щелкнуть на кнопке Пуск панели задач Windows.  
Откроется меню кнопки Пуск.
2. Щелкнуть в строке Программы, щелкнуть на пункте Microsoft Visual Studio.NET, а затем снова щелкнуть на Microsoft Visual Studio.NET.

Приложение Visual Basic .NET начнет работать, и откроется окно Start Page (Стартовая страница), показанное на рис. 2.1.

Когда окно Start Page открыто, вы можете:



- ✓ приступить к созданию нового проекта (если это входит в ваши планы);
- ✓ загрузить существующий проект (таким образом вы сможете вносить изменения в уже созданные программы);
- ✓ настроить пользовательский интерфейс Visual Basic .NET.

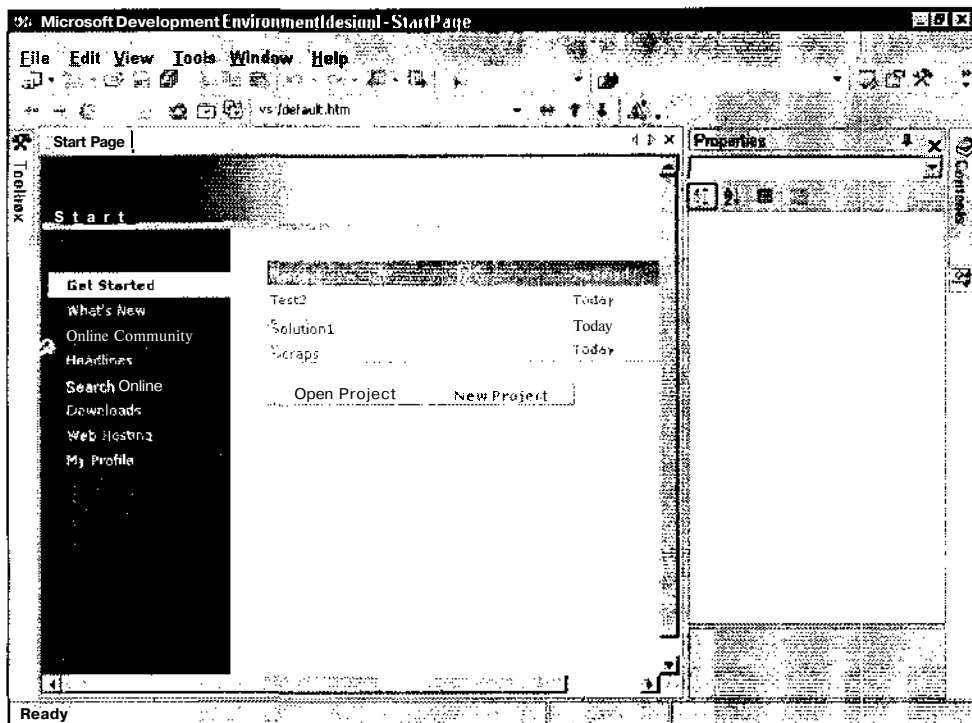


Рис. 2.1. Окно StartPage открытия которого начинается работа Visual Basic .NET

## Приступаем к созданию нового проекта



В мире Visual Basic .NET *проект* рассматривается как один или несколько файлов, которые работают совместно с целью создания единой программы.

Новый проект создается следующим образом

**1. Откройте окно создания проекта, воспользовавшись одним из перечисленных ниже методов:**

- в окне Start Page щелкните на кнопке **New Project** (Создать проект);
- выберите команду **File⇒New⇒Project** (Файл⇒Создать⇒Проект);
- нажмите комбинацию клавиш **<Ctrl+Shift+N>**.

Вне зависимости от того, какой из трех методов вы выберете, Visual Basic .NET отобразит диалоговое окно **New Project** (Новый проект), показанное на рис. 2.2.

- Щелкните на папке **Visual Basic Projects (Проекты Visual Basic)** в разделе **Project Types (Тип проекта)**.

В разделе **Templates (Шаблоны)** откроется список различных шаблонов Visual Basic.NET.

- Щелкните на шаблоне, который вы хотите использовать при создании своей программы.

Если вы хотите повторить пример, рассмотренный далее, щелкните на шаблоне **Windows Application**.

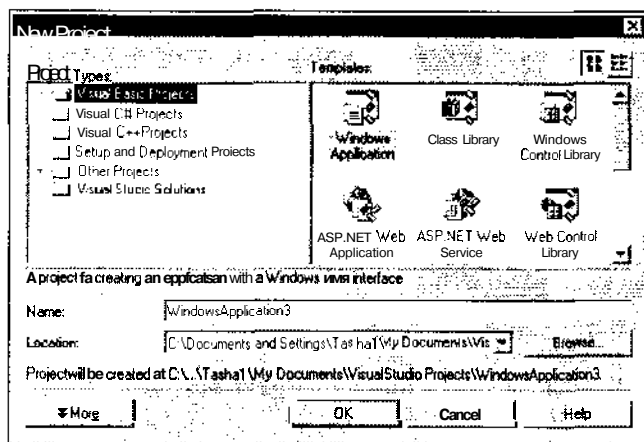


Рис. 2.2. Диалоговое окно *New Project* содержит опции, отвечающие за создание новой программы



Только опытные программисты могут использовать в своей работе шаблоны **ASP.NET Web Service** и **Windows Control Library**, поэтому не применяйте их, пока не научитесь создавать более простые (**Windows Application**) программы.

- Щелкните в строке **Name (Название)** и укажите в ней имя нового проекта.

Вы можете пропустить шаги с пятого по седьмой, если не хотите создавать специальную папку для записи нового проекта.

- Щелкните на кнопке **Browse (Обзор)**.

Откроется диалоговое окно **Project Location (Размещение проекта)**.

- Укажите папку, в которой хотите сохранить будущий проект **Visual Basic .NET**.

Для записи нового проекта вы можете не только выбрать уже существующую папку, но и создать новую.

- Щелкните на кнопке **Open (Открыть)**.

Вы снова перейдете в диалоговое окно **New Project**.

- Щелкните на кнопке **OK**.

Visual Basic .NET откроет пустую форму, чтобы вы могли начать создавать пользовательский интерфейс для своей программы.

## Открытие существующих проектов

Может случиться так, что вы захотите открыть существующий проект для внесения в него некоторых изменений. Чтобы сделать это, выполните следующие шаги.

**1. Откройте диалоговое окно Open Project (Открыть проект), выбрав один из следующих методов:**

- в окне **Start Page** щелкните на кнопке **Open Project**;
- выберите команду **File⇒Open⇒Project**;
- нажмите комбинацию клавиш **<Ctrl+Shift+O>**.

**2. Дважды щелкните на папке, в которой содержится нужный вам проект.**

Откроется список файлов, относящихся к вашему проекту.

**3. Щелкните на файле проекта, который хотите открыть.**

Когда курсор мыши указывает на какой-либо файл, рядом с ним открывается маленькое окошко с определением типа этого файла (например, Visual Basic .NET Project file).

**4. Щелкните на кнопке Open.**

После этого вы можете приступить к внесению изменений в свой проект.



Чтобы ускорить процесс открытия существующих проектов, Visual Basic .NET предлагает еще две возможности.

- ✓ В окне **Start Page** отображаются названия нескольких проектов, сохраненных последними. Щелкните на нужном имени, и Visual Basic .NET откроет для вас этот проект.
- ✓ Выберите команду **File⇒Recent Project** (Файл⇒Последние проекты), чтобы увидеть список последних открываемых проектов. Щелкните на названии проекта, который вы хотите загрузить, и он тут же появится на экране.

## *Добро пожаловать в пользовательский интерфейс Visual Basic .NET*

Будете вы создавать новый проект или откроете уже существующий — в любом случае вам необходимо уметь пользоваться интерфейсом приложения Visual Basic .NET. Основные составляющие пользовательского интерфейса Visual Basic .NET представлены на рис. 2.3, однако далеко не все элементы интерфейса могут быть одновременно отображены на экране.

- ✓ **Открывающиеся меню.** С их помощью можно получить доступ к любой команде Visual Basic .NET, однако постоянно использовать их в своей работе не очень удобно.
- ✓ **Панели инструментов.** Содержат кнопки наиболее часто используемых команд Visual Basic .NET, однако они также не являются самым удобным средством в работе с Visual Basic .NET.



Создавая пользовательский интерфейс для своей программы, для выбора объектов, которые будут нарисованы в окне формы, вы используете панель Toolbox. Когда объекты нарисованы, следующим шагом будет настройка их отображения в окне Properties. И наконец, когда пользовательский интерфейс разработан, вы можете переключиться на вкладку Code и приступить к написанию команд BASIC, с тем чтобы созданный интерфейс приводил в движение реальные механизмы.

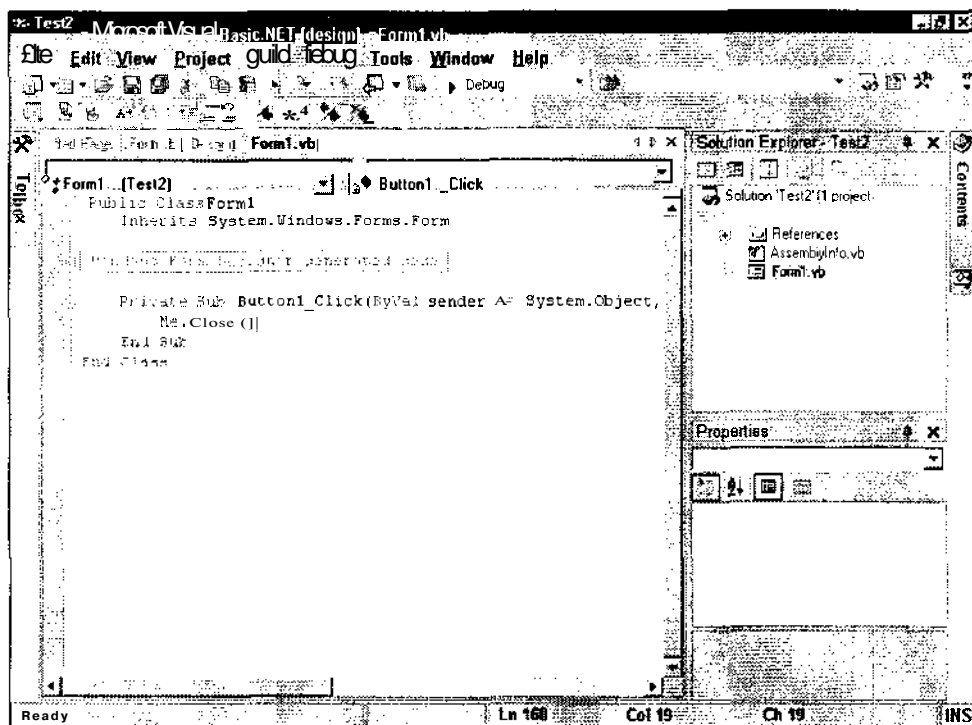
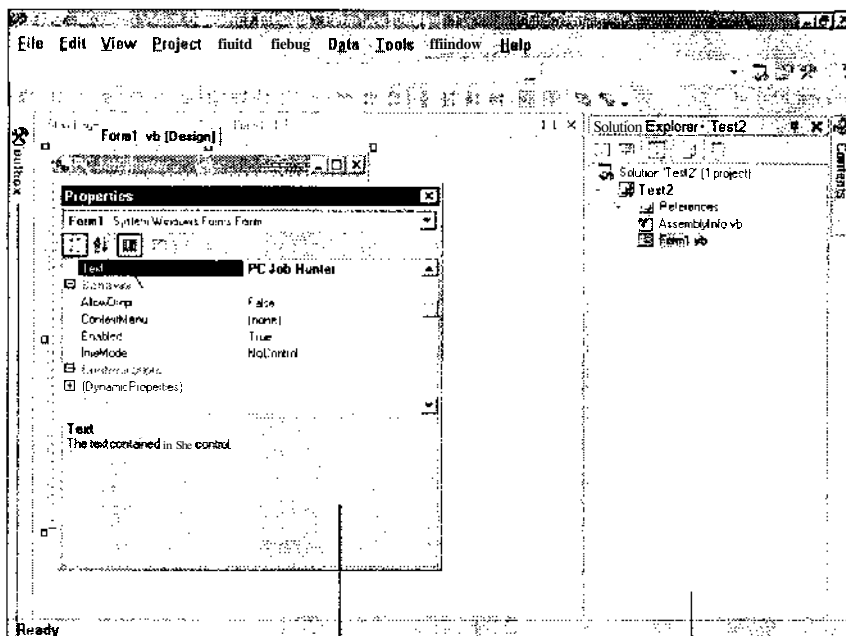


Рис.2.4. Щелкнув на вкладке Code, вы можете просмотреть и отредактировать коды BASIC, для того чтобы сделать объекты формы (кнопки, флажки и т.п.) действительно рабочими

## Манипулирование окнами

В Visual Basic .NET имеется три способа отображения окон на экране (рис. 2.5).

- ✓ **Плавающее окно.** Окно ни к чему не привязано и может быть размещено в любом месте экрана.
- ✓ **Закрепленное окно.** Окно растянуто вдоль нижней, верхней или боковой стороны экрана.
- ✓ **Скрытое окно.** Окно автоматически сворачивается после того, как курсор мыши покидает его пределы.



Скрытое окно

Плавающее окно

Закрепленное окно

Рис. 2.5. Три способа отображения окна на экране

### Как сделать окно плавающим

Чтобы окно стало плавающим и его можно было разместить в любом месте экрана, сделайте следующее.

1. Для начала **отобразите** окно на экране. Выберите команду View (Вил) и щелкните на имени нужного окна (например, Properties или Solution Explorer).
2. **Поместите курсор мыши на заголовке окна, нажмите левую кнопку мыши и перетащите окно на середину экрана.**  
Visual Basic .NET при перетаскивании отображает серый контур окна, так что вы сможете увидеть, как оно будет размещено после того, как вы отпустите кнопку мыши.
3. Отпустите кнопку мыши,  
Поздравляем! Вы только что создали плавающее окно.



Если дважды щелкнуть на заголовке плавающего окна, оно снова станет закрепленным.

### Как сделать окно закрепленным

Вот как можно прикрепить окно к какой-нибудь стороне экрана.

1. **Отобразите окно на экране. Выберите команду View и щелкните на имени нужного окна (например, Solution Explorer или Toolbox).**
2. **Поместите курсор мыши на заголовке окна, нажмите левую кнопку мыши и перетащите окно к какой-нибудь стороне экрана.**  
Visual Basic .NET при перетаскивании отображает серый контур окна, так что вы сможете увидеть, как оно будет размещено после того, как вы отпустите кнопку.

3. Отпустите кнопку мыши после того, как контур окна окажется "привязанным" к стороне экрана.

В результате выполнения перечисленных действий окно становится закрепленным.



Дважды щелкнув на заголовке окна, закрепленное окно можно снова сделать плавающим.

## Скрытие окон

Чтобы освободить место на экране, окно можно временно скрыть, оставив видимым только его название, отображаемое как корешок вкладки возле одной из сторон экрана. Когда вы захотите снова увидеть окно на экране, поместите курсор на корешок вкладки, и оно снова отобразится во всей своей красе.

А вот как можно скрыть окно.

1. Пройдите шаги 1 и 2, описанные в разделе "Как сделать окно закрепленным".

Когда окно отображается как закрепленное возле одной из сторон экрана, в его правом верхнем углу появляется значок автоматического скрывтия.

2. Щелкните на значке автоматического скрывтия.

Visual Basic .NET скроет окно и отобразит корешок вкладки возле боковой стороны экрана.

3. В любой момент, когда вы снова захотите увидеть окно, поместите курсор мыши на корешок вкладки.



Скрыть можно сразу все закрепленные окна (это не касается плавающих), выбрав команду **Window⇒Auto Hide All** (Окно⇒Скрыть все).

## Заккрытие окна

Если вы хотите совсем закрыть окно, достаточно щелкнуть на соответствующей кнопке, расположенной в его правом верхнем углу.

# Выход из Visual Basic .NET

Как бы вы ни любили проводить свое время за программированием на Visual Basic .NET, иногда все же необходимо оторваться от компьютера и пойти поспать (или уделить немного внимания **своей** жене и детям). Для выхода из Visual Basic .NET вы можете воспользоваться одним из следующих методов:

- ✓ выбрать команду **File⇒Exit** (Файл⇒Выход);
- ✓ нажать комбинацию клавиш **<Alt+F4>**;
- ✓ щелкнуть на кнопке закрытия в окне Visual Basic .NET.

Если вы не сохранили открытую на текущий момент программу, Visual Basic .NET откроет диалоговое окно и предложит сохранить результат проделанной вами работы. После щелчка на кнопке **No** все изменения, внесенные в программу в течение последнего сеанса работы с Visual Basic .NET, будут безвозвратно утеряны. Поэтому, если вы хотите сохранить результаты своего труда, щелкните на кнопке **Yes**, потому как второго такого шанса уже не будет.

После этого Visual Basic .NET плавно завершит свою работу, и вы вновь вернетесь к знакомому окну Windows.

# Создаем пользовательский интерфейс

*В этой главе...*

- > Обзор основных компонентов пользовательского интерфейса
- Как нарисовать интерфейс
- > Изменение свойств созданных элементов

**П**ользовательский интерфейс предназначен для отображения информации на экране и для получения данных от пользователя. Если вы создадите простой и понятный интерфейс, написанная вами программа также будет казаться доступной и удобной. Сделайте интерфейс сложным и запутанным, и тогда можно будет продать кому-нибудь эту “серьезную” программу и написать книгу на 500 страниц с объяснением того, как “проста” эта программа в использовании.

Чтобы дать вам общее представление о возможностях Visual Basic .NET, в этой главе (а также в главе 5) мы рассмотрим основные этапы создания пользовательского интерфейса:

- ✓ этап рисования интерфейса;
- ✓ определение свойств элементов интерфейса;
- ✓ написание кодов BASIC (этот этап описан в главе 4)

## Основные компоненты пользовательского интерфейса

Стандартный пользовательский интерфейс состоит из окон, в которых отображаются различные элементы, в первую очередь текст и картинки. Окна могут заполнять собою весь экран или только его часть. Одновременно могут быть открыты несколько окон. Они могут как карты накрывать друг друга, а могут быть просто расположены рядом. На языке Visual Basic .NET окна называются *формами*.

Если вы создаете новую форму, то вначале она будет пустой. Чтобы сделать ее чем-то полезной, нужно нарисовать на ней объекты. *Объектом* может служить кнопка, текстовое поле, картинка, переключатель и т.д. В дальнейшем пользователи смогут общаться с вашей программой посредством щелканья на кнопках, ввода текста или манипулирования с объектами, содержащимися в форме.

## Использование панели Toolbox для рисования объектов

Чтобы нарисовать объект, нужно воспользоваться панелью элементов Toolbox, которая обычно находится в левой части экрана (если вы ее еще не переместили в какое-нибудь другое место). На панели расположены значки и перечислены соответствующие им имена, представляющие различные типы объектов, которые вы можете нарисовать в окне формы (рис. 3.1).



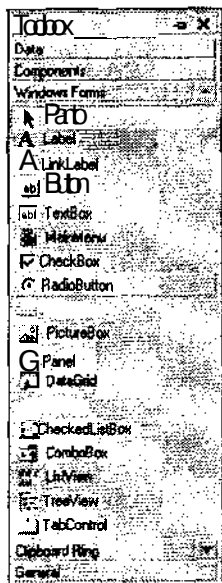


Рис. 3.1. На панели Toolbox вы найдете значки объектов всех типов, которые можно нарисовать в окне формы

Чтобы нарисовать любой объект, нужно придерживаться следующей последовательности действий.

1. Щелкните на кнопке объекта в панели Toolbox, чтобы сообщить Visual Basic о своем желании нарисовать его в окне формы.
2. Поместите курсор мыши в то место формы, где должен быть нарисован объект.
3. Нажмите левую кнопку мыши и перетяните курсор, чтобы нарисовать объект.



Если вы дважды щелкнете на кнопке объекта в панели Toolbox, Visual Basic .NET автоматически нарисует этот объект в окне открытой в данный момент формы.

## Создание вашего первого пользовательского интерфейса

Чтобы немедленно приступить к практическому освоению Visual Basic .NET, воспользуйтесь следующим руководством по созданию реально работающего пользовательского интерфейса.

1. Запустите Visual Basic .NET и выберите команду **File⇒New⇒Project** (Файл⇒Создать⇒Проект).

Visual Basic .NET отобразит диалоговое окно **New Project** (Создать проект), в котором вы сможете выбрать тип создаваемой программы.

2. Щелкните на значке **Windows Application** (Приложение Windows).
3. Щелкните в строке **Name** (Название), наберите слово **Hello**, а затем щелкните на кнопке **OK**.

Visual Basic .NET отобразит пустую форму, именуемую **Form1**.

4. Поместите курсор мыши над правым нижним углом формы (прямо над маленьким квадратиком, называемым маркером, который будет отображен

напротив угла формы) так, чтобы он принял вид двунаправленной стрелки. Нажмите левую кнопку мыши и перетащите курсор, придав окну формы нужный размер.

5. Выберите команду **View**⇒**Toolbox**, чтобы на своем обычном месте в левой части экрана отобразилась панель **Toolbox**.

(Пропустите этот шаг если панель **Toolbox** уже отображена.)

6. В панели **Toolbox** щелкните на значке **Button** (Кнопка).

7. Поместите курсор в окно формы, а затем перетащите его, чтобы нарисовать кнопку (рис. 3.2).

Не стоит особо беспокоиться по поводу точного размещения объекта в окне формы. Попробуйте лишь придать ему вид, подобный тому, который вы видите на рис. 3.2.

8. В панели **Toolbox** щелкните на кнопке **PictureBox** (Рисунок) и нарисуйте объект в окне формы.

Повторите этот процесс еще два раза, чтобы выделить места для трех рисунков. На рис. 3.3 показано, как эти места могут быть распределены. (Обратите внимание: на рисунке все три объекта выделены, благодаря чему вы их видите.)

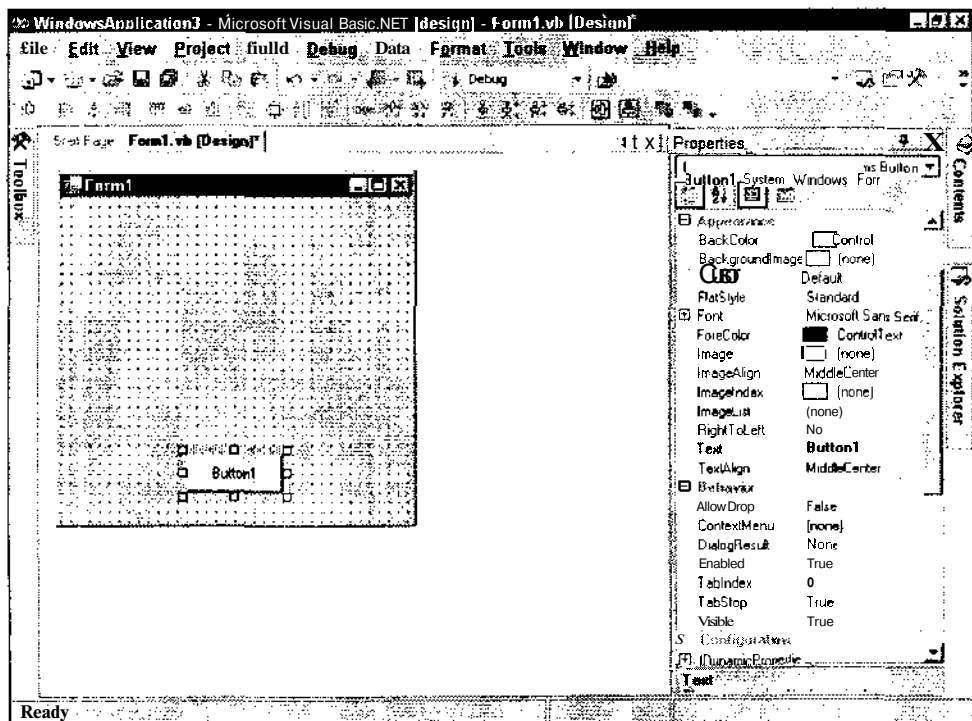


Рис. 3.2. Создание кнопки в окне формы

9. Щелкните на значке объекта **Label** (Надпись) и нарисуйте его в окне формы (рис. 3.4).

10. Выберите команду **File**⇒**Save All** (Файл⇒Сохранить все).

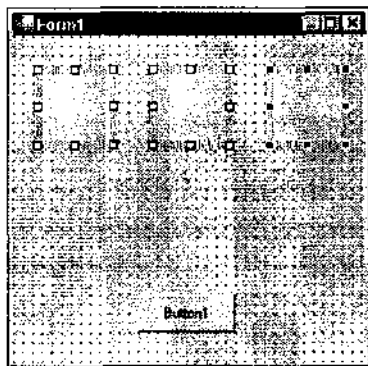


Рис. 3.3. Выделение места для трех рисунков

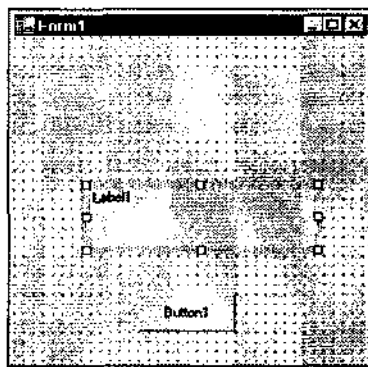


Рис. 3.4. Создание надписи в окне формы

#### 11. Нажмите клавишу <F5> (или выберите команду Debug⇒Start).

Visual Basic .NET запустит вашу программу, и вы сможете увидеть созданный интерфейс. Поскольку вы пока еще не изменили установленные по умолчанию свойства объекта, ваш пользовательский интерфейс будет выглядеть пустым и обезличенным. Как сделать его более презентабельным, вы узнаете в следующем разделе.

#### 12. Щелкните на кнопке закрытия в правом верхнем углу формы.

Visual Basic .NET остановит выполнение программы, и вы сможете продолжить процесс ее создания.

## Определение свойств элементов интерфейса

Прорисовка пользовательского интерфейса — это только первый шаг на пути к разработке программы Visual Basic .NET. Теперь нужно определить свойства каждого созданного объекта интерфейса.



Хотя каждый объект имеет целый набор установленных по умолчанию свойств, совсем не обязательно изменять каждое из них. В большинстве случаев достаточно модифицировать два или три свойства.

## Зачем нужны свойства

Свойства определяют характеристики объектов, такие как место расположения, размер, форма, цвет и т.д. Почти каждый объект имеет целый набор свойств, а чтобы легче было в них ориентироваться, свойства разбиты на несколько категорий (рис. 3.5).

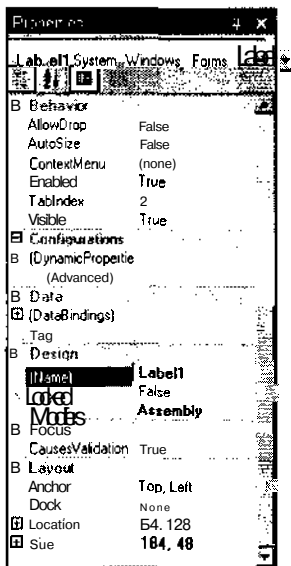


Рис. 3.5. В окне *Properties* все свойства организованы по категориям

Каждый объект может иметь свойства сразу нескольких категорий. Чаще всего вы можете встретить следующие из них:

- ✓ **Appearance** (Отображение). Свойства этой категории определяют цвета, выравнивание, текстовые надписи, которые будут отображаться на объекте.
- ✓ **Behavior** (Поведение). Определяют, как объект будет себя вести, когда пользователь щелкнет кнопкой мыши или нажмет клавишу.
- ✓ **Data** (Данные). Связывают объект с информацией, сохраненной в базе данных.
- ✓ **Design** (Проект). Определяют имя объекта, может ли он перемещаться и будет ли отображаться для других частей программы.
- ✓ **Focus** (Фокус). Определяют действия объекта в момент выбора его пользователем.
- ✓ **Layout** (Размещение). Определяют размер объекта и размещение его на экране.



Свойство **Name**, относящееся к категории **Design**, — это то свойство, которое почти всегда изменяется. Visual Basic .NET автоматически присваивает объектам стандартные имена, такие как `Text1` и `Button3`. Однако если вам нужно в тексте программы указать определенный объект, намного легче это сделать, если присвоить ему легко запоминающееся имя.

## Внесение изменений в свойства объектов

Свойства объектов могут изменяться на двух этапах:

- ✓ на стадии разработки интерфейса;
- ✓ во время выполнения программы.

На стадии разработки интерфейса в основном определяются размер, цвет и местоположение объекта.



В большинстве случаев свойства объектов устанавливаются именно на данном этапе. Наиболее важное **свойство**, которое обычно изменяется, — это имя объекта.

Чтобы свойства объектов можно было изменять в процессе выполнения программы, нужно написать соответствующие коды BASIC. Таким образом поддерживается диалог с пользователем. Например, изменяя свойства объектов, можно отобразить на экране какое-нибудь сообщение или переместить объект в другое место.

## Изменение свойств объектов на стадии разработки интерфейса

Изменить свойства объектов в процессе создания интерфейса очень просто. Делается это следующим образом.

1. **Щелкните на объекте, свойства которого необходимо изменить.**
2. **Откройте окно Properties (Свойства) и щелкните на названии нужного свойства.**  
Чтобы открыть окно Properties, нажмите клавишу <F4>.
3. **Наберите или выберите из списка новое значение этого свойства.**

Ничего сложного, не так ли? Если вы будете следовать нашим советам и повторять на компьютере все приведенные в этой книге примеры, вам довольно часто придется изменять свойства объектов. Поэтому, когда нужно будет изменить набор свойств для одного или нескольких объектов, в книге будет просто приведена таблица, подобная представленной ниже.

Объект	Свойство	Значение
Form	Text	Hello, world!

Это значит, что вам нужно будет выполнить следующие действия.

1. Щелкнуть на объекте Form (Форма).
2. В окне Properties (Свойства) щелкнуть на свойстве Text (Текст).
3. В поле свойства Text набрать He! ] o, world! (Здравствуй, мир!).

### Тест на проверку полученных вами знаний

1. Из каких двух основных частей состоит пользовательский интерфейс?
  - а. Из простого в использовании интерфейса и трех томов руководства по использованию этого интерфейса.
  - б. Из огромного сложного меню и большой кнопки "Выход из программы".
  - в. Из монитора и клавиатуры.
  - г. Из форм и объектов.
2. Как можно изменить свойства объектов?
  - а. Никак. Они сами иногда меняются.
  - б. Послать объекту сообщение с просьбой измениться.
  - в. Воспользоваться окном Properties во время создания интерфейса либо написать коды BASIC для изменения свойств объектов в процессе выполнения программы.
  - г. Сидя перед монитором, громко назвать свою фамилию, нужный объект и новое свойство.

## Определение свойств для интерфейса вашей первой программы

Вот как можно определить свойства формы, которую вы начали создавать в этой главе.

1. Запустите приложение Visual Basic .NET. (Если оно уже работает, пропустите этот шаг.)
2. Выберите команду File⇒Recent Projects⇒Hello.sln (Файл⇒Последние проекты⇒Hello.sln).

Выполняя действия, описанные в разделе “Создание вашего первого пользовательского интерфейса”, на шаге 3 вы должны были дать своему проекту название Hello. Если же вы указали какое-нибудь другое имя, то сейчас выберите именно его.

3. Щелкните на рисунке, который расположен крайним слева.
4. Нажмите клавишу <F4>, чтобы открыть окно Properties.
5. Дважды щелкните на свойстве Name (оно относится к категории Design) и наберите picSmile.
6. Щелкните на свойстве Image (Изображение), которое относится к категории Appearance (Отображение).

Появится кнопка с тремя точками (...).

7. Щелкните на кнопке с тремя точками (...).  
Откроется диалоговое окно Open.
8. Откройте папку с пиктограммами. (Обычно найти ее можно с помощью команды Programs⇒Microsoft Visual Studio.NET⇒Common7⇒Graphics⇒Icons).
9. Внутри папки Icons найдите папку Misc и откройте ее.

Visual Basic .NET отобразит список пиктограмм, сохраненных в папке Misc.

10. Выберите пиктограмму FACE02 и щелкните на кнопке Open.

Visual Basic .NET отобразит в области объекта picSmile маленькое улыбающееся лицо.

11. Щелкните на свойстве SizeMode (Размер, категория Behavior), затем — на кнопке со стрелкой и в открывшемся списке выберите пункт StretchImage.

Visual Basic .NET увеличит вставленный рисунок до размеров выделенной для него области.

12. Щелкните на надписи Label1.

13. Щелкните на свойстве BorderStyle (Тип границ, категория Appearance), потом — на кнопке со стрелкой и выберите пункт Fixed3D.

Visual Basic .NET придаст объекту Label1 объемный вид.

14. Дважды щелкните на свойстве Name (категория Design) и наберите lblMessage.

15. Дважды щелкните на свойстве Text (категория Appearance) и удалите установленное по умолчанию значение.

16. В соответствии с табл. 3.1 измените указанные свойства для остальных объектов.

17. Выберите команду File⇒Save All или нажмите комбинацию клавиш <Ctrl+Shift+S>.

Поздравляем! Вы только что определили все необходимые свойства для своего первого пользовательского интерфейса. Если вы все сделали правильно, ваша форма должна выглядеть так, как показано на рис. 3.6.

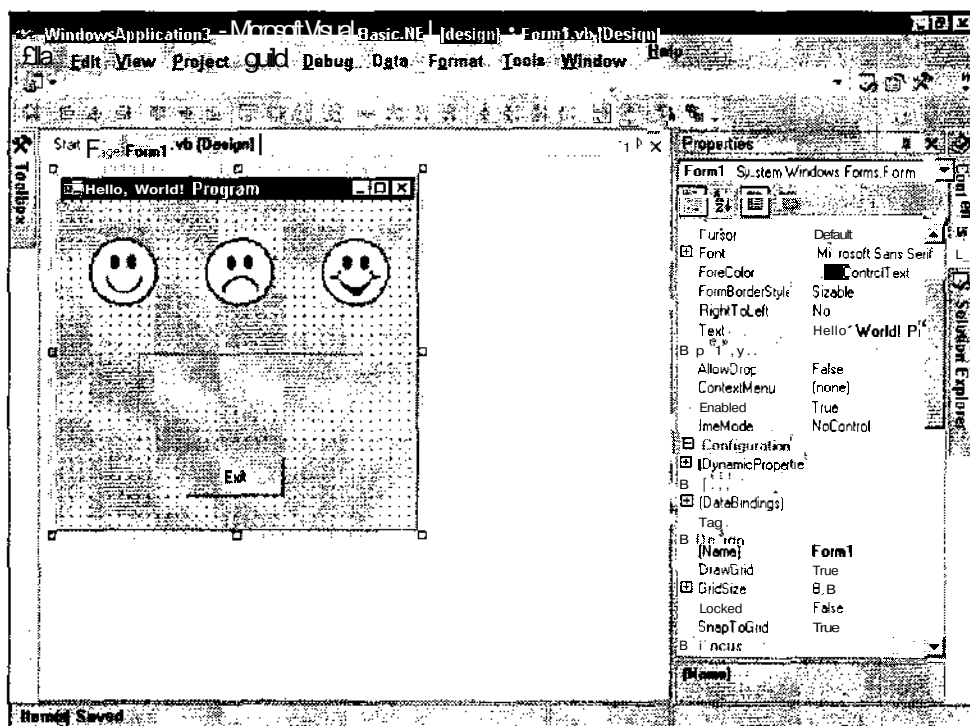


Рис. 3.6. Так должен выглядеть пользовательский интерфейс после определения всех необходимых свойств

Таблица 3.1. Свойства, которые нужно изменить для завершения разработки своего пользовательского интерфейса

Объект	Свойство	Значение
Form	Text	Hello, World! Program
PictureBox2	Name	picFrown
	Image	FACE04
	SizeMode	StretchImage
PictureBox3	Name	picHappy
	Image	FACE03
	SizeMode	StretchImage
Button 1	Name	btnExit
	Text	Exit

# Приступаем к написанию кодов BASIC

*В этой главе...*

- > Что могут делать коды BASIC
- Создание работающих процедур
- > Написание кодов BASIC для вашей программы

**В**аш компьютер начнет что-нибудь делать лишь при условии, что ему будут даны пошаговые инструкции. Если вы пропустите какой-то шаг или дадите неточные указания, компьютер не будет знать, что ему делать дальше. (Вообще-то компьютер знает, что ему делать, просто он не сможет понять, что вы от него хотите.)

Программисты называют отдельную инструкцию *командой*. Обычная команда BASIC выглядит примерно так:

```
Taxes = Income * FlatTaxRate
```

Набор команд программисты называют *кодом*. Если вы хотите найти *общий* язык с программистами (а они не разговаривают ни о чем, кроме как о программах), вам нужно изучить их сленг и этикет.

Итак, никогда не говорите программисту: "Покажи мне свой набор команд". Он вас скорее всего не поймет, а если и поймет, то сделает вид, что не понял. Вместо этого скажите: "Покажи мне свои коды".

Обычно код выглядит приблизительно следующим образом:

```
Income = 90000
```

```
FlatTaxRate = .95
```

```
Taxes = Income * FlatTaxRate
```

Совокупность кодов, которая заставляет компьютер делать что-нибудь полезное (например, играть с вами в игру, считать что-либо или изображать на экране летающий гамбургер), называется *программой*.

## *Что представляют собой коды BASIC*

Чтобы заставить компьютер что-то сделать, нужно дать инструкции, которые он сможет понять. При использовании Visual Basic .NET все команды, которые даются компьютеру, должны быть написаны на языке BASIC.

Как и все другие языки, BASIC оперирует собственным набором команд, каждой из которых соответствует свое ключевое слово. Некоторые из ключевых слов перечисленных ниже.

Loop	Function	Sub	End
Do	Integer	Case	If
Else	Select	Then	For

Коды BASIC сплошь состоят из ключевых слов, соответствующих командам языка BASIC. Всякий раз, когда компьютер видит ключевое слово, он сразу же думает: "О, это специальная инструкция для меня, теперь я знаю, что делать!"



Программа может состоять лишь из одной команды, а может вмещать в себя миллионы команд. Короткие программы чаще всего выполняют что-нибудь не очень значительное, например отображают на экране надписи наподобие "Привет, Хозяин!" Большие программы, как правило, способны на нечто более интересное, но с ними обычно и проблем побольше.

Теоретически, вы можете написать программу, содержащую миллион или более команд. Однако те программисты, которые пытались это сделать, рисковали попасть к психиатру задолго до завершения своей работы.

Чтобы превратить процесс программирования в максимально приятное занятие, принято большие программы разбивать на несколько маленьких, которые, к счастью, потом могут работать вместе. Программы, написанные на Visual Basic .NET, обычно состоят из трех частей (рис. 4.1). Вот описание каждой такой части.

- ✓ **Пользовательский интерфейс.** Информация о нем будет сохранена в файле с расширением .VB. (Для его создания коды BASIC вам писать не придется.)
- ✓ **Процедуры управления событиями.** Они сохраняются в том же файле, что и пользовательский интерфейс. (Эти процедуры создаются с использованием кодов BASIC для отображения информации на экране и получения данных от пользователей.)
- ✓ **Файлы модулей и классов.** Состоят из кодов BASIC и предназначены для проведения вычислений и манипулирования данными, полученными от пользователей. Visual Basic .NET сохраняет модули и классы в файлах с расширением .VB.

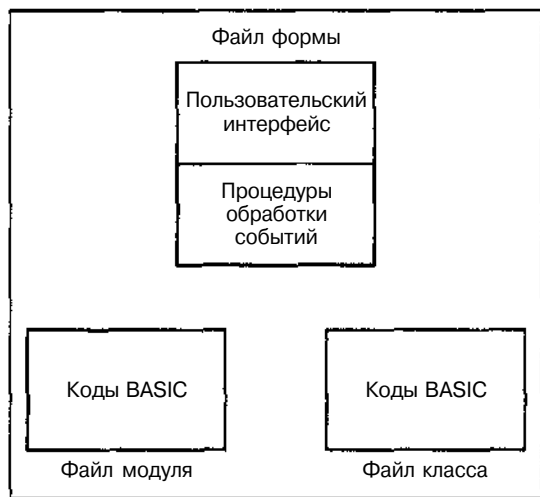


Рис. 4.1. Части, из которых обычно состоит программа Visual Basic .NET



Файлы модулей содержат коды BASIC, называемые основными процедурами (они будут рассмотрены в четвертой части книги). В Visual Basic .NET нет жесткого предписания использовать эти файлы для сохранения части программы, однако их применение помогает правильно организовать структуру программы и сделать ее удобной для модификации.



Поскольку файлы модулей и классов не обязательны, вы можете сохранить всю программу (коды для управления данными и коды пользовательского интерфейса) в одном файле. Однако если вы так сделаете, то ваша программа станет трудной для чтения и малопонятной (имеются в виду коды программы).



В интерфейсе большинства программ используется некоторое количество форм, поэтому вы можете сохранить коды BASIC, относящиеся только к вашим конкретным формам, в отдельных файлах. Те же коды, которые относятся к двум или нескольким формам одновременно, лучше сохранить в файлах модулей или классов.

Далее в этой главе рассматривается процесс создания процедур управления событиями. Обычно такие процедуры сообщают компьютеру, что нужно делать, если пользователь производит какое-нибудь действие, например, щелкает кнопкой мыши на объекте формы или нажимает какую-то клавишу в то время, когда выделен определенный объект формы.



Не для каждого объекта нужны процедуры обработки событий. Они необходимы только для тех объектов, на которых можно щелкнуть мышью или выделить как-то еще (такowymi, в частности, являются кнопки и переключатели). Более подробно о процедурах обработки событий можно прочитать в четвертой части этой книги.

## Написание процедур обработки событий

Прежде чем приступить к написанию процедуры обработки событий для объекта формы, этот объект нужно создать (точнее, нарисовать в окне формы).

Затем нужно настроить свойства созданных объектов и дать всем им имена, которые не трудно будет вспомнить. Если вы этого не сделаете, то вам придется столкнуться с именами объектов, устанавливаемыми Visual Basic .NET по умолчанию (такими, как `RadioButton1` или `Text30x3`).

### Быстрый способ создания процедуры обработки событий

Многие объекты, к числу которых относятся те же кнопки и переключатели, должны как-то реагировать на различные события, скажем на пользовательское щелканье мышкой. Чтобы сделать их способными на ответные действия, нужно написать код BASIC. Как вы уже могли догадаться, одним из самых распространенных событий является событие, носящее название `Click` (Щелчок) и обозначающее, что пользователь навел курсор на объект и щелкнул кнопкой мыши.

Итак, чтобы создать процедуру обработки событий для какого-то определенного объекта формы, вы должны выполнить следующие действия.

- 1. Откройте форму, содержащую объект, для которого нужно написать процедуру обработки событий.**

Для того чтобы сделать это, можно выполнить команду `View⇒Designer` (Вид⇒Конструктор), нажать комбинацию клавиш `<Shift+F7>` или щелкнуть на вкладке `Design`. Если форма не появится на экране, выберите команду `View⇒Solution Explorer` (Вид⇒Разработчик задач) и дважды щелкните на названии нужной формы. Visual Basic .NET отобразит ее на экране.

Режим, при котором на экране отображается форма, называется *режимом конструктора*.

- 2. Дважды щелкните на объекте, для которого хотите написать процедуру обработки событий.**

Если этим объектом будет кнопка, произведите на ней двойной щелчок. Visual Basic .NET отобразит пустой код (его выполнение не влечет за собой никаких действий) процедуры обработки событий, имеющий приблизительно такой вид:

```
Protected Sub Button1_Click(ByVal sender As Object, _
    ByVal e As System.EventArgs)
End Sub
```

Режим, при котором отображается окно с кодами BASIC, называется *режимом кодов*. Можете пока не обращать внимания на всякий "мусор", который заключен в круглые скобки.

3. Между первой и последней строкой напишите код BASIC, который должен выполняться после наступления события. Это может выглядеть примерно так:

```
Protected Sub Button1_Click(ByVal sender As Object, _  
    ByVal e As System.EventArgs)  
    Me.Close ()  
End Sub
```

## Обычный способ создания процедуры обработки событий

Каждый раз, когда вы рисуете новый объект, Visual Basic .NET сохраняет его имя в списке объектов, который называется Class Name (Имена классов). Этот список показан на рис. 4.2.

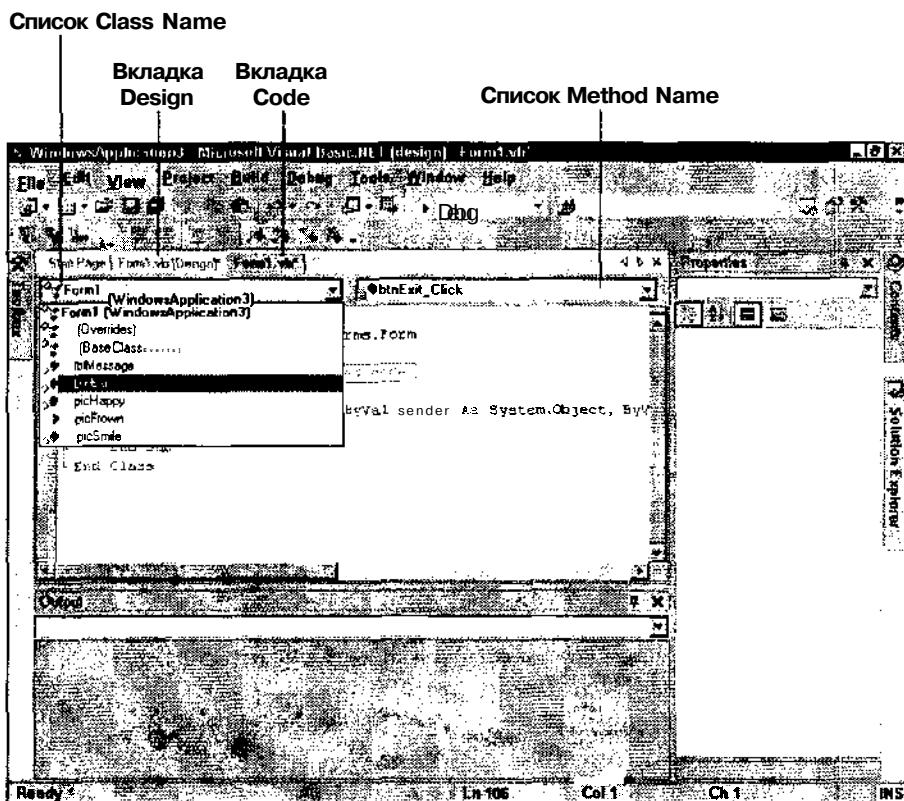


Рис. 4.2. Открыв список Class Name в окне кодов, выберите нужный объект

Для создания процедуры обработки событий вы должны сообщить Visual Basic:

- ✓ имя нужного объекта;
- ✓ событие, на которое этот объект должен отреагировать.

Делается это следующим образом.

### 1. Переключитесь в режим кодов.

Чтобы переключиться в режим кодов, выберите команду **View⇒Code** (Вид⇒Коды), нажмите клавишу <F7> или щелкните на вкладке **Code**. Visual Basic .NET отобразит коды BASIC, отвечающие за работу вашего пользовательского интерфейса.

### 2. Щелкните на списке Class Name и выберите объект, для которого хотите написать процедуру обработки событий.

Например, если вы хотите написать процедуру обработки событий для определенной кнопки, найдите в списке Class Name ее имя и щелкните на нем.

Если вы напишете процедуру обработки событий для объекта, а потом измените его имя, Visual Basic .NET не сможет правильно на это отреагировать, так как будет думать, что вы просто создали новый объект, в точности похожий на старый. Это означает, что переименованному объекту не будут соответствовать никакие процедуры обработки событий. Поэтому переименуйте все объекты до того, как начнете писать для них процедуры обработки событий.

### 3. Щелкните на списке Method Name (Имя метода) и выберите в нем событие, на которое должен отреагировать объект.

Visual Basic .NET создаст пустую процедуру обработки событий.

### 4. В новой процедуре обработки событий напишите код BASIC, который будет выполнен в случае наступления указанного события.

#### Тест на проверку полученных вами знаний

##### 1. Что такое "ключевые слова"?

- а. Слова, которые вы произнесете, когда вам принесут счет за ужин в ресторане, где стакан воды стоит 25 долларов
- б. Слова, которые произносятся на церемонии бракосочетания
- в. Специальные инструкции, которые используются при написании программ
- г. Произносимые за столом тосты

##### 2. Зачем нужны процедуры обработки событий?

- а. Они сообщают компьютеру, когда наступят такие важные события, как трансляция футбольного матча или ваш день рождения
- б. Они вырубят компьютер, если за него садится неопытный пользователь
- в. Они сообщают компьютеру, как нужно реагировать на события, совершаемые пользователем в отношении объектов интерфейса

## Что могут делать коды BASIC

Пустую процедуру обработки событий нужно заполнить командами BASIC, чтобы она смогла выполнить определенное действие. Обычно написанием кодов BASIC преследуются две основные цели:

- ✓ проведение вычислений и получение их результатов;
- ✓ изменение свойств (отображения) других объектов пользовательского интерфейса.

Если вы хотите посчитать, сколько котят могут привести все котята, приведенные вашей кошкой, если каждая кошка будет приводить по пять котят в год, то Visual Basic .NET сможет это сделать, если вы сообщите ему все необходимые исходные данные. А вообще, команда BASIC, производящая вычисления, может выглядеть приблизительно так:

```
PayThis = 5000 * 0.5
```

Когда результат будет вычислен, вы, наверное, захотите отобразить его на экране. Чтобы сделать это, нужно изменить свойство какого-нибудь объекта интерфейса (например, тексто-

вого поля или надписи). Следовательно, чтобы отобразить сообщение на экране, сначала для него нужно создать объект в окне формы. Пусть этим объектом будет текстовое поле.

Затем вновь созданному текстовому полю нужно присвоить какое-нибудь имя, скажем, `txtMessage`. И наконец, чтобы отобразить что-нибудь в текстовом поле, нужно изменить свойство `Text` объекта `txtMessage`. Например, так:

```
txtMessage.Text = "Display Me!"
```

Данная команда отобразит на экране, в текстовом поле `txtMessage`, сообщение `Display Me!` Аналогичный результат можно получить, изменив в режиме конструктора свойство `Text` текстового поля `txtMessage`, как это показано на рис. 4.3.



Используя коды BASIC, невозможно изменить все свойства объекта. Некоторые из них (например, имя объекта) можно изменить только в режиме конструктора в окне `Properties`.

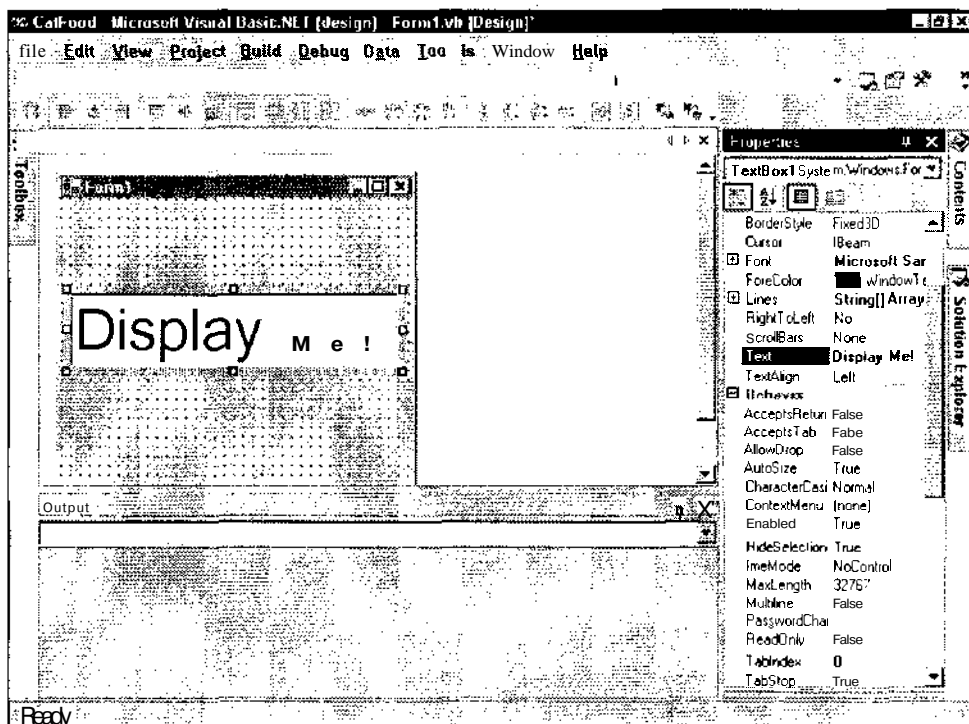


Рис. 4.3. Результате изменения свойства объекта изменяется и способ его отображения на экране

## Как работают процедуры обработки событий

Инструкции, записанные в процедуре обработки событий, запускаются в действие лишь после наступления определенного события, например, щелчка кнопкой мыши на объекте. Каждый раз, когда пользователь будет щелкать на объекте, тот же набор инструкций будет выполняться снова и снова. Исключение составляет только процедура, содержащая команду окончания работы программы.

Чтобы остановить выполнение программы Visual Basic .NET, нужно создать процедуру обработки событий, которая будет выглядеть приблизительно так:

```
Protected Sub Button1_Click(ByVal sender As Object, _
    ByVal e As System.EventArgs)
    Me.Close()
End Sub
```

Как только пользователь щелкнет на кнопке `Button1`, произойдет следующее.



1. Visual Basic .NET спросит: "Эй! Какое имя у объекта, на котором только что щелкнул пользователь?"
2. Ответ не заставит себя долго ждать, и Visual Basic .NET узнает, что пользователь щелкнул на объекте с именем `Button1`.
3. Visual Basic .NET сразу же спросит: "А есть ли какие-то инструкции на тот случай, если пользователь щелкнет на объекте `Button1`?" Такая инструкция будет обнаружена в процедуре обработки событий `Protected Sub Button1_Click()`.
4. Visual Basic .NET начнет читать первую инструкцию процедуры обработки событий `Button1_Click()`. В данном случае эта инструкция будет единственной и будет иметь вид `Me.Close()`. Visual Basic .NET интерпретирует ее как указание закрыть форму, обозначенную ключевым словом `Me` (она же открытая и активная в данный момент форма). Когда Visual Basic .NET закроет эту форму, программа будет остановлена.
5. Visual Basic .NET остановит выполнение программы и удалит форму с экрана и из оперативной памяти. Причем все это произойдет настолько быстро, что вы даже не успеете ничего заметить.

## Написание кодов BASIC для своей первой ftftoifiaauibt

Как говорится, пока сам не попробуешь — ничему не научишься. Поэтому рекомендуем вам последовать нашим советам и, выполнив все перечисленные ниже шаги, создать работающую программу, которая даже будет способна произвести впечатление на неискушенных пользователей. В приведенной ниже последовательности действий показано, как можно написать коды BASIC для программы, которую вы начали создавать в третьей главе.

Не переживайте, если вам будет что-то не понятно из того, что нужно будет набрать. Цель данного упражнения состоит лишь в том, чтобы показать, как просто может быть создана работающая программа Visual Basic .NET.

1. **Запустите приложение Visual Basic .NET (если вы, конечно, еще этого не сделали. А если сделали, выберите команду `File⇒Open⇒Project`).**  
Visual Basic .NET отобразит диалоговое окно **Open Project** (Открыть проект).
2. **Щелкните на проекте `Hollo`, а затем — на кнопке `Open`.**  
(Можете пропустить эти шаги, если на вашем экране еще отображается форма, созданная в третьей главе.)
3. **Дважды щелкните на улыбающемся лице (это первый рисунок формы).**  
Visual Basic .NET отобразит пустую процедуру обработки событий для рисунка `picSmile`.
4. **Наберите коды BASIC для процедуры `picSmile_Click`.**



```

picFrown.BorderStyle =_
    System.Windows.Forms.BorderStyle.FixedSingle
picHappy.BorderStyle =_
    System.Windows.Forms.BorderStyle.None
lblMessage.Text = "Good-bye, cruel world!"

```

End Sub

8. Щелкните на вкладке **Design**, выберите команду **View⇒Designer** или нажмите <Shift+F7>, чтобы опять отобразить на экране форму.

9. Дважды щелкните на рисунке picHappy (он крайний справа).

Visual Basic .NET отобразит пустую процедуру обработки событий picHappy.

10. Наберите следующие коды для процедуры picHappy:

```

Private Sub picHappy_Click(ByVal sender As_
    System.Object, ByVal e As System.EventArgs)_
    Handles picHappy.Click
picSmyle.BorderStyle =_
    System.Windows.Forms.BorderStyle.None
picFrown.BorderStyle =_
    System.Windows.Forms.BorderStyle.None
picHappy.BorderStyle =_
    System.Windows.Forms.BorderStyle.FixedSingle
lblMessage.Text = "I'm going to DisneyWorld!"

```

End Sub

11. Щелкните на вкладке **Design**, выберите команду **View⇒Designer** или нажмите <Shift+F7>, чтобы снова увидеть свою форму.

12. Дважды щелкните на кнопке btnExit (это кнопка, на которой отображается надпись Exit).

Visual Basic .NET отобразит пустую процедуру обработки событий btnExit.

13. Наберите приведенные далее коды для процедуры btnExit:

```

Protected Sub btnExit_Click(ByVal sender As_
    System.Object, ByVal e As System.EventArgs)_
    Handles btnExit.Click
Me.Close()

```

End Sub

14. Нажмите клавишу <F5> или выберите команду **Debug⇒Start**, чтобы запустить написанную вами программу.

Если вы набрали все коды без ошибок, Visual Basic .NET отобразит на экране пользовательский интерфейс вашей программы (рис. 4.5).



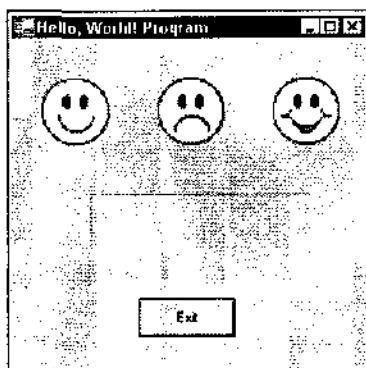


Рис. 4.5. Программа Hello World! начинает свою работу

**15. Щелкните на улыбающемся лице (первое по счету).**

Visual Basic .NET отобразит сообщение Hello World!, как это показано на рис. 4.6.

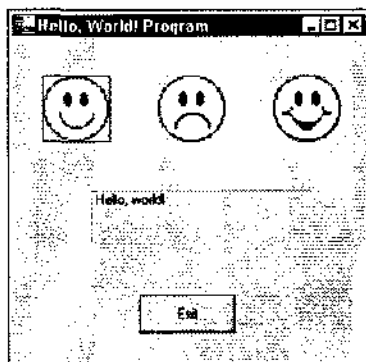


Рис. 4.6. Вот что произойдет, если вы щелкнете на улыбающемся лице

**16. Щелкните на грустном лице, которое расположено посередине.**

Visual Basic .NET отобразит сообщение Good-bye, cruel world! (рис. 4.7).

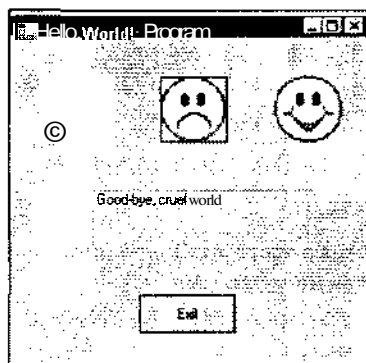


Рис. 4.7. Сообщение, которое появится после щелчка на грустном лице

### 17. Щелкните на счастливом лице (крайнее справа).

Для этого рисунка Visual Basic .NET отобразит сообщение I'm going to Disney World! (рис. 4.8).

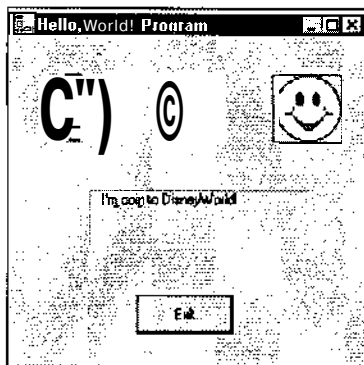


Рис. 4.8. Вот что случится, если вы щелкнете на счастливом лице

### 18. Щелкните на кнопке с надписью Exit.

Работа вашей программы будет завершена, и вы вернетесь к пользовательскому интерфейсу Visual Basic .NET.

Итак, проект Hello, World! успешно завершен. Теперь вы на собственном опыте убедились, как легко и быстро с помощью Visual Basic .NET можно создать качественную программу с отличным интерфейсом.



## Часть I

# Создание пользовательского интерфейса

Эта история о том, как Чарли Ван Гог, программист Visual Basic и внучатый племянник Винсента Ван Гога, получил задание разработать пользовательский интерфейс для создаваемой базы данных.

Извини, Чак,  
но начальство опять  
забракowało твою работу



### *В этой части...*

Пользовательский интерфейс дает возможность другим людям пользоваться вашей программой. Чем сложнее и запутанней интерфейс, тем труднее понять саму программу. Поэтому, чем логичнее и проще будет созданный вами интерфейс, тем большее число людей захочет его использовать.

Эта часть — одна из лучших в книге. Здесь вам не придется набирать длинные и непонятные коды, учить загадочные команды или запоминать какие-то комбинации клавиш. Вы будете только рисовать всякие фигурки на экране, а компьютер в это же время будет создавать вашу программу.

# Создание пользовательского интерфейса: разберемся в деталях

*В этой главе...*

- Подробности разработки пользовательского интерфейса
- Рисование объектов интерфейса
- Перемещение, копирование и удаление объектов
- Создание недоступных объектов

**Н**ужно понимать одну очень простую вещь. В действительности ни у кого нет желания использовать вашу программу. Большинство людей, не задумываясь, отдали бы предпочтение рыбалке, походам по магазинам или чему-то еще. Но зато людям может понадобиться результат, который можно получить с помощью вашей программы. Если бы они могли получить его каким-нибудь другим, более простым способом, то они бы так и сделали. Но так как других способов у них, скорее всего, нет, придется применять вашу программу.

Это означает, что на самом деле люди хотят, чтобы программа "читала их мысли", а затем волшебным образом сама выполняла всю необходимую работу. Поскольку эти желания не реальны, то лучшее, что вы можете для них сделать, так это создать программу, максимально простую в использовании. Если она будет понятна даже маленьким детям, то большинство других людей, наверное, тоже смогут в ней разобраться.

## *До того как приступить к созданию интерфейса*

Создать пользовательский интерфейс — вовсе не означает напичкать диалоговые окна яркими рисунками и разными эффектными штучками в надежде, что люди со временем сами во всем разберутся. Главная цель интерфейса — сделать программу понятной и доступной пользователям. А потому при создании пользовательского интерфейса не следует упускать из виду следующие моменты.

### Вспомните, для кого создается программа

Перед тем как приступить к разработке интерфейса, подумайте, кто может стать пользователем вашей программы. Возможно, это будут менеджеры компаний, которые хорошо знакомы с компьютерными технологиями, а может быть, "бумажные клерки", которые раньше видели компьютеры только по телевизору.

Когда вы определите круг лиц, которые должны стать пользователями вашей программы, создайте интерфейс, максимально точно отображающий их прежний опыт работы и способ восприятия, даже если такой интерфейс будет не приемлем для других категорий пользователей. Например, бухгалтеры легко привыкнут к электронным таблицам, поскольку они будут им напоминать их родные бухгалтерские книги. Машинистки же быстро освоятся с текстовыми редакторами, чей интерфейс так похож на чистый лист бумаги для набора текста.

Но представьте, что все текстовые редакторы будут заполнены строками и столбцами, как электронные таблицы. Такой текстовый редактор быстро вызовет депрессию у любой машинистки (зато бухгалтер будет себя чувствовать с ними как рыба в воде).

Таким образом, знание программистом потребностей и привычек будущих пользователей является залогом того, что создаваемый интерфейс будет понят и воспринят адекватно. Ведь в конечном счете кто как не пользователь должен оценить результаты вашего труда.

## Не дайте пользователю заблудиться

Не для кого не секрет, что многие люди чувствуют себя угнетенными в огромных мегаполисах, где так легко заблудиться и потом долго бродить в поисках знакомых ориентиров. Вспомните, свои ощущения в тот момент, когда вы понятия не имели, где находитесь и в какую сторону следует двигаться.

Это чувство беспомощности и есть причиной того, что потерявшиеся маленькие дети так безутешно плачут, а сбитые с толку пользователи готовы беспорядочно щелкать на всех кнопках одновременно.

Хороший интерфейс должен ориентировать людей, чтобы они всегда знали, в каком месте программы когда находятся и как оттуда в случае чего можно выйти. В некоторых пользовательских интерфейсах, где-нибудь в нижней части экрана, постоянно отображается сообщение наподобие "Страница 2 из 5". Благодаря этому пользователь всегда знает, сколько всего страниц содержит документ и на какой странице он сейчас находится.

Итак, не забывайте, что пользовательский интерфейс — это "карта" программы, поэтому снабдите его всей необходимой информацией, чтобы пользователи могли в ней свободно ориентироваться и чувствовать себя уверенно.

## Сделайте навигацию очевидной

Помимо предоставления пользователям четких ориентиров, хороший интерфейс должен также ясно показывать, как из текущей позиции можно двигаться вперед, назад или куда-то еще. Если интерфейс отображает в нижней части экрана надпись "'Страница 4 из 25'", подумайте, как помочь пользователю перейти на следующую или на предыдущую страницу. Можно, например, в нижней части экрана по углам нарисовать стрелку "вперед" и стрелку "назад". Или, скажем, добавить кнопки "Предыдущая страница" и "Следующая страница".

Если пользователь будет понимать, куда он попадет на следующем шаге, какую клавишу нужно нажать или где щелкнуть кнопкой мыши, чтобы перейти в нужную позицию, программа будет казаться надежной и заслуживающей доверия.

## Будьте снисходительны

Цель интерфейса — это поддержание нормального диалога с пользователем. Если ваша программа будет вести себя вызывающе и после каждого сколько-нибудь неверного движения пользователя высвечивать всякие резкие и обидные сообщения наподобие "Файл MPR.DLL удален", никто просто не захочет ею пользоваться. С помощью интерфейса нужно корректно объяснять пользователю, к чему могут привести те или иные его действия и что лучше сделать в конкретной ситуации.



Отнеситесь к пользователям с уважением. Пусть ваша программа скроет или затемнит кнопки или команды меню, которые не доступны на данный момент. Если пользователь нажимает не ту клавишу или щелкает не в том месте, пусть программа отобразит сообщение с объяснением, к чему может привести это действие, и с вопросом, действительно ли пользователь этого хочет. Всем нравятся понятные и предсказуемые программы, поэтому и вы должны делать их такими.

## Не забывайте о простоте и удобстве

Большинство программ предоставляют пользователям сразу несколько способов для выбора определенной команды. Вы можете щелкнуть на кнопке, выбрать команду из раскрывающегося меню или нажать комбинацию клавиш (например, <Ctrl+F2>). Из трех указанных способов щелчок на кнопке считается самым простым, а нажатие комбинации клавиш — самым сложным.

Сделайте так, чтобы пользователи имели простой и быстрый доступ к часто используемым командам через кнопки или команды меню. И помните, что нет необходимости присваивать комбинации клавиш всем возможным командам.

Хотя нажатие комбинации клавиш — наиболее быстрый способ получить нужный результат, он же и самый сложный для запоминания. Если вы назначаете команде комбинацию клавиш, желательно, чтобы они ассоциировались друг с другом. Например, для команды Open (Открыть) лучше назначить комбинацию <Ctrl+O>, чем что-то вроде <Shift+F11>. Ведь довольно просто запомнить, что команде Open соответствует клавиша <O>, а с чем ассоциируется клавиша <F11>, придумать сложно.

## Переходим к созданию интерфейса

В Visual Basic .NET интерфейс состоит из форм и объектов. Формы определяют размер, размещение и фон окон, которые представляют вашу программу. Объектами являются различные элементы, такие как кнопки или переключатели, способные поддерживать диалог с пользователем. На рис. 5.1 и 5.2 показаны примеры стандартных пользовательских интерфейсов, состоящих из:

- ✓ форм (они же окна);
- ✓ кнопок и переключателей;
- ✓ текстовых полей;
- ✓ надписей;
- ✓ рисунков.

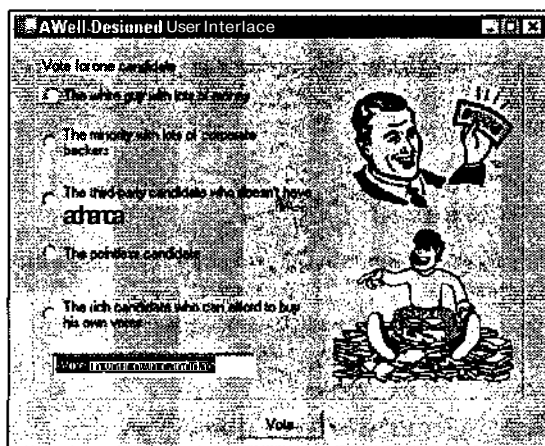


Рис. 5.1. Правильно организованный интерфейс позволяет пользователю легко сделать свой выбор



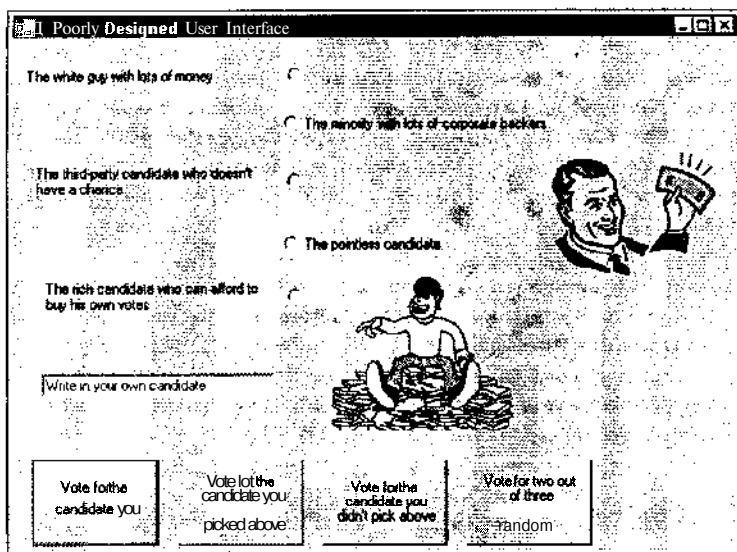


Рис.5.2. Плохо организованный интерфейс делает выбор не таким очевидным

## Создание форм

При разработке пользовательского интерфейса первым делом нужно создать форму. В окне формы будет отображаться информация для пользователя, и там же пользователь сможет оставлять информацию для программы. Когда вы создаете новое приложение для Windows (Windows application), Visual Basic .NET автоматически создает для вас одну пустую форму. Но поскольку программы обычно состоят из нескольких окон, вам, скорее всего, придется создавать дополнительные формы (окна).

Чтобы создать форму, выберите команду **Project ⇒ Add Windows Form** (Проект ⇒ Добавить форму Windows) или же щелкните на кнопке **Add New Item** (Добавить новый элемент), выберите пункт **Windows Form** и щелкните на кнопке **Open**.



На самом деле Visual Basic .NET может автоматически создавать и другие виды форм, но при разработке программ для Windows (например, игр или вычисляющих программ) наиболее часто используется именно **Windows Form**.

## Рисование объектов в окне формы

После того как форма будет создана, вы можете приступить к рисованию элементов. Это делается следующим образом.

1. Выберите команду **View ⇒ Toolbox** (Вид ⇒ Панель Toolbox) или нажмите комбинацию клавиш **<Ctrl+Alt+X>**. (Пропустите этот шаг, если панель элементов Toolbox уже есть на экране.)

Откроется панель Toolbox, которая показана на рис. 5.3. Возможно, вам нужно будет щелкнуть на кнопке **Windows Forms**, чтобы отобразить все объекты пользовательского интерфейса, доступные при создании форм Windows.

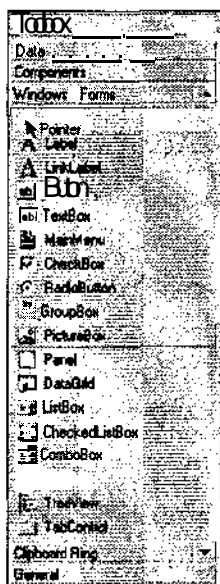


Рис. 5.3. На панели Toolbox вы найдете все объекты, которые можно нарисовать в окне формы

2. Щелкните на объекте, который хотите нарисовать, например на TextBox (Текстовое поле) или RadioButton (Переключатель). В табл. 5.1 приведен список объектов пользовательского интерфейса, отображаемых на панели Toolbox.

Курсор мыши примет форму перекрестия.

3. Поместите курсор над тем местом в окне формы, где должен быть нарисован объект, нажмите левую кнопку мыши и перетяните курсор в сторону. Когда контур объекта примет нужный размер, отпустите кнопку мыши.

Visual Basic .NET отобразит в указанном вами месте выбранный объект.

**Таблица 5.1. Инструменты панели Toolbox**

Название инструмента	Выполняемое им действие
Pointer (Указатель)	Выделяет объекты
Label (Надпись)	Создает в окне формы надпись
LinkLabel (Ссылка)	Создает в окне формы гиперссылку
Button (Кнопка)	Создает прямоугольную кнопку
TextBox (Текстовое поле)	Создает поле, в котором пользователь может набирать текст
MainMenu (Главное меню)	Создает открывающееся меню
CheckBox (Флажок проверки)	Создает флажок проверки
RadioButton (Переключатель)	Создает переключатель

Название инструмента	Выполняемое им действие
GroupBox (Группа)	Рисует в окне формы рамку для выделения группы объектов, например, переключателей
PictureBox (Окно с рисунком)	Выделяет область для отображения графики
Panel (Панель)	Рисует прямоугольный блок, на котором могут размещаться другие объекты
DataGrid(Таблица данных)	Создает таблицу для отображения информации из файла базы данных
ListBox (Список)	Создает список
CheckedListBox (Список флажков)	Создает список, элементами которого являются флажки
ComboBox(Поле со списком)	Создает поле со списком
ListView (Текстовый список)	Создает текстовый список
TreeView(Дерево)	Создает объект, при щелчке на котором отображается текст в виде дерева
TabControl (Вкладки)	Создает одну или несколько вкладок, на каждой из которых могут быть размещены другие объекты



Чтобы быстро создать какой-либо объект, дважды щелкните на нем в панели **Tool-box**. Например, если вы хотите создать кнопку, дважды щелкните в панели **Tool-box** на инструменте **Button**, и Visual Basic .NET создаст его в окне формы автоматически.

## Изменение свойств объектов

После того как объект будет создан, нужно изменить свойства, задаваемые по умолчанию, определив для него цвет, размер, отображаемую надпись и т.д. Каждый объект имеет целый набор свойств, но совсем не обязательно изменять каждое из них.



Есть два свойства, которые необходимо изменять практически для каждого объекта. Это свойство **Name** (Имя) и свойство **Text** (Текст). Свойство **Name** идентифицирует каждый объект вашего интерфейса. Свойство **Text** определяет текст, который отображается на самом объекте (например, на объекте **Button** может отображаться слово **OK** или **Отмена**).

Для изменения свойств объектов необходимо произвести такие действия.

- Откройте окно **Solution Explorer**, воспользовавшись одним из трех методов.
  - Нажмите комбинацию клавиш **<Ctrl+Alt+L>**.
  - Выберите команду **View⇒Solution Explorer**.
  - На панели инструментов щелкните на кнопке **Solution Explorer**.
- Щелкните на названии формы, содержащей объект, свойства которого нужно изменить.

Visual Basic .NET отобразит эту форму и все ее объекты.

- Щелкните на объекте, свойства которого нужно изменить.

Окно Properties отобразит все свойства, доступные для этого объекта. Если окна Properties нет на экране, нажмите клавишу <F4>.

- Щелкните на свойстве, которое нужно изменить.

Свойства изменяются по-разному, но в любом случае это будет один из следующих методов.

- Наберите текст (если изменяется, например, свойство Text (Текст)) или введите число (если изменяется, скажем, свойство Size (Размер) или Location (Размещение)).
- Щелкните на стрелке и из открывшегося списка выберите одно из доступных для свойства значений (например, значение True или False для свойства Visible (Видимый)).
- Щелкните на кнопке с тремя кнопками при необходимости отобразить диалоговое окно для установки сразу нескольких опций (как, например, в случае свойства Font (Шрифт)).

## Переименование объектов

Каждый объект имеет свое имя (свойство Name), по которому Visual Basic .NET может его идентифицировать. (Всем нам в детстве родители дают какие-то имена, чтобы другие люди для привлечения нашего внимания не говорили нам "Эй, ты!".) Свойство Name можно найти в окне Properties (оно относится к категории Design).



Объекты Visual Basic .NET, расположенные в окне одной формы, должны иметь разные имена. Вы можете попытаться дать двум разным объектам одно и то же имя, но это останется не более чем попыткой, так как Visual Basic .NET с такой установкой никогда не согласится. (Можно, правда, дать одинаковые имена объектам, расположенным в разных формах, но тогда вы легко можете запутаться в кодах своей программы.

Каждый раз, когда вы создаете новый объект, Visual Basic .NET автоматически присваивает ему стандартное имя. Например, если вы создаете первую кнопку, Visual Basic .NET называет ее Button1, а если вы создаете вторую кнопку, Visual Basic .NET называет ее Button2 и т.д.



Имя объекта никогда не отображается на экране. Длина имен может достигать 40 символов, но они не должны включать знаки пунктуации и пробелы. Вы можете назвать объект каким угодно именем, но большинство программистов, использующих Visual Basic .NET, начинают имена с трехсимвольных префиксов, которые приведены в табл. 5.2. Эти префиксы помогают потом вспомнить, к объекту какого типа относится данное имя.

Таблица 5.2. Рекомендуемые префиксы для имен объектов

Объект	Префикс	Пример имени
Button	btn	btnYourZip
CheckBox	chk	chkZipper

Объект	Префикс	Пример имени
ComboBox	cmb	cmbBoBo
DataGrid	dat	datFanOver
Form	frm	frm1001N
Label	lbl	lblName
ListBox	lst	lstPresidents
MainMenu	mnu	mnuRex
PictureBox	pic	picPrettyDog
RadioButton	rad	radStat101
TextBox	txt	txtReadHere

Чтобы изменить имя формы, сделайте следующее.

1. **Откройте окно Solution Explorer.**

Для этого нажмите комбинацию клавиш <Ctrl+Alt+L>, выберите команду **View⇒Solution Explorer** или на панели инструментов щелкните на кнопке **Solution Explorer**.

2. **Щелкните на форме, которую хотите переименовать.**

3. **Откройте окно Properties.**

Чтобы сделать это, нажмите клавишу <F4>, выберите команду **View⇒Properties Window**, в окне **Solution Explorer** щелкните на кнопке **Properties Window** или щелкните правой кнопкой мыши в окне формы и выберите команду **Properties**.

4. **Дважды щелкните на свойстве Name (оно принадлежит категории Design) и наберите новое имя.**

А вот как можно изменить имя любого объекта.

1. **Щелкните на объекте, который вы хотите переименовать.**

Вокруг объекта появятся маркеры.

2. **Откройте окно Properties.**

Чтобы сделать это, нажмите клавишу <F4>, выберите команду **View⇒Properties Window**, в окне **Solution Explorer** щелкните на кнопке **Properties Window** или щелкните правой кнопкой мыши в окне формы и выберите команду **Properties**.

3. **Дважды щелкните на свойстве Name (категория Design) и наберите новое имя.**

## Отображение текста на объекте

Большинство объектов может отображаться на экране вместе с текстом, указанным посредством свойства **Text**. Это свойство можно выбрать в окне **Properties** — оно относится к категории Appearance (Отображение). Ниже приведен список объектов, которые наиболее часто используются для отображения текста.

✓ кнопки (**Button**);

- ✓ флажки проверки (**CheckBox**);
- ✓ список флажков (**CheckedListBox**);
- ✓ надпись (**Label**);
- ✓ рамка (**GroupBox**);
- ✓ переключатель (**RadioButton**);
- ✓ текстовое поле (**TextBox**).

Если речь идет о форме, то текст, указанный в свойстве **Text**, отображается в заголовке окна. В случае других объектов текст отображается прямо на них самих. Например, текст кнопки будет отображен на самой кнопке.



Установленные по умолчанию значения свойств **Name** и **Text** будут оставаться такими же до тех пор, пока вы их не измените. Когда вы рисуете в окне формы, скажем, флажок, рядом с ним сразу появляется надпись, наподобие **Check1**.

Чтобы изменить свойство **Text** какого-нибудь объекта, вам нужно выполнить следующие действия.

### 1. Щелкните на объекте, текст которого нужно изменить.

Вокруг объекта появятся маркеры.

### 2. Откройте окно **Properties**.

Чтобы сделать это, нажмите клавишу <F4>, выберите команду **View** ⇒ **Properties Window**, в окне **Solution Explorer** щелкните на кнопке **Properties Window** или щелкните правой кнопкой мыши в окне формы и выберите команду **Properties**.

### 3. Дважды щелкните на свойстве **Text** (категория **Appearance**) и наберите любой текст.

Visual Basic .NET отобразит этот текст на выделенном объекте.



Текст можно сделать стильным, привлекающим внимание (или раздражающим), изменив свойство **Font** (Шрифт), также принадлежащее к категории **Appearance**. Только не переусердствуйте и не превратите текст в трудно читаемый.

## Тест на проверку полученных вами знаний

### 1. Почему так важно знать, кто будет **пользователем** вашей программы?

- а. Потому что в противном случае можно создать программу, которая не сможет выполнять все пожелания пользователей, и они обидятся.
- б. Это позволит создать интерфейс с учетом потребностей и привычек будущих **пользователей**.
- в. Потому, что если среди будущих пользователей окажутся мои враги, программу я писать не буду.
- г. Не важно, кто будет пользователем программы. Главное, сколько мне за нее заплатят.

### 2. Интерфейс должен быть картой для **пользователя**. Как вы это понимаете?

- а. Это ваше утверждение. Почему я должен его понимать?
- б. Все любят путешествовать, почему бы не подарить людям праздник.
- в. Программа не должна напоминать **"бродилку"**, из которой нет **выхода**.
- г. Интерфейс всегда должен показывать **пользователю**, в каком месте программы он находится и куда приведут его дальнейшие **действия**.

## Настройка размеров объектов

После того как объект будет создан, у вас может возникнуть желание изменить его размеры. В Visual Basic .NET это можно сделать двумя способами:

- ✓ используя мышь;
- ✓ изменив свойства **Width** (Ширина) и **Height** (Высота).

Чтобы изменить размер объекта с использованием мыши, выполните следующие действия.

1. Щелкните на объекте, размер которого нужно изменить.

Вокруг границ этого объекта появятся маркеры.

2. Поместите курсор мыши над границей объекта так, чтобы он принял вид двунаправленной стрелки.
3. Нажмите левую кнопку мыши и перетащите курсор. Когда объект примет нужный размер, отпустите кнопку мыши.

А вот как изменятся размер объекта при использовании окна **Properties**.

1. Щелкните на объекте, размер которого нужно изменить.
2. Откройте окно **Properties**.

Чтобы сделать это, нажмите клавишу <F4>, выберите команду **View⇒Properties Window**, в окне **Solution Explorer** щелкните на кнопке **Properties Window** или щелкните правой кнопкой мыши в окне формы и выберите команду **Properties**.

3. Щелкните на знаке “плюс”, который отображается слева от свойства **Size** (Размер), принадлежащего к категории **Layout** (Размещение).

Visual Basic .NET отобразит значения свойств **Width** (Ширина) и **Height** (Высота).

4. Укажите новые значения для свойств **Width** и **Height**.

Мышь лучше использовать в том случае, когда размеры не должны быть заданы с точностью до миллиметра. Вручную задавать значения свойств **Width** и **Height** нужно тогда, когда необходима абсолютная точность, или если вам нравится, когда расположенные рядом объекты имеют одинаковые размеры.



Вместо того чтобы изменять значения свойств **Width** и **Height**, можно изменить значение свойства **Size**. Значение свойства **Size** отображается в формате **Width, Height**. Это значит, что если нужно свойству **Width** присвоить значение 97, а свойству **Height** — значение 23, то для свойства **Size** следует указать значение 97, 23.

## Перемещение объектов на экране

В окне формы объекты могут быть расположены где угодно. Visual Basic .NET предоставляет в ваше распоряжение два способа определения позиции объекта в окне формы:

- ✓ с использованием мыши;
- ✓ путем изменения свойств **X** и **Y**, принадлежащих к категории **Layout** (Размещение).

Чтобы изменить положение объекта с помощью мыши, сделайте следующее.

1. Щелкните на объекте, который нужно переместить, таким образом, чтобы вокруг его границ отобразились черные маркеры.

2. **Расположите курсор мыши над объектом (но не над маркерами объекта). Затем нажмите левую кнопку мыши и перетащите объект на новое место.**
3. **Отпустите кнопку мыши.**



Используйте этот способ тогда, когда нет необходимости точно выравнять объект относительно окна формы и других объектов. Если же положение объекта должно быть задано с абсолютной точностью, укажите в окне **Properties** значения свойств **X** и **Y**.

Значение свойства **X** определяет расстояние от левого края формы до левого края объекта. Значение свойства **Y** определяет расстояние от верхнего края формы до верхнего края объекта.

Изменить размещение объекта с использованием окна **Properties** можно следующим образом.

1. **Щелкните на объекте, который вы хотите переместить.**

Visual Basic .NET выделит выбранный вами объект и отобразит вокруг его границ маркеры.

2. **Откройте окно Properties.**

Чтобы сделать это, нажмите клавишу <F4>, выберите команду **View⇒Properties Window**, в окне **Solution Explorer** щелкните на кнопке **Properties Window** или щелкните правой кнопкой мыши в окне формы и выберите команду **Properties**.

3. **В категории Layout найдите свойство Location (Положение) и щелкните на знаке “плюс”, который отображается слева от названия свойства.**

Visual Basic .NET отобразит текущие значения свойств **X** и **Y**.

4. **Укажите новые значения для свойств X и Y.**



Вместо того чтобы изменять значения свойств **X** и **Y**, можно изменить значение свойства **Location**, которое отображается в формате **X ,Y**. Другими словами, если **X** имеет значение 45, а **Y** — значение 2, то свойство **Location** будет иметь значение 45, 2.

## Прикрепление объектов к сторонам формы

Любой объект можно отобразить растянутым по всей длине или ширине формы и прикрепленным к одной из ее сторон. Даже если пользователь увеличит или уменьшит размеры формы, объект останется прикрепленным к ее стороне.

Чтобы прикрепить объект, выполните следующие действия.

1. **Щелкните на объекте, который вы хотите растянуть вдоль одной из сторон формы.**

Вокруг границ объекта появятся маркеры.

2. **Откройте окно Properties.**

Чтобы сделать это, нажмите клавишу <F4>, выберите команду **View⇒Properties Window**, в окне **Solution Explorer** щелкните на кнопке **Properties Window** или щелкните правой кнопкой мыши в окне формы и выберите команду **Properties**.

3. **В категории Layout щелкните на свойстве Dock (Прикрепить).**

Рядом появится кнопка со стрелкой.

4. **Щелкните на кнопке со стрелкой, направленной вниз.**

Откроется графическое меню, в котором будут отображаться серые прямоугольники, характеризующие способы прикрепления объекта к стороне формы (рис. 5.4).



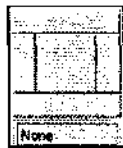


Рис. 5.4. Меню прикрепления, где можно выбрать сторону, к которой будет прикреплен данный объект

5. Щелкните на той серой кнопке, которая отвечает нужному расположению объекта.

Visual Basic .NET сразу же растянет объект и прикрепит его к указанной вами стороне.



Если объект ранее уже был прикреплен к какой-либо стороне, щелкните на кнопке **None**, и данная опция будет снята.

## Закрепление объектов

Когда вы прикрепляете объект, он автоматически изменяет свои размеры и становится растянутым вдоль всей **формы**. Если вы не хотите, чтобы объект менял свою форму, можно просто закрепить его положение относительно одной или нескольких сторон.

Закрепление объекта относительно какой-то стороны означает, что как бы пользователь не изменял размеры окна формы, расстояние между объектом и указанной стороной будет оставаться неизменным. Чтобы закрепить объект, сделайте следующее.

1. Щелкните на объекте, который вы хотите закрепить.

Вокруг границ этого объекта отобразятся маркеры.

2. Откройте окно Properties.

Чтобы сделать это, нажмите клавишу <F4>, выберите команду **View⇒Properties Window**, в окне **Solution Explorer** щелкните на кнопке **Properties Window** или щелкните правой кнопкой мыши в окне формы и выберите команду **Properties**.

3. В категории **Layout** щелкните на свойстве **Anchor** (Закрепить).

Появится кнопка с направленной вниз стрелкой.

4. Щелкните на кнопке со стрелкой.

Появится графическое меню, схематически отображающее объект и направленные к четырем сторонам прямоугольники, символизирующие способы закрепления (рис. 5.5).



Рис. 5.5. В меню закрепления вы можете указать стороны, относительно которых будет закреплен данный объект

**5. Щелкните на одном или нескольких четырехугольниках, направленных к нужным сторонам.**

Теперь каждый раз, когда будет открываться форма, расстояние между объектом и стороной, относительно которой он закреплен, будет оставаться неизменным. Таким образом, если, скажем, объект закреплен относительно верхней и левой сторон, то как бы пользователь не изменял размеры окна, расстояние между объектом и этими сторонами будет оставаться прежним.

## Копирование созданного объекта

После того как объект будет создан и свойства его будут настроены нужным образом, вам наверняка захочется просто скопировать его, чтобы создать еще один или несколько таких же объектов. Это избавит вас от необходимости снова и снова проходить через все этапы построения объектов.

Чтобы скопировать объект, сделайте следующее.

**1. Щелкните на объекте, который нужно скопировать.**

Вокруг его границ отобразятся маркеры.

**2. Нажмите комбинацию клавиш <Ctrl+C>, выберите команду Edit⇒Copy (Правка⇒Копировать) или щелкните на кнопке Copy.**

**3. Нажмите комбинацию клавиш <Ctrl+V>, выберите команду Edit⇒Paste (Правка⇒Вставить) или щелкните на кнопке Paste.**

Visual Basic .NET создаст еще один такой же объект— прямо над тем, который вы скопировали.

**4. Переместите скопированный объект на новое место.**

О том, как это делается, было рассказано в разделе "Перемещение объектов на экране".



После того как с помощью операции копирования вы создадите новый объект, ему нужно будет присвоить какое-нибудь запоминающееся имя (Visual Basic .NET автоматически дает новым объектам стандартные имена, наподобие Button3. Если вас такое имя устраивает, можете его оставить.)

## Удаление объектов

Может случиться так, что вы нарисуете какой-то объект, но потом вдруг решите, что он вам вовсе не нужен. Чтобы избавиться от такого объекта, сделайте следующее.

**1. Щелкните на объекте, который нужно удалить.**

**2. Нажмите клавишу <Delete> или выберите команду Edit⇒Delete (Правка⇒Удалить).**



Если до того как будет произведена операция удаления вы нажмете комбинацию клавиш <Ctrl+Z>, удаленный объект можно будет впоследствии восстановить.

## Выделение сразу нескольких объектов для перемещения, копирования или удаления

До того как переместить, скопировать или удалить объект, его нужно выделить (т.е. щелкнуть на нем). Если необходимо переместить, скопировать или удалить сразу несколько объектов, выделить таковые вы сможете одним из двух методов:

- 1 ✓ с использованием мыши;
- ✓ посредством щелчков на нужных объектах при нажатой клавише <Ctrl> или <Shift>.

Чтобы выделить объекты, используя мышь, сделайте следующее.

1. **Поместите курсор мыши слева и сверху от группы объектов, которые вы хотите выделить. (Но только не над каким-то объектом.)**

Если вы поместите курсор мыши прямо над каким-либо объектом, вы сможете только лишь перетаскивать его, но не выделять группу объектов.

2. **Нажмите левую кнопку мыши и перетащите курсор вниз и вправо от группы объектов, которые нужно выделить.**

Visual Basic .NET отобразит пунктирную линию вокруг области выделения.

3. **Отпустите кнопку мыши.**

Visual Basic .NET отобразит серые прямоугольники вокруг всех выделенных вами объектов.

Чтобы выделить группу объектов вторым способом, выполните такие действия.

1. **Щелкните на первом объекте, который вы хотите выделить.**

Visual Basic .NET отобразит вокруг этого объекта черные маркеры.

2. **Поместите курсор мыши над вторым объектом.**

3. **Нажав и удерживая в таком состоянии клавишу <Ctrl> или <Shift>, щелкните на втором объекте.**

Visual Basic .NET отобразит серые прямоугольники над этим объектом и над тем, который уже был выделен.

4. **Повторяйте второй и третий шаги до тех пор, пока не выделите все объекты.**

## Определение для объектов свойства TabIndex

Наряду с использованием для выделения объектов мыши, Visual Basic .NET предоставляет возможность выбора таковых с помощью клавиши <Tab>. Для определения очередности, в которой объекты будут выделяться при нажатии клавиши <Tab>, используется свойство, носящее название *TabIndex*.

Объект, у которого свойство *TabIndex* имеет значение 0, выделяется сразу же при запуске программы. Если пользователь нажимает клавишу <Tab>, выделяется объект, у которого свойство *TabIndex* имеет значение 1 и т.д.



Значение свойства *TabIndex* первого объекта, который вы создадите в своей форме, будет равным 0, второго объекта — 1 и т.д.

Свойство TabIndex определяет порядок, в котором Visual Basic .NET выделяет объекты, если пользователь нажимает клавишу <Tab>, клавиши управления курсором (это те, на которых нарисованы стрелки) или комбинацию клавиш <Shift+Tab>.

- ✓ Путем нажатия клавиши <Tab>, клавиши со стрелкой вниз и клавиши со стрелкой вправо выделяется объект, который имеет значение свойства TabIndex, на единицу большее, чем значение свойства TabIndex у текущего объекта.
- ✓ Нажатием комбинации клавиш <Shift+Tab>, клавиши со стрелкой вверх и клавиши со стрелкой влево выделяется объект, который имеет значение свойства TabIndex, на единицу меньшее, чем значение свойства TabIndex у выделенного в данный момент объекта. Для выделения переключателей можно использовать только клавиши со стрелками, но не клавишу <Tab> и не комбинацию клавиш <Shift+Tab>.
- ✓ Нажатием на пробел или на клавишу <Enter> выбирается выделенный объект. (Это равносильно щелчку на объекте мышью.)



Некоторые объекты (например, окна с рисунками) вообще не имеют свойства TabIndex, поэтому, какие бы клавиши вы ни нажимали бы, выделить их вы не сможете.

Вот как изменяется значение свойства TabIndex.

1. Выберите команду View⇒Tab Order (Вид⇒Порядок Tab).

Visual Basic .NET отобразит значения свойства TabIndex прямо над самими объектами (рис. 5.6).

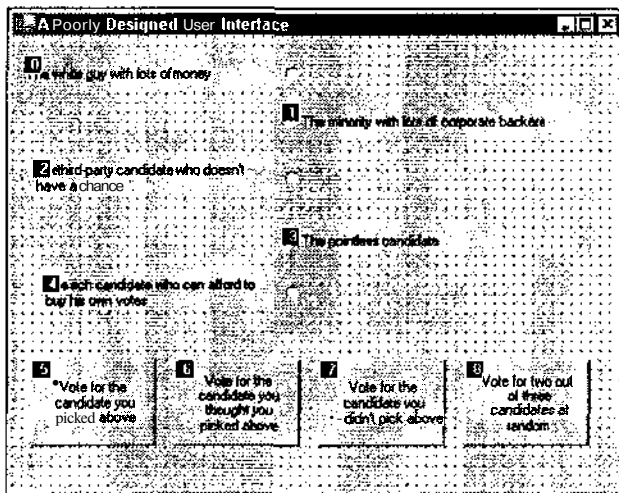


Рис. 5.6. Значения свойства TabIndex отображают порядок, в котором будут выделяться объекты при нажатии клавиши <Tab>

2. Щелкните на объекте, который должен, по-вашему мнению, выделяться первым (значение свойства TabIndex для него должно стать нулевым). Visual Basic .NET установит для него значение 0.
3. Щелкните на объекте, который должен **выделяться** следующим (ему будет присвоено значение, на единицу большее, чем у того объекта, на котором вы щелкнули в предыдущий раз).

4. Повторяйте шаг 3 до тех пор, пока всем объектам не будут присвоены нужные значения.
5. Выберите команду View⇒Tab Order.

Visual Basic .NET скроет значения свойства **TabIndex** для всех объектов.



Если по какой-то причине вы не хотите, чтобы пользователи могли выделять объект с помощью клавиши <Tab>, присвойте свойству **TabStop** (категория **Behavior**) значение **False** (Ложь). Это свойство доступно лишь тогда, когда снято отображение в окне формы значений свойства **TabIndex**.

## Выделение объектов серым цветом

Если вы не хотите, чтобы пользователь щелкал на каком-то объекте (например, на кнопке или флажке), можно выделить его серым цветом, как показано на рис. 5.7. Такой объект как бы говорит пользователю: "Придет время, когда на мне можно будет щелкать, но сейчас этого делать не надо".

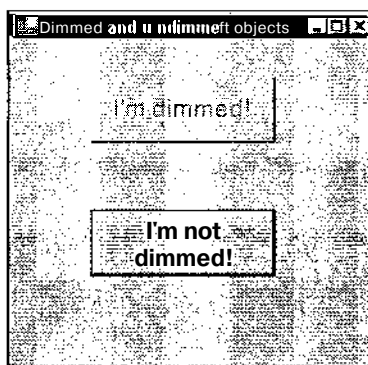


Рис. 5.7. Обычный объект, который выделен серым

Чтобы выделить объект серым цветом, сделайте следующее.

1. Щелкните на объекте, который вы хотите выделить серым.
2. Откройте окно Properties.

Чтобы сделать это, нажмите клавишу <F4>, выберите команду View⇒**Properties Window**, в окне **Solution Explorer** щелкните на кнопке **Properties Window** или щелкните правой кнопкой мыши в окне формы и выберите команду **Properties**.

3. Щелкните на свойстве **Enabled** (Разрешенный), которое расположено под категорией **Behavior**, и присвойте ему значение **False**.

Выделенный серым объект не способен ни на какие действия. Если же вы хотите, чтобы он был недоступным не все время, а только иногда, напишите для него соответствующие коды BASIC.



Чтобы вы получили представление о беспрецедентной простоте кодов BASIC, ниже приведен код, который снимает выделение серым с кнопки (имя которой, скажем, **btnExit**). Для этого нужно присвоить свойству кнопки **Enabled** значение **True** (Истина). Вот как это делается:

```
BtnExit.Enabled = True
```

Если вы хотите с помощью кода BASIC сделать кнопку недоступной, присвойте свойству **Enabled** значение **False**. В случае кнопки `btnExit` это будет выглядеть так;

```
BtnExit.Enabled = False
```

Итак, с помощью кодов BASIC можно выделять кнопку серым или снимать с нее выделение в процессе выполнения программы. Это может быть ответом на какие-либо действия, производимые пользователем (например, на ввод текста, перемещение курсора, беспомощное щелканье мышью и т.п.).

## Как сделать объект невидимым

Помимо выделения недоступного объекта серым (что может смущать пользователей, поскольку объект все равно остается перед глазами и на нем хочется щелкнуть), таковой можно вовсе убрать с экрана.

Чтобы отменить отображение объекта на экране, вы должны выполнить такие действия.

- 1. Щелкните на объекте, который отображать на экране не нужно.
- 2. Откройте окно **Properties**.

Чтобы сделать это, нажмите клавишу `<F4>`, выберите команду **View⇒Properties Window**, в окне **Solution Explorer** щелкните на кнопке **Properties Window** или щелкните правой кнопкой мыши в окне формы и выберите команду **Properties**.

- 3. Находясь в категории **Behavior**, щелкните на свойстве **Visible (Видимый)** и присвойте ему значение **False**.

Убрать объект с экрана можно также с помощью кодов BASIC. Вот пример кода, который убирает с экрана кнопку `btr.New`:

```
BtnNew.Visible = False
```

Как и выделенные серым, скрытые на экране объекты становятся недоступными пользователю. Чтобы сделать объект доступным, нужно его снова отобразить на экране, присвоив свойству **Visible** значение **True**. Вот пример кода, который позволит снова отобразить на экране кнопку `btnNew`:

```
BtnNew.Visible = True
```

## Изменение текста, отображаемого на объекте

Приведенный ниже пример демонстрирует, насколько легко и быстро можно создать код BASIC, изменяющий текст, который отображается на кнопке. Вначале в окне формы нарисуйте кнопку и измените ее свойства, как показано в табл. 5.3.

Таблица 5.3. Новые значения для свойств кнопки

Объект	Свойство	Значение
Button 1	Name	btnChangeMe
	Text	Change Text

Дважды щелкните на кнопке `btnChangeMe` и наберите код для создания процедуры управления событиями:

```
Protected Sub btnChangeMe_Click(ByVal sender As_  
    System.Object, ByVal e As System.EventArgs)  
    btnChangeMe.Text = "It works!"  
End Sub
```

Запустив программу, щелкните на кнопке, где написано `Change Text`. Visual Basic .NET самым волшебным образом заменит текст на кнопке надписью `It works!` Чтобы остановить выполнение этой доброй и наивной программы, щелкните на кнопке закрытия окна формы.

# Разработка форм

*В этой главе...*

- Создание форм и определение способа их отображения
- Изменение внешнего вида формы
- Открытие и закрытие формы

**С**амым главным элементом пользовательского интерфейса является окно, которое в Visual Basic .NET называется формой. Большинство программ, написанных на языке Visual Basic .NET, используют для своей работы не менее одной формы, а более сложные программы применяют по две, три и более форм.

Программа может использовать одну форму, в частности, для отображения набора кнопок. Если пользователь щелкает на определенной кнопке, открывается вторая форма, содержащая, скажем, имена, адреса и телефоны сотрудников, у которых он одолжил деньги до зарплаты, но пока еще ничего им не вернул.

## Создание форм

Когда вы приступите к разработке проекта Windows Applications, Visual Basic .NET автоматически создаст одну пустую форму, чтобы вы могли сразу начать создавать свой пользовательский интерфейс. Но если ваша программа не будет отличаться особым аскетизмом и простотой, вам наверняка понадобится еще парочка форм.

Вот каким образом можно создать новую форму.

1. Выберите команду **Project⇒Add Windows Form (Проект⇒Добавить форму)** или щелкните на панели инструментов на кнопке со стрелкой, расположенной справа от значка **Add New Item**, и выберите пункт **Add Windows Form**. Откроется диалоговое окно **Add New Item** (Добавить новый элемент).

2. Укажите имя новой формы в текстовом поле **Name**.

Когда вы вводите имя формы, не нужно указывать расширение файла .VB; оно будет добавлено к имени файла автоматически.

3. Щелкните на кнопке **Open**.

Visual Basic .NET отобразит новую форму, готовую к применению.



Возьмите за привычку периодически сохранять результаты проделанной вами работу, тем более что для этого нужно всего лишь нажать комбинацию клавиш **<Ctrl+Shift+S>** или выбрать команду **File⇒Save All (Файл⇒Сохранить все)**. Если ваш компьютер зависнет или если вдруг пропадет напряжение в сети, вы потеряете только ту часть работы, которая была выполнена уже после последнего сохранения.

## Переименование форм

При создании формы Visual Basic .NET автоматически присваивает ей стандартное имя, наподобие **Form3.vb**. Но вы можете дать форме более подходящее имя.



1. Откройте окно Solution Explorer.

Чтобы сделать это, нажмите комбинацию клавиш <Ctrl+Alt+L>, выберите команду **View**⇒**Solution Explorer** или щелкните на панели инструментов на значке Solution Explorer.

2. Щелкните на форме, которой вы хотите присвоить новое имя.

Можно также щелкнуть правой кнопкой мыши в окне формы, выбрать в открывшемся списке команду **Rename** (Переименовать) и указать новое имя. Если вы поступите именно так, то можете не выполнять шаги 3 и 4, указанные ниже.

3. Откройте окно Properties.

Чтобы сделать это, нажмите клавишу <F4>, выберите команду **View**⇒**Properties Window**, в окне Solution Explorer щелкните на значке Properties Window или щелкните правой кнопкой мыши в окне формы и выберите команду Properties.

4. Дважды щелкните на свойстве Name (Имя), относящемся к категории Design, и наберите новое имя.



Visual Basic .NET всегда добавляет к имени формы расширение .VB, например: frmMain.vb.

## Отображение нескольких форм

Поскольку программы обычно состоят из двух и более форм, иногда возникает необходимость отобразить на экране сразу несколько форм. Вот как это делается.

1. Откройте окно Solution Explorer.

Чтобы открыть данное окно, нажмите комбинацию клавиш <Ctrl+Alt+L>, выберите команду **View**⇒**Solution Explorer** или щелкните на панели инструментов на значке Solution Explorer.

2. Дважды щелкните на форме, которую хотите открыть.

Visual Basic .NET отобразит выбранную вами форму.



Visual Basic .NET может в каждый отдельный момент времени отображать на экране только одну форму. Однако если вы выбрали для просмотра несколько форм, Visual Basic .NET поместит каждую из них на отдельной вкладке. Имя формы и слово Design будут отображаться на корешке соответствующей вкладки, и все корешки будут постоянно видны на экране. Для того чтобы отобразить какую-то форму, достаточно щелкнуть на ее корешке. А чтобы просмотреть коды BASIC, относящиеся к форме, нужно щелкнуть на вкладке кодов.

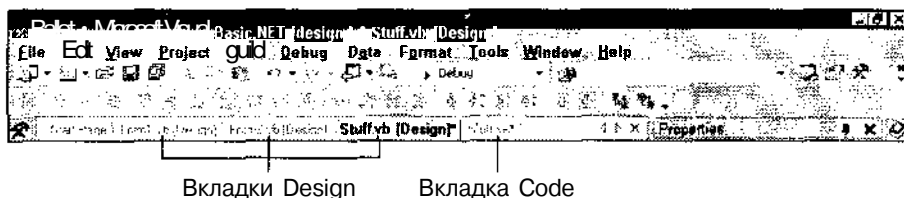


Рис. 6.1. Вы можете быстро перейти к нужной форме, щелкнув на вкладке с ее именем

# Изменение внешнего вида формы

Если вы уже создали форму и успели разместить в ее окне какие-то объекты (например, кнопки или флажки), то теперь вам наверняка захочется поработать немного над ее внешним видом, чтобы она не выглядела как невзрачное серое пятно.



Изменяя цветовую гамму пользовательского интерфейса, вы можете сделать его либо приятным и привлекательным, либо, наоборот, отталкивающим и раздражающим. Если выбранные вами цвета не понравятся пользователям, они просто не захотят пользоваться вашей программой.

## Итак, раскрасим вашу форму

По умолчанию все создаваемые Visual Basic .NET формы имеют серую фоновую заливку. В этом нет ничего страшного, но иногда все же хочется добавить хоть какой-то цвет, чтобы окно формы не выглядело как фрагмент бортовой части линкора или эсминца.

Изменить цвет фона в окне формы можно следующим образом.

1. Откройте окно Solution Explorer.

Чтобы сделать это, нажмите комбинацию клавиш <Ctrl+Alt+L>, выберите команду **View**⇒ **Solution Explorer** или щелкните на панели инструментов на значке **Solution Explorer**.

2. Дважды щелкните на форме, вил которой хотите изменить.

Вокруг выбранной вами формы появятся маркеры.

3. Откройте окно Properties.

Чтобы сделать это, нажмите клавишу <F4>, выберите команду **View**⇒ **Properties Window**, в окне Solution Explorer щелкните на значке **Properties Window** или щелкните правой кнопкой мыши в окне формы и выберите команду **Properties**.

4. Щелкните на свойстве **BackColor** (Цвет фона), относится к категории **Appearance**.

Рядом появится кнопка со стрелкой.

5. Щелкните на кнопке со стрелкой.

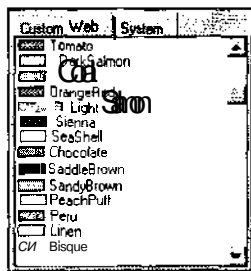


Рис. 6.2. Выберите в этом окне цвет фона для своей формы

Откроется диалоговое окно (рис. 6.2), в котором вам на выбор будут представлены три типа фоновых цветов:

- Custom — обычный набор цветов и их оттенков;

- Web— цвета, разработанные для создания Web-страниц (что не мешает вам использовать их при создании собственной программы);
  - System — цвета, используемые в среде Windows (например, цвета кнопок на панелях инструментов).
6. Щелкните на одной из вкладок и выберите понравившийся цвет.  
Visual Basic .NET заполнит форму указанными цветом.

## Создание фоновых рисунков

Вы можете не только заполнить окно формы каким-нибудь цветом, но и применить в качестве фонового рисунка какое-либо изображение, например, логотип компании, горный пейзаж или портрет вашего босса.

Чтобы сделать некий рисунок фоновым, необходимо выполнить следующие действия.

1. Откройте окно Solution Explorer.  
Чтобы сделать это, нажмите комбинацию клавиш <Ctrl+Alt+L>, выберите команду **View⇒Solution Explorer** или щелкните на панели инструментов на значке Solution Explorer.
2. Дважды щелкните на форме, вид которой хотите изменить.  
Вокруг выбранной вами формы появятся маркеры.
3. Откройте окно Properties.  
Чтобы сделать это, нажмите клавишу <F4>, выберите команду **View⇒Properties Window**, в окне Solution Explorer щелкните на значке Properties Window или щелкните правой кнопкой мыши в окне формы и выберите команду Properties.
4. Щелкните на свойстве **BackgroundImage** (Фоновый рисунок), относящемся к категории Appearance.  
Рядом с ним появится кнопка с тремя точками.
5. Щелкните на кнопке с тремя точками.  
На экране появится диалоговое окно Open.
6. Выберите изображение, которое вы хотите использовать в качестве фонового рисунка, и щелкните на кнопке Open.  
Возможно, вам придется открыть несколько папок, чтобы добраться до файла, в котором сохранен нужный рисунок. Если окно формы больше, чем выбранный вами рисунок, Visual Basic .NET заполнит форму несколькими копиями этого рисунка.



Чтобы удалить фоновый рисунок, на пятом шаге щелкните правой кнопкой мыши на кнопке с тремя точками и выберите команду Reset (Сброс).

## Виды границ

Границы способны не только посеять вражду между соседними государствами, но и помочь вам определить способ отображения формы на экране монитора. Visual Basic .NET предоставляет в ваше распоряжение семь таких способов. Все они описаны ниже, а соответствующие примеры можно увидеть на рис. 6.3.

- ✓ None. Границы и строка заголовка не отображаются, но все объекты формы будут видны на экране. Пользователь не может переместить, изменить размеры или свернуть форму такого типа.
- ✓ FixedSingle. Отображаются строка заголовка, кнопки управления окном и кнопка закрытия окна. Пользователь может перемещать, сворачивать и разворачивать окно, но не может изменять его размеры.
- ✓ Fixed3D. Работает точно так же, как и окно типа FixedSingle: единственное, что добавлено, — это эффект трехмерности вокруг границ формы. Пользователь может перемещать, сворачивать и разворачивать окно такого типа, но не может изменять его размеры.
- ✓ FixedDialog. Отображаются строка заголовка, кнопки управления окном и кнопка закрытия окна. Пользователь может перемещать, сворачивать и разворачивать окно, но не может изменять его размеры.
- ✓ Sizable. Отображаются строка заголовка, кнопки управления окном и кнопка закрытия окна. Пользователь может перемещать, сворачивать и разворачивать окно, а также изменять его размеры. Этот стиль используется для всех форм по умолчанию.
- ✓ Fixed ToolWindow. Отображаются строка заголовка и кнопка закрытия окна. Пользователь может перемещать такое окно, но не может сворачивать, разворачивать и изменять его размеры.
- ✓ Sizable ToolWindow. Отображаются строка заголовка и кнопка закрытия окна. Пользователь может перемещать такое окно (перетаскивая его за строку заголовка) и изменять его размеры (перетаскивая стороны или углы окна).

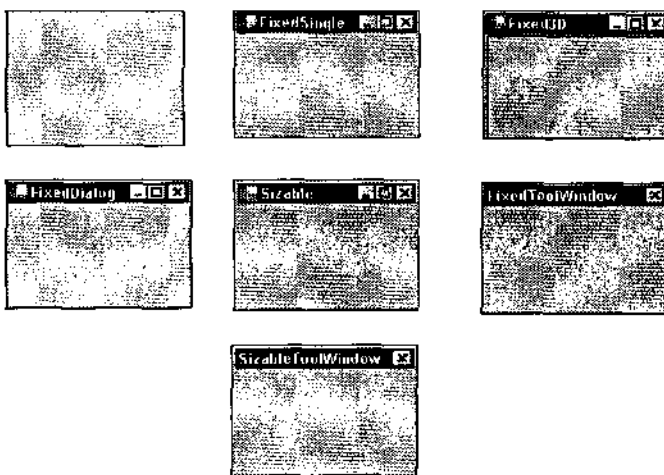


Рис. 6.3. Семь способов отображения окна формы на экране

Таким образом, каждый из описанных способов не только позволяет отображать различные объекты на экране, но и определяет, может ли пользователь перемещать окно и изменять его размеры.

А вот как можно изменить вид границ формы.

1. Откройте окно Solution Explorer.

Чтобы сделать это, нажмите комбинацию клавиш <Ctrl+Alt+L>, выберите команду View⇒Solution Explorer или щелкните на панели инструментов на значке Solution Explorer.

2. **Дважды** щелкните на форме, вид которой вы хотите изменить.

Вокруг выбранной вами формы появятся маркеры.

3. Откройте окно Properties.

Чтобы сделать это, нажмите клавишу <F4>, выберите команду View⇒Properties Window, в окне Solution Explorer щелкните на значке Properties Window или щелкните правой кнопкой мыши в окне формы и выберите команду Properties.

4. Щелкните на свойстве FormBorderStyle (Вид границы), относящемуся к категории Appearance.

Рядом с формой появится кнопка с направленной вниз стрелкой.

5. Щелкните на кнопке со стрелкой и выберите тип границы (например, Fixed3D).

Visual Basic .NET изменит способ отображения формы на экране.

## Сворачивание и разворачивание окна формы

Форма может отображаться на экране в развернутом виде (в таком случае она будет занимать все пространство экрана), в свернутом виде (на экране будет отображаться только ее заголовок) и в нормальном виде (окно формы будет занимать только часть экрана). Все три вида отображения формы представлены на рис. 6.4.

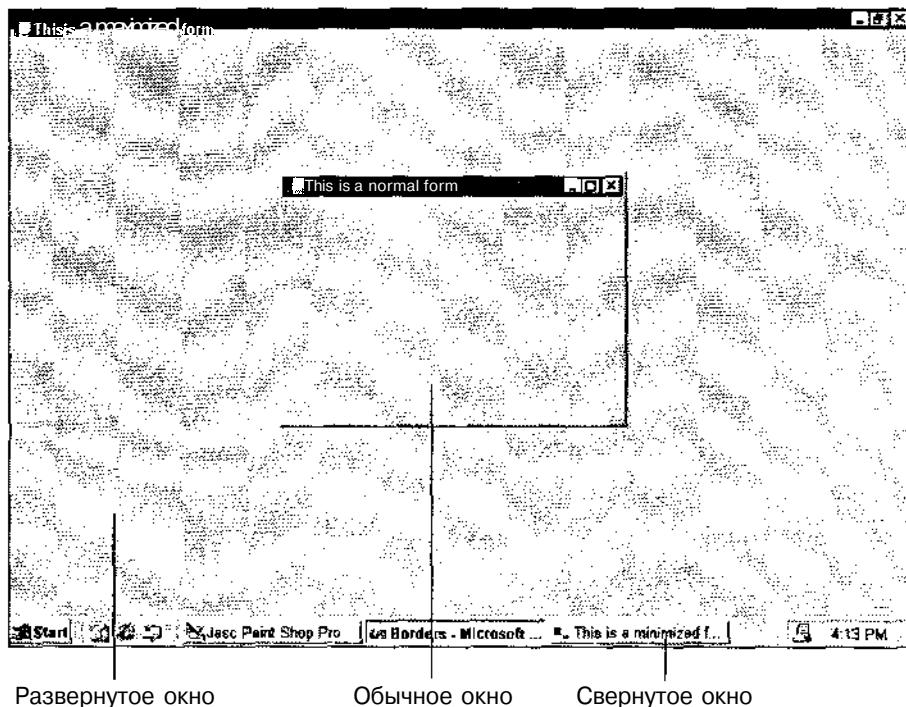


Рис. 6.4. Развернутое, обычное и свернутое окно формы

Чтобы заставить окно формы в процессе выполнения программы открыться в каком-то определенном виде, вы должны выполнить следующие действия.

1. Откройте окно Solution Explorer.

Чтобы сделать это, нажмите комбинацию клавиш <Ctrl+Alt+L>, выберите команду **View⇒Solution Explorer** или щелкните на панели инструментов на значке Solution Explorer.

2. Дважды щелкните на форме, параметры которой нужно изменить.

Вокруг выбранной вами формы появятся маркеры.

3. Откройте окно Properties.

Чтобы сделать это, нажмите клавишу <F4>, выберите команду **View⇒Properties Window**, в окне Solution Explorer щелкните на значке Properties Window или щелкните правой кнопкой мыши в окне формы и выберите команду Properties.

4. Щелкните на свойстве **WindowState (Вид окна)** категории **Layout**.

Рядом с формой появится кнопка с направленной вниз стрелкой.

5. Щелкните на кнопке со стрелкой и выберите нужную опцию (например, **Normal** или **Minimized**).

Большинство форм отображают кнопки управления окном (кнопки сворачивания и разворачивания окна), благодаря чему пользователь может в любое время свернуть или развернуть таковое. Конечно же, если вы хотите лишить его такой возможности, можете удалить эти кнопки с экрана.



Если вы определите, что стилем отображения окна формы на экране будет **None**, **Fixed Single**, **Fixed ToolWindow** или **Sizable ToolWindow**, кнопки управления окном отображаться не будут.

Скрыть или отобразить на экране кнопки управления окном вы можете следующим образом.

1. Откройте окно Solution Explorer.

Чтобы сделать это, нажмите комбинацию клавиш <Ctrl+Alt+L>, выберите команду **View⇒Solution Explorer** или щелкните на панели инструментов на значке Solution Explorer.

2. Дважды щелкните на форме, параметры которой нужно изменить.

Вокруг выбранной вами формы появятся маркеры.

3. Откройте окно Properties.

Чтобы сделать это, нажмите клавишу <F4>, выберите команду **View⇒Properties Window**, в окне Solution Explorer щелкните на значке Properties Window или щелкните правой кнопкой мыши в окне формы и выберите команду Properties.

4. Щелкните на свойстве **MinimizeBox (Кнопка сворачивания окна)** или **MaximizeBox (Кнопка разворачивания окна)**, которые относятся к категории **Windows Style**, и выберите значение **True** или **False**.



Если вы планируете разрешить пользователю разворачивать окно или изменять его размеры, не забудьте закрепить объекты формы (кнопки, флажки, переключатели и т.п.), поскольку в противном случае их будет кидать в разные стороны как лодки во время шторма или же они будут вообще пропадать с экрана. О том, как можно закрепить объекты в окне формы, было рассказано в главе 5.

## Размещение формы на экране

Чтобы получить гарантию того, что во время работы программа будет выглядеть именно так, как вы себе это представляли, необходимо точно указать, в каком месте экрана должна открываться каждая форма. Если вы этого не сделаете, формы могут открываться, например, одна над другой или, скажем, в тех частях экрана, на которые вы не рассчитывали.



Если для свойства **WindowState** вы определили значение **Minimized** или **Maximized**, указывать местонахождение формы на экране не имеет смысла, поскольку в первом случае открытая форма BASIC будет занимать все пространство экрана, а во втором случае она будет представлена только своим заголовком на панели задач Windows.

Чтобы определить положение открывающейся формы на экране, вам нужно для свойства **StartPosition** (Начальная позиция) категории **Layout** выбрать одно из **пяти** возможных значений.

- ✓ **Manual**. Позволяет с использованием свойства **Location** (категория **Layout**) вручную определить **позицию** формы на экране.
- ✓ **CenterScreen**. Форма открывается в **центре** экрана.
- ✓ **WindowsDefaultLocation**. Позволяет программе самостоятельно определить оптимальное место для открытия формы (хотя вам оно может показаться не самым оптимальным).
- ✓ **WindowsDefaultBounds**. Позволяет программе самостоятельно определить оптимальное место и оптимальный размер открываемой формы (хотя вам они могут показаться не такими уж и оптимальными).
- ✓ **CenterParent**. Форма открывается в центре окна другой формы.

Чтобы выбрать один из этих пяти вариантов, выполните следующие действия.

### 1. Откройте окно **Solution Explorer**.

Чтобы сделать это, нажмите комбинацию клавиш <Ctrl+Alt+L>, выберите команду **View⇒Solution Explorer** или щелкните на панели инструментов на значке **Solution Explorer**.

### 2. Дважды щелкните на форме, параметры открытия которой нужно изменить.

Вокруг выбранной вами формы появятся маркеры.

### 3. Откройте окно **Properties**.

Для этого нажмите клавишу <F4>, выберите команду **View⇒Properties Window**, в окне **Solution Explorer** щелкните на значке **Properties Window** или щелкните правой кнопкой мыши в окне формы и выберите команду **Properties**.

### 4. Щелкните на свойстве **StartPosition** (Начальная позиция) категории **Layout**.

Рядом появится кнопка с направленной вниз стрелкой.

### 5. Щелкните на кнопке со стрелкой и выберите нужную опцию (например, **CenterScreen**).



Для определения местонахождения формы на экране в процессе выполнения программы можно также использовать код BASIC. Вот как этот код может выглядеть: `ActiveForm.Location = New Point fx, Y)`

Команда ActiveForm указывает Visual Basic .NET на необходимость переместить активную в данный момент форму в нужную позицию. Значения X и Y определяют, какое состояние должно быть между окном формы и левой стороной экрана (X) и между окном формы и верхней стороной экрана (Y).

## Удаление и добавление форм

Не исключено, что в один прекрасный момент вы решите, что какая-то форма для нормальной работы вашей программы вовсе и не нужна. В таком случае вы вынуждены будете ее удалить.

### 1. Откройте окно Solution Explorer.

Чтобы сделать это, нажмите комбинацию клавиш <Ctrl+Alt+L>, выберите команду View⇒Solution Explorer или щелкните на панели инструментов на значке Solution Explorer.

### 2. Щелкните на форме, которую собираетесь удалить.

### 3. Выберите команду Project⇒Exclude From Project (Проект⇒Убрать из проекта) или щелкните правой кнопкой мыши на форме и выберите команду Exclude From Project.

Visual Basic .NET удалит форму из окна Solution Explorer.



Если вы удалите форму, которая ранее была сохранена, она все еще будет сохраняться на диске вашего компьютера. Правда, теперь она уже не будет частью вашего проекта Visual Basic .NET. Чтобы физически удалить все следы существования этой формы, используйте Windows Explorer. Вы найдете файл этой формы и удалите его из памяти компьютера.



Если вы хотите добавить к своему проекту уже существующую форму (например ту, которую недавно из него удалили), выберите команду Project⇒Add Existing Item (Проект⇒Добавить существующий элемент) или нажмите комбинацию клавиш <Shift+Alt+A> и щелкните на форме, которую хотите включить в проект.

## Тест на проверку полученных вами знаний

1. Что такое форма?
  - а. Термин Visual Basic .NET, обозначающий окно программы.
  - б. Философское понятие, противоположное "содержанию" программы.
  - в. Какая-то бесполезная **штсыковина**, которая при каждом запуске Visual Basic .NET отображается на экране в ожидании того, что я начну ее чем-то заполнять.
  - г. Форма - это моя реакция на Visual Basic .NET. Если я пишу программу и она начинает работать, значит, я в хорошей **форме**. Но если что-то не получается, тогда я в плохой форме, и лучше меня в это время не трогать.
2. Если ваша программа состоит из нескольких форм, как можно просмотреть каждую из них?
  - а. Никак. Это одна из причин, по которой программирование считается невероятно сложной задачей.
  - б. Необходимо купить еще несколько **компьютеров**, чтобы можно было просматривать каждую форму на отдельном мониторе.
  - в. Нужно щелкнуть на вкладке Design той формы, которую я хочу просмотреть. Если для нужной формы такая вкладка еще не отображается, нужно открыть окно Solution Explorer и дважды щелкнуть на названии этой формы.



# Выбор формы, которая будет отображаться первой

Когда ваша программа начинает функционировать, то обычно первой открывается форма, созданная в самом начале процесса разработки проекта. Если вы хотите, чтобы в первую очередь открывалась какая-то другая форма, выполните такие действия.

**1. Откройте окно Solution Explorer.**

Чтобы сделать это, нажмите комбинацию клавиш <Ctrl+Alt+L>, выберите команду View⇒Solution Explorer или щелкните на панели инструментов на значке Solution Explorer.

**2. Щелкните правой кнопкой мыши на названии проекта.**

Откроется контекстное меню.

**3. Выберите пункт Properties (Свойства).**

Откроется диалоговое окно Property Pages (Свойства страниц), показанное на рис. 6.5.

**4. Щелкните на кнопке со стрелкой в поле со списком Startup Object (Запускаемый при старте объект), чтобы увидеть список всех форм своего проекта.**

**5. Выберите форму, которая должна отображаться первой, и щелкните на кнопке OK.**

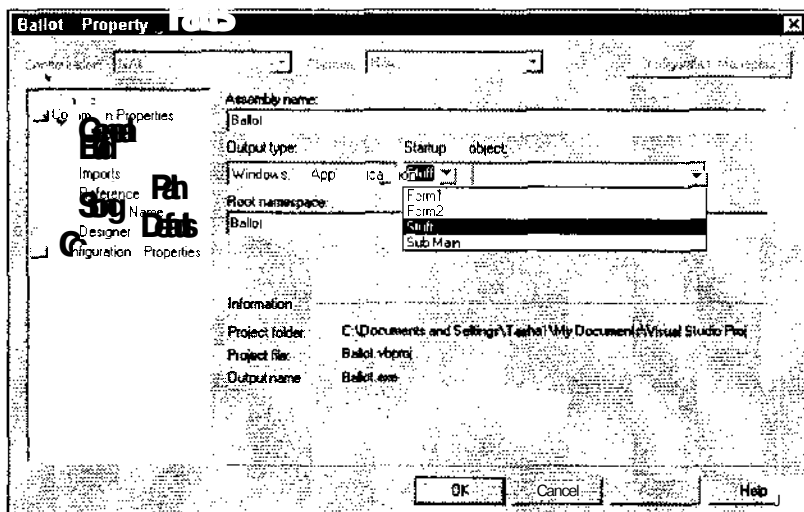


Рис. 6.5. В этом диалоговом окне вы можете выбрать форму, которая будет открываться сразу же после запуска программы

## Открытие, скрытие и закрытие форм

То, что ваша программа включает в себя несколько форм, конечно же, очень хорошо, но в процессе работы программы в каждый момент времени обычно отображается только одна форма. Чтобы можно было убрать с экрана одну форму и отобразить вместо нее другую,

нужно написать код BASIC, который бы говорил программе: "Хорошо, теперь вот эту форму покажи на экране, а ту, которая там сейчас, спрячь".



Открытие, скрытие и закрытие форм требует написания специальных кодов BASIC, и таковые будут подробно рассмотрены в четвертой части книги. Сейчас, в составшейся части настоящей главы, мы дадим лишь краткий обзор этой темы, поэтому пока не пытайтесь глубоко вникнуть во все тонкости работы приведенных здесь кодов BASIC.

## Открытие форм

Для того чтобы открыть (или закрыть) форму, нужно знать, какое ей присвоено имя. В окне Solution Explorer вы можете просмотреть список имен всех форм (они выглядят приблизительно так: `frmMain.vb`), используемых вашей программой.

Если вы уже знаете, как называется форма, которую необходимо отобразить на экране, нужно написать следующий код BASIC:

```
Dim oForm As FormName
oForm = New FormName()
oForm.Show()
oForm = Nothing
```

Возможно, кому-то будет трудно понять, как он работает, поэтому ниже мы дадим краткое пояснение.

1. Первая строка приказывает Visual Basic .NET: "Определи объект `oForm`, который будет представлять собой форму, именуемую `FormName`". Таким образом, если вы хотите открыть форму, носящую название `frmMain`, нужно написать следующий код:

```
Dim oForm As frmMain
```

(Вместо `oForm` вы можете дать объекту любое другое имя. Буква `o` в начале имени просто напоминает вам, что это объект.)

2. Вторая строка дает следующее указание Visual Basic .NET: "Создай новый объект, который будет представлять собой форму `FormName`". (Основное различие между первой и второй строками заключается в том, что первая строка говорит Visual Basic .NET только о необходимости создать объект, представляющий собой определенную форму, а вторая строка уже сама создает таковой. Если вам это сейчас не совсем очевидно, ничего страшного, пропустите это место и читайте дальше.)
3. Третья строка говорит Visual Basic .NET: "Отобрази форму, представляемую объектом `cForm`".
4. А четвертая строка приказывает ему: "Установи для объекта, именуемого `oForm`, значение `Nothing` (Ничего), чтобы освободить память, которую он до этого занимал".



Последний шаг очень важен, поскольку при открытии слишком большого количества форм без освобождения занимаемой ими оперативной памяти ресурс таковой быстро исчерпается и ваша программа вызовет сбой в работе компьютера.

## Скрытие (и восстановление) форм

Если вы хотите временно убрать форму с экрана, напишите такой код:

```
FormName.Hide()
```

После того как форма будет скрыта, вам в какой-то момент наверняка понадобится восстановить ее отображение на экране. Для этой цели следует написать код BASIC с использованием команды Show:

```
FormName.Show()
```

## Закрывание форм

Если форма скрыта, то это означает, что она просто не отображается на экране, но по-прежнему присутствует в памяти компьютера. Чтобы убрать форму из оперативной памяти, ее нужно закрыть. Это вам поможет сделать такой код.

```
FormName.Close()
```



Чтобы завершить выполнение программы, нужно закрыть все ее формы. По крайней мере хотя бы одна форма должна давать возможность запустить команду для выхода (это может быть кнопка Exit или команда контекстного меню File⇒Exit). Код BASIC, который закрывает последнюю форму, выглядит приблизительно так:

```
Me.Close()
```



Если вы просматривали коды, которые Visual Basic .NET генерирует автоматически при создании каждой новой формы, вы, наверное, замечали код, похожий на этот:

```
Form1 = Me
```

Данный код означает, что текущая форма будет представляться в дальнейшем словом Me. Теперь для указания на эту форму вам не нужно набирать ее полное имя (например, frmMainWindow), а достаточно ввести слово Me.

# Элементы как средство предоставления пользователю возможности выбора

*В этой главе...*

- Создание кнопок
- > Создание флажков и переключателей
- Отображение текста на объектах

**У** чтобы продуктивно работать, многие программы должны периодически получать данные от пользователя. Например, программе для каких-то целей может понадобиться информация о вашем семейном положении. Вместо того чтобы заставлять пользователя вводить эту информацию вручную, можно просто предложить ему выбрать один из вариантов: "Женат", "Не женат", "Жестоко разведен". Если пользователь должен выбирать только один из возможных вариантов, для этой цели часто используются два или несколько переключателей.

В этой главе рассказывается, как создавать кнопки, флажки и переключатели для предоставления пользователю возможности выбора.

## *Нажми на кнопку — получишь результат...*

Нажатие кнопки — это самая простая задача для кого бы то ни было. Даже маленький ребенок может нажать на кнопку, чтобы включить или выключить, скажем, настольную лампу.

Поскольку кнопки сами по себе настолько просты и знакомы нам с детства, многие программы используют их в своем интерфейсе и позволяют на них "нажимать" с помощью мыши. Вместо того чтобы заставлять вас блуждать по разным меню в поисках нужной команды, программа позволяет просто щелкнуть на одной из кнопок, которые всегда находятся перед глазами. Единственное, что вам нужно сделать, — это определить, на какой же кнопке все-таки щелкнуть.

На самом деле, кнопка — это всего лишь область на экране, где пользователь может щелкнуть с помощью мыши. На большинстве кнопок отображается текст, указывающий, какая команда выполняется при щелчке (например, OK, Cancel, Open, Exit).

Чтобы создать кнопку, вам нужно выполнить действия, перечисленные ниже.

1. Выберите команду **View⇒Toolbox** или нажмите комбинацию клавиш **<Ctrl+Alt+X>**. (Пропустите этот шаг, если панель **Toolbox** уже отображается на экране.)

На экране появится панель **Toolbox**. Возможно, вам придется щелкнуть на кнопке **Windows Forms**, чтобы найти инструмент **Button** (Кнопка).

2. Щелкните на инструменте **Button**.

Курсор мыши примет вид перекрестия. Если вы дважды щелкнете на инструменте **Button** панели **Toolbox**, Visual Basic .NET автоматически создаст кнопку в окне

формы. Потом вы сможете переместить ее на нужное место и придать ей необходимые размеры.

3. Поместите курсор мыши над тем местом, где должна быть нарисована кнопка, нажмите левую кнопку мыши и перетяните курсор. Когда контур создаваемой кнопки примет нужные размеры, отпустите кнопку мыши.

Visual Basic .NET нарисует в окне формы кнопку.



После того как кнопка будет создана, вам еще придется написать коды BASIC, чтобы щелчок на ней вызывал определенные действия. Подробно о написании кодов BASIC рассказывается в четвертой части книги.

## Создание флажков и переключателей

Группа флажков как бы говорит пользователю: "Выберите все, что считаете нужным". Сказанное можно продемонстрировать следующим примером:

Если я, предположим, собираюсь в поход, мне нужно взять с собой (отметьте все, что вам требуется):

- ☐ зубную щетку;
- ☐ спальный мешок;
- ☐ ящик пива;
- ☐ заграничный паспорт.

Переключатели действуют примерно так же, как кнопки на пульте обычного телевизора. В каждый конкретный момент времени вы можете выбрать для просмотра только один канал. И из группы переключателей пользователь может каждый раз выбирать только один из предложенных вариантов. В данном случае пример должен быть несколько иным:

Мой месячный заработок (можно выбрать только один пункт):  
отсутствует;

- ☐ О до \$100;
- ☐ О до \$1000;
- ☐ О вам такая сумма и не снилась.

Вот как можно создать флажок или переключатель.

1. Выберите команду View⇌Toolbox или нажмите комбинацию клавиш <Ctrl+Alt+X>. (Пропустите этот шаг, если панель Toolbox уже отображается на экране.)

На экране появится панель Toolbox. Возможно, вам придется щелкнуть на кнопке **Windows Forms**, чтобы найти инструменты **CheckBox** (Флажок) и **RadioButton** (Переключатель).

2. Щелкните на кнопке инструмента **CheckBox** или на кнопке инструмента **RadioButton**.

Курсор мыши примет вид перекрестия. Если вы дважды щелкнете на кнопке инструмента **CheckBox** или **RadioButton**, Visual Basic .NET автоматически создаст флажок или переключатель в окне формы. Потом вы сможете переместить его в нужное место и придать ему необходимые размеры.

3. Поместите курсор мыши над тем местом, где должен быть нарисован объект, нажмите левую кнопку мыши и перетяните его. Когда контур объекта примет нужные размеры, отпустите кнопку мыши.

Visual Basic .NET нарисует в окне формы флажок или переключатель.



В отличие от обычных кнопок, флажки и переключатели не требуют создания кодов BASIC для своей активизации. Вам лишь нужно будет написать код, определяющий, какой из предложенных вариантов был выбран пользователем. О том, как это делается, подробно описывается в главе 16, но забегаая вперед, скажем, что свойство, определяющее, выбран объект или **нет**, называется **Checked** и относится к категории Appearance. Если флажок или переключатель выбран, это свойство принимает значение True, в противном случае — False.

## Выравнивание флажков и переключателей

Флажки и переключатели обычно отображаются слева от сопровождающего их текста.

Но Visual Basic .NET этим не ограничивается и предлагает еще восемь способов расположения флажков и переключателей относительно текста (рис. 7.1).

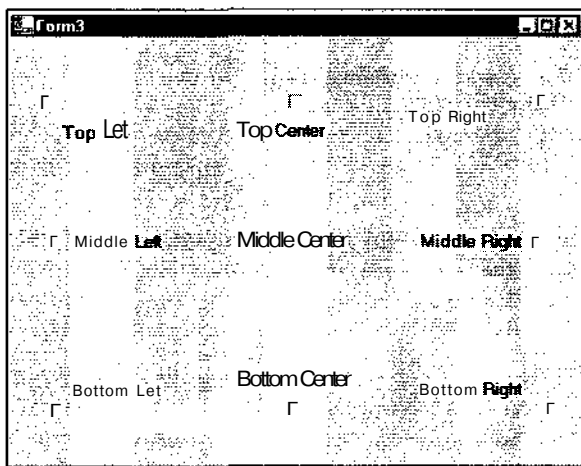


Рис. 7.1. Девять способов выравнивания флажков и переключателей относительно сопровождающего их текста

Вы можете изменить способ выравнивания флажка или переключателя следующим образом.

1. Щелкните на флажке или переключателе, положение которого хотите выровнять.
2. Откройте окно Properties.

Чтобы сделать это, нажмите клавишу <F4>, выберите команду **View⇄Properties Window**, в окне Solution Explorer щелкните на кнопке **Properties Window** или щелкните правой кнопкой мыши в окне формы и выберите команду **Properties**.

3. Щелкните на свойстве **CheckAlign** (Выравнивание флажка) категории Appearance.

Откроется графическое меню, состоящее из девяти прямоугольников, которые соответствуют положениям флажка или переключателя относительно текста (рис. 7.2).

4. Щелкните на одном из прямоугольников и выберите способ выравнивания.

Visual Basic .NET отобразит флажок или переключатель в соответствии со сделанным вами выбором.



Рис. 7.2. Определив значение свойства *CheckAlign*, можно выбрать способ отображения флажка или переключателя относительно текста



Чтобы отобразить флажок или переключатель в определенной позиции, вам, возможно, придется изменить размеры самого объекта.



Вместо изменения свойства *CheckAlign* (определяет положение флажка или переключателя) можно изменить свойство *TextAlign* (Выравнивание текста) категории *Appearance*, которое определяет положение текста относительно флажка или переключателя.

## Группировка флажков и переключателей

Очень редко флажки и переключатели используются по одному. Почти всегда они собираются в группы, как стайки рыбок на мелководье. Чтобы выделить группу флажков или переключателей, можно использовать инструмент *GroupBox* (Группа), который рисует рамку вокруг нескольких объектов. После того как рамка нарисована, внутри нее можно размещать флажки и переключатели.



В предыдущих версиях Visual Basic инструмента *GroupBox* еще не было. Вместо него использовался другой инструмент, называемый *Frame* (Рамка). Они очень похожи, но все же немного отличаются. Если вы уже программировали на Visual Basic, не забывайте, что *GroupBox* работает несколько иначе, чем *Frame*.



Если вы создадите в окне формы несколько переключателей, то пользователь в любой момент времени сможет выбрать только один из них. Если же вы, используя инструмент *GroupBox*, выделите их в две отдельные группы, пользователь сможет выбрать сразу два переключателя: по одному переключателю в каждой группе (рис. 7.3).

А вот как можно создать группу.

1. Выберите команду **View⇌Toolbox** или нажмите комбинацию клавиш <Ctrl+Alt+X>. (Пропустите этот шаг, если панель *Toolbox* уже отображается на экране.)

На экране появится панель *Toolbox*. Возможно, вам придется щелкнуть на кнопке *Windows Forms*, чтобы найти кнопку инструмента *GroupBox* (Группа).

2. Щелкните на кнопке *GroupBox*.

Курсор мыши примет вид перекрестия. Если вы дважды щелкнете на кнопке инструмента *GroupBox*, Visual Basic .NET автоматически нарисует рамку группы в окне формы. Потом вам, скорее всего, придется переместить ее в нужное место и изменить размеры.

3. Поместите курсор мыши над тем местом, где должен быть нарисован объект, нажмите левую кнопку мыши и перетяните курсор. Когда контур рамки группы примет нужные размеры, отпустите кнопку мыши.

Visual Basic .NET нарисует в окне формы рамку для группы объектов. Теперь вы можете приступить к созданию флажков и переключателей внутри области, выделенной для одной группы.

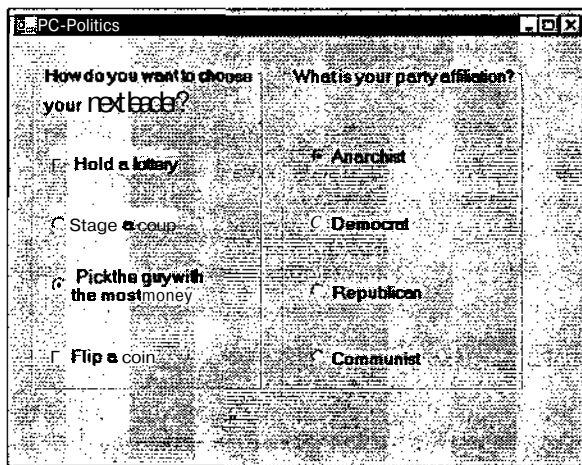


Рис. 7.3. Используя инструмент GroupBox, переключатели можно разбить на две отдельные группы



Чтобы визуально выделить группу объектов, нужно для объекта **GroupBox** изменить свойство **BackColor** (Цвет фона), относящееся к категории **Appearance**. После того как вы это сделаете, цвет фона внутри области, выделенной рамкой группы, будет отличаться от фоновой цвета всего остального окна.



Щелкните на рамке группы таким образом, чтобы вокруг нее отображались маркеры, после чего вам достаточно будет произвести двойной щелчок на кнопке инструмента **CheckBox** или **RadioButton**, чтобы Visual Basic .NET автоматически создал флажок или переключатель, принадлежащие этой группе.

### Тест на проверку полученных вами знаний

1. В чем главное различие между флажками и переключателями?
  - а. Пользователь может выбрать сразу несколько флажков и только один переключатель.
  - б. Окошко флажка квадратное, а переключателя - круглое.
  - в. Вообще-то различия никакой не существует, но я до сих пор не могу понять, что нужно сделать, чтобы выбрать сразу два переключателя.
  - г. Ваш вариант.
2. Каким свойством определяется текст, который будет отображаться на кнопках, флажках и переключателях?
  - а. Текст определяется свойствами моего характера и воображения - что захочу, то и напишу.
  - б. Текст определяется свойствами решаемых программой задач.
  - в. Определяющее значение имеют свойства и привычки пользователей, для которых создается программа.
  - г. Свойством **Text**, принадлежащим к категории **Appearance**.



После того как флажок или переключатель будет нарисован внутри рамки группы, он постоянно будет оставаться в ее пределах. При перемещении рамки группы вместе с нею перемещаются и все расположенные внутри нее флажки и переключатели.

## *Отображение текста на кнопках, флажках, переключателях и рамках групп*

После того как кнопка, флажок, переключатель и рамка группы будут нарисованы в окне формы, нужно определить поясняющий текст, который должен на них отображаться. Это достигается путем изменения свойства **Text**.

Чтобы изменить значение свойства **Text**, сделайте следующее.

- 1. Щелкните на объекте, для которого нужно изменить текст.**

Visual Basic .NET отобразит маркеры вокруг выделенного объекта.

- 2. Откройте окно Properties.**

Чтобы сделать это, нажмите клавишу <F4>, выберите команду **View⇒Properties Window**, в окне **Solution Explorer** щелкните на значке **Properties Window** или щелкните правой кнопкой мыши в окне формы и выберите команду **Properties**.

- 3. Дважды щелкните на свойстве Text категории Appearance.**

Visual Basic .NET выделит текущее значение свойства **Text**.

- 4. Наберите текст, который должен отображаться на объекте.**

Изменить шрифт и начертание отображаемого на объекте текста вы также можете, воспользовавшись свойством **Font (Шрифт)** категории **Appearance**.



# Использование текстовых полей и надписей

*В этой главе...*

- Создание надписей и текстовых полей
- Настройка параметров отображения текста

**Н**есмотря на огромные возможности графической составляющей пользовательского интерфейса, иногда кнопок, флажков и переключателей бывает недостаточно для поддержки нормального диалога с пользователем. Время от времени программе необходимо отобразить на экране слово, предложение, абзац, а то и целую новеллу. Нередко и у пользователя возникает желание набрать “доброе” слово (а то и два) в адрес программы.

Если вы хотите просто отобразить текст для пользователя, примените объект, именуемый **Label** (Надпись). Если вы хотите отобразить текст и позволить пользователю набрать что-нибудь в ответ, воспользуйтесь объектом **TextBox** (Текстовое поле).

Таким образом, текстовые поля выполняют сразу две функции:

- 1. ✓ отображают текст на экране;
- 2. ✓ предоставляют пользователю возможность набрать текст для программы.

Текстовое поле является одним из наиболее гибких объектов пользовательского интерфейса, поскольку позволяет получить от пользователя почти любую информацию (текстовую и числовую). Если текстовых полей в вашей программе будет достаточно много, то не исключено даже, что вы повысите общий уровень грамотности наших сограждан.

## Создание надписей и текстовых полей

В повседневной жизни нас окружает множество надписей, таких как номера на дверях квартир, названия на обложках журналов, цифры и буквы на номерах автомобилей. Очень часто надписи привлекают наше внимание к тому, чего бы мы, не будь их, и не заметили.

Текстовые поля хоть и могут отображать текст, используются они в основном для предоставления пользователю возможности вводить какие-то данные для программы, например свое имя, пароль или телефонный номер. Часто надписи применяются вместе с текстовыми полями с целью сообщить пользователю, какого рода информацию он должен набрать в текстовом поле.



И надписи, и текстовые поля могут отображать текст на экране. Основное различие между ними состоит в том, что внести изменения в текст текстового поля пользователь может, а в текст надписи — нет.

Чтобы создать надпись или текстовое поле, необходимо выполнить такие действия.

1. Выберите команду **View⇒Toolbox** или нажмите комбинацию клавиш **<Ctrl+Alt+X>**. (Пропустите этот шаг, если панель **Toolbox** уже отображается на экране.)

На экране появится панель Toolbox. Возможно, вам придется щелкнуть на кнопке **Windows Forms**, чтобы найти инструменты **Label** (Надпись) и **TextBox** (Текстовое поле).

**2. Щелкните на кнопке Label или TextBox.**

Курсор мыши примет вид перекрестия. Если вы дважды щелкнете на инструменте **Label** или **TextBox**, Visual Basic .NET автоматически создаст соответствующий объект в окне формы. Потом вы сможете переместить его в нужное место и придать ему необходимые размеры.

**3. Поместите курсор мыши над тем местом, где должен быть нарисован объект, нажмите левую кнопку мыши и перетяните курсор. Когда контур объекта примет нужные размеры, отпустите кнопку мыши.**

Visual Basic .NET создаст надпись или текстовое поле (в зависимости от того, на кнопке какого инструмента вы щелкнули) в окне формы.

**4. Откройте окно Properties.**

Чтобы сделать это, нажмите клавишу <F4>, выберите команду **View⇒Properties Window**, в окне **Solution Explorer** щелкните на значке **Properties Window** или щелкните правой кнопкой мыши в окне формы и выберите команду **Properties**.

**5. Дважды щелкните на свойстве Text (категория Appearance) и наберите текст, который должен отображаться внутри текстового поля или надписи.**

Почти во всех случаях для текстового поля лучше удалить значение свойства **Text**, чтобы на экране оно отображалось пустым и готовым для набора пользователем своего текста.



Для того чтобы текст отображался каким-то особым образом, используйте свойство **Font** (Шрифт), принадлежащее к категории **Appearance**.

Вам не нужно писать коды BASIC, чтобы сделать надпись и текстовое поле рабочими. Однако код BASIC нужно будет написать для считывания данных, сохраненных в качестве значения свойства **Text**, — таким образом программа узнает, какую информацию оставил для нее пользователь. О том, как это делается, подробно рассказывается в главе 16.

## Изменение внешнего вида отображаемого текста

Чтобы текст выглядел как-то по-особенному, можно изменить шрифт, его размер либо способ начертания (полуужирный или курсив). Делается это так.

**1. Щелкните на надписи или текстовом поле, способ отображения текста которого вы хотите изменить.**

**2. Откройте окно Properties.**

Чтобы сделать это, нажмите клавишу <F4>, выберите команду **View⇒Properties Window** в окне **Solution Explorer** щелкните на значке **Properties Window** или щелкните правой кнопкой мыши в окне формы и выберите команду **Properties**.

**3. Щелкните на свойстве Font (категория Appearance).**

Рядом с ним появится кнопка с тремя точками.

**4. Щелкните на кнопке с тремя точками.**

На экране появится диалоговое окно **Font**.

5. Установите шрифт, его размер, способ начертания, задайте нужные эффекты (например, **Strikeout** (Зачеркивание)), а когда закончите, щелкните на кнопке ОК.



Тексты надписей и текстовых полей одновременно могут отображаться лишь одним шрифтом, с одним стилем и одним размером. Если вам вдруг захочется отобразить текст одного объекта разными шрифтами, выбросите эту идею из головы — у вас все равно ничего не получится.

## Раскрашивание текста надписей и текстовых полей

Если вы любили в детстве писать слова на асфальте разноцветными мелками, то вам понравится идея изменения цветов при отображении текста объектов Visual Basic .NET.

Обычно Visual Basic .NET отображает черный текст на белом (для текстовых полей) или на сером (для надписей) фоне. Для того чтобы "картинка ожила", можно изменить как цвет текста, так и цвет фона.



Цвет, которым заполняется область надписи или текстового поля (цвет фона), определяется свойством **BackColor**. Цвет, которым отображается сам текст, определяется свойством **ForeColor**. Оба свойства относятся к категории **Appearance**.

Чтобы изменить цвет фона или цвет текста, сделайте следующее.

1. Щелкните на надписи или текстовом поле, цвет которого вы хотите изменить.
2. Откройте окно **Properties**.

Чтобы сделать это, нажмите клавишу <F4>, выберите команду **View⇌Properties Window**, в окне **Solution Explorer** щелкните на значке **Properties Window** или щелкните правой кнопкой мыши в окне формы и выберите команду **Properties**.

3. Щелкните на свойстве **BackColor** или **ForeColor** (категория **Appearance**).

Рядом с ним появится кнопка с направленной вниз стрелкой.

4. Щелкните на кнопке с направленной вниз стрелкой.

Откроется диалоговое окно с тремя вкладками: **Custom**, **Web** и **System**.

5. Щелкните на нужной вкладке и выберите подходящий цвет.

Visual Basic .NET примет внесенные изменения и отобразит цветной текст.



Изменив цвет отображения текста, вы можете выделить определенную информацию, чтобы обратить на нее внимание пользователя (или, наоборот, сделать ее малозаметной). Однако помните, что слишком большая цветовая гамма может произвести плохое впечатление, а кроме того, некоторые люди могут быть цветоаномалами или дальтониками. Это означает, что какие-то цвета и оттенки они различают плохо или вовсе не различают, а потому часть текста могут просто не заметить. Поэтому применять одновременно различные цвета нужно очень осторожно.

## Настройка отображения границ

Обычно объект **Label** (Надпись) отображает текст, но не отображает каких бы то ни было границ. Границы объекта **TextBox** (Текстовое поле) видны на экране, но особой оригинальностью они не отличаются. Конечно же, это не единственно возможный вариант, и вы без труда можете изменить способ отображения границ на экране.

1. Щелкните на текстовом поле или надписи, границы которых хотите изменить.
2. Откройте окно **Properties**.  
Чтобы сделать это, нажмите клавишу <F4>, выберите команду **View⇒Properties Window**, в окне **Solution Explorer** щелкните на значке **Properties Window** или щелкните правой кнопкой мыши в окне формы и выберите команду **Properties**.
3. Щелкните на свойстве **BorderStyle (Тип границы)**, относящемся к категории **Appearance**, и выберите один из трех пунктов:
  - None (устанавливается по умолчанию для надписей);
  - FixedSingle;
  - Fixed3D (устанавливается по умолчанию для текстовых полей).

## Выравнивание текста относительно границ объекта

Visual Basic .NET может выравнивать текст надписей по горизонтали и по вертикали, а текстовых полей — только по горизонтали.



Пользователь не сможет определить, как выровнен текст, если на экране не отображаются границы объекта.

Вот что нужно сделать, чтобы выровнять текст.

1. Щелкните на надписи или текстовом поле, текст которого нужно выровнять.
2. Откройте окно **Properties**.  
Чтобы сделать это, нажмите клавишу <F4>, выберите команду **View⇒Properties Window**, в окне **Solution Explorer** щелкните на значке **Properties Window** или щелкните правой кнопкой мыши в окне формы и выберите команду **Properties**.
3. Щелкните на свойстве **TextAlign (Выравнивание текста)** категории **Appearance** и выберите одну из следующих опций.

В случае текстового поля:

- Left (Слева);
- Right (Справа);
- Center (По центру).

В случае надписи:

- TopLeft (Слева сверху, устанавливается по умолчанию);
- Top Right (Справа сверху);
- Top Center (По центру сверху);
- MiddleLeft (Слева посередине);
- Middle Right (Справа посередине);
- Middle Center (В центре посередине);
- Bottom Left (Слева внизу);
- Bottom Right (Справа внизу);
- Bottom Center (По центру внизу).

# Настройка текстовых полей

Текстовые поля со свойствами, установленными по умолчанию, могут быть не очень удобными в использовании. Если вы набираете слишком много текста, он будет смещаться, и набранный ранее текст уйдет из поля зрения. Для управления курсором внутри текстового поля используются обычные в таких случаях клавиши.

- ✓ <Delete>. Удаляет символ справа от курсора.
- ✓ **Клавиша возврата**. Удаляет символ слева от курсора.
- ✓ **Стрелка влево**. Перемещает курсор на один символ влево.
- ✓ **Стрелка вправо**. Перемещает курсор на один символ вправо.
- ✓ <Ctrl>+Стрелка влево. Перемещает курсор на одно слово влево.
- ✓ <Ctrl>+Стрелка вправо. Перемещает курсор на одно слово вправо.
- ✓ <Home> (или <Ctrl+Home>). Перемещает курсор на начало строки.
- ✓ <End> (или <Ctrl+End>). Перемещает курсор в конец строки.
- ✓ <Shift>+любая клавиша управления курсором. Выделяет текст.

По своему усмотрению вы можете настроить текстовое поле так, чтобы оно прокручивало текст как текстовый процессор, принимало пароли (отображая на экране вместо букв какой-нибудь символ, например звездочку) или позволяло вводить только текст, длина которого не превышала бы определенного значения.

## Прокрутка текста

Если пользователь, набирая текст, достигает конца текстового поля, то при наборе каждого нового символа текст начинает смешаться, из-за чего ранее набранные символы один за другим уходят с экрана. Но текст можно заставить прокручиваться, что делает возможным отображение на экране сразу нескольких строк. И сделать это вовсе не трудно.

1. Щелкните на текстовом поле, текст которого должен прокручиваться.

2. Откройте окно Properties.

Чтобы сделать это, нажмите клавишу <F4>, выберите команду View⇨Properties Window, в окне Solution Explorer щелкните на значке Properties Window или щелкните правой кнопкой мыши в окне формы и выберите команду Properties..

3. Щелкните на свойстве Multiline (В несколько строк) категории Behavior.

Рядом с ним появится кнопка со стрелкой.

4. Щелкните на кнопке со стрелкой и выберите значение True.

Пока вы не присвоите свойству Multiline (В несколько строк) значение True (Истина), вы не сможете изменить высоту текстового поля.

5. Щелкните на текстовом поле, которое вы выбрали на шаге 1, и измените его высоту.

Или же щелкните на свойстве Height (Высота) категории Layout и укажите новое значение высоты текстового поля.

6. Убедитесь, что свойство Wordwrap (Прокрутка), относящееся к категории Behavior, имеет значение True.

Свойство Wordwrap имеет значение True по умолчанию, однако если вы изменили его на False, вам придется снова вернуть ему значение True. В следующий раз, когда вы запустите программу, это текстовое поле будет состоять из нескольких строк, и текст будет переходить из одной строки в другую.

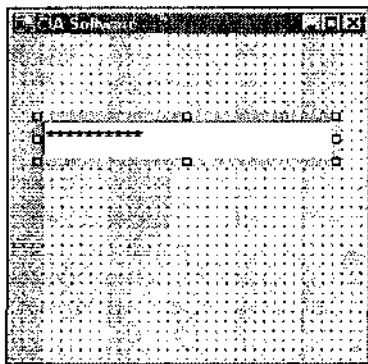


Вы можете присвоить свойству ScrollBars (Полосы прокрутки) категории Appearance значение Vertical. В таком случае, если весь текст невозможно будет поместить в текстовое поле, из-за его большого объема, то при необходимости можно будет использовать полосу прокрутки.

## Создание полей для ввода паролей

Если вы работаете на ЦРУ, СБУ, КГБ, ФБР, ГРУ или любую другую организацию, для которой свои и чужие секреты — это смысл жизни, вас наверняка заинтересует возможность создания текстовых полей для ввода паролей.

Вместо того чтобы отображать на экране все, что вы набираете, поля для ввода паролей на месте любой буквы или знака будут содержать какой-нибудь один символ, например, звездочку (\*). На рис. 8.1 показано окно с таким полем, в котором уже набран пароль Супер-Тайна, но, как вы видите, вместо букв отображаются только звездочки.



*Рис. 8.1. Когда пользователь набирает текст в поле для ввода паролей, на экране все время отображается один и тот же символ, например, звездочка*

Чтобы создать поле для ввода паролей, нужно определить символ, который будет отображаться на экране при наборе любого знака или буквы.

1. Щелкните кнопкой мыши на текстовом поле, которое должно принимать секретные сведения.
2. Откройте окно Properties.

Чтобы сделать это, нажмите клавишу <F4>, выберите команду View⇌ Properties Window, в окне Solution Explorer щелкните на значке Properties Window или щелкните правой кнопкой мыши в окне формы и выберите команду Properties.

3. Щелкните на свойстве Multiline (категория Behavior).  
Рядом с ним появится кнопка со стрелкой, направленной вниз.
4. Щелкните на кнопке со стрелкой и выберите значение False.

Если вы не сделаете этого, т.е. оставите для поля возможность состоять из нескольких строк, скрывать информацию оно не сможет. Таким образом Visual Basic .NET дает понять, что не стоит создавать огромные пароли.

5. Дважды щелкните на свойстве **PassChar** (Символ пароля), относящемся к категории **Behavior**, и наберите какой-нибудь символ (например, звездочку), который будет заменять собой на экране любой введенный знак.



На экран может выводиться только какой-нибудь один символ. Как вы уже могли заметить, наиболее часто для этой цели используется звездочка (\*).

## Ограничение длины текста

Чтобы приучить пользователя к лаконичности, можно ограничить максимальную длину текста, принимаемого текстовым полем. Это предотвратит попытки некоторых пользователей вводить в текстовые поля пространственные сочинения в стиле "как я провел прошлое лето".

Определить максимальное количество символов, которое будет приниматься текстовым полем, можно следующим образом.

1. Щелкните кнопкой мыши на текстовом поле, которое не должно принимать слишком длинные "послания" от пользователя.
2. Откройте окно **Properties**.

Чтобы сделать это, нажмите клавишу <F4>, выберите команду **View⇒Properties Window**, в окне **Solution Explorer** щелкните на значке **Properties Window** или щелкните правой кнопкой мыши в окне формы и выберите команду **Properties**.

3. Дважды щелкните на свойстве **MaxLength** (Максимальная длина), относящемся к категории **Behavior**, и укажите любое положительное число.

Это число и будет определять максимальное количество вводимых символов. Если таким числом будет 0 (нуль), значит ограничения сняты и пользователь может вводить любое количество символов.



Свойство **MaxLength** определяет максимальное количество символов, которое может быть набрано в текстовом поле пользователем. Однако это ограничение не распространяется на текст, который вы отображаете в текстовом поле с помощью свойства **Text**.

### Тест на проверку полученных вами знаний

1. Для каких целей могут быть использованы текстовые поля?
  - а. Ими можно заполнить окно формы, чтобы оно не казалось пустым.
  - б. Пользователи могут применять текстовое поле как книжку с предложениями.
  - в. Текстовое поле может быть использовано для вывода информации на экран и для получения данных от пользователя.
  - г. Текстовое поле можно сделать секретным и записывать туда все, что я думаю о своем боссе.
2. Если свойству **PasswordChar** текстового поля присвоено значение \* (звездочка), а свойству **MaxLength** - значение 10, то что это значит?
  - а. Одну минутку, я должен вернуться на несколько страниц назад и прочитать об этом еще раз.
  - б. Это значит, что мой пароль может состоять максимум из 10 звездочек.
  - а. В этом текстовом поле можно набирать значения, состоящие максимум из 10 символов, и при их наборе вместо символов будут отображаться звездочки.





# Использование списков и полей со СПИСКОМ

*В этой главе...*

- > Создание списков и полей со списком
- > Сортировка элементов, входящих в список
- Настройка параметров отображения списков

**Ф**лажки и переключатели предоставляют пользователю возможность выбирать нужные опции из числа доступных вариантов, но при этом возникает небольшая проблема. Дело в том, что каждый такой вариант занимает место в окне формы. К тому же, чем больше флажков и переключателей отображается на экране, тем более запутанной становится общая картина и менее очевидным сам выбор.

Другим способом предоставления пользователю возможности выбора является использование списков и полей со списком. Оба эти объекта способны предоставить гораздо большее количество вариантов, чем флажки и переключатели, но места на экране они занимают намного меньше. Более того, поля со списком, помимо предоставления готовых вариантов, могут дать пользователю возможность набрать свой вариант ответа, используя объект как обычное текстовое поле.

## Создание списков и полей со списком

Списки предоставляют в распоряжение пользователя перечень возможных вариантов ответа, один из которых он может выбрать. Если среди предложенных вариантов нет того, который устроил бы пользователя, значит, ему не повезло. Все равно придется выбирать один из готовых вариантов.

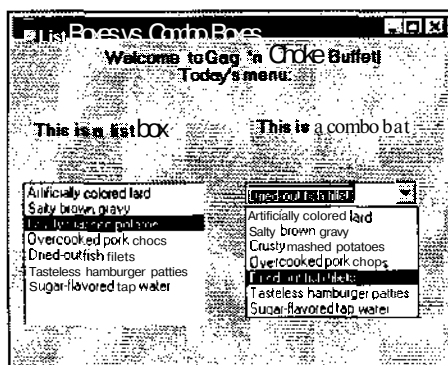


Рис. 9.1. Вот как выглядят СПИСОКИ и поле со СПИСКОМ

Поля со списком также содержат перечень возможных вариантов ответа. Но в отличие от списка, если пользователя не устраивает ни один из предложенных вариантов, поле со списком может принять от пользователя его вариант ответа. На рис. 9.1 показаны примеры использования в окне формы списка и поля со списком. Обратите внимание, что поле со списком открывает свой набор возможных вариантов лишь после того, как пользователь щелкнет на кнопке со стрелкой, в то время как список может постоянно отображать возможные варианты.

## Как нарисовать объекты в окне формы

Работа списков похожа на обслуживание в ресторане быстрого питания. Вы можете выбрать для себя только то, что есть в меню; при этом обслуживающий персонал понятия не имеет, как готовится что-либо другое. Поля со списком, напротив, похожи на серьезные заведения, где вы, просмотрев меню, можете сказать что-нибудь наподобие: "Я вижу, что у вас ресторан для вегетарианцев, но мне приготовьте, пожалуйста, поросенка с хреном".

Чтобы нарисовать список или поле со списком в окне формы, выполните следующие действия.

1. Выберите команду **View⇒Toolbox** или нажмите комбинацию клавиш **<Ctrl+Alt+X>**. (Пропустите этот шаг, если панель **Toolbox** уже отображается на экране.)

На экране появится панель **Toolbox**. Возможно, вам придется щелкнуть на кнопке **Windows Forms**, чтобы найти инструменты **ListBox** (Список) и **ComboBox** (Поле со списком).

Если вы дважды щелкнете на инструменте **ListBox** или **ComboBox**, Visual Basic .NET нарисует для вас эти объекты автоматически.

2. Щелкните на кнопке инструментов **ListBox** или **ComboBox**.

Курсор мыши примет вид перекрестия. Если вы хотите создать список, элементами которого будут флажки (как это показано на рис. 9.2), щелкните на кнопке инструмента **CheckBoxListBox** (Список флажков).

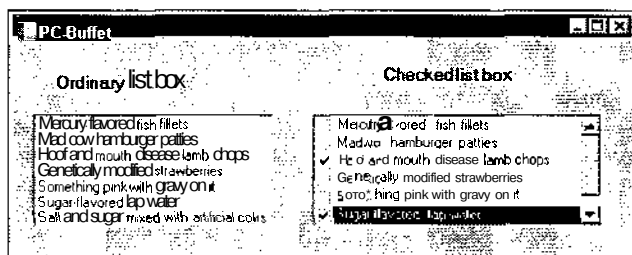


Рис. 9.2. Сравните между собой обычный список и список флажков

3. Поместите курсор мыши над тем местом, где должен быть создан объект.
4. Нажмите левую кнопку мыши и перетяните курсор. Когда контур объекта примет нужные размеры, отпустите кнопку мыши.

Visual Basic .NET нарисует в окне формы список или поле со списком и сразу же присвоит им какие-нибудь безликие имена, наподобие **ListBox3** или **ComboBox2**.

## Настройка параметров поля со списком

В Visual Basic .NET поля со списком могут быть заданы в одном из трех стилей (рис. 9.3).

- ✓ Simple (Простой). Отображает список элементов, который всегда открыт для просмотра. В отличие от двух других стилей, поля со списком, отображенные в стиле **Simple**, в процессе выполнения программы не изменяют своей высоты, по-

этому вам нужно будет самостоятельно определить высоту объекта, при которой пользователям будет удобно просматривать список возможных вариантов.

- ✓ DropDown (Раскрывающийся). Вначале на экране отображается значение, назначенное для свойства Text. Если пользователь щелкает на кнопке со стрелкой, на экране появляется список готовых вариантов.
- ✓ DropDownList (Раскрывающийся список). Вначале отображается пустое поле (вне зависимости от того, присвоено ли какое-то значение свойству Text). Когда пользователь щелкает на кнопке со стрелкой, на экране отображается список готовых вариантов.

В отличие от двух других стилей, в поле, отображаемом в стиле DropDownList, пользователь не может набирать свои варианты ответа.

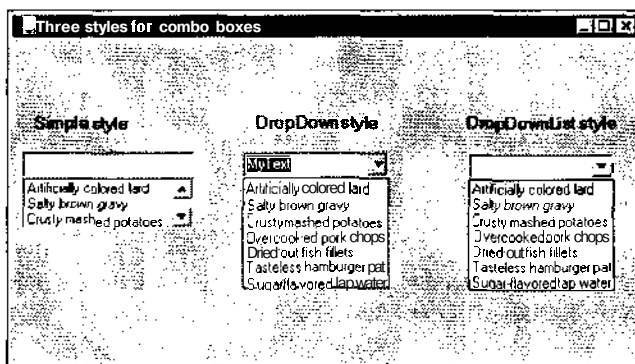


Рис. 9.3. Три стиля отображения полей со списком

### Тест на проверку полученных вами знаний

1. Какое главное различие между списком и полем со списком?
  - а. В поле со списком пользователь может указать свой вариант ответа, а в списке он может только выбрать один из предложенных вариантов.
  - б. Первый объект **состоит** ИЗ ОДНОЙ части (список), второй - из двух частей (поле плюс список).
  - в. Кажется, это как-то связано с обслуживанием в ресторанах. Наверное, в поле со списком варианты **"ловкуснее"**.
  - г. Никакого различия не **существует**. Не зря же они описываются в одной главе.
2. Почему помимо флажков и переключателей для предоставления выбора используются списки и поля со списком?
  - а. Для разнообразия.
  - б. Списки И ПОЛЯ СО СПИСКОМ занимают на экране меньше места; к **тому** же в поле со списком пользователь может ввести собственный вариант ответа.
  - в. Насчет списков не знаю, а поля со списком просто симпатичней. Они так **"прикольно"** раскрываются и **закрываются**.
  - г. Выбор - дело серьезное. Если пользователя не устраивают варианты, предлагаемые посредством флажков и переключателей, он может поискать что-нибудь в списках.

Чтобы указать, в каком стиле должно отображать поле со списком, необходимо выполнить такие действия.

1. Щелкните на поле со списком, которое вы хотите изменить.  
(Имеется в виду, что само поле вы уже создали.)
2. Откройте окно Properties.  
Чтобы сделать это, нажмите клавишу <F4>, выберите команду View⇒Properties Window, в окне Solution Explorer щелкните на значке Properties Window или щелкните правой кнопкой мыши в окне формы и выберите команду Properties.
3. В категории Appearance найдите свойство DropDownStyle (Стиль поля со списком) и щелкните на нем.  
Рядом с ним появится кнопка со стрелкой.
4. Щелкните на кнопке со стрелкой, чтобы отобразить список возможных вариантов.  
(Минуточку. А ведь само поле свойства тоже является полем со списком!)
5. Выберите стиль поля со списком, например Simple или DropDownList.  
Visual Basic .NET отобразит поле со списком в указанном вами стиле.

## *Наполнение списка и поля со списком элементами для выбора*

После того как список или поле со списком созданы, для них нужно определить набор элементов, из числа которых пользователь сможет делать свой выбор. (В противном случае эти объекты будут просто пустыми.) В Visual Basic .NET это можно сделать двумя способами:

- ✓ с использованием свойства Items (Элементы), относящегося к категории Data (Данные);
- ✓ посредством написания кодов BASIC.

Вот как можно добавить элементы, используя свойство Items.

1. Щелкните на списке или на поле со списком, к которому вы хотите добавить элементы.
2. Откройте окно Properties.  
Чтобы сделать это, нажмите клавишу <F4>, выберите команду View⇒Properties Window в окне Solution Explorer, щелкните на значке Properties Window или щелкните правой кнопкой мыши в окне формы и выберите команду Properties.
3. Щелкните на свойстве Items (категория Data).  
Рядом с ним появится кнопка с тремя точками.
4. Щелкните на кнопке с тремя точками.  
Откроется диалоговое окно String Collection Editor, показанное на рис. 9.4.
5. Наберите значение элемента, который должен присутствовать в списке для выбора, и нажмите клавишу <Enter>.
6. Повторяйте шаг 5 до тех пор, пока не введете значения всех элементов, из числа которых пользователь будет делать свой выбор.
7. Щелкните на кнопке ОК.

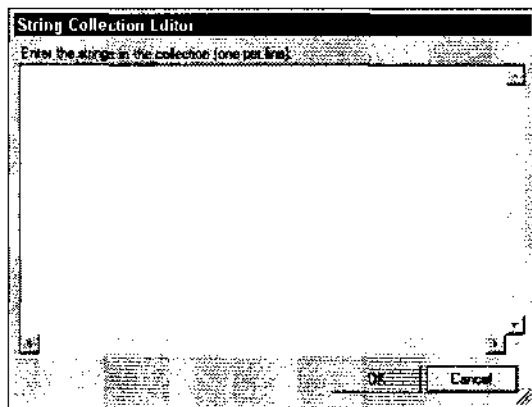


Рис. 9.4. В диалоговом окне String Collection Editor можно определить набор элементов, из числа которых пользователь сможет сделать свой выбор

Если вы захотите как-то изменить список доступных пользователю вариантов ответа в процессе выполнения программы, вам нужно будет написать коды BASIC. Чтобы добавить к набору элементов списка или поля со списком еще один элемент (который будет отображаться ниже доступных ранее элементов), вам достаточно всего лишь написать следующий код BASIC:

```
BoxName.Items.Add ("Add me")
```



Вот какие действия совершаются в процессе выполнения этого кода.

1. Слово `BoxName` говорит Visual Basic .NET о необходимости найти список или поле со списком, именуемое `BoxName`. (Разумеется, если вы хотите добавить элемент к своему списку или полю со списком, вам нужно в этом месте указать именно его имя.)
2. Команда `Items.Add` говорит Visual Basic .NET, что нужно подготовиться к добавлению нового элемента к объекту, именуемому `BoxName`.
3. Код ("Add me") указывает Visual Basic .NET на необходимость добавления строки Add me (Добавь меня) в нижнюю часть списка элементов для выбора, принадлежащего объекту `BoxName`. (В своей программе вы можете ввести любую строку, также заключенную в кавычки.)



Обычно команда `Items.Add` добавляет новый элемент в нижнюю часть уже существующего списка возможных вариантов. Однако если вы присвоите свойству **Sorted** (Сортировать) категории `Behavior` значение `True`, все элементы будут отсортированы в алфавитном порядке и новый элемент займет свое место в соответствии с этим порядком. Более подробно о сортировке элементов читайте в разделе "Сортировка элементов списков и полей со списком".

Если вы хотите вставить новый элемент в какую-то определенную позицию в списке, воспользуйтесь вместо команды `Add` командой `Insert`:

```
BoxName.Items.Insert (X, "Add me")
```



Вот как работает этот код BASIC.

1. Слово `BoxName` указывает Visual Basic .NET, что нужно найти список или поле со списком, именуемое `BoxName`. (Если вы хотите добавить элемент к своему списку или полю со списком, вам нужно в этом месте указать соответствующее имя.)

2. Команда `Items.Insert` сообщает Visual Basic .NET о необходимости подготовиться к добавлению нового элемента в определенную позицию списка элементов, принадлежащего объекту `BoxName`.
3. Код (`X, "Add me"`) дает указание Visual Basic .NET добавить строку `Add me` в позицию номер `X`. Первый элемент в списке имеет позицию номер 0, второй — позицию номер 1 и т.д. Если вы хотите, чтобы строка `Add me` была второй в списке элементов, вам нужно набрать (`1, "Add me"`), поскольку именно вторая позиция в списке обозначается номером 1.

## Сортировка элементов списков и полей со списком

Порядок, в котором вы добавляете элементы в список доступных для выбора вариантов, сохранится и при отображении данного списка на экране. Но при необходимости вы можете отсортировать список элементов в алфавитном порядке.

Когда происходит сортировка элементов по алфавиту, регистр не учитывается. Например, для Visual Basic .NET элементы "Варяг" и "ВАРЯГ" являются абсолютно равнозначными.

Для того чтобы отсортировать элементы по алфавиту, необходимо выполнить следующие действия.

### **J. Щелкните на списке или на поле со списком, элементы которого хотите отсортировать.**

2. Откройте окно `Properties`.

Чтобы сделать это, нажмите клавишу `<F4>`, выберите команду `View⇒Properties Window`, в окне `Solution Explorer` щелкните на значке `Properties Window` или щелкните правой кнопкой мыши в окне формы и выберите команду `Properties`.

3. Щелкните на свойстве `Sorted` (категория `Behavior`).

4. Щелкните на кнопке со стрелкой и выберите значение `True` (или `False`, если хотите отменить сортировку).

Если выбрано значение `True`, элементы списка автоматически будут отсортированы в алфавитном порядке.

Visual Basic .NET позволяет сортировать элементы в порядке от А до Z. (Если элементы состоят из цифр, то для них порядок сортировки будет от 0 до 9. Если элементами являются и числа и текстовые строки, то в начале списка будут идти отсортированные числа, а за ними отсортированные строки.) Элементы нельзя отсортировать в обратном порядке, т.е. от Z до А (только разве что перевернув монитор вверх ногами).

Если алфавитный порядок сортировки вас не устраивает, можете отсортировать элементы вручную в окне `Sorting Collection Editor` или указав точное расположение в списке каждого элемента с помощью команды `Items.Insert`.

## Удаление элементов из списка

Добавлять и сортировать элементы — это не единственное, что вы должны уметь делать для составления хороших списков. Не исключено, что может возникнуть необходимость удалить лишние или случайно внесенные элементы либо те элементы, которые уже потеряли свою актуальность, — в общем, просто что-то удалить.

В Visual Basic .NET это можно сделать двумя путями.

1. Во-первых, с использованием команды `Items.RemoveAt`, указывающей, какой именно элемент нужно удалить.

- ✓ Во-вторых, с помощью команды `Items.Clear`, которая позволяет удалить весь список сразу.

Вы не можете использовать команду `Items.RemoveAt`, не зная позицию элемента, подлежащего удалению. Например, чтобы удалить третий элемент списка, именуемого `lstToDo`, нужно написать следующий код BASIC:

```
LstToDo.Items.RemoveAt (2)
```



Первый элемент занимает нулевую позицию (0), второй — первую (1), третий — вторую (2) и т.д.



Если вы хотите удалить выделенные на данный момент элементы, напишите такой код: `ComboBox1.Items.RemoveAt (ComboBox1.SelectedIndex)`

Если вы собираетесь посредством команды `Items.Clear` удалить сразу весь набор элементов списка или поля со списком, достаточно указать имя этого объекта. Например, чтобы очистить весь список элементов поля со списком, носящего название `cbHideIn`, воспользуйтесь таким кодом BASIC:

```
CbHideIn.Items.Clear ()
```



Прежде чем применить команду `Clear`, подумайте, действительно ли нужно удалять весь список элементов, и убедитесь, что это именно тот список.

## Сделайте список удобным

Чтобы немного оживить список и выделить его из общей серой массы безликих потоков информации, попробуйте изменить его шрифт, начертание и размер.

Обычно Visual Basic .NET использует для отображения текста шрифт MS Sans Serif, но вы можете назначить для этого любой другой имеющийся у вас шрифт. (Шрифт MS Sans Serif похож на шрифт Helvetica, а используемый Visual Basic .NET шрифт MS Serif похож на Times Roman).

Вот как можно изменить шрифт, используемый для отображения элементов списков и полей со списком.

1. Щелкните на списке или на поле со списком, параметры которых нужно изменить.
2. Откройте окно **Properties**.  
Чтобы сделать это, нажмите клавишу <F4>, выберите команду **View⇌Properties Window**, в окне Solution Explorer щелкните на значке **Properties Window** или щелкните правой кнопкой мыши в окне формы и выберите команду **Properties**.
3. Щелкните на свойстве **Font (Шрифт)** категории **Appearance**.  
Рядом с ним появится кнопка с тремя точками.
4. Щелкните на кнопке с тремя точками.  
На экране будет отображено диалоговое окно **Font**.
5. Определите шрифт, начертание, размер текста, наличие дополнительных эффектов, например **strikeout** (зачеркивание), и щелкните на кнопке **OK**.





Все хорошо в меру. Однако новички часто настолько увлекаются изменением шрифтов и используют их в таком количестве, что это переходит все разумные пределы. Если у вас нет действительно уважительной причины для изменения шрифтов, начертания, размера текста, оставьте установки, используемые Visual Basic .NET по умолчанию.

Чем более примечательным вы сделаете свой список или поле со списком, тем быстрее пользователь обратит на него внимание. Однако помните, что вы создаете удобную для пользователей программу, а не шедевр живописного искусства. Если вам хочется творить прекрасное, возьмите мольберт и выйдите на природу. Если же вы хотите создать полезную программу и заработать на этом миллион долларов, сделайте ее простой, понятной и удобной в использовании.

# Настройка отображения пользовательского интерфейса

*В этой главе...*

- Установка одинаковых размеров для нескольких объектов
- Выравнивание объектов
- Установка одинаковых интервалов между объектами
- Фиксирование положения объектов

**П**осле того как вы создали форму и разместили в ней некоторое количество объектов (например, кнопки, текстовые поля, переключатели), можете упорядочить их отображение, используя следующие возможности Visual Basic .NET:

- ✓ придавать сразу нескольким объектам одинаковые размеры;
- ✓ выравнивать объекты относительно друг друга;
- ✓ размещать объекты на одинаковом расстоянии друг от друга;
- ✓ фиксировать текущие позиции и размеры объектов.

## *Установка одинаковых размеров для нескольких объектов*

Если вы создали некоторое количество подобных объектов, например группу флажков, переключателей или кнопок, и хотите, чтобы все они имели одинаковые размеры, поручите эту работу Visual Basic .NET. Он сделает это все автоматически.

От вас лишь потребуется выполнить следующие действия.

1. Щелкните на первом объекте, который должен стать такого же размера, как и другие объекты.
2. Удерживая нажатой клавишу <Ctrl>, поочередно выполните щелчки на всех остальных объектах, которые должны иметь одинаковые размеры.

Visual Basic .NET выделит все указанные вами объекты.

3. Щелкните на объекте, размеры которого послужат образцом для всех остальных, выделенных на первом и втором шаге, объектов.

Объект, который послужит образцом для других, должен быть выделен последним. В таком случае его размеры останутся прежними, а все остальные объекты примут его параметры.

4. Задайте команду **Format⇒Make Same Size** (Формат⇒Одинаковый размер), а затем выберите одну из следующих опций:

- Width (Ширина) — ширина всех объектов станет одинаковой;

- Size to Grid (Размер по сетке) — размеры всех объектов будут выровнены по линиям сетки, отображаемым в окне формы;
- Height (Высота) — высота всех объектов станет одинаковой;
- Both (Оба параметра) — высота и ширина всех объектов станут одинаковыми.

Как только вы выберете одну из этих опций, размеры выделенных объектов сразу же будут изменены.



Если вам вдруг не понравятся проведенные преобразования, отмените их нажатием комбинации клавиш <Ctrl+Z> или щелкните на кнопке Undo (Отменить) стандартной панели инструментов.

## Выравнивание объектов

Большое количество произвольно расположенных элементов может создавать впечатление хаоса, царящего в окне формы. Чтобы сделать общую картину более приглядной, выровняйте объекты одним из следующих способов (рис. 10.1).

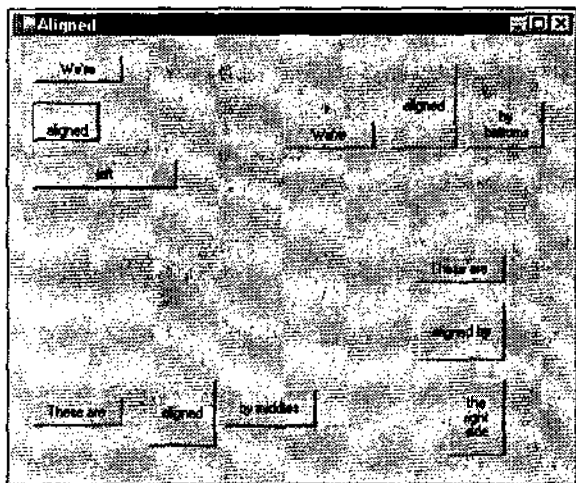


Рис. 10.1. Различные способы выравнивания объектов

- ✓ Lefts (По левому краю) — местоположение всех объектов выравнивается по левому краю;
- ✓ Centers (По центру) — через середину каждого объекта проходит условная вертикальная линия;
- ✓ Rights (По правому краю) — местоположение всех объектов выравнивается по правому краю;
- ✓ Tops (По верхнему краю) — местоположение всех объектов выравнивается по их верхнему краю;
- ✓ Middles (По середине) — середины каждого объекта выравниваются вдоль горизонтальной линии;
- ✓ Bottoms (По нижнему краю) — местоположение всех объектов выравнивается по их нижнему краю;

- ✓ **To Grid (По линиям сетки)** — все объекты выравниваются по линиям сетки, отображаемым в окне формы.



Вы можете изменить интервалы между линиями сетки, щелкнув в окне формы (но не на каком-либо ее объекте). Откроется окно Properties, где в категории Design вы можете изменить значения свойств GridSize Width (Ширина сетки) и GridSize Height (Высота сетки). Чем больше значение вы зададите, тем большее расстояние будет между линиями сетки.

При необходимости выровнять несколько объектов, выполните следующие действия.

1. Удерживая нажатой клавишу **<Ctrl>**, **поочередно щелкните на всех объектах**, которые хотите выделить.

Visual Basic .NET выделит каждый выбранный вами объект.

2. **Щелкните на объекте, положение которого должно быть ориентиром для всех остальных объектов.**

Объект, выделенный последним, останется на месте, а все остальные будут выровнены с учетом его положения.

3. Активируйте команду **Format⇒Align (Формат⇒ Выровнять)**, а затем выберите одну из доступных опций, например **Left** или **Middle**.

Visual Basic .NET выровняет выделенные вами объекты по указанному объекту.



Вы всегда можете отменить внесенные изменения, нажав комбинацию клавиш **<Ctrl+Z>** или щелкнув на кнопке Undo (Отменить) стандартной панели инструментов.

## Определение расстояния между объектами

Упорядочить местоположение объектов в окне формы можно также путем определения точного расстояния между ними.

Вот как это делается.

1. Удерживая нажатой клавишу **<Ctrl>**, **выполните поочередные щелчки на всех объектах**, местоположение которых вы **хотите выровнять, определив расстояние между ними**.

Visual Basic .NET выделит все указанные вами объекты.

2. Щелкните **на объекте, с учетом месторасположения которого должны быть выровнены остальные выделенные объекты**.

Объект, выделенный последним, останется на своем месте, в то время как все другие объекты будут перемещены.

3. Задайте команду **Format⇒Horizontal Spacing/Vertical Spacing (Формат⇒ Горизонтальное расстояние/Вертикальное расстояние)**, а затем выберите один из следующих вариантов.

- **Make Equal (Сделать равным)**— объекты размещаются таким образом, что вертикальные или горизонтальные промежутки между ними становятся одинаковыми;
- **Increase (Увеличить)**— все выделенные объекты перемещаются в сторону от объекта, выделенного на шаге 2;

- Decrease (Уменьшить) — все выделенные объекты размещаются притык к объекту, выделенному на шаге 2;
- Remove (Передвинуть) — Все выделенные объекты размещаются вплотную друг к другу.



Если вам не понравятся выполненные Visual Basic .NET перемещения объектов, отмените их, нажав комбинацию клавиш <Ctrl+Z> или щелкнув на кнопке Undo (Отменить) стандартной панели инструментов.

### Тест на проверку полученных вами знаний

1. Зачем нужно выравнивать объекты?
  - а. Чтобы у пользователей глаза не разбегались в разные стороны.
  - б. Это позволит создать ужасный и беспорядочный пользовательский интерфейс.
  - в. Таким образом можно создать аккуратный и упорядоченный интерфейс.
  - г. Минуточку, я, кажется, уже выбрал вариант в.
2. Когда вы выделяете группу объектов, желая их выровнять или изменить их размеры, какой из объектов Visual Basic .NET в любом случае оставит без изменений?
  - а. Тот, который был выделен последним.
  - б. Тот, который "ровнее" всех остальных.
  - в. Это непредсказуемо. Как Visual Basic .NET решит, так и будет.
  - г. Обычно Visual Basic .NET меняет все объекты, причем до неузнаваемости.

## Выравнивание по центру

Иногда, чтобы придать интерфейсу более эстетический вид, целесообразно разместить один или несколько объектов точно по центру формы. К счастью, вам необязательно делать это вручную, Visual Basic .NET может сам выровнять объекты как по горизонтали, так и по вертикали.

Чтобы выровнять объекты по центру, сделайте следующее.

1. Щелкните на одном или нескольких объектах, которые должны быть выровнены.

Чтобы выделить сразу несколько объектов, поочередно щелкайте на них, удерживая нажатой клавишу <Ctrl>.

2. Выберите команду Format⇒Center in Form (Формат⇒По центру формы), а затем сделайте установку Horizontally или Vertically, а возможно, и обе сразу.

Visual Basic .NET разместит выделенные объекты точно по центру формы.

## Фиксация положения объектов

Когда вы фиксируете положение объектов, вы как бы приклеиваете их к определенному месту в окне формы, с тем чтобы в дальнейшем никто другой не мог случайно или намеренно переместить их в другое место. Этот прием особенно полезен в том случае, когда над созданием программы работают сразу несколько программистов. Если один из них зафиксирует положения объектов, другие уже не смогут просто так испортить его работу.



Зафиксированные объекты нельзя переместить на другое место, но их по-прежнему можно удалить, можно изменить любые их свойства.

## Фиксирование положения всех объектов в окне формы

Для того чтобы зафиксировать положение и размеры всех объектов, созданных в какой-то форме, необходимо выполнить такие действия.

1. Щелкните в окне формы на любом объекте (например, на текстовом поле),
2. Выберите команду **Format⇒Lock Controls** (**Формат⇒Заблокировать**).

Visual Basic .NET зафиксирует положение всех объектов, принадлежащих этой форме.

Теперь, если вы щелкнете на каком-нибудь из таких объектов, Visual Basic .NET отобразит вокруг него белые границы (рис. 10.2), и это будет означать, что положение объекта зафиксировано и его нельзя перемещать или изменять его размеры.

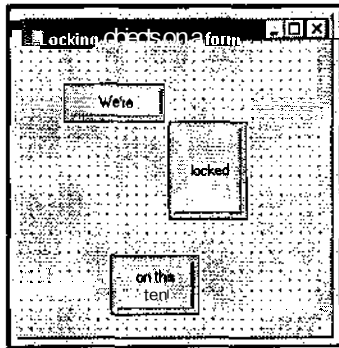


Рис. 10.2. Привыделении в окне формы зафиксированных объектов вокруг них отображаются белые границы



Чтобы снять фиксацию со всех объектов, еще раз выберите команду **Format⇒Lock Controls**.

## Фиксирование положения отдельных объектов

Далеко не всегда возникает необходимость в фиксировании положения сразу всех объектов в окне формы. Вы можете зафиксировать положение лишь одного или нескольких объектов, с тем чтобы оставить за собой возможность перемещать и изменять размеры всех остальных объектов. Вот что нужно сделать, чтобы зафиксировать положение только одного объекта.

1. Щелкните на объекте, положение которого вы хотите зафиксировать.
2. Откройте окно Properties.

Чтобы сделать это, нажмите клавишу <F4>, выберите команду **View⇒Properties Window**, в окне Solution Explorer щелкните на значке Properties Window или щелкните правой кнопкой мышки в окне формы и выберите команду Properties.

**3. Щелкните на свойстве Locked (Зафиксировать) категории Design.**

Рядом с ним появится кнопка с направленной вниз стрелкой.

**4. Щелкните на кнопке со стрелкой и выберите значение True (Истина).**

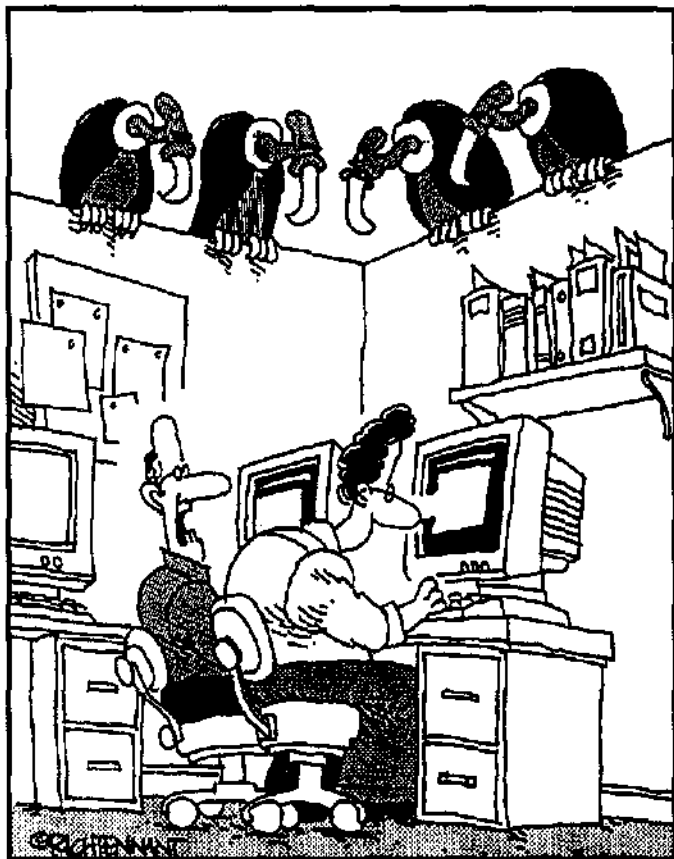
Теперь при выделении этого объекта вокруг него будут отображаться белые границы, информирующие о том, что объект зафиксирован.



Чтобы разблокировать данный объект, повторите все те же действия, только вместо значения True выберите значение False.

## Часть III

# Создание меню



У меня иногда чувство, что этот проект в любую минуту может приказать всем голою жить



### *В этой части...*

Раскрывающиеся меню являются прекрасным средством для организации доступа ко всем опциям, предлагаемым программой. Пользователю, чтобы заставить программу что-то сделать, нужно будет просто выбрать соответствующую команду из меню.

Эта часть книги посвящена описанию процесса разработки раскрывающихся меню для ваших программ. Очень скоро вы сможете убедиться, что в этом нет ничего сложного. (Настоящие трудности могут возникнуть **тогда, когда вы** приступите к разработке той части программы, которая отвечает за выполнение реальных действий и подучение конкретных результатов. На этом этапе часто "спотыкаются" даже самые опытные программисты.)

# Разработка раскрывающихся меню

*В этой главе...*

- Создание меню
- Назначение имен
- Настройка меню

**Р**аскрывающиеся меню позволяют разбивать по группам и компактно отображать наборы команд, выполняющих подобные действия. Благодаря этому вам не нужно будет засорять пользовательский интерфейс лишними наборами кнопок, флажков, переключателей, дополнительными панелями инструментов и т.д.

## Основные компоненты строки меню

В заголовках меню большинства программ наиболее часто встречаются такие названия, как File, Edit, Window и Help.

Меню **File** (Файл) обычно содержит команды, имеющие прямое отношение к операциям с файлами (рис. 11.1). В число таковых входят операции открытия, закрытия, сохранения, печать файлов; здесь же находится и команда выхода из программы, позволяющая оторваться от компьютера и пойти чего-нибудь перекусить.

Меню **Edit** (Правка), как правило, объединяет команды, относящиеся к процессу редактирования, например такие, как Undo (Отменить), Redo (Повторить), Cut (Вырезать), Copy (Копировать), Paste (Вставить), Clear (Очистить) и Select All (Выделить все).

Меню **Window** (Окно) обычно содержит команды, касающиеся открытия, закрытия, выравнивания окон, а также команды переключения между разными окнами.



Рис. 11.1. Пример раскрывающегося меню, созданного с помощью Visual Basic.NET



Меню Window присутствует только в тех программах, которые позволяют открывать сразу несколько окон (в частности, в текстовых процессорах). Если же программа работает лишь с одним окном, в меню Window просто нет необходимости.

В меню Help (Справка) вы найдете команды, с помощью которых можно получить справочную информацию о самой программе.

Обычно заголовки меню File и Edit расположены рядом друг с другом, а заголовки меню с командами, которые вы вряд ли найдете в других программах, расположены между заголовками меню Edit и Window. Например, в большинстве текстовых процессоров есть меню Tools (Сервис), где собраны команды наподобие Тезаурус, Расстановка переносов, Макрос и некоторые другие, о которых 99 процентов населения планеты не имеют ни малейшего представления.



Если вы создаете меню, заголовки которых вряд ли можно встретить в интерфейсах других программ (иными словами, если это не Edit, Window и не какие-либо другие общеизвестные заголовки меню), постарайтесь сделать так, чтобы ваши заголовки были максимально информативны и чтобы пользователи хотя бы приблизительно представляли, какие команды они смогут найти в этом меню.

## Создание меню для интерфейса вашей программы

Чтобы создать раскрывающееся меню, сделайте следующее.

1. Щелкните на форме, в окне которой должно быть создано раскрывающееся меню.
2. На панели ToolBox дважды щелкните на кнопке инструмента MainMenu (Главное меню).

Visual Basic .NET создаст в окне формы пустое меню и отобразит в нижней части экрана отдельное окно со значком MainMenu1 (рис. 11.2).

3. В окне формы щелкните в поле, где отображаются слова Type Here (Наберите здесь) и введите название заголовка меню, например File или Формат.

Visual Basic .NET отобразит введенный вами текст как заголовок меню или как название команды меню, и создаст рядом еще одно поле Type Here для ввода нового заголовка меню или команды меню.

4. Повторите шаг 3 столько раз, сколько заголовков и команд меню вы хотите создать.



Чтобы иметь возможность получить доступ к меню посредством нажатия "горячей клавиши", перед буквой, которая должна быть подчеркнута, наберите символ &. Например, чтобы создать типичное меню File (с подчеркнутой буквой!'), наберите &File.



Если вы щелкнете на команде созданного вами раскрывающегося меню, ничего не произойдет, поскольку вы еще не написали коды BASIC, объясняющие программе, что нужно делать при выборе какой-либо команды меню.

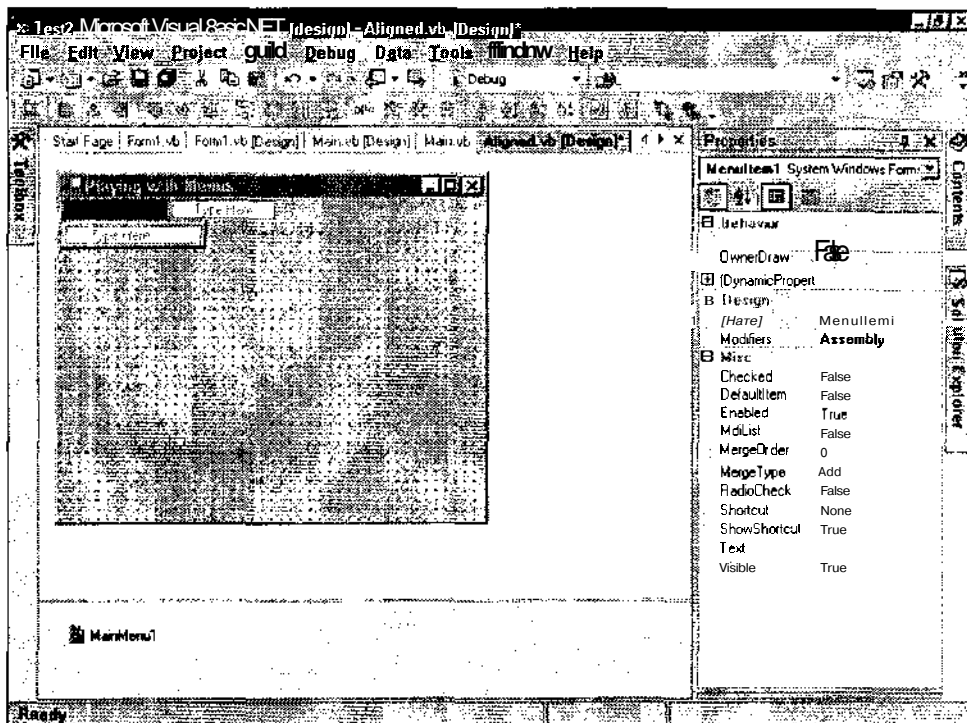


Рис. 11.2. Вот как выглядит только что созданное открывающееся меню

## Добавление и удаление заголовков меню и команд меню

К созданному набору раскрывающихся меню со временем вам, возможно, потребуется добавить еще парочку пунктов или, наоборот, что-нибудь удалить. В первом случае ваши действия должны быть таковыми.

1. Откройте форму, содержащую меню, которые вы хотите отредактировать.
2. Щелкните на нужном раскрывающемся меню.  
Если вы щелкнете на заголовке меню, Visual Basic .NET откроет само меню.
3. Далее выберите одно из двух действий в зависимости от того, что вы намерены сделать.
  - Чтобы добавить новый заголовок меню, щелкните на уже созданном заголовке. Visual Basic .NET разместит новый заголовок меню слева от заголовка, на котором вы щелкнули.
  - Чтобы создать новую команду меню, щелкните на уже существующей команде. Новая команда будет размещена над командой, на которой вы щелкнули.
4. Нажмите клавишу <Insert>.

Visual Basic .NET вставит в строку меню новый заголовок меню или новую команду в раскрывающееся меню.

Чтобы удалить заголовок меню или команду меню, сделайте следующее.

1. Откройте форму, содержащую меню, которое вы хотите отредактировать.
2. Щелкните на нужном раскрывающемся меню.

Если вы щелкнете на заголовке меню, Visual Basic .NET откроет само меню.

### 3. Нажмите клавишу <Delete>.

Если выбранный вами элемент содержит в себе команды меню, на экране появится диалоговое окно с вопросом, действительно ли вы хотите удалить этот элемент.

### 4. Щелкните на кнопке Yes.



Когда вы удаляете заголовок меню или команду меню, все коды BASIC, которые были написаны для этого элемента, остаются в тексте программы, поэтому их тоже нужно найти и удалить.

## Перемещение заголовков и команд меню

Вместо того чтобы удалять раскрывающиеся меню и их команды, иногда разумнее просто переместить их в другое место. Вот как это делается.

### 1. Откройте форму, содержащую раскрывающиеся меню, которые нужно отредактировать.

### 2. Далее выберите один из двух следующих вариантов действий.

- Щелкните на заголовке меню, которое вы хотите переместить. Когда вы перемещаете заголовок меню, все присоединенные к нему команды перемещаются вместе с ним.
- Щелкните на заголовке меню, в котором содержится команда, которую вы хотите переместить. Затем щелкните на самой команде.

### 3. Нажмите левую кнопку мыши и перетащите заголовок или команду меню на новую позицию.

Заголовок меню можно переместить таким образом, чтобы он сам стал командой в другом раскрывающемся меню.

### 4. Отпустите левую кнопку мыши, когда заголовок или команда меню достигнут нужной позиции.

## Присвоение меню имен

Всем заголовкам и командам меню необходимо присвоить имена, с тем чтобы в дальнейшем их можно было идентифицировать. Имена не должны содержать пробелов и знаков пунктуации.

Хорошей практикой является использование в названиях заголовков и команд меню префиксов `mnu`, например:

```
✓ mnuFile;  
✓ mnuWindow;  
✓ mnuFileOpen.
```

Для Visual Basic .NET нет никакой разницы, состоит имя из заглавных букв или из прописных. Если вам это действительно необходимо, можете использовать и имена наподобие таких:

```
✓ MNufile;  
✓ mNuWiNDow;  
✓ MNUfileOPEN.
```

Правда, подобные имена не просто трудны для чтения — они могут создать вам репутацию неграмотного или несерьезного человека. Поэтому не создавайте себе лишних проблем и придерживайтесь наработанных общих стандартов.



Чтобы легче было идентифицировать команду, расположенную под каким-то заголовком меню, включите в имя этой команды название заголовка. Например, если меню File имеет название `mnuFile`, присвойте командам `Open`, `Save`, `Exit` этого меню имена `mnuFileOpen`, `mnuFileSave`, `mnuFileExit`.

Для того чтобы присвоить имя заголовку или команде меню, необходимо сделать следующее.

1. **Откройте форму, содержащую раскрывающиеся меню, которые нужно отредактировать.**
2. **Далее выберите один из следующих вариантов.**
  - Щелкните на заголовке меню, которому вы хотите присвоить имя.
  - Щелкните на заголовке меню, содержащем команду, которой вы хотите дать имя. Затем щелкните на самой команде.Visual Basic .NET выделит выбранный вами элемент.
3. **Откройте окно `Properties`.**

Чтобы сделать это, нажмите клавишу `<F4>`, выберите команду `View⇒Properties Window`, в окне `Solution Explorer` щелкните на кнопке **Properties Window** или щелкните правой кнопкой мыши в окне формы и выберите команду `Properties`.
4. **Дважды щелкните на свойстве `Name (Имя)` категории `Design`.**

Visual Basic .NET выделит текущее значение этого свойства.
5. **Наберите новое имя для заголовка или команды меню.**

## Настройка меню

Существует несколько приемов, способных сделать меню более удобными в использовании. Так, с помощью разделительных линий команды одного меню можно разбить на отдельные группы, можно отображать напротив активных в данный момент команд флажки, назначать командам комбинации клавиш для быстрого доступа, выделять серым или убирать с экрана недоступные на данный момент команды.

### Добавление в меню разделительных линий

Разделительные линии визуально делят команды раскрывающегося меню на отдельные группы (рис. 11.3). Если в такие группы объединены команды, выполняющие аналогичные действия, это облегчает пользователям поиск и выбор нужных на данный момент команд.

Вот как можно добавить в меню разделительную линию, сделайте следующее.

1. **Откройте форму, содержащую раскрывающиеся меню, которые нужно отредактировать.**
2. **Щелкните на заголовке меню, к которому вы хотите добавить разделительную линию.**

Visual Basic .NET откроет выбранное вами меню.
3. **Щелкните на команде, над которой должна располагаться разделительная линия.**
4. **Нажмите клавишу `<Insert>`.**

Visual Basic .NET отобразит пустое поле прямо над выбранной вами командой меню.
5. **Щелкните в пустом поле и наберите тире `-`.**

Visual Basic .NET в качестве команды меню отобразит тире. В следующий раз, когда вы запустите свою программу, тире волшебным образом превратится в разделительную линию.

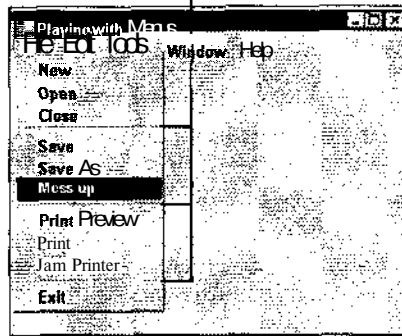


Рис. 11.3. Разделительные линии облегчают пользователю поиск нужных команд

## Тест на проверку полученных вами знаний

1. Чем так полезны раскрывающиеся меню?
  - а. Они создают у пользователей иллюзию то-го, что данная программа очень серьезная.
  - б. С их помощью можно разбить на группы и компактно разместить команды, выполняющие подобные действия, что делает использование программы намного более удобным.
  - в. Тем, что в них всегда можно найти команду Help и команду Exit.
  - г. Если они такие полезные, то почему люди до сих пор должны изучать 500-страничные "Руководства пользователей"?
2. Как можно создать раскрывающееся меню для программ Visual Basic .NET?
  - а. Необходимо дважды щелкнуть в панели Toolbox на кнопке MainMenu.
  - б. Чтобы начать создавать раскрывающиеся меню, нужно пройти 4-годичный курс обучения языкам C++ и Java.
  - в. Нужно нарисовать 3 окна формы меню и поискать на панели инструментов кнопку Раскрыть/Заккрыть.
  - г. А что, Visual Basic .NET действительно может создавать раскрывающиеся меню?

## Назначение комбинаций клавиш

Наиболее часто используемым командам можно назначить *комбинации клавиш*, которые будут предоставлять пользователям возможность быстрого доступа к этим командам без необходимости поиска таковых во всех раскрывающихся меню. Например, команде Save (Сохранить) обычно назначается комбинация клавиш <Ctrl+S>, команде Copy (Копировать) — комбинация <Ctrl+C> и т.д. Некоторые из команд, которым назначены комбинации клавиш, показаны на рис. 11.4.

Назначенные командам комбинации клавиш отображаются в раскрывающихся меню рядом с этими командами. Благодаря этому пользователи могут достаточно быстро запомнить комбинации клавиш, соответствующие командам, которыми они чаще всего пользуются.

Чтобы назначить команде меню комбинацию клавиш, вы должны выполнить такие действия.

1. Откройте форму, раскрывающиеся меню которой вы хотите отредактировать.
2. Щелкните на заголовке меню, в котором содержится нужная команда.  
Visual Basic .NET откроет данное меню.



Рис. 11.4. Комбинации клавиш позволяют получить быстрый доступ к наиболее часто используемым командам

3. Щелкните на команде, которой вы хотите назначить комбинацию клавиш.

4. Откройте окно Properties.

Чтобы сделать это, нажмите клавишу <F4>, выберите команду View⇌Properties Window, в окне Solution Explorer щелкните на кнопке Properties Window или щелкните правой кнопкой мыши в окне формы и выберите команду Properties.

5. Щелкните на свойстве Shortcut (Комбинация клавиш), которое расположено в категории Misc (Разное).

Рядом с ним появится кнопка с направленной вниз стрелкой.

6. Щелкните на кнопке со стрелкой.

Откроется выпадающий список, в котором вы увидите все возможные комбинации клавиш (рис. 11.5).

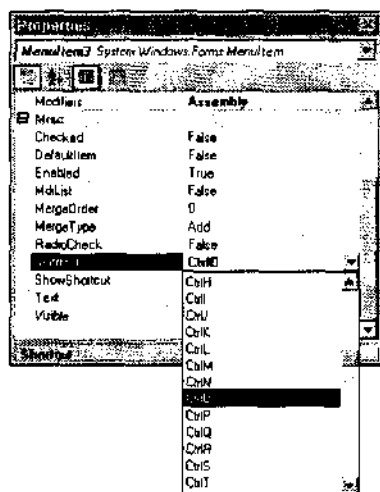


Рис. 11.5. Выберите для команды подходящую комбинацию клавиш

7. Щелкните на комбинации клавиш, которую вы хотели бы назначить данной команде.



Постарайтесь подобрать такую комбинацию клавиш, которая как-то ассоциировалась бы с командой или с ее названием (например, <Ctrl+S> для команды Save, <Ctrl+C> для команды Copy и т.д.).

Visual Basic .NET не позволит вам назначить одну и ту же комбинацию клавиш разным командам. Если вы попытаетесь это сделать, на экран будет выведено сообщение об ошибке.

Когда ваша программа будет запущена, указанные комбинации клавиш будут отображаться в меню напротив соответствующих им команд. Но пока вы не напишете для команд меню коды BASIC, объясняющие программе, что нужно делать, при выборе пользователем той или иной команды, нажатие комбинаций клавиш ни к чему не приведет.





Если вы назначите свойству ShowShortcut (Показывать комбинации клавиш), относящемуся к категории Misc, значение False, комбинации клавиш в меню отображаться не будут.

## Отображение флажков рядом с командами меню

Флажок, отображаемый рядом с командой меню, визуально демонстрирует тот факт, что эта команда на данный момент уже выбрана.



Флажки могут отображаться рядом только с командами меню, но никак не с заголовками меню.

Visual Basic .NET может отображать флажки двух видов: обычные и флажки-переключатели, которые выглядят как большие круглые точки.

Для того чтобы рядом с командой меню отображался флажок, необходимо сделать следующее.

1. Откройте форму, содержащую меню, которые нужно отредактировать.
2. Щелкните на заголовке меню, к командам которого нужно добавить флажки. Visual Basic .NET откроет нужное меню.
3. Щелкните на команде, рядом с которой должен отображаться флажок.
4. Откройте окно Properties.

Чтобы сделать это, нажмите клавишу <F4>, выберите команду View⇌Properties Window, в окне Solution Explorer щелкните на кнопке Properties Window или щелкните правой кнопкой мыши в окне формы и выберите команду Properties.

5. Щелкните на свойстве Checked (Проверка) категории Misc (Разное).

Рядом с ним появится кнопка с направленной вниз стрелкой.

6. Щелкните на кнопке со стрелкой и выберите значение True.

Теперь при каждом запуске вашей программы в раскрывающемся меню рядом с командой будет отображаться флажок.

А вот что нужно сделать, чтобы рядом с командой меню отображался флажок-переключатель.

1. Откройте форму, содержащую меню, которые нужно отредактировать.
2. Щелкните на заголовке меню, к командам которого нужно добавить флажки. Visual Basic .NET откроет раскрывающееся меню.
3. Щелкните на команде, рядом с которой должен отображаться флажок-переключатель.
4. Откройте окно Properties.

Чтобы сделать это, нажмите клавишу <F4>, выберите команду View⇌Properties Window, в окне Solution Explorer щелкните на кнопке Properties Window или щелкните правой кнопкой мыши в окне формы и выберите команду Properties.

5. Щелкните на свойстве Checked (Проверка), которое относится к категории Misc (Разное).

Рядом с ним появится кнопка с направленной вниз стрелкой.

6. Щелкните на кнопке со стрелкой и выберите значение True.

7. Щелкните на свойстве RadioCheck (Флажок-переключатель), которое также относится к категории Misc (Разное).

Рядом с ним появится кнопка с направленной вниз стрелкой.

Обратите внимание, что свойство **Checked** отвечает за отображение флажка, а свойство **RadioCheck** — за отображение флажка-переключателя.

## 8. Щелкните на кнопке со стрелкой и выберите значение True.

Теперь при каждом последующем запуске программы в раскрывающемся меню рядом с командой будет отображаться **флажок-переключатель**.

Если вы решили, что рядом с какой-то командой меню должен отображаться флажок, то, естественно, захотите, чтобы при наступлении определенных событий этот флажок исчезал. Что для этого нужно? Конечно же, написать соответствующие коды BASIC.

Чтобы флажок рядом с командой меню не отображался, нужно свойству **Checked** (или **RadioCheck**) присвоить значение False. Приведенная ниже команда удаляет изображение флажка, соответствующего команде `mnuFont12`:

```
mnuFont12.Checked = False
```

Чтобы отобразить флажок, используя коды BASIC, нужно написать команды, которые присваивали бы свойству **Checked** (или **RadioCheck**) значение True. Ниже приведен код, который включает отображение флажка-переключателя рядом с командой, именуемой `mnuFontHelvetica`:

```
mnuFontHelvetica.Checked = True
```

```
mnuFontHelvetica.RadioCheck = True
```



Если вы хотите отобразить рядом с командой меню флажок-переключатель, нужно присвоить значение True одновременно двум свойствам, **Checked** и **RadioCheck**. Чтобы отобразить обычный флажок, достаточно присвоить значение True только свойству **Checked**.

## Выделение серым команд меню

Иногда попытка использовать некоторые команды не приводит к положительному результату. Например, пока вы не выделите какую-либо текстовую информацию, нет смысла использовать команду **Cut** (Вырезать) или **Copy** (Копировать). Чтобы предостеречь пользователей от попытки применить недоступные на данный момент команды, можно выделить названия таковых серым (название команды в раскрывающемся меню будет отображаться серым цветом). И хотя сама команда по-прежнему будет находиться в меню, пользователь будет знать, что задействовать ее невозможно.

Чтобы выделить команду серым, вы должны выполнить такие действия.

1. **Откройте форму, содержащую меню, которые нужно отредактировать.**
2. **Щелкните на заголовке меню, где расположена команда, которую нужно выделить серым.**

Visual Basic .NET активизирует раскрывающееся меню.

3. **Щелкните на команде, которую вы хотите выделить серым.**
4. **Откройте окно Properties.**

Чтобы сделать это, нажмите клавишу <F4>, выберите команду **View⇒Properties Window**, в окне **Solution Explorer** щелкните на кнопке **Properties Window** или щелкните правой кнопкой мыши в окне формы и выберите команду **Properties**.

5. **Щелкните на свойстве Enabled (Доступная) категории Misc (Разное).**

Рядом с ним появится кнопка с направленной вниз стрелкой.

6. **Щелкните на кнопке со стрелкой и выберите значение False.**

При каждом следующем запуске программы эта команда будет выделяться серым.

Если команда выделена серым, то, естественно, в определенные моменты времени выделение с нее должно сниматься, т.е. команда должна снова становиться доступной. А для этого вам опять понадобится написать код BASIC. Чтобы снять с команды выделение серым, нужно ее свойству **Enabled** присвоить значение **True**. Ниже приведен код, который снимает выделение с команды, названной именем `mnuEditCut`:

```
MnuEditCut.Enabled = True
```

Чтобы команда стала недоступной в процессе выполнения программы, также нужно написать код BASIC, присваивающий ее свойству **Enabled** значение **False**. Вот код, который делает недоступной (выделяет ее серым) команду `mnuEditCopy`:

```
MnuEditCopy.Enabled = False
```

## Скрытие команд меню

Вы можете не только выделять команды меню серым, но и вовсе убирать их с экрана на какое-то время. Например, некоторые программы не отображают никаких заголовков меню, за исключением меню **File** и **Help**, до тех пор, пока пользователь не откроет или не создаст новый файл. (В самом деле, зачем отображать меню **Edit**, если редактировать пока нечего?)

При необходимости удалить элемент меню с экрана сделайте следующее.

1. **Откройте форму, содержащую меню, которые нужно отредактировать.**
2. **Щелкните на заголовке меню, где расположена команда, отображение которой нужно отменить.**

Visual Basic .NET активизирует раскрывающееся меню.

3. **Щелкните на команде, которую нужно убрать с экрана.**
4. **Откройте окно Properties.**

Чтобы сделать это, нажмите клавишу <F4>, выберите команду **View** → **Properties Window**, в окне **Solution Explorer** щелкните на кнопке **Properties Window** или щелкните правой кнопкой мыши в окне формы и выберите команду **Properties**.

5. **Щелкните на свойстве Visible (Видимый), относящемся к категории Misc (Разное).**

Рядом с ним появится кнопка с направленной вниз стрелкой.

6. **Щелкните на кнопке со стрелкой и выберите значение False.**

В следующий раз, когда вы запустите программу, этой команды в меню вы не найдете.

Если команда в определенный момент времени на экране отсутствует, то наверняка наступит время, когда ее нужно будет отобразить снова и сделать доступной. Для этого вам опять-таки понадобится написать код BASIC. Чтобы отобразить команду на экране, нужно ее свойству **Visible** присвоить значение **True**. Ниже приведен код, который возвращает на экран заголовок меню с именем `mnuEdit`:

```
MnuEdit.Visible = True
```

Чтобы убрать команду с экрана в процессе выполнения программы, также нужно написать код BASIC, присваивающий свойству **Visible** значение **False**. Ниже приведен код, который отменяет отображение заголовка меню, именуемого `mnuTools`:

```
MnuTools.Visible = False
```

В заключение еще раз напомним вам, что раскрывающиеся меню призваны облегчить работу с программой и придать ей более профессиональный вид. Поэтому не жалейте времени и усилий на создание качественного пользовательского интерфейса, действуйте для этого все доступные средства, и пользователи непременно скажут вам спасибо.

# Подменю, расширяемые меню и выпадающие меню

*В этой главе...*

- > Создание подменю
- Создание динамически расширяемых меню
- > Создание выпадающих меню

**К**оличество команд, которые можно разместить в раскрывающемся меню, всегда ограничивается высотой экрана. А что делать, если вы создаете суперпрограмму, в которой столько команд, что всех раскрывающихся меню просто не хватает для их размещения? По-видимому, вам придется заняться созданием подменю (или еще раз подумать, нужно ли создавать такую огромную программу).

## Создание подменю

Подменю — это одно раскрывающееся меню, вложенное в другое раскрывающееся меню. Таким образом можно создавать целую иерархию команд, размещая отдельные из них на разных уровнях вложенности. Например, в интерфейсе многих программ можно встретить меню **Format** (Формат), внутри которого расположены такие команды, как **TypeStyle** (Начертание), **Font** (Шрифт) и **Size** (Размер). Если вы щелкнете на команде **Font**, на экране отобразится подменю со списком всех доступных для использования шрифтов. Пример вложенных подменю приведен на рис. 12.1.

Visual Basic .NET позволяет создавать системы подменю, включающие до четырех уровней вложенности (рис. 12.1). Хотя использование такого количества вложенных подменю может показаться вам заманчивым, большинство программ применяют только один уровень вложенности. Опыт показывает, что обнаружить команду, расположенную на третьем или четвертом уровне вложенности, очень непросто.

Находящийся рядом с элементом меню треугольник, повернутый в виде стрелки, свидетельствует о том, что за этим элементом скрывается подменю. Когда вы создаете подменю, Visual Basic .NET добавляет отображение такой стрелки автоматически.

Создается подменю следующим образом.

- 1. Откройте форму, содержащую раскрывающееся меню, к которому вы хотите добавить подменю.**
- 2. Щелкните на заголовке раскрывающегося меню.**

Visual Basic .NET отобразит меню на экране.

- 3. Щелкните на команде, за которой должно раскрываться подменю.**

Справа от выбранной команды Visual Basic .NET отобразит поле **Type Here** (Наберите здесь).

4. Щелкните в поле **Type Here** и наберите название первой команды подменю.  
Visual Basic .NET отобразит поля **Type Here** справа и снизу под только что созданной командой подменю.
5. Повторяйте шаг 4 до тех пор, пока не наберете названия всех команд, которые должны отображаться в подменю.  
Если вы щелкнете в поле **Type Here**, расположенном справа от новой команды подменю, вы создадите новое подменю.

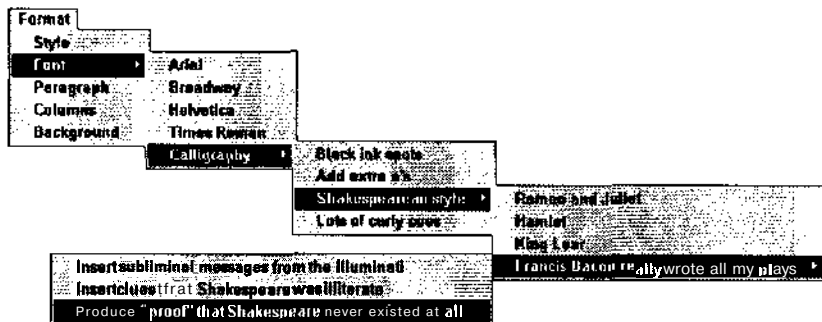


Рис. 12.1. Вы можете спрятать команду за несколькими уровнями подменю



Чем больше вы создадите подменю и чем сложнее будет их структура, тем труднее будет пользователям находить нужные команды и тем неудобнее будет ваша программа в использовании.



Вместо многоуровневой системы подменю многие программы используют для общения с пользователем диалоговые окна. (Об их создании подробно рассказывается в главе 13.) Диалоговые окна позволяют за один раз выбрать сразу несколько команд и установок, что значительно удобнее, чем блуждание по разным подменю в поиске лишь одной команды или установки.

## Изменение команд меню в процессе выполнения программы

Иногда возникают ситуации, когда команду меню нужно изменить в процессе выполнения программы. Так, в интерфейсе большинства программ имеется команда **Undo** (Отменить), которая почти всегда изменяется. После того, как пользователь выбирает команду **Undo**, таковая исчезает с экрана, а на ее месте появляется команда **Redo** (Повторить).

Чтобы команда меню изменялась в процессе выполнения программы, нужно написать соответствующий код BASIC, изменяющий для этой команды значение свойства **Text** (Текст). Приведенный ниже код изменяет для команды `mnuEditUndo` название на экране **Undo** на **Redo**:

```
MnuEditUndo.Text = "Redo"
```

Следующий код возвращает на экран название **Undo**:

```
MnuEditUndo.Text = "Undo"
```

# Создание динамически расширяемых меню

В интерфейсе большинства программ в меню File отображается список из четырех или пяти открываемых последними файлов (рис. 12.2). Другой случай: если вы открываете в одной и той же программе несколько окон, в меню Window можно увидеть список файлов, открытых на данный момент.

Такие меню называются динамически расширяемыми, и используются они для быстрого доступа к определенным файлам в процессе работы программы. Visual Basic .NET предлагает два способа создания динамически расширяемых меню:

- \* ✓ в процессе разработки интерфейса программы;
- ✓ в процессе выполнения программы.

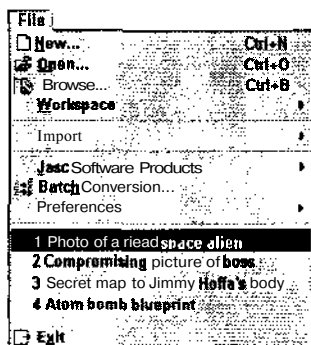


Рис. 12.2. Динамически расширяемое меню File, которое вы можете увидеть в программе PaintShop Pro

## Создание расширяемых меню в режиме конструктора

Если вы наперед знаете, что в процессе работы программы в раскрывающееся меню должно быть добавлено несколько новых пунктов, то можете заранее создать некоторое количество пустых строк в меню, и позднее, используя коды BASIC, заполнить их конкретными элементами.

Чтобы сделать это, выполните следующие действия.

1. Откройте форму, меню которой должно динамически расширяться.
2. Щелкните на заголовке раскрывающегося меню.

В нижней части открывшегося меню будет отображаться поле Type Here.

3. Щелкните в поле Type Here, наберите любой символ и нажмите клавишу возврата, чтобы стереть его.

Таким образом вы создадите в раскрывающемся меню пустую строку.

4. Откройте окно Properties.

Чтобы сделать это, нажмите клавишу <F4>, выберите команду View⇨Properties Window, в окне Solution Explorer щелкните на кнопке Properties Window или щелкните правой кнопкой мыши в окне формы и выберите команду Properties.

**5. Щелкните на свойстве Visible (Видимый), которое относится к категории Misc (Разное).**

Рядом с названием свойства появится кнопка с направленной вниз стрелкой.

**6. Щелкните на кнопке со стрелкой и выберите значение False.**

Теперь созданный вами пустой элемент меню не будет отображаться на экране.

**7. Дважды щелкните на свойстве Name (Имя) категории Design и укажите новое имя для только что созданного элемента меню.**

Назовите элемент каким-нибудь запоминающимся именем, потому что вам еще придется его использовать при написании кодов BASIC для отображения новых строк в раскрывающемся меню.

Итак, выполнив все перечисленные действия, вы создали в раскрывающемся меню резервную пустую строку, однако чтобы заполнить эту строку текстом и отобразить ее на экране (т.е. для того чтобы меню стало действительно динамически расширяемым), нужно написать коды BASIC.

Предположим, вы назвали созданную пустую строку именем `mnuFileOne`. Ниже приведен код BASIC, который выводит эту строку на экран и заполняет ее текстом.

```
MnuFileOne.Visible = True  
MnuFileOne.Text = "1 C:\My Documents\Flame.txt"
```



1. Первая строка говорит Visual Basic .NET: "Сделай команду меню, именуемую `mnuFileOne`, видимой, чтобы она смогла отображаться на экране".
2. Вторая строка дает Visual Basic .NET другое указание: "Отобрази текст `1 C:\My Documents\Flame.txt` в команде меню, именуемой `mnuFileOne`".

Выполнение двух этих шагов создает у пользователей иллюзию, что раскрывающееся меню динамически расширяется в процессе выполнения программы.

## Добавление новых элементов меню в процессе выполнения программы

Недостаток рассмотренного выше способа создания динамически расширяемых меню заключается в том, что вы не всегда можете заранее сказать, сколько именно новых элементов нужно будет добавить. Поэтому, чтобы сделать программу более гибкой (правда, и более сложной), вы должны написать коды BASIC, предназначенные для добавления в меню по мере необходимости новых элементов в процессе выполнения самой программы.



Более подробно использование кодов BASIC будет рассмотрено в четвертой части этой книги. Поэтому вы можете сейчас лишь бегло просмотреть настоящий раздел и вернуться к нему позже, когда будете больше знать о работе кодов BASIC.

А теперь попробуем создать динамически расширяемое меню с использованием кодов BASIC.

**1. В окне формы создайте открывающееся меню и присвойте ему какое-нибудь запоминающееся имя, например, `mnuFirstMenu`.**

О том, как создаются раскрывающиеся меню, рассказано в главе 11.

**2. Откройте редактор кодов для этой формы.**

**3. Наберите приведенный ниже код.**

Код, который вы видите ниже, в Visual Basic .NET называется *методом*. По существу, метод представляет собой маленькую программу внутри большой программы. Обычно методы создаются для решения каких-то несложных задач.

```
Public Sub AddMenuItem(ByVal NewStuff As String)
    Dim myMenuItemNew As New MenuItem()
    myMenuItemNew.Text = NewStuff
    mnuFileMenu.MenuItems.Add(myMenuItemNew)
End Sub
```

4. Введите следующий код куда-нибудь в текст программы, чтобы привести в действие только что набранный метод, добавив тем самым в раскрывающееся меню новый элемент.

```
AddMenuItem ("Мой новый элемент меню")
```



Чтобы помочь вам понять, как работает приведенный здесь метод, ниже дано пошаговое описание выполняемых им действий.

```
Public Sub AddMenuItem(ByVal NewStuff_
    As String)
    Dim myMenuItemNew As New MenuItem()
    myMenuItemNew.Text = NewStuff
    mnuFileMenu.MenuItems.Add(myMenuItemNew)
End Sub
```

1. Первая строка сообщает Visual Basic .NET: "Этот метод называется AddMenuItem (ему можно дать любое понравившееся вам имя). Метод принимает строку (String) и сохраняет ее как переменную NewStuff".
2. Вторая строка приказывает Visual Basic .NET: "Создай переменную myMenuItemNew (ее можно назвать по-другому), которая будет представлять собой объект, именуемый MenuItem".
3. Третья строка говорит Visual Basic .NET: "Присвой значение переменной NewStuff свойству Text объекта, представляемого переменной myMenuItemNew".
4. Четвертая строка дает указание Visual Basic .NET: "Добавь в раскрывающееся меню, именуемое mnuFileMenu, новый элемент, представляемый переменной myMenuItemNew".
5. Наконец, пятая строка информирует Visual Basic .NET: "На этом список инструкций данного метода заканчивается".



В приведенном выше примере курсивом выделены имена, которые не являются обязательными; вы можете присваивать таковые по своему усмотрению. Например, метод можно назвать не AddMenuItem (Добавление элемента меню), а, скажем, GrowMenu (Расширяемое меню). Не важно, какие именно имена вы присваиваете. Главное, чтобы вы их потом корректно использовали.



## Тест на проверку полученных вами знаний

1. Объясните, зачем иногда нужно менять команды в раскрывающихся меню.
  - а. Чтобы пользователи не расслаблялись и периодически задумывались, а все ли правильно они делают?
  - б. Чтобы один и тот же интерфейс не надоедал пользователям, нужно через каждые пять минут изменять расположение команд в меню.
  - в. Чтобы менять местами такие команды, как Undo (Отменить) и Redo (Повторить).
  - г. Ничто не вечно, в том числе и порядок команд в раскрывающихся меню.
2. Для чего создаются подменю?
  - а. Чтобы подальше спрятать от пользователей наиболее важные команды.
  - б. Чтобы иметь возможность компактно отображать команды, выполняющие подобные действия, без необходимости создавать отдельные раскрывающиеся меню.
  - в. Большая разветвленная система меню вкушает уважение к столь "серьезной" программе.
  - г. Подменю должно быть побольше, чтобы пользователи постоянно находили в них для себя что-то новое и интересное.

## Создание выпадающих меню

Многие программы предоставляют пользователям доступ к командам, не только посредством раскрывающихся меню, но и через выпадающие меню, которые, согласно терминологии Visual Basic .NET, называются также *контекстными меню*. Контекстное меню появляется на экране каждый раз, когда пользователь щелкает где-либо правой кнопкой мыши. Есть два способа создания контекстных меню:

- ✓ посредством создания новых команд, которые будут отображаться в контекстном меню;
- ✓ путем копирования в контекстные меню уже существующих команд из раскрывающихся меню.

## Создание команд для контекстных меню

Для того чтобы создать для контекстного меню новую команду, вы должны выполнить следующие действия.

1. Откройте форму, для которой нужно создать контекстное меню.
2. В панели Toolbox дважды щелкните на кнопке инструмента ContextMenu (Контекстное меню).

Кнопка ContextMenu относится к категории Windows Forms, но на экране она обычно не отображается, а чтобы ее увидеть, нужно нажать кнопку с направленной вниз стрелкой и прокрутить вниз список инструментов.

После того как вы дважды щелкнете на этой кнопке, Visual Basic .NET отобразит значок ContextMenu в отдельном окне нижней части экрана, а в самой форме появится объект ContextMenu (рис. 12.3).

3. В окне формы щелкните на объекте ContextMenu.  
Прямо под объектом ContextMenu откроется поле Type Here.
4. Щелкните в поле Type Here (Наберите здесь) и введите название команды, которая должна отображаться в контекстном меню.

5. Повторяйте шаг 4 до тех пор, пока не укажете названия всех команд, которые должны быть доступны через это контекстное меню.



После того как контекстное меню будет создано, вы в любой момент сможете его открыть для редактирования, щелкнув на значке **ContextMenu** в окне, расположенном в нижней части экрана. Visual Basic .NET отобразит само меню в верхней части окна формы, и вы сможете добавлять, удалять или перемещать его элементы. Более подробную информацию о редактировании меню можно найти в главе 11.

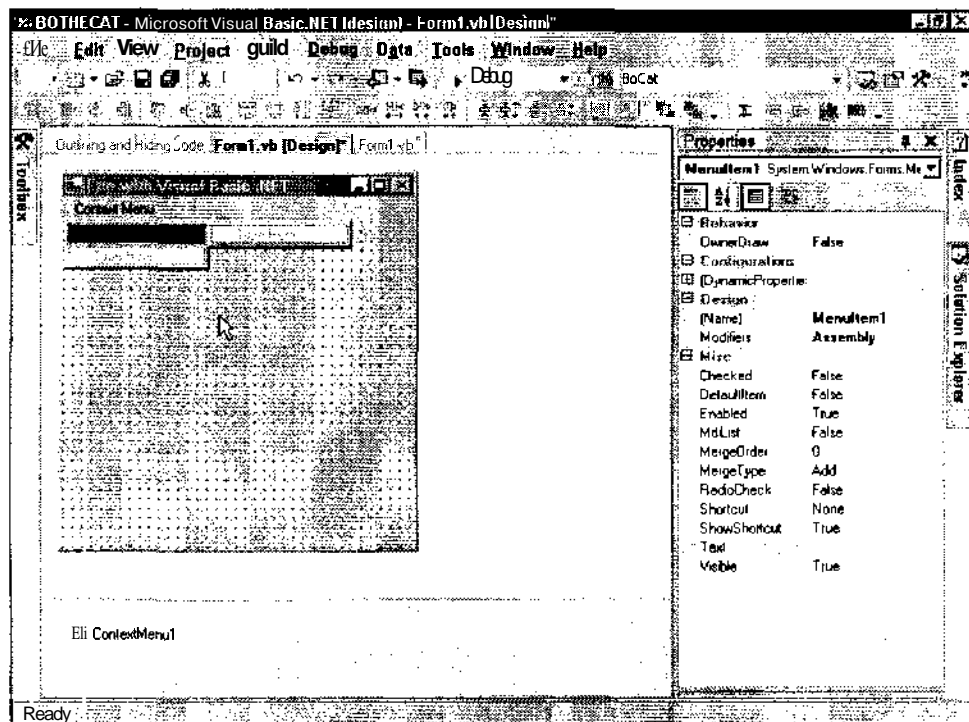


Рис. 12.3. Создание контекстного меню для формы вашей программы

## Как сделать контекстное меню выпадающим

После того как контекстное меню будет создано и вы присвоите ему (надеюсь, вы уже умеете это делать) какое-нибудь запоминающееся имя, нужно сделать так, чтобы оно появлялось на экране при каждом щелчке пользователя в окне формы правой кнопкой мыши (т.е. чтобы оно стало выпадающим).

1. Откройте форму, для которой создано контекстное меню.
2. Откройте окно Properties.

Чтобы сделать это, нажмите клавишу <F4>, выберите команду **View**→**Properties Window**, в окне **Solution Explorer** щелкните на кнопке Properties Window или щелкните правой кнопкой мыши в окне формы и выберите команду **Properties**.

3. Щелкните на свойстве **ContextMenu** (Контекстное меню) категории **Behavior** (Поведение).

Рядом с ним появится кнопка с направленной вниз стрелкой.

**4. Щелкните на кнопке со стрелкой.**

Visual Basic .NET отобразит имена всех контекстных меню, созданных для этой формы.

**5. Щелкните на названии того контекстного меню, которое, по-вашему мнению, должно отображаться на экране при каждом щелчке правой кнопкой мыши в окне формы.**

Вот и все. Теперь, стоит пользователю щелкнуть правой кнопкой мыши, контекстное меню тут же появится на экране.

## Копирование команд в контекстное меню

После того как контекстное меню будет создано и вы сделаете его выпадающим, вам еще придется написать для него коды BASIC, отвечающие за то, чтобы выбор любой из команд меню приводил к каким-нибудь действиям.

Поскольку в контекстных меню зачастую содержатся команды, которые уже имеются в раскрывающихся меню, вы можете просто скопировать эти команды из раскрывающихся меню в контекстные.

Вот как это можно сделать.

**1. Откройте форму с раскрывающимся меню, команды которого нужно скопировать.**

**2. Чтобы открыть раскрывающееся меню, щелкните на его заголовке.**

Вначале вам, возможно, придется щелкнуть на значке меню в окне, расположенном в нижней части экрана. Visual Basic .NET отобразит все команды, содержащиеся в указанном вами меню.

**3. Щелкните правой кнопкой мыши на той команде, которую нужно скопировать.**

Рядом откроется выпадающее меню.

**4. Выберите команду Copy (Копировать).**

**5. В окне, расположенном внизу экрана, щелкните на значке контекстного меню, в которое должна быть скопирована команда.**

Visual Basic .NET отобразит это контекстное меню в окне формы.

**6. Щелкните правой кнопкой мыши в поле Type Here контекстного меню.**

Рядом откроется выпадающее меню.

**7. Выберите команду Paste (Вставить).**

Visual Basic .NET вставит скопированную команду в выбранное вами поле контекстного меню.



Чтобы те команды, которые были скопированы из других меню, стали рабочими, для них также нужно будет написать коды BASIC, даже несмотря на то, что в других меню коды для этих команд уже имеются.

# Диалоговые окна

*В этой главе...*

- > Создание диалоговых окон
- > Добавление в диалоговые окна пиктограмм и кнопок
- Использование стандартных диалоговых окон

**Р**аскрывающиеся меню могут в значительной степени облегчить для пользователей процесс общения с программой (при условии, конечно, что пользователи знают, как обращаться с этими меню). Но почти все программы для диалога с пользователями помимо меню применяют и так называемые диалоговые окна.

Очень часто программы используют диалоговые окна с целью сообщить пользователю, что именно они делают в данный момент. Например, на экране может появиться сообщение наподобие "Печать страницы 4 из 75" или "Программа выполнила недопустимую операцию и сейчас будет закрыта".

К тому же с помощью диалоговых окон программа может задавать пользователям вопросы, такие например, как "Остановить печать?" или "Вы действительно хотите завершить выполнение программы?" Большие диалоговые окна могут быть буквально напичканы различными опциями, позволяя пользователям за один раз сделать немалое количество установок и ответить на множество вопросов. Подобно тому как большинство программ, написанных под Windows, используют похожие раскрывающиеся меню (File, Edit, Help), имеют между собой много общего и диалоговые окна, используемые в разных программах.

## Создание простых диалоговых окон

Небольшие диалоговые окна обычно отображают какое-нибудь сообщение для пользователя и одну или несколько командных кнопок. Типичное диалоговое окно состоит из следующих обязательных элементов:

- ✓ строка заголовка;
- ✓ сообщение;
- ✓ бросаемая в глаза пиктограмма;
- ✓ одна или несколько кнопок.

Строка заголовка определяет цель или назначение диалогового окна, например: "Сохранение документа". Сообщением называется текст, который отображается в диалоговом окне, скажем, такой: "Проверка документа завершена". Пиктограмма делает визуальный акцент на той информации, которая передается посредством диалогового окна. Кнопки позволяют сделать выбор, и обычно их количество колеблется от одной до трех штук.

Простейшее диалоговое окно отображает на экране сообщение и кнопку ОК. В этом случае пользователю ничего не остается, кроме как прочитать сообщение и щелкнуть на кнопке. Чтобы создать такое окно, вам нужно просто набрать команду `MsgBox` и указать текст, который должен отображаться в строке заголовка и в самом окне.

Приведенный ниже код BASIC создает простое диалоговое окно, показанное на рис. 13.1.

MsgBox ("Ready to try Linux yet?", , "Windows just\_crashed again")

Это простейшее диалоговое окно совершенно безобидно. Оно всего лишь появляется на экране и исчезает, как только пользователь щелкает на кнопке ОК.

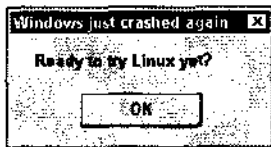


Рис. 13.1. Вот как выглядит простейшее диалоговое окно

## Добавление пиктограмм

Пиктограммы помогают привлечь внимание пользователя к диалоговому окну. Visual Basic .NET предлагает на ваш выбор четыре пиктограммы (рис. 13.2).



Critical (Критическая)



Exclamation (Предупреждающая)



Information (Информационная)



Question (Вопрос)

Рис. 13.2. Четыре пиктограммы, которые могут быть добавлены для отображения в простейших диалоговых окнах

- ✓ Critical (Критическая) — красный кружок с белым крестиком внутри — окна с такой пиктограммой появляются, как правило, в экстремальных случаях и сопровождают действительно важные вопросы, типа такого: "Еще один подобный шаг, и последние данные будут удалены с вашего компьютера. Вы уверены, что хотите этого?"
- ✓ Exclamation (Предупреждающая) — восклицательный знак — акцентирует внимание на предупреждениях, о которых пользователь обязательно должен знать, например: "Память перегружена! Удалите ненужные программы."
- ✓ Information (Информационная) — буква "i" — привлекает внимание к не очень интересным сообщениям, наподобие такого: "Печать всех 3 056 страниц вашего документа может занять некоторое время. Щелкните на кнопке ОК, когда будете готовы."
- ✓ Question (Вопрос) — вопросительный знак — сопровождает безобидные вопросы, вроде "Вы действительно не хотите выходить из этой программы?"

Чтобы разместить в диалоговом окне пиктограмму, вначале нужно написать такой код:

```
Imports.Microsoft.VisualBasic.MsgBoxStyle
```



Приведенная выше строка должна быть самым первым кодом в тексте вашей программы, иначе Visual Basic .NET не сможет ее правильно интерпретировать и программа будет работать некорректно.

Если такая строка присутствует в кодах вашей программы, при создании диалоговых окон вы можете добавлять слова *Critical*, *Exclamation*, *Information* или *Question* между сообщением диалогового окна и его заголовком, например:

```
MsgBox ("Ready to try Linux yet?", Critical, _  
        "Windows Crashed Again")
```

Такой код создает диалоговое окно, показанное на рис. 13.3.



Рис. 13.3. Отображение пиктограммы *Critical* в диалоговом окне



Если вы не добавили в код программы строку `Imports Microsoft.VisualBasic`, при создании диалогового окна вместо слова *Critical* нужно будет набрать `Microsoft.VisualBasic.MsgBoxStyle.Critical`:

```
MsgBox ("Ready to try Linux yet?", __  
        "Windows Crashed Again")
```



Visual Basic .NET позволяет отобразить в диалоговом окне только одну стандартную пиктограмму. Если вас такое ограничение не устраивает, можете создать свое диалоговое окно вручную. Для этого откройте отдельную форму, свойству `BorderStyle` присвойте значение `Fixed Dialog`, создайте в окне формы кнопки и рисунки. В этом случае вы можете разместить в окне столько рисунков, сколько посчитаете нужным.

Только не забывайте, что такой способ создания диалогового окна потребует от вас самостоятельно добавить все необходимые кнопки, надписи, рисунки, а для того чтобы окно стало рабочим, нужно будет еще написать коды BASIC. Если же вы решили не усложнять себе жизнь, а хотите создать диалоговое окно просто и быстро, воспользуйтесь командой `MsgBox`.

## Добавление командных кнопок

В стандартном диалоговом окне можно разместить от одной до трех командных кнопок. В табл. 13.1 приведены шесть возможных вариантов наличия командных кнопок.

Таблица 13.1. Возможные варианты наличия командных кнопок в стандартном диалоговом окне

Отображение	Значение	Константа
Кнопка OK	0	OKOnly
Кнопки OK и Cancel	1	OKCancel
Кнопки Abort, Retry, Ignore	2	AbortRetryIgnore

Отображение	Значение	Константа
Кнопки Yes, No, Cancel	3	YesNoCancel
Кнопки Yes и No	4	YesNo
Кнопки Retry и Cancel	5	RetryCancel

Чтобы добавить в окно командные кнопки, выберите один из возможных вариантов (например, кнопки ОК и Cancel) и наберите соответствующую константу в коде программы, между сообщением диалогового окна и его заголовком:

```
MsgBox ("File not found", OKCancel, "Cryptic Error_
      Message")
```



Чтобы иметь возможность использовать перечисленные в табл. 13.1 константы для добавления в диалоговые окна комбинации командных кнопок, первой строкой в кодах вашей программы должна быть такая:

```
Imports.Microsoft.VisualBasic.MsgBoxStyle
```

Если такой строки в начале программы не будет, то при добавлении командных кнопок придется использовать код `Microsoft.VisualBasic.MsgBoxStyle`:

```
MsgBox ("File not found",_
      Microsoft.VisualBasic.MsgBoxStyle.OKCancel,_
      "Cryptic Error Message")
```



Чтобы добавить в диалоговое окно и пиктограмму, и командные кнопки, укажите их в коде программы рядом, поставив между ними команду `Or`. Таким образом, если вы хотите отобразить в диалоговом окне информационную пиктограмму и кнопки Yes, No (рис. 13.4), наберите следующий код:

```
MsgBox ("File not found", YesNo Or Information,_
      "Cryptic Error Message")
```

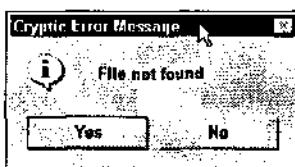


Рис. 13.4. Диалоговое окно с пиктограммой и командными кнопками

## Как определить, на какой кнопке щелкнул пользователь

Если в диалоговом окне имеется только кнопка ОК, все предельно просто: щелчок пользователя на таковой приводит к закрытию диалогового окна. Но как быть, если кнопок две или три? В этом случае у пользователя есть выбор, и, следовательно, вам нужно будет написать коды BASIC для определения того:

- ✓ на какой кнопке щелкнул пользователь;
- ✓ что делать программе, если пользователь выбрал ту или иную кнопку.

Семи кнопкам, которые может выбрать пользователь, соответствуют определенные числовые значения (табл. 13.2).

**Таблица 13.2. Командные кнопки, которые могут быть выбраны пользователем**

Кнопка	Числовое значение	Константа
OK	1	OK
Cancel (Отмена)	2	Cancel
Abort (Прервать)	3	Abort
Retry (Повторить)	4	Retry
Ignore (Пропустить)	5	Ignore
Yes (Да)	6	Yes
No (Нет)	7	No

Чтобы программа могла определить, на какой кнопке щелкнул пользователь, нужно написать код, в котором переменной будет присвоено значение, возвращаемое командой MsgBox:

```
Dim intReply As Integer
intReply = MsgBox ("File not found", OKCancel, _
    "Cryptic Message")
```



Этот код отображает диалоговое окно с кнопками OK и Cancel. Если пользователь щелкает на кнопке OK, переменной intReply присваивается значение 1 (см. табл. 13.2). Если пользователь щелкает на кнопке Cancel, переменной intReply присваивается значение 2.

## использование стандартных диалоговых окон

С помощью простых диалоговых окон невозможно решить все проблемы, поэтому вам наверняка захочется создать более сложные и в то же время хорошо всем знакомые диалоговые окна, наподобие File Save (Сохранение документа) или Print (Печать). Чтобы помочь вам в этом нелегком деле, Visual Basic .NET предлагает на выбор несколько разработанных заранее стандартных диалоговых окон, каждое из которых представлено своим инструментом на панели Toolbox:

- ✓ OpenFileDialog (Открытие документа);
- ✓ SaveFileDialog (Сохранение документа);
- ✓ ColorDialog (Цвет);
- ✓ FontDialog (Шрифт);
- ✓ PrintDialog (Печать);
- ✓ PageSetupDialog (Параметры страницы).





Кнопки инструментов для создания стандартных диалоговых окон относятся к категории Windows Forms панели Toolbox. Если вы не видите их в данный момент на экране, щелкните на кнопке любого инструмента (например, TextBox) и, нажимая кнопку с направленной вниз стрелкой, прокрутите весь список инструментов панели Toolbox,

### Тест на проверку полученных вами знаний

1. Для чего в диалоговых окнах отображаются пиктограммы?
  - а. На тот случай, если за компьютер сел пользователь, который не умеет читать.
  - б. Чтобы привлечь внимание пользователей и сделать визуальный акцент на отображаемом сообщении. Например, пиктограмма "Критическая" призвана привлечь внимание пользователя к сообщению, к которому он **должен** отнестись **очень** серьезно.
  - в. Обычно пользователи игнорируют сообщения, **которые не** сопровождаются яркими картинками.
  - г. Никто не знает для чего. Наверное, традиция такая.
2. Что нужно сделать, чтобы стандартное диалоговое окно (например, Open или Save As) было создано автоматически?
  - а. Написать специальный код BASIC: По щучьему велению, по моему хотению, окно (такое-то) создайся!
  - б. Открыть нужное диалоговое окно и нажать на компьютере кнопку RESET.
  - в. В панели Toolbox дважды щелкнуть на кнопке соответствующего инструмента (например, OpenFileDialog или SaveFileDialog).
  - г. Создать нужное окно 10 или 20 раз, после чего у вас это будет получаться почти автоматически.

## Создание окна OpenFileDialog

Диалоговое окно OpenFileDialog позволяет пользователю выбрать файл, который должен быть открыт. Здесь же могут быть отображены только те файлы, которые удовлетворяют определенным условиям, например имеют расширение .TXT или .EXE.

Чтобы создать диалоговое окно OpenFileDialog, выполните следующие действия.

1. Откройте форму.
2. На панели Toolbox дважды щелкните на кнопке инструмента OpenFileDialog. Эта кнопка относится к категории Windows Forms. Возможно, чтобы ее увидеть, вам придется прокрутить вниз список элементов данной категории (рис. 13.5.) Visual Basic .NET отобразит значок созданного диалогового окна (и присвоенное ему стандартное имя, наподобие OpenFileDialog1) в специальном окне, расположенном в нижней части экрана.

После того как диалоговое окно OpenFileDialog будет добавлено в форму, вам еще придется написать код BASIC, который должен открывать данное окно на экране. Вот этот код:

```
OpenFileDialog1.ShowDialog()
```



OpenFileDialog1 — это стандартное имя, которое Visual Basic .NET присваивает диалоговому окну автоматически. Если вы впоследствии присвоите данному окну какое-нибудь иное имя, то его нужно будет указать в приведенном выше коде вместо имени OpenFileDialog.

Если вы хотите определить список файлов, которые должны отображаться в окне OpenFileDialog, то нужно будет воспользоваться специальным приемом, который называют *применение фильтра*. Фильтр определяет для Visual Basic .NET, файлы какого типа нужно отображать (например, только с расширением .TXT или .BAT).

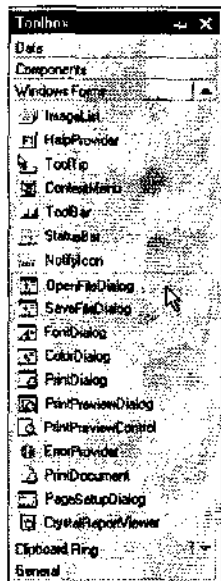


Рис. 73.5. Чтобы найти кнопку `OpenFileDialog`, прокрутите вниз список инструментов панели `Toolbox`

Фильтр состоит как бы из двух частей: из своего названия, которое отображается в списке диалогового окна, и из собственно фильтра. Примеры некоторых фильтров и соответствующих им названий приведены в табл. 13.3. Для большей наглядности названия фильтров включают в себя обозначения самих фильтров.

Так, текстовые файлы обычно имеют расширение `.TXT`, но некоторые из них имеют еще и другое расширение — `.ASC`. Таким образом, название `Text Files (*.TXT)` говорит о том, что в диалоговом окне будут отображены только те текстовые файлы, которые имеют расширение `.TXT` (те же текстовые файлы, которые имеют расширение `.ASC`, отображаться не будут).

Таблица 13.3. Фильтры и их названия

Название	Фильтр
All Files (*.*) (Все файлы)	*.*
Text Files (*.TXT) (Текстовые файлы)	*.TXT
Batch Files (*.BAT) (Командные файлы)	*.BAT
Executable Files (*.EXE) (Выполняемые файлы)	*.EXE

Вот как определяются фильтры и их названия.

1. Откройте форму, которая содержит диалоговое окно `OpenFile`.
2. Щелкните на значке `OpenFileDialog`, который отображается в окне, расположенном под окном формы.
3. Откройте окно `Properties`.

Чтобы сделать это, нажмите клавишу `<F4>`, выберите команду `View ⇒ Properties Window`, в окне `Solution Explorer` щелкните на кнопке `Properties Window` или щелкните правой кнопкой мыши в окне формы и выберите команду `Properties`.

- Щелкните на свойстве **Filter (Фильтр)** категории **Misc (Разное)** и наберите **фильтр, который хотите использовать**.

Фильтр состоит из короткого описания файлов, которые будут отображены:

```
Text files (*.txt)| *.txt
```

В этом примере текст, набранный до вертикальной черты, будет отображаться в диалоговом окне в списке **Files Of type (Тип файлов)**. А текст, набранный после вертикальной черты, является фильтром, в соответствии с которым будут отбираться файлы для отображения.



Для того чтобы набрать вертикальную черту, необходимо нажать клавишу <Shift>, а затем — клавишу, на которой изображена косая черта (\).

Вы можете создать и несколько фильтров сразу:

```
Text files (*.txt)| *.txt| All Files (*.*) [ *.*
```

При этом не забывайте каждый следующий фильтр отделять от предыдущего вертикальной чертой.

Если вы создаете несколько фильтров одновременно, вам нужно сообщить Visual Basic .NET, какой из них должен отображаться сразу после открытия окна **OpenFile**. Для этого свойству **FilterIndex (Номер фильтра)** нужно присвоить определенное числовое значение.

Когда вы создаете несколько фильтров, первому из них присваивается номер 1, второму — номер 2 и т.д. Таким образом, в предыдущем примере, чтобы при открытии диалогового окна сразу отображался фильтр `Text files (*.txt)`, свойству **FilterIndex** нужно присвоить значение 1.

Итак, чтобы определить, какой фильтр должен отображаться в диалоговом окне первым, сделайте следующее.

- Откройте форму, для которой создано диалоговое окно **OpenFile**.
- Щелкните на значке **OpenFileDialog** в окне, расположенном в нижней части экрана.
- Откройте окно **Properties**.  
Чтобы сделать это, нажмите клавишу <F4>, выберите команду **View⇒Properties Window**, в окне **Solution Explorer** щелкните на кнопке **Properties Window** или щелкните правой кнопкой мыши в окне формы и выберите команду **Properties**.
- Щелкните на свойстве **FilterIndex** категории **Misc (Разное)** и наберите номер фильтра, который должен отображаться сразу при открытии диалогового окна.

В следующий раз, когда вы запустите свою программу и откроете диалоговое окно **OpenFile**, сразу же будет включен указанный вами фильтр.

## Как определить, какой файл выбран пользователем в диалоговом окне **OpenFile**

После того как пользователь определил в диалоговом окне **OpenFile**, какой файл он хочет открыть, и щелкнул на кнопке **OK**, программа должна будет как-то узнать, а что именно выбрал пользователь. Путь к указанному пользователем файлу сохраняется как значение свойства **Filename** диалогового окна **OpenFile**. Поэтому вам нужно будет написать код, который создает новую переменную и присваивает ей значение свойства **Filename**:

```
Dim strWhatFile As String  
strWhatFile = OpenFileDialog1.filename
```



В свойстве `Filename` запоминается как само название выбранного файла, так и полный путь к нему (например, `C:\Мои документы\Личное\Резюме.txt`). Если пользователь закрыл диалоговое окно щелчком на кнопке `Cancel`, свойству `Filename` присваивается значение `""` (пустая строка).

## Создание окна `SaveFile`

Диалоговое окно `SaveFile` выглядит почти так же, как и окно `OpenFile`. Разве что заголовки у этих окон разные. (В заголовке диалогового окна `OpenFile` отображается слово `Open` (Открыть), а в заголовке окна `SaveFile` отображаются слова `Save As` (Сохранить как).)

Чтобы создать диалоговое окно `SaveFile`, сделайте следующее.

1. **Откройте форму.**
2. **На панели `Toolbox` дважды щелкните на кнопке инструмента `SaveFileDialog`. (Эта кнопка расположена в категории `Windows Forms`. Возможно, чтобы ее увидеть, вам придется прокрутить вниз список элементов этой категории.)**

Visual Basic .NET отобразит значок созданного диалогового окна и его название (что-то наподобие `SaveFileDialog1`) в окне, расположенном в нижней части экрана.

После того как окно создано, для его отображения следует набрать следующий код:

```
SaveFileDialog1.ShowDialog()
```



Как и в окне `OpenFile`, в окне `SaveFile` вы можете использовать фильтр для отбора тех файлов, которые могут быть здесь показаны. Чтобы получить такую возможность, нужно изменить значение свойства `Filter` диалогового окна `SaveFile`. Как это сделать, рассказывается выше, в разделе "Создание окна `OpenFile`".

## Как определить, какой файл выбран пользователем в окне `SaveFile`

Как и в случае с окном `OpenFile`, название файла, который выбран в окне `SaveFile`, сохраняется в качестве значения свойства `Filename`. Чтобы программа определила, какой именно выбор сделал пользователь, нужно создать новую переменную и присвоить ей значение свойства `Filename`:

```
Dim strWhatFile As String  
strWhatFile = SaveFileDialog.Filename
```



Диалоговое окно `SaveFile` само по себе файлы не сохраняет. Чтобы сохранить файл, вам нужно будет написать дополнительный код BASIC, который отдаст компьютеру распоряжение физически сохранить файл на диске.

## Создание окна `Color`

Диалоговое окно `Color` (Цвет) позволяет пользователю выбрать один из стандартных цветов или же определить и задать собственный цвет (рис. 13.6).

Вот что нужно сделать для создания такого диалогового окна.

1. **Откройте форму.**
2. **На панели `Toolbox` дважды щелкните на кнопке инструмента `ColorDialog`. (Эта кнопка расположена в категории `Windows Forms`. Возможно, чтобы ее увидеть, вам нужно будет прокрутить вниз список инструментов этой категории.)**

Visual Basic .NET отобразит значок созданного диалогового окна и его название (что-то наподобие ColorDialog1) в окне, расположенном в нижней части экрана.

После того как диалоговое окно создано, для его отображения нужно набрать следующий код:

```
ColorDialog1.ShowDialog()
```

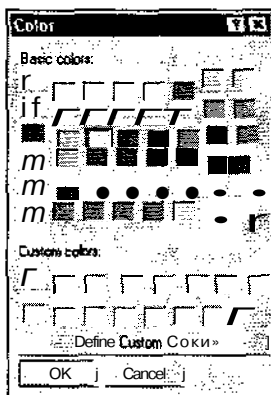


Рис. 13.6. Диалоговое окно Color позволяет пользователю выбрать нужный цвет

## Как определить, какой цвет был выбран пользователем

Информация о выборе пользователя сохраняется как значение свойства **Color** диалогового окна Color. Это значение можно использовать затем для определения цвета отдельных объектов (ForeColor) или их фона (BackColor). Например, фоновый цвет для объекта Button1 устанавливается следующим кодом:

```
Button1.BackColor = ColorDialog1.Color
```

## Создание окна Font

Диалоговое окно **Font** (Шрифт) позволяет пользователям определить шрифт, размер и начертание набираемого ими текста (рис. 13.7). Каждый раз, когда пользователь выбирает какую-то новую опцию, в диалоговом окне отображается пример того, как будет выглядеть текст при такой установке.

Чтобы создать диалоговое окно **Font**, выполните такие действия.

1. **Откройте форму.**
2. **На панели Toolbox дважды щелкните на кнопке инструмента FontDialog. (Эта кнопка относится к категории Windows Forms. Возможно, чтобы ее увидеть, вам придется прокрутить вниз список инструментов данной категории.)**

Visual Basic .NET отобразит значок созданного диалогового окна и его название (что-то наподобие FontDialog1) в окне, расположенном в нижней части экрана.

После того как диалоговое окно **Font** создано, для его отображения нужно набрать следующий код:

```
FontDialog1.ShowDialog()
```

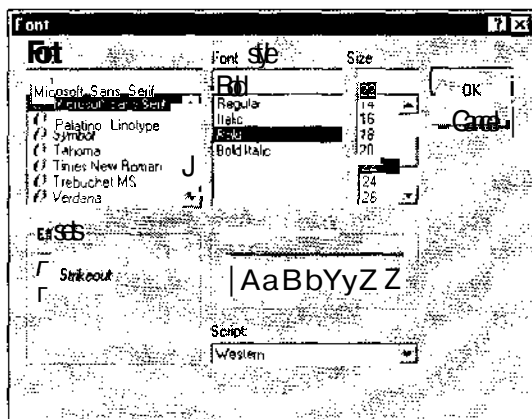


Рис. 13.7. В диалоговом окне Font можно определить шрифт, начертание и размер текста

## Как определить, какие опции были выбраны пользователем в окне Font

Когда диалоговое окно Font открыто, пользователь может выбрать шрифт текста, его размер и еще несколько дополнительных установок. Вся информация о заданных пользователем установках сохраняется в виде значений свойства **Font** (категория **Misc**) диалогового окна Font. Если вы щелкнете на знаке “плюс”, отображаемом рядом с названием этого свойства в окне Properties, Visual Basic .NET откроет список дополнительных свойств, хранящих всю информацию о выборе пользователя (табл. 13.4).

Таблица 13.4. Свойства, запоминающие установки, сделанные пользователем в диалоговом окне Font

Свойство	Значение
Name (Имя)	Название выбранного шрифта (например, Times New Roman)
Size (Размер)	Число, обозначающее размер шрифта (например, 3 или 12)
Bold (Полужирный)	Значение False или True
Italic (Курсив)	Значение False или True
Strikeout (Зачеркнутый)	Значение False или True
Underline (Подчеркнутый)	Значение False или True

Чтобы программа знала, какую опцию выбрал пользователь, нужно написать код BASIC. Выглядеть он должен приблизительно так:

```
Dim strWhatFont As String
strWhatFont = FontDialog1.Font.Name
```



Для получения данных об установках, сделанных пользователем в диалоговом окне **Font**, вначале нужно указать название диалогового окна (в нашем примере это FontDialog1), далее название свойства **Font**, а затем название того свойства, данные о котором нужно получить (например, Name или Bold).

## Создание окна Print

Диалоговое окно Print (Печать) позволяет пользователям выбрать принтер, определить диапазон печати и количество требуемых копий. Окно Print показано на рис. 13.8.

Чтобы создать такое диалоговое окно, нужно сделать следующее.

1. Откройте форму.
2. На панели Toolbox дважды щелкните на кнопке инструмента PrintDialog. (Эта кнопка расположена в категории Windows Forms. Возможно, чтобы ее увидеть, вам придется прокрутить вниз список инструментов данной категории.) Visual Basic .NET отобразит значок созданного диалогового окна и его название (что-то наподобие PrintDialog1) в окне, расположенном в нижней части экрана.
3. На панели Toolbox дважды **щелкните** на кнопке инструмента PrintDocument. (Эта кнопка расположена в категории Windows Forms. Возможно, чтобы ее увидеть, вам нужно будет прокрутить вниз список инструментов данной категории.) Visual Basic .NET отобразит в окне, расположенном в нижней части экрана, значок печатаемого документа и его название (что-то наподобие PrintDocument1). Обратите внимание: если вы назовете печатаемый документ каким-нибудь своим именем, это же имя нужно будет указать и на шаге 7.
4. На панели Toolbox дважды щелкните на кнопке инструмента PrintDialog. Значок этого диалогового окна отобразится в окне, расположенном в нижней части экрана.
5. Откройте окно Properties. Чтобы сделать это, нажмите клавишу <F4>, выберите команду View⇨Properties Window, в окне Solution Explorer щелкните на кнопке Properties Window или щелкните правой кнопкой мыши в окне формы и выберите команду Properties.
6. Щелкните на свойстве Document (Документ) категории Misc (Разное). Рядом появится кнопка с направленной вниз стрелкой.
7. Щелкните на кнопке со стрелкой и выберите имя печатаемого документа (в нашем случае это PrintDocument1).

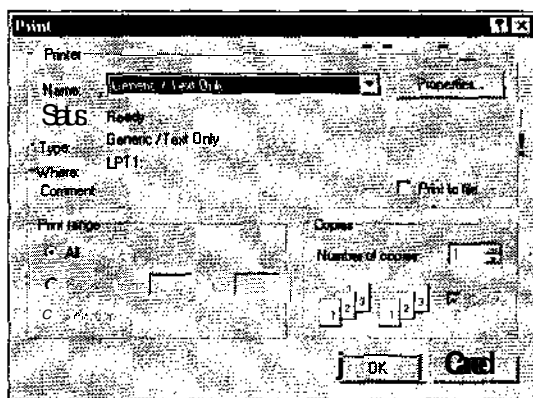


Рис. 13.8. В этом диалоговом окне можно указать страницы, которые должны быть распечатаны



объект `PrintDocument` имеет свойство `DocumentName` (Имя документа), относящееся к категории `Misc`. По значению данного свойства диалоговое окно `Print` определяет, какой документ должен быть выведен на печать.

После того как объекты `PrintDialog` и `PrintDocument` созданы, в текст программы нужно добавить код, посредством которого диалоговое окно `Print` будет отображаться на экране:

```
PrintDialog1.ShowDialog()
```



Диалоговое окно `Print` само по себе ничего не печатает, оно лишь может принять установки от пользователя. Вам же еще нужно будет написать коды `BASIC`, которые должны реально выводить что-то на печать.

## Создание окна `PageSetup`

В диалоговом окне `PageSetup` (Параметры страницы) пользователи могут определить размеры печатаемых страниц и величину их полей (рис. 13.9).

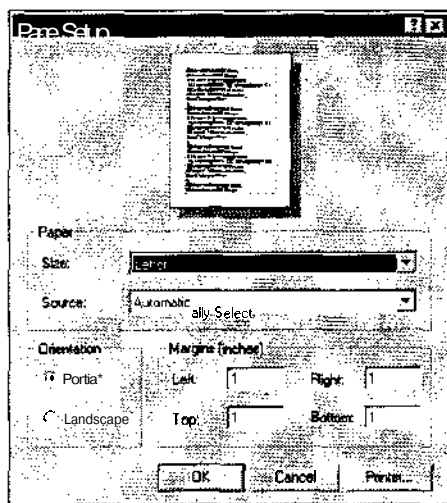


Рис. 13.9. В диалоговом окне `PageSetup` можно указать размер выводимых на печать страниц

Вот как создается диалоговое окно `PageSetup`.

1. Откройте форму.
2. На панели `Toolbox` дважды щелкните на кнопке инструмента `PrintDocument`. (Эта кнопка расположена в категории `Windows Forms`. Возможно, чтобы ее увидеть, вам придется прокрутить вниз список инструментов категории.)

`Visual Basic .NET` отобразит в окне, расположенном в нижней части экрана, значок печатаемого документа и его название (что-то наподобие `PrintDocument1`). Обратите внимание: если вы назовете печатаемый документ каким-нибудь своим именем, это же имя нужно будет указать и на шаге 6.

3. На панели `Toolbox` дважды щелкните на кнопке инструмента `PageSetupDialog`.

Значок этого диалогового окна отобразится в окне, расположенном в нижней части экрана.



#### 4. Откройте окно **Properties**.

Чтобы сделать это, нажмите клавишу <F4>, выберите команду **View**⇒**Properties Window**, в окне **Solution Explorer** щелкните на кнопке **Properties Window** или щелкните правой кнопкой мыши в окне формы и выберите команду **Properties**.

#### 5. Щелкните на свойстве **Document (Документ)** категории **Misc (Разное)**.

Рядом с названием свойства появится кнопка с направленной вниз стрелкой.

#### 6. Щелкните на кнопке со стрелкой и выберите имя печатаемого документа (в нашем случае это `PrintDocument1`).



Объект **PrintDocument** имеет свойство **DocumentName** (Имя документа), относящееся к категории **Misc**, по значению которого диалоговое окно **PageSetup** определяет, размеры и поля страниц какого документа устанавливаются пользователем. Вам нужно будет написать код **BASIC** для определения имени этого документа, а также коды **BASIC** для изменения (с учетом установок, сделанных пользователем) параметров печатаемых страниц.

Добавив к форме объект **PrintDocument** и диалоговое окно **PageSetup**, вы сможете отобразить на экране окно **PageSetup**, набрав в тексте программы такой код **BASIC**:

```
PageSetupDialog1.ShowDialog()
```

## Часть IV

# Основы создания кодов



Кстати, ты знаешь, что коты обожают печатать  
на клавиатуре? Я своего Тишку уже второй час  
не могу отогнать от компьютера

## *В этой части...*

Ура-а! **Наконец-то** начинаются **главы**, рассказывающие о том, как создать собственные коды BASIC и таким образом заставить компьютер делать то, что вы так давно от него ждете. До сих пор нами рассматривались лишь приемы построения пользовательского интерфейса (и написания некоторых сопутствующих ему кодов BASIC). Но каждому понятно, что просто красиво выглядеть недостаточно (разве что только в том случае, если вы пытаетесь выскочить замуж за старенького **мультимиллионера**, жить которому осталось от силы **пару** месяцев). Представьте, что созданный вами интерфейс будет не только радовать **глаз**, но и начнет адекватно реагировать **на** все ваши действия.

Даже если вам кажется, что освоение кодов BASIC — занятие не из легких, очень скоро вы сможете убедиться в обратном. Сами по себе коды BASIC — это всего лишь наборы пошаговых инструкций для **компьютера**, точно указывающих ему, что нужно делать. Поэтому приготовьтесь начать отдавать компьютеру указания (другими **словами**, писать коды BASIC), и вы увидите, что процесс программирования может быть простым и приятным. Более того, он войдет в привычку, как рисование пользовательских интерфейсов или как периодические **“уходы”** в любимую компьютерную игру.

# Написание процедур обработки событий

В этой главе...

- Использование редактора кодов
- Создание процедур обработки событий
- > Просмотр процедур обработки событий

Любое действие, совершаемое пользователем в отношении компьютера, будь то щелчок кнопкой мыши, нажатие клавиши или набор текста какого-либо послания на клавиатуре, называется *событием*. Как только событие происходит, Visual Basic .NET начинает искать код BASIC, который бы объяснил компьютеру, как он должен на это реагировать. Коды BASIC, содержащие инструкции на тот случай, если наступит то или иное событие, называются *процедурами обработки событий*.

Любая программа Visual Basic .NET может содержать в себе тысячи процедур обработки событий. Однако если вы создадите такую программу, то это значит, что либо программа будет невероятно сложной и огромной, либо программист из вас никудышный.

Сколько бы в программе ни было процедур обработки событий, возникает вопрос: а как Visual Basic .NET определит, какую процедуру в какой момент нужно использовать?

Ответ на этот вопрос прост. Любое событие совершается в отношении какого-нибудь элемента пользовательского интерфейса. Например, пользователь обычно щелкает кнопкой мыши только тогда, когда курсор наведен на какой-то объект интерфейса (на кнопку, текстовое поле или на команду меню).

Каждый объект может иметь одну или несколько процедур обработки событий, и каждая процедура соответствует какому-то определенному событию (например, щелчку кнопкой мыши или нажатию клавиши).

## Работа с редактором кодов

Редактор кодов — это окно, где вы можете набирать коды BASIC, благодаря которым ваша программа начнет делать что-то полезное. Вот как открывается редактор кодов.

1. Выберите команду View⇒Solution Explorer или нажмите комбинацию клавиш <Ctrl+Alt+L>.

На экране появится окно Solution Explorer.

2. Щелкните в окне формы и выберите команду View⇒Code, нажмите клавишу <F7> или щелкните на кнопке View Code в окне Solution Explorer.

Visual Basic .NET откроет редактор кодов для выбранной вами формы.

Если вы хотите написать коды и сохранить их в файле, который не относится к формам, на шаге 2 нужно щелкнуть на имени этого файла.





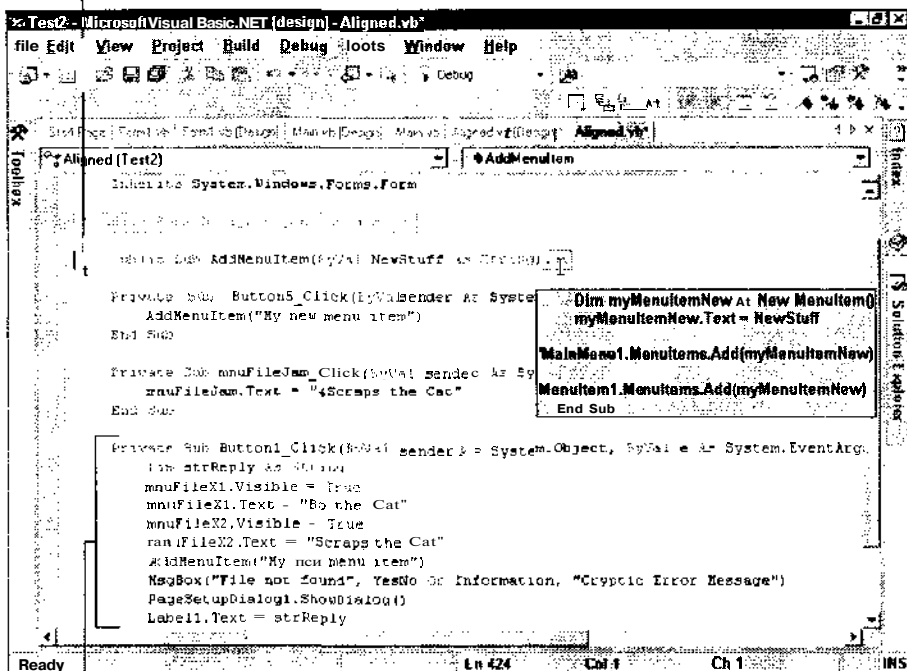
После того как вы в первый раз откроете редактор кодов для формы, его корешок будет отображаться рядом с корешком вкладки, где открыта сама форма (тот корешок, на котором вы видите имя формы и слово [Design]). Чтобы снова переключиться в редактор кодов, достаточно щелкнуть на данном корешке.

## Разворачивание и сворачивание кодов BASIC

В идеальном варианте все коды BASIC должны помещаться на одном экране. В таком случае вы или кто-либо другой сможете просматривать и модифицировать их без необходимости прокручивать экран. Если ваши программы короткие и лаконичные, они будут просты для чтения, понимания и, следовательно, для внесения в них изменений.

К сожалению, в действительности коды BASIC редко размещаются на одном экране, а это значит, что вам или кому-то еще приходится постоянно заниматься его прокруткой. Чтобы облегчить вашу участь, Visual Basic .NET позволяет сворачивать и разворачивать отдельные фрагменты кодов BASIC.

Свернутый фрагмент кодов



Развернутый фрагмент кодов

Рис. 14. \.Сворачивая и разворачивая отдельные фрагменты кодов BASIC, вы сможете быстрее понять весь текст программы



Visual Basic .NET изначально знает, как сворачивать и разворачивать коды отдельных процедур. Но если вы хотите сами определить, какой фрагмент кодов должен быть свернут, выделите его и выберите команду Edit⇒Outlining⇒Hide Selection (Правка⇒Выделение⇒Скрыть выделенное).

Чтобы свернуть фрагмент кода BASIC, достаточно щелкнуть на знаке “минус”, который отображается слева от данного фрагмента. Visual Basic .NET свернет этот фрагмент и отобразит рядом три точки, обозначая тем самым, что здесь скрыт отдельный фрагмент программы.

При необходимости повторно развернуть на экране скрытый фрагмент, щелкните на знаке “плюс”, который отображается слева от него. Окно редактора кодов со свернутыми и развернутыми фрагментами кодов программы показано на рис. 14.1.

## Виды событий

Все события могут быть разбиты на две категории.

- ✓ **События, инициируемые пользователем.** Наступают, когда пользователь нажимает какие-либо клавиши, набирает что-то на клавиатуре или производит какие-то действия с помощью мыши (перемещает курсор, щелкает кнопками и т.п.).
- ✓ **События, генерируемые системой.** Наступают, когда программа Visual Basic .NET сама производит какие-то действия, например открывает форму или изменяет содержимое текстового поля.

Хотя Visual Basic .NET может реагировать на множество событий, вам, скорее всего, понадобится, чтобы создаваемый интерфейс реагировал только на некоторые из них, например на нажатие определенных клавиш или на щелчки на конкретных объектах. Как только Visual Basic .NET увидит, что произошло какое-то событие, он сразу же начнет искать инструкции, которые вы оставили на этот случай.

Например, когда пользователь щелкает кнопкой мыши, Visual Basic .NET определяет, что произошло событие (“Ага, это был щелчок кнопкой мыши”). Далее он идентифицирует объект, на котором щелкнул пользователь (“Пользователь щелкнул на кнопке ОК”).

Затем Visual Basic .NET определяет, не написана ли для объекта процедура обработки событий, объясняющая компьютеру, что нужно делать, если произошло именно это событие.

## Создание процедур обработки событий

Один объект может реагировать сразу на несколько событий. Например, кнопка может отреагировать на щелчок на ней и на нажатие клавиши <Enter>.

На одно и то же событие могут реагировать сразу несколько объектов. Например, кнопка и переключатель могут отреагировать на один и тот же щелчок мыши, но обычно у каждого из них имеются свои инструкции относительно того, что необходимо делать дальше.

Чтобы создать процедуру обработки событий, вам нужно выполнить следующие действия.

1. **Определите, какая часть пользовательского интерфейса должна реагировать на событие.**
2. **Откройте редактор кодов.**
3. **Определите событие, на которое должен отреагировать Visual Basic .NET.**
4. **Напишите код BASIC, начинающий выполняться при наступлении указанного события.**



Прежде чем приступить к написанию процедур управления событиями, убедитесь, что всем объектам пользовательского интерфейса уже присвоены имена. Если вы создадите процедуру для объекта, а затем переименуете его, вам нужно будет внести исправления в код процедуры.

Вот три составляющие пользовательского интерфейса, которые могут реагировать на события:

- I ✓ формы;
- \* ✓ объекты (кнопки, флажки, поля со списками и т.п.);
- I ✓ раскрывающиеся меню.

Для того чтобы создать процедуру обработки событий для любого элемента пользовательского интерфейса, необходимо сделать следующее.

- 1. Откройте форму, содержащую объект, для которого должна быть создана процедура обработки событий.**

Если вы дважды щелкнете на объекте (скажем, на кнопке или даже на самой форме), Visual Basic .NET сразу же откроет редактор кодов и создаст в нем процедуру, наиболее типичную для данного объекта.

- 2. Откройте редактор кодов, для чего нажмите клавишу <F7> или выберите команду View⇒Code.**

Visual Basic .NET отобразит его на экране (см. рис. 14.1).

- 3. Щелкните на списке Class Name (Название класса).**

Откроется список названий объектов (таких как Label1 и Button2). Однако к этому времени, и вы, надо полагать, об этом помните, необходимо было присвоить объектам собственные имена, поэтому названия объектов в открывшемся списке не должны совпадать с автоматически генерируемыми именами.

- 4. Щелкните на списке Method Name (Название метода).**

Откроется список событий, на которые может отреагировать выбранный вами объект. Среди них будут, к примеру, события Click (Щелчок кнопкой мыши) и MouseHover (На объект наводится курсор).

- 5. Щелкните на названии события, на которое должен реагировать объект (например, Click).**

Visual Basic .NET создаст пустую процедуру обработки событий. Теперь вы можете приступить к написанию кодов BASIC, с тем чтобы новая процедура начала реально что-то делать (например, изменять свойства других объектов формы).

Вот как создается процедура обработки событий для команды раскрывающегося меню.

- 1. Щелкните на заголовке меню, в котором содержится нужная команда.**
- 2. Дважды щелкните на названии команды, для которой требуется создать процедуру обработки событий.**

Visual Basic .NET создаст для этой команды пустую процедуру обработки событий, и вам останется только наполнить ее реальным содержанием.

## Из каких частей состоит процедура обработки событий

Когда вы впервые создаете процедуру обработки событий, Visual Basic .NET сразу же отображает пустую процедуру в окне редактора кодов. Пустая процедура обработки событий состоит из двух строк:

```
Private Sub Button1_Click(ByVal sender As System.
    Object, ByVal e As System.EventArgs) _
    Handles Button1.Click
```

```
End Sub
```

Первая строка любой процедуры обработки событий состоит из следующих частей.

- ✓ Private Sub. Идентифицирует процедуру как подпрограмму.
- ✓ Название объекта. В данном примере объекту "кнопка" было присвоено имя Button1.
- Символ подчеркивания (\_).
- ✓ Название события. В данном примере этим событием является щелчок кнопкой мыши (Click).
- ✓ Пара круглых скобок, содержащих в себе данные, которые необходимы для работы подпрограммы.
- % Слово Handles и следующее за ним ключевое слово. По ключевому слову событию (в данном случае Button1.Click) ставится в соответствие процедура обработки событий.



Не старайтесь сейчас вникнуть во все технические тонкости приведенной выше процедуры обработки событий. На данном этапе важно, чтобы вы научились правильно идентифицировать различные части процедур обработки событий.

Рассмотренная процедура как бы говорит компьютеру: "Здесь содержатся инструкции, которые нужно выполнять каждый раз, когда пользователь щелкает мышью на кнопке, имеющей Button1".

Поскольку в данной процедуре никакие инструкции пока не содержатся, щелчок на кнопке Button1 не приведет к выполнению компьютером каких-либо действий.



При любом изменении имени объекта не забывайте вносить соответствующие изменения во все написанные для него процедуры обработки событий. Иначе Visual Basic .NET не сможет разобраться, какая процедура для какого объекта написана.

## Разделение окна редактора кодов на две части

Если ваша программа будет содержать большое количество процедур обработки событий, не трудно догадаться, что их коды не смогут одновременно поместиться на экране. Одним из способов решения этой проблемы является сворачивание отдельных частей программы, с тем чтобы временно скрыть коды, в отображении которых на данный момент нет необходимости. (Этот прием описан в разделе "Разворачивание и сворачивание кодов BASIC".)

Другим способом, который может помочь вам просматривать одновременно разные части программы, является разделение окна редактора кодов на две части. В разделенном надвое окне можно просматривать разные фрагменты кодов BASIC одной и той же программы.



Окно редактора кодов можно разделить только на две части (но не на три или, скажем, четыре).

Для того чтобы разделить окно редактора кодов, выполните такие действия.

1. Выберите команду Window⇒Split (**Окно⇒Разделить**).  
Visual Basic .NET разделит окно редактора кодов на две части.
2. Поместите курсор над полосой разделения, нажмите левую кнопку мыши и, перетянув курсор, настройте размеры верхнего и нижнего окна.  
Когда полоса разделения примет нужное положение, отпустите кнопку мыши (рис. 14.2).



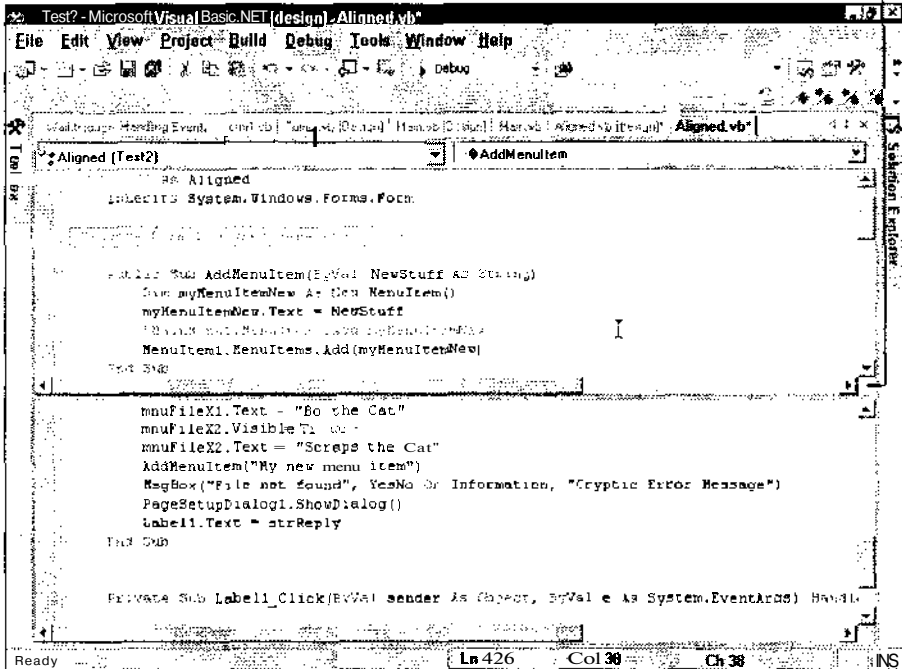


Рис./4.2. Разделив окно редактора кодов на две части, вы сможете одновременно видеть на экране разные фрагменты кода в своей программе



При необходимости снять разделение и вернуть окну редактора кодов прежний вид, выберите команду **Window** ⇒ **Remove Split** (**Окно** ⇒ **Снять разделение**).

## Использование редактора кодов

Работа редактора кодов похожа на работу обычного текстового процессора. В табл. 14.1 приведен список команд (вызываются нажатием соответствующих клавиш), которые вы можете использовать при редактировании своих процедур обработки событий.

Таблица 14.1. Клавиши, используемые при редактировании кодов BASIC

Клавиши	Вызываемые действия
<Delete>	Удаляет символ, находящийся справа от курсора
<Клавиша возврата>	Удаляет символ слева от курсора
<Home>	Переносит курсор на начало строки
<End>	Переносит курсор в конец строки
<Ctrl+Home>	Переносит курсор на первую строку кодов программы

Клавиши	Вызываемые действия
<Ctrl+End>	Переносит курсор на последнюю строку кодов программы
<Ctrl+Стрелка вниз>	Прокручивает окно редактора кодов на одну строку вниз без перемещения курсора
<Ctrl+Стрелка вверх>	Прокручивает окно редактора кодов на одну строку вверх без перемещения курсора
<Ctrl+Page Down>	Переход на следующую строку редактора кодов (не путайте со следующей строкой кодов вашей программы)
<Ctrl+Page Up>	Переход на одну строку выше в редакторе кодов (но не на предыдущую строку кодов вашей программы)
<Ctrl+Стрелка вправо>	Переход на одно слово вправо
<Ctrl+Стрелка влево>	Переход на одно слово влево
<Page Down>	Переход на следующую страницу редактора кодов
<Page Up>	Переход на предыдущую страницу редактора кодов
<Insert>	Включение и выключение режима вставки
<Ctrl+X>	Вырезать выделенный фрагмент текста
<Ctrl+C>	Скопировать выделенный фрагмент текста
<Ctrl+V>	Вставить вырезанный или скопированный фрагмент текста
<Ctrl+Z>	Отменить последнее действие
<Ctrl+F>	Найти указанное слово
<F1>	Вызов справки Visual Basic .NET
<F6>	Переключение между частями окна редактора кодов (если оно было разделено надвое)
<Ctrl+H>	Поиск и замена указанных слов другими словами
<Ctrl+P>	Отображение диалогового окна Print (Печать)

Чтобы помочь вам в написании кодов BASIC, редактор кодов автоматически выделяет цветом зарезервированные ключевые слова. Благодаря этому вы видите, какие команды являются зарезервированными в BASIC ключевыми словами, а какие были созданы вами.



Если вы при работе в редакторе кодов допустите ошибку (например, случайно удалите строку программы), нажмите комбинацию клавиш <Ctrl+Z>, и ваше последнее действие будет отменено.

# Просмотр процедур обработки событий

Обычно программы Visual Basic .NET включают в себя несколько процедур обработки событий, каждая из которых хранится в отдельном файле. Это избавляет вас от необходимости прокручивать окно редактора кодов в поисках нужной процедуры. Вы просто будете пользоваться списками Class Name и Method Name, расположенными в верхней части окна редактора кодов.

Список Class Name содержит перечень всех созданных для данной формы объектов, таких как кнопки, переключатели, раскрывающиеся меню, а список Method Name — перечень всех возможных событий, на которые отреагируют объекты.

Чтобы найти нужную процедуру обработки событий, используя списки Class Name и Method Name, сделайте следующее.

## 1. Щелкните на списке Class Name.

Visual Basic .NET отобразит перечень доступных объектов (рис. 14.3). Если вы, предположим, создали кнопку и присвоили ей имя btnClickMe, вы обязательно найдете ее в этом списке.

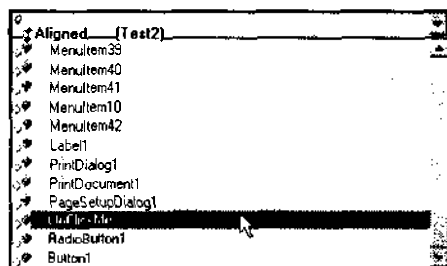


Рис. 14.3. В списке Class Name вы найдете имена всех объектов, созданных для открытой на данный момент формы

## 2. Щелкните на имени нужного объекта.

## 3. Щелкните на списке Method Name.

Visual Basic .NET отобразит список всех возможных событий, на которые может отреагировать выбранный вами объект (рис. 14.4). Названия тех событий, для которых уже созданы процедуры обработки, будут отображаться полужирным шрифтом.

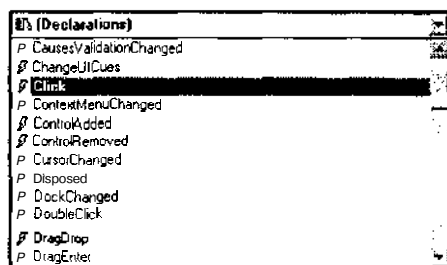


Рис. 14.4. Примерно так выглядит открытый список Method Name

## 4. Щелкните на названии нужного события.

Visual Basic .NET откроет соответствующую процедуру обработки событий.



Если вы щелкнете на событии, название которого дано полужирным шрифтом, Visual Basic .NET откроет уже созданную процедуру обработки событий. Если же вы щелкнете на событии, название которого отображено нормальным шрифтом (не полужирным), Visual Basic .NET откроет для него пустую процедуру обработки событий,

В табл. 14.2 приведен список некоторых (но далеко не всех) событий, на которые могут отреагировать объекты пользовательского интерфейса.

**Таблица 14.2. Наиболее часто используемые события**

Название	Событие
Click	Пользователь щелкает один раз кнопкой мыши на данном объекте
DoubleClick	Пользователь дважды (два раза подряд) щелкает кнопкой мыши на данном объекте
DragDrop	Пользователь наводит курсор на объект, нажимает кнопку мыши, перетаскивает курсор и отпускает кнопку мыши
DragOver	Пользователь наводит курсор на объект, нажимает кнопку мыши, перетаскивает курсор в сторону
GotFocus	Объект становится выделенным вследствие нажатия пользователем клавиши <Tab>, щелчка кнопкой мыши или в результате открытия формы
KeyDown	Пользователь нажимает какую-нибудь клавишу
KeyPress	Пользователь нажимает и отпускает одну из ANSI-клавиш, к которым относятся символьные клавиши, их комбинации с клавишей <Ctrl>, клавиша <Enter> и клавиша возврата
KeyUp	Пользователь отпускает клавишу
LostFocus	Выделение снимается с объекта вследствие нажатия пользователем клавиши <Tab>, щелчка кнопкой мыши на другом объекте или в результате закрытия формы
MouseDown	Пользователь нажимает кнопку мыши
MouseHover	Пользователь помещает курсор мыши на объект
MouseUp	Пользователь отпускает кнопку мыши

Название процедуры обработки событий определяется комбинацией имени объекта и названия события. Поскольку имена всех объектов, созданных для одной формы, должны быть уникальными, все процедуры обработки событий будут иметь разные имена.



Для каждого объекта может быть написано целое множество процедур обработки событий, что позволит программе по-разному реагировать на различные события, например такие, как размещение курсора над объектом и щелчок на нем кнопкой мыши. Но в большинстве случаев для объектов создается лишь по несколько процедур обработки событий.

# Написание процедур обработки событий

Чтобы задуманный вами пользовательский интерфейс заработал, нужно создать процедуры обработки событий и заполнить их кодами BASIC. Рабочие процедуры обработки событий могут выполнять одну или более из перечисленных ниже задач.

- ✓ Считывать значения, которые пользователь передает объектам пользовательского интерфейса (получение данных от пользователя).
- ✓ Обрабатывать данные, которые представляют собой значения свойств объектов пользовательского интерфейса (вычисление результата).
- ✓ Изменять значения свойств объектов пользовательского интерфейса (отображение на экране полученных результатов).

## Получение данных от пользователя

Чтобы получить данные от пользователя, нужно создать переменную (этот процесс описан в главе 15), прочесть значение, присвоенное свойству объекта пользовательского интерфейса (о том, как это делается, вы узнаете из главы 16), и присвоить это значение созданной переменной, благодаря чему другие части программы смогут использовать его для вычисления результата.

## Вычисление результата

Процесс вычисления результата подразумевает использование для получения новых данных арифметических команд или команд обработки текстовых значений, описанных соответственно в главах 17 и 18.



Любые коды BASIC, которые были написаны для вычисления результатов, должны быть сохранены в файлах модулей (см. главу 27) или в файлах классов (см. главу 31). Таким образом, если ваша программа работает некорректно, вы можете быстро определить источник возникновения ошибки. Если программа возвращает неверный результат, значит, ошибку нужно искать в файлах модулей или классов. Если результат вычисляется правильно, но по какой-то причине на экране отображается не то, что нужно, значит, ошибка в кодах процедуры обработки событий.

## Отображение полученных результатов на экране

Чтобы отобразить полученные данные для пользователя, необходимо присвоить новые значения свойствам объектов пользовательского интерфейса (например, свойству Text текстового поля для отображения вычисленного значения или какого-нибудь сообщения). Таким образом, пользовательский интерфейс выполняет две основные функции: принимает данные от пользователя и отображает для него полученные результаты.

## Процедура, которую должна иметь любая программа

Самая простая и самая важная процедура обработки событий, которая имеется в каждой программе, — это процедура, останавливающая выполнение программы. Приведенная ниже процедура при щелчке пользователя на кнопке Button1 также сообщает Visual Basic .NET о необходимости закончить выполнение программы:

```
Private Sub Button1_Click(ByVal sender As System._  
                          Object, ByVal e As System.EventArgs) _
```

Handles Button1.Click

Me.Close()

End Sub

Если среди созданных вами процедур обработки событий не будет ни одной, останавливающей выполнение программы, пользователю, чтобы выйти из программы, не останется ничего другого, как перезагрузить компьютер или вообще его выключить. Поскольку такой способ выхода из программы не является самым оптимальным (более того, он просто ужасен), никогда не забывайте создавать одну или несколько процедур для цивилизованного завершения работы программы.

### Тест на проверку полученных вами знаний

1. Что такое события? Каких видов они бывают?
  - а. Событиями мы называем получение Нобелевских премий программистами за написание исключительно гениальных программ.
  - б. Для компьютера событием считается разлитый на клавиатуре кофе, а также треснутый от неудачного движения пользователя монитор.
  - в. События - это любые изменения, на которые может отреагировать программа. События могут быть инициированными пользователем (посредством нажатия клавиш, щелчков кнопкой мыши) и сгенерированными системой (открытие форм, изменение свойств объектов).
  - г. События - это то, что откладывается у вас в памяти. События могут быть запоминающимися и плохо запоминающимися.
2. Для чего в редакторе кодов нужны списки Class Name и Method Name?
  - а. В списке Class Name содержится перечень классных объектов, а в списке Method Name содержится перечень методов, с помощью которых можно сделать классной всю вашу программу.
  - б. Список Class Name содержит перечень всех объектов, для которых могут быть созданы процедуры обработки событий, а список Method Name включает перечень всех событий, на которые могут отреагировать объекты.
  - в. Трудно сказать, ясно лишь одно: не следует усложнять себе жизнь и лучше их вообще не трогать.
  - г. В списке Class Name приводится классификация неопытных и бестолковых пользователей, а в списке Method Name - методы борьбы с ними.



# Использование переменных

*В этой главе...*

- > Чтение данных
- У Использование переменных
- > Определение области видимости переменных
- Представление объектов с помощью переменных

**П**осле того как вы определите для себя, что именно должна делать ваша программа, можете приступить к написанию кодов BASIC. Сразу возникает вопрос: а что должно произойти, когда пользователь наберет имя, адрес или номер телефона в ответ на вопрос, заданный программой? Очевидно, что программа должна принять эту информацию от пользовательского интерфейса и что-то дальше с ней делать. Если какая-то информация должна быть сохранена только в течение некоторого промежутка времени (чтобы затем быть обработанной и измененной), для ее хранения используются *переменные*.

## Чтение данных

Любая информация, которую программа получает извне, обозначается термином *данные*. Почти все, даже самые простые, программы получают данные, обрабатывают их и передают дальше.

Например, текстовый процессор получает информацию от пользователя в виде последовательности набираемых букв и символов, компонует их в виде документа, аккуратно выводит все это на печать, \* — и вот приказ об увольнении половины сотрудников фирмы готов.

Итак, информация на входе, производятся какие-то действия и преобразования, — и информация уже на выходе. Любая программа Visual Basic .NET различает данные двух видов, а именно числа и строки.

*Числа* могут быть положительными и отрицательными, целыми и дробными, и вообще любым сочетанием цифр, которое вы можете себе представить (включая телефонные номера и коды к сейфу, где деньги лежат).

*Строки* — это наборы символов. *Символами* называется все, что вы можете набирать на клавиатуре, включая буквы, знаки пунктуации и (не падайте в обморок) числа.



С технической точки зрения, компьютер понимает только числа. Когда компьютер работает со строками, он использует числа для представления каждого символа.

Программа может интерпретировать наборы цифр и как числа, и как строки. Например, большинство программ воспринимает вводимые пользователем телефонные номера как строки, а, скажем, возраст, вес или суммы потраченных денег — как числа.



Одна буква рассматривается как строка. Одно предложение \* — это тоже строка. Вы можете полностью набрать первый том романа "Война и мир" — и это тоже будет строка. В строке может быть любое количество букв, пробелов, цифр и других символов.



# Переменные и их значения

Набранные вами числа и строки программа в дальнейшем должна будет извлечь и использовать в своей работе. Поэтому она как бы спрашивает себя: "Куда же мне записать эту информацию? Ага, запишу-ка я ее в месте, называемом ТелефонныйНомер (переменная)." Затем, когда эту информацию нужно будет обработать, программа обратится к переменной ТелефонныйНомер и использует эти данные в своей работе.

Переменные могут содержать в себе самые разные данные, которые к тому же время от времени могут меняться (потому они и называются *переменными*). Информация, которую переменная содержит в данный момент, называется *значением* переменной.

## Использование переменных

Все переменные можно разделить на две группы:

- V переменные, которые вы создаете сами;
- ✓ переменные, которые созданы изначально и представляют собой свойства всех объектов пользовательского интерфейса.

Каждый раз, когда вы рисуете в окне формы новый объект, Visual Basic .NET автоматически создает для него целый набор переменных (которые уже известны вам как *свойства*) со значениями, заданными по умолчанию. Просмотреть значения свойств объектов можно в окне Properties (Свойства), для чего нужно нажать клавишу <F4> или выбрать команду View⇒Properties Window.

Значениями свойств могут быть числа (представляющие, например, высоту и ширину объекта), логические значения False и True (определяющие, в частности, будет отображаться объект на экране или нет) и строки (в том числе текст, отображаемый на кнопках или в текстовых полях). Таким образом, свойства отвечают за способ отображения объектов на экране.



Переменные можно рассматривать как имена, которыми обозначаются различные фрагменты информации. Свойства являются специальными именами, которыми обозначается информация об отображении и поведении объектов пользовательского интерфейса.

Чтобы создать собственную переменную, вам нужно вначале объявить ее, по сути, сказав таким способом компьютеру: "Я хочу создать новую переменную и сообщая тебе, какой тип данных она должна содержать".

## Объявление переменных

Чтобы объявить переменную, нужно указать:

- ✓ имя переменной;
- ✓ тип данных, которые будут сохраняться как значения этой переменной.

Для этого вам следует написать такой код:

`Dim ИмяПеременной As ТипДанных`

Существует три причины, по которым Visual Basic .NET требует от вас указывать тип данных для каждой создаваемой переменной.

- ✓ Во-первых, вам самим будет легче потом разобраться, данные какого типа содержатся в той или иной переменной.

- ✓ Во-вторых, предотвращается сохранение в переменной данных других типов, что неминуемо привело бы к некорректной работе программы.
- ✓ И в-третьих, таким образом достигается более эффективное использование памяти, поскольку для хранения данных различных типов требуется разное количества места на диске.



Старайтесь присваивать переменным имена, которые будут как-то описывать хранящуюся в них информацию (например, `ВсегоКоплате` или `КодТовара`). Для Visual Basic .NET безразлично, назовете вы переменную `ПВМОЕСУ` или `ДатаПродажи`. Но если имя переменной не будет ассоциироваться с ее значением, написанная программа будет сложной для понимания, в частности при внесении в нее изменений.

Как уже было отмечено, данные могут быть представлены в виде строк и чисел. Если вы хотите создать переменную, содержащую текстовую информацию и носящую имя `Регион`, вам нужно набрать следующую команду:

```
Dim Регион As String
```

Если переменная будет содержать числовую информацию, нужно будет указать тип числовых данных. В табл. 15.1 приведен список всех возможных типов данных (для Visual Basic .NET) и объемы памяти, которые требуются для хранения одного значения каждого типа.

**Таблица 15.1. Типы данных Visual Basic .NET**

Тип данных	Объем памяти, байт	Диапазон принимаемых данных
Boolean	2	True (1) и False (0)
Byte	1	От 0 до 255
Char	2	От 0 до 65535
Date	8	Даты от 01.01.0001 до 31.12.9999
Decimal	16	+/-79228162514264337593543950335 без десятичной запятой; +/-7,9228162514264337593543950335 с 28 знаками после запятой
Double	8	От -1,79769313486231E+308 до -4,94065645841247E-324 (отрицательные числа); от 4,94065645841247E-324 до 1,79769313486231E+308 (положительные числа)
Integer	4	От -2 147 483 648 до 2 147 483 647
Long	8	От -9 223 372 036 854 775 808 до 9 223 372 036 854 775 807
Short	2	От -32 768 до 32 767
Single	4	От -3,402823E+38 до -1,401298E-45 (отрицательные числа); от 1,401298E-45 до 3,402823E+38 (положительные числа)
String	Разный	От 0 до почти 2 миллиардов унифицированных символов



Под унифицированными символами подразумевается специальный стандарт, который каждой букве, цифре, знаку, а также буквам иностранных языков ставит в соответствие свой уникальный номер. Более подробную информацию об этом стандарте вы можете найти по адресу [www.unicode.org](http://www.unicode.org).

Итак, если вы хотите создать переменную, именуемую Возраст, можете набрать такой код:

```
Dim Возраст As Integer
```

или такой:

```
Dim Возраст As Byte
```

Отличие между типом данных Byte и Integer заключается в том, что Byte-переменная может принимать значения из диапазона от 0 до 255, в то время как Integer-переменная может принимать значения из диапазона от -2 147 483 648 до 2 147 483 647. Но поскольку возраст человека вряд ли превысит 255 лет, лучше объявить, что переменная имеет тип Byte, тем более что в этом случае она будет занимать только 1 байт памяти. А если вы объявите, что она имеет тип Integer, то потребуется 4 байта памяти.



Старайтесь для каждой переменной из подходящих для нее типов данных использовать тот, который занимает наименьший объем памяти. В таком случае и вся ваша программа для своего выполнения потребует меньшего объема оперативной памяти, а следовательно, будет работать более эффективно.



Попытка присвоить переменной значение, превышающее наибольшее из возможных (или меньшее, чем самое малое), которые допустимы для данных ее типа, приведет к сбою в работе программы. Например, если переменной, имеющей тип Byte, присвоить, скажем, значение 2000, Visual Basic .NET воспримет это как ошибку и прекратит выполнение программы.

Можно также в одной строке объявить сразу несколько переменных:

```
Dim Переменная1, Переменная2 As ТипДанных
```

Например, если нужно определить, что переменные Доход и Расход будут иметь тип Integer, наберите следующий код:

```
Dim Доход, Расход As Integer
```



Но в Visual Basic .NET есть и другой способ объявления переменных: вместо кода

```
Dim МойТекст As String
```

можно набрать

```
Dim МойТекст$
```

Первый код более длинный, но зато более понятный. Во втором случае использовано нечто, называемое *символами объявления типов*. Набрать такой код проще и быстрее, но прочесть и понять его по прошествии некоторого времени будет сложнее.



Если вы хотите, чтобы код вашей программы был легко читаемым, и вас не пугает перспектива набора таких длинных слов, как `As String` или `As Integer`, используйте первый метод. Но если вы хотите сэкономить время и усилия, если вас не волнует, как будет читаться код программы, применяйте второй метод. Ниже приведена небольшая таблица, содержащая символы объявления типов и примеры их использования.

Тип данных	Символ	Пример	Что соответствует
Decimal	@	Dim Loot@	Dim Loot As Decimal
Double	#	Dim Chip#	Dim Chip As Double
Integer	%	Dim Age%	Dim Age As Integer
Long	&	Dim Doc&	Dim Doc As Long
Single	!	Dim Hook!	Dim Hook As Single
String	\$	Dim Name\$	Dim Name As String

## Присвоение имен переменным

Вы можете назвать переменную любым именем и присвоить ей любое значение. Однако неразумно называть переменную, предположим, именем ТелНомер и присваивать ей значение, являющееся чьим-то почтовым адресом.



Присваивайте переменным имена, определяющие каким-то образом данные, которые будут в них храниться. Например, если вы назвали переменную именем ТелНомер, то ее следует использовать для хранения именно телефонных номеров.

Давая переменным имена, вы должны неукоснительно придерживаться перечисленных ниже правил, иначе Visual Basic .NET не примет их. Итак, все имена переменных должны:

- ✓ начинаться с букв;
- ✓ состоять максимум из 255 символов (минимум — из одного символа);
- ✓ состоять только из букв, цифр и символов подчеркивания ( \_ ); пробелы и знаки пунктуации здесь не допустимы;
- ✓ не быть зарезервированными в Visual Basic .NET словами, такими как End или Sub.

Если все назначенные вами имена будут удовлетворять перечисленным критериям, все будет хорошо. (Это, конечно же, не гарантирует, что ваша программа сразу начнет работать так, как вы хотите, но, по крайней мере, к вам претензий со стороны Visual Basic .NET не будет.) Примеры допустимых в Visual Basic .NET имен приведены ниже.

Телефон

Доход\_за\_прошлый\_месяц

Ставка75

А вот примеры имен, на использование которых Visual Basic .NET ни за что не согласится.

10Негритят (Начинается с цифры.)

Десять негритят (Содержит пробел.)

Sub (Совпадает с зарезервированным в Visual Basic .NET ключевым словом.)



Чтобы по имени переменной можно было определить, к какому типу данных она носит, некоторые программисты начинают такие имена с трехсимвольных пфиксов, например:

intПремияльные (Нетрудно понять, что для переменной определен тип Integer.)

lngПроценты (Такой префикс означает, что для переменной определен тип Long.)

strПетиция (Эта переменная имеет тип String.)

Поскольку префиксы смотрятся не очень эстетично (хотя с точки зрения правильности присвоения имен — вполне корректно), в наших примерах встречаться они будут крайне редко.

### Тест на проверку полученных вами знаний

1. Для чего **написан** следующий код BASIC?

```
Dim Mon As Integer
Dim Dollars
```

а. Этот код **должен ввести** в заблуждение других программистов, которые попытаются внести изменения в мою программу.

б. Таким образом объявляются переменные Mon (**тип Integer**) и Dollars (**тип Decimal**).

в. Код оценивает рыночную стоимость программы и определяет, за сколько **долларов ее можно продать**.

г. Код **запускает надстройку Visual Basic .NET**, конвертирующую **монгольские тулрики** в **надские доллары** и наоборот.

2. Зачем нужно объявлять переменные?

а. Чтобы другие программисты знали, что данные переменные вы забронировали для собственного использования.

б. Объявленные переменные впоследствии будут опубликованы в Internet с целью установления гиперссылок на вашу программу.

в. **Это связано с защитой авторских прав.** Объявив переменные, вы становитесь их собственником и можете рассчитывать **на** свою долю прибыли **от реализации создаваемой программы**.

г. Объявляя переменную, вы даете компьютеру **указание создать переменную** определенного **типа** и присвоить ей указанное имя.

## Присвоение переменным числовых значений

Теперь, когда вы знаете, как создать переменную и присвоить ей имя, возникает другой вопрос: а как присвоить этой переменной конкретное значение? Ответ очень прост: нужно использовать обычный знак равенства (=).

Чтобы присвоить переменной значение, наберите следующий код BASIC:

ИмяПеременной = Значение

Итак, вам не нужно говорить компьютеру: "Последний раз тебе повторяю: или ты присваиваешь переменной Ириска значение 36, или я выдергиваю штепсель из розетки!" Достаточно будет набрать код:

Ириска = 36



Переменная в каждый отдельный момент времени может иметь только одно значение. Если переменная уже имеет какое-то значение, но вы захотите присвоить ей новое, старое значение бесследно исчезает.

Вы можете написать сразу две команды:

Ириска = 36

Ириска = 57

Visual Basic .NET посмотрит на первую строку и скажет себе: "Ага, переменной Ириска нужно присвоить значение 36". Потом он, взглянув на вторую строку, присвоит этой переменной значение 57, а о значении 36 напрочь забудет.

## Присвоение переменным текстовых значений

Текстовые значения присваиваются точно так же, как и числовые. Единственное различие состоит в том, что текстовое значение должно быть заключено в кавычки. Благодаря этому Visual Basic .NET может понять, где оно начинается, а где заканчивается.

Например, можно присвоить значение, состоящее из одного слова:

Имя = "Майкл"

Или из двух слов:

Имя = "Майкл Джексон"

Или из любого другого количества слов, заключенных в кавычки:

Имя = "Степан Эдуардович Розенштейн-младший"

Не все текстовые переменные состоят из букв. Если вам нужно, скажем, чтобы значением переменной был телефонный номер или, например, некий секретный код, наберите следующее:

ТелНомер = "555-1234"

СекретныйКод = "123-45-6789"

Вы спросите, а что произойдет, если кавычки не будут набраны? Если просто набрать

ТелНомер = 555-1234

СекретныйКод = 123-45-6789

Без кавычек эти значения будут восприняты не как строки, а как числа, над которыми к тому же еще нужно произвести операцию вычитания. Поэтому вместо телефонного номера переменная ТелНомер примет значение, равное -679 (это разница 555-1234). В свою очередь, переменная СекретныйКод примет значение -6711.



Итак, вовсе не важно, состоит значение из букв или из цифр, но если вы хотите, чтобы оно воспринималось как текст, возьмите его в кавычки.

## Присвоение переменным значений других переменных

Помимо непосредственного присвоения переменным конкретных текстовых или числовых значений, одним переменным можно присваивать значения других переменных. Сделать это можно, набрав следующий код BASIC:

ПерваяПеременная = ВтораяПеременная

Например, если переменной Василий нужно присвоить значение переменной НизкийIQ, достаточно ввести код

Dim НизкийIQ, Василий As Byte

НизкийIQ = 9

Василий = НизкийIQ



1. Первая строка дает Visual Basic .NET указание: "Создай две переменные, назови их НизкийIQ и Василий, и чтобы каждая из них могла принимать значения из диапазона от 0 до 255 (тип данных Byte)".
2. Вторая строка говорит Visual Basic .NET: "Присвой переменной НизкийIQ значение 9".
3. А третья строка приказывает: "Присвой переменной Василий значение переменной НизкийIQ". Поскольку переменная НизкийIQ имеет в данный момент значение 9, переменной Василий также будет присвоено значение 9.

## Присвоение переменным значений свойств объектов

Чтобы отобразить какое-то сообщение на экране, нужно изменить свойства надписи или текстового поля. Следовательно, изменяя свойства объектов, вы сообщаете пользователю некую информацию.

Поскольку свойства объектов, по сути, являются теми же переменными, присваивать им новые значения можно точно так же, как и другим переменным. Например, если вы хотите, чтобы кнопка `Button1` имела высоту, равную 43 пунктам, наберите следующий код:

```
Button1.Height = 43
```

Таким образом вы говорите Visual Basic .NET: "Найди объект, названный `Button1`, и присвой его свойству `Height` значение 43".

Если вы хотите отобразить на экране надпись, измените значение свойства **Text** текстового поля или этой надписи, например:

```
TextBox1.Text = "Этот текст появится на экране."
```



Единственным свойством, которое нельзя изменить, набрав код BASIC, является свойство **Name**. Изменить значение этого свойства (т.е. присвоить объекту новое имя) можно только с помощью окна **Properties** (Свойства) в режиме конструктора.

## Область видимости переменных

При выполнении программы ошибки возникают чаще всего тогда, когда одна часть программы начинает использовать значения переменных, которые предназначены для применения в другой ее части. Чтобы не вносить путаницу и отделить одни данные от других, Visual Basic .NET использует прием, называемый, согласно терминологии программирования, *определением области видимости переменных*.

*Область видимости* переменных определяет, в какой части программы может быть использована та или иная переменная. В Visual Basic .NET переменная может быть видимой в пределах:

- ✓ блока;
- ✓ процедуры;
- ✓ модуля;
- ✓ проекта.

### Переменные, видимые в пределах блока

Такие переменные имеют наименьшую область видимости, поскольку она ограничена пределами одного блока, такого например, как `If-Then-End If` (см. главу 22) или `For-Next` (см. главу 25). Чтобы создать переменную, видимую только в пределах своего блока, нужно с использованием ключевого слова `Dim` объявить ее внутри этого блока. Например:

```
If Доход > Расход Then
    Dim Message As String
    Message = "Поздравляем! Проект рентабелен."
    TextBox1.Text = Message
End If
```

В приведенном примере Visual Basic .NET создает во второй строке переменную Message, которой ранее не существовало. Затем, когда будет достигнута последняя строка, End If, Visual Basic .NET забудет об указанной переменной, как будто ее вовсе и не было.



Хотя такие переменные используются только пока выполняются коды их блока, они запоминают последние присвоенные им значения. Поэтому, если в дальнейшем программа будет опять выполнять коды этого же блока, не забудьте присвоить переменным исходные значения, чтобы не возникали ошибки из-за повторного использования старых значений.

## Переменные, доступные в пределах процедуры

Процедура — это небольшая подпрограмма, создаваемая, как правило, для решения какой-то одной задачи. Например, процедура обработки событий указывает компьютеру, что нужно делать, если пользователь, скажем, щелкнул на кнопке Button1. Переменные, которые создаются специально для таких процедур, видимы только в их пределах. Таким образом, если возникает ошибка и переменная принимает неправильное значение, вы легко можете определить источник возникновения ошибки (он будет где-то в пределах данной процедуры). Более подробную информацию о процедурах, и о процедурах обработки событий в частности, вы можете найти в главах 27 и 14.

Чтобы создать переменную, чьей областью видимости будет определенная процедура, объявите ее в самом начале этой процедуры. Например:

```
Private Sub Button1_Click(ByVal sender As System.__  
    Object, _ByVal e As System.EventArgs)_  
    Handles Button1.Click  
    Dim Количество As Integer  
  
End Sub
```

Теперь любая команда, набранная между строками Dim Количество As Integer и End Sub, может использовать значение переменной Количество. С другой стороны, для кодов всей остальной программы этой переменной не существует.

## Переменные, доступные в пределах модуля

Переменные данного типа могут быть использованы всеми кодами BASIC, сохраненными в том же файле, где эти переменные объявлены. Такой файл может быть файлом формы или файлом модуля, о котором достаточно подробно рассказывается в главе 27. Как и файл формы, файл модуля обычно состоит из одной или нескольких процедур.



Поскольку переменные, видимые в пределах модуля, могут быть использованы любыми принадлежащими ему кодами, то в случае возвращения переменной неверного значения для поиска источника возникновения ошибки вам придется просматривать все коды этого файла.

Чтобы создать переменную, видимую в пределах всего файла, замените ключевое слово Dim словом Private и наберите приведенную ниже строку в начале кодов в файле модуля или сразу после строки Public Class в файле формы.

```
Private ИмяПеременной As ТипДанных
```



Увидев ключевое слово `Private`, Visual Basic .NET определит, что эта переменная создается для использования любыми кодами BASIC в пределах данного файла. Если ваша программа будет состоять из нескольких файлов, каждая из таких переменных будет доступной для использования только в пределах своего файла.

## Глобальные переменные

Глобальными называются те переменные, чьей областью видимости является вся программа. Другими словами, переменные указанного типа могут быть использованы любым кодом данной программы. Применять их следует очень осторожно, поскольку, если такая переменная примет неверное значение, вам придется в поисках ошибки просматривать коды всей программы.

Если ошибку возвращает переменная, видимая в пределах модуля, понятно, что источник ее возникновения нужно искать в том файле, где эта переменная была объявлена. Если же ошибочное значение принимает переменная, видимая в пределах блока или процедуры, круг поиска ошибки сужается до размеров соответствующего блока или процедуры.

Чтобы создать глобальную переменную, замените ключевое слово `Dim` словом `Public` и наберите следующую строку в начале кодов любого файла программы:

```
Public ИмяПеременной As ТипДанных
```

## использование переменных для представления объектов

В основном переменные используются для представления различных типов данных (чисел или строк). Однако переменные могут быть использованы и для представления сразу целых объектов. Наиболее часто объектом, представляемым с помощью переменных, является форма.



Часть VII настоящей книги посвящена объектно-ориентированному программированию, поэтому, если сейчас вам идея использования переменных для представления объектов не очень понятна, сильно не переживайте.

Допустим, вам нужно открыть какую-то форму в процессе выполнения программы. Для этого понадобится набрать такой код:

```
Dim Форма As ИмяФормы
Форма = New ИмяФормы()
Форма.Show()
```



1. Первая строка говорит Visual Basic .NET: "Создай переменную, которая будет представлять собой форму". Если вы хотите, к примеру, создать переменную `МояФорма` (ей можно присвоить любое другое имя), которая будет представлять форму `frmWindow` (сохраненную в файле `frmWindow.vb`), наберите код  
`Dim МояФорма As frmWindow`
2. Вторая строка дает Visual Basic .NET указание: "Создай новую копию формы, которая представлена переменной `Форма`".
3. А третья строка приказывает Visual Basic .NET: "Отобрази на экране объект, представленный переменной `Форма`". В данном случае заданная переменная представляет собой форму, именуемую `ИмяФормы`.

# Получение данных от пользователя

*В этой главе...*

- Использование свойства **Text**
- Использование свойства **Checked**
- Получение числовой информации
- Выбор сразу нескольких элементов из списка

**7.7** Пользовательский интерфейс является лицом вашей программы, и, разумеется, это лицо должно иметь достойный и даже привлекательный вид. Но, к сожалению, для хорошей программы этого вовсе не достаточно. Если вы не хотите, чтобы ваша программа напоминала симпатичную, но совершенно глупую девицу, которая не может связать двух слов, позаботьтесь о том, чтобы она была в состоянии адекватно реагировать на действия пользователей и поддерживать с ними разумный диалог.

Чтобы пользовательский интерфейс действительно заработал, программа должна:

- ✓ принять информацию от пользователя;
- ✓ обработать полученные данные и вычислить результат;
- ✓ отобразить конечный результат на экране.

Например, если пользователь выбрал некоторый элемент в поле со списком, программа не имеет ни малейшего понятия о том, какой выбор был сделан. Вам же выбранный элемент хорошо виден на экране, поэтому вы, наверное, уже сказали компьютеру: “Ну и глупая же ты машина! Если я вижу, какой элемент выбран, то почему же ты этого не видишь?”

Однако то, что видите на экране вы, совсем не обязательно должен знать компьютер. Чтобы объяснить ему, какие именно действия были предприняты пользователем, нужно написать код BASIC, который должен собрать информацию, сохраненную в качестве значений свойств объектов. Наиболее часто для хранения данных, полученных от пользователя, используются следующие свойства объектов:

- ✓ **Text** (для хранения текстовой информации);
- ✓ **Checked** (для хранения логических значений);
- ✓ **Value** (для хранения числовой информации).

## *использование свойства Text для получения текстовой информации*

Свойство **Text** объектов пользовательского интерфейса предназначено для хранения текстовых данных, полученных от пользователя. Все значения этого свойства имеют тип **String**. Вот перечень объектов, которые используют свойство **Text** для получения текстовой информации:

- ✓ **TextBox** (Текстовое поле);
- ✓ **ComboBox** (Поле со списком);

- 1 ✓ ListBox (Список);
- 2 ✓ CheckedListBox (Список флажков);
- 3 ✓ RichTextBox (Расширенное текстовое поле);
- 4 ✓ DomainUpDown.

Текстовое поле, расширенное текстовое поле и поле со списком позволяют пользователю самому набрать текст. (Если для свойства Style поля со списком установлено значение DropDownList, пользователь сможет выбрать только одно из заранее определенных значений.)



Поскольку пользователь имеет возможность набрать для этих объектов любой текст (включая ругательства и разного рода бесполезную информацию), вы можете написать дополнительные коды BASIC для отсеивания неприемлемых данных. В противном случае, если пользователь введет данные, которые программа не в состоянии будет правильно интерпретировать, она может зависнуть, прекратить работу или просто выдать неверный результат.

Список, поле со списком, список флажков и объект DomainUpDown отображают готовые варианты ответа, из числа которых пользователь и делает свой выбор. Поскольку все возможные варианты ответа определены заранее, от пользователя может быть принято только допустимое значение.

Чтобы программа определила, какую текстовую информацию оставил для нее пользователь, переменной нужно присвоить то же значение, которое сохранено в свойстве Text. Вот как это делается:

*ИмяПеременной* = *ИмяОбъекта*.Text



Вам нужно заранее объявить о создании переменной, имеющей тип String, и вместо слова *ИмяОбъекта* набрать название реального объекта. Например, если вам нужно узнать, какую информацию оставил пользователь в текстовом поле txtMessage, и сохранить таковую в качестве значения переменной strInfo, наберите следующий код BASIC:

```
Dim strInfo As String
strInfo = txtMessage.Text
```



Текстовые поля можно также использовать для получения от пользователя числовой информации. Но при этом не следует забывать, что Visual Basic .NET интерпретирует введенные в текстовые поля числа как текст. Чтобы преобразовать строки в числа, следует воспользоваться специальными командами BASIC, наподобие CInt и CSng. О том, как строки преобразовываются в числовые значения различных типов данных, подробно рассказано в главе 18.

## Получение логической информации

Среди элементов пользовательского интерфейса есть два типа объектов, которые почти всегда отображаются группами. Это флажки и переключатели. Чтобы определить, какой из флажков или переключателей был выбран пользователем, нужно проверить для каждого из них значение свойства Checked.

Это свойство может принимать лишь одно из логических значений (тип данных Boolean):

- 1 ✓ True (объект выбран);
- 2 ✓ False (объект не выбран).

Напротив невыбранного флажка или переключателя отображается пустое окошко, напротив выбранного — • флажок или большая точка.



Для использования данных, сохраненных в качестве значений свойства Checked, использовать отдельную переменную необязательно. Вы можете ссылаться в кодах BASIC непосредственно на значение свойства Checked. Например, вам необязательно писать следующий код:

```
Dim Flag As Boolean
Flag = RadioButton1.Checked
If Flag = True Then
    TextBox1.Text = "Соглашение достигнуто!"
End If
```

Вместо этого достаточно набрать

```
If RadioButton1.Checked = True Then
    TextBox1.Text = "Соглашение достигнуто!"
End If
```

Более того, вы можете даже опустить = True и набрать только

```
If RadioButton1.Checked Then
    TextBox1.Text = "Соглашение достигнуто!"
End If
```



Несколько подробнее об использовании логических значений в блоках If-Then будет рассказано в главе 22. Сейчас же вам нужно лишь просмотреть приведенные выше примеры, чтобы получить о них хотя бы общее представление.

## Получение числовых данных

Если значение, которое определяет пользователь, может принадлежать к очень большому диапазону (например, один раз будет введено число 0,0093, а в другой раз — число 85 157), для получения информации лучше использовать текстовые поля. Но в большинстве случаев возможные значения будут принадлежать небольшому диапазону чисел, например от 0 до 100.

Если вы хотите, чтобы пользователь выбрал значение из заранее определенного интервала, можете использовать один из следующих объектов пользовательского интерфейса:

- ✓ NumericUpDown (тип данных Decimal);
- ✓ Horizontal/VerticalScrollBar (тип данных Integer);
- ✓ TrackBar (тип данных Integer);
- ✓ DateTimePicker (тип данных Long).

Вместо того чтобы заставлять пользователя набирать какое-то число, перечисленные выше объекты позволяют прокрутить список возможных значений в поиске такового и просто щелкнуть на нужном значении, которое потом будет сохранено как значение свойства Value.

Чтобы сохраненными данными можно было оперировать, переменной нужно присвоить значение свойства Value. Для этого наберите такой код:

```
ИмяПеременной = ИмяОбъекта.Value
```



Вам нужно заранее объявить о создании переменной, имеющей определенный тип данных. Вместо кода *ИмяОбъекта* нужно набрать название реального объекта. Например, если вам нужно узнать, какое число сохранено объектом `TrackBar`, именуемым `TrackBar1`, и сразу же сохранить его в качестве значения переменной `intHoMep`, наберите следующий код BASIC:

```
Dim intHoMep As Integer  
intHoMep = TrackBar1.Value
```

## Выбор из списка сразу нескольких элементов

Если при работе со списком или полем со списком вы присвоите свойству `SelectionMode` значение `MultiSimple` или `MultiExtended`, пользователь, удерживая нажатой клавишу `<Shift>` или `<Ctrl>`, сможет выбрать из списка элементов сразу несколько возможных вариантов. В этом случае свойство `Text` запомнит значение только первого выбранного элемента.

Чтобы определить, какие именно элементы списков были выбраны пользователем, нужно воспользоваться значениями свойств `SelectedItem` и `SelectedIndices`.

В свойстве `SelectedItem` содержатся значения выбранных элементов, а в свойстве `SelectedIndices` — их номера. Например, если пользователь выбрал первый и третий элементы списка, свойство `SelectedIndices` будет содержать значение 0 (это номер первого элемента) и значение 2 (номер третьего элемента).

## Сколько элементов выбрал пользователь?

Чтобы определить, сколько именно элементов выбрал пользователь, наберите следующий код BASIC:

```
Dim Количество As Integer  
Количество = Список.SelectedItems.Count
```

В приведенном выше примере переменной вместо имени `Количество` можно присвоить любое другое имя, а вместо слова `Список` нужно указать настоящее название вашего списка или поля со списком.



Количество выбранных пользователем элементов можно подсчитать только для списка или поля со списком, у которого свойству `SelectionMode` присвоено значение `MultiSingle` или `MultiExtended`.

## Определение выбранных пользователем элементов

Если вы знаете, сколько элементов списка было выбрано пользователем, то можете создать цикл (циклы подробно описаны в главах 24 и 25), чтобы получить значения всех этих элементов, которые Visual Basic .NET хранит как значения свойства `SelectedItem`. Значение отдельно выбранного элемента определяется кодом

```
ИмяСписка.SelectedItems.Item(x)
```

Здесь вместо слова `ИмяСписка` нужно указать действительное название вашего списка или поля со списком, а вместо буквы `x` — номер выбранного элемента. Например, если поль-

зователь выбрал три элемента из списка, именуемого `ListBox1`, первому из них соответствует нулевой номер, а его значение определяется кодом

```
ListBox1.SelectedItems.Item(0)
```

Второй выбранный элемент имеет номер 1, а его значение, соответственно, определяется кодом

```
ListBox1.SelectedItems.Item(1)
```

Третий выбранный элемент имеет порядковый номер 2 и т.д., что показано на рис. 16.1.

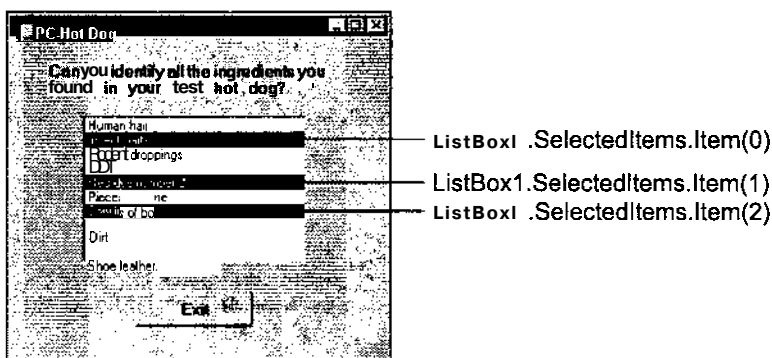


Рис./6.1. Пользователь выбрал в этом списке сразу несколько элементов



Если пользователь выбирает в списке сразу несколько элементов, сохраненные Visual Basic .NET значения элементов имеют тип данных `Collection` (будет рассмотрен в главе 20). После извлечения эти данные нужно будет преобразовать в тип `String`, используя для этого команду `CStr`.

Ниже приведен пример кода BASIC, определяющего значения всех выбранных элементов списка.

```
Dim K As Integer
For K = 0 to (BoxName.SelectedItems.Count - 1)
    MsgBox (CStr(BoxName.SelectedItems.Item(K), , _
        "Этот элемент был выбран")
Next
```



1. В первой строке создается переменная `K`, имеющая тип `Integer`.
2. Во второй строке создается цикл `For-Next`, начинающийся нулевой итерацией и заканчивающийся итерацией, номер которой на единицу меньше количества выбранных пользователем элементов. Например, если было выбрано четыре элемента, цикл начнется нулевой, а закончится третьей итерацией, т.е. всего их будет четыре. (Если вам сейчас это кажется не очень понятным, не переживайте, так как более подробно цикл `For-Next` описывается в главе 25.)
3. В третьей строке для отображения окна с сообщением используется команда `MsgBox`. Заголовком окна будет надпись "Этот элемент был выбран", а сообщением — сам выбранный элемент. На каждой итерации для отображения следующего выбранного элемента открывается новое окно. Таким образом, если пользователь выбрал, предположим, три элемента, окно с сообщением тоже будет открываться три раза.
4. Четвертая строка обозначает конец цикла `For-Next`.

### Тест на проверку полученных вами знаний

1. Что делает следующий код BASIC:  
`ЧтоЭто = VScrollBar.Value`
  - а. Заставляет Visual Basic .NET вывести на экран подсказку, объясняющую, чем является `VScrollBar.Value`.
  - б. Если Visual Basic .NET не может понять, для чего вы набрали `VScrollBar.Value`, на экране появляется такой вопрос.
  - в. Ничего не делает, поскольку для Visual Basic .NET такой тон обращения является оскорбительным.
  - г. Переменной `ЧтоЭто` присваивается значение свойства `Value` объекта `VScrollBar`.
2. При каком условии в списке или в поле со списком можно выбрать сразу несколько вариантов?
  - а. Только в том случае, если там имеются эти несколько вариантов.
  - б. Главное, чтобы пользователь действительно захотел выбрать сразу несколько вариантов, а при большом желании, как известно, все возможно.
  - в. Если программа качнет "глючить" и "забудет", что какой-то вариант уже был выбран, тогда можно будет выбрать еще парочку вариантов.
  - г. Если свойству `SelectionMode` этого списка или поля со списком присвоено значение `MultiSimple` или `MultiExtended`.

# Займемся математикой

*В этой главе...*

- > Сложение, вычитание, умножение и деление чисел
- > Операторы Not, And, Or и Xor
- > Операторы сравнения
- Приоритет операторов

**П**оступившие от пользователя данные (будь то числовые значения или текстовые) необходимо обработать, т.е. произвести над ними вычисления и получить результат.

Для получения любого нового результата данные, поступившие от пользователя, нужно определенным образом изменить — модифицировать, преобразовать или что-то еще с ними сделать. Любые изменения подразумевают выполнение определенных операций, а специальные команды, изменяющие данные для получения нового результата, называются *операторами*.

В языке Visual Basic .NET существует три вида операторов:

- ✓ арифметические;
- ✓ логические;
- ✓ операторы сравнения.

## Арифметические операторы

*Арифметические* операторы, по существу, превращают ваш красивый дорогой компьютер в пятидолларовый калькулятор. Они занимаются сложением, вычитанием, умножением, делением чисел и числовых переменных. Наиболее часто употребляемые арифметические операторы представлены в табл. 17.1.

**Таблица 17.1. Арифметические операторы**

Оператор	Выполняемая операция
+	Сложение двух чисел
-	Вычитание
*	Умножение
/	Деление, результатом которого будет число с плавающей запятой (тип Decimal), например 3,14; 15,2; 652,1248
\	Деление, результатом которого будет целое число
Mod	Вычисление остатка от деления
^	Возведение в степень
&	Объединение (конкатенация) двух строк





Тип переменных, участвующих в одной операции, обязательно должен совпадать. Если переменные, которые должны участвовать в вычислениях, относятся к разному типу, приведите их вначале к одному типу данных. О том, как это можно сделать, рассказано ниже, в разделе “Изменение типов переменных”.

## Сложение двух чисел

При необходимости к одному числу добавить другое используется оператор сложения ( + ), например;

```
Dim X, Y, Sum As Single
```

```
X = 10
```

```
Y = 15.4
```

```
Sum = X + Y
```

В данном случае переменной Sum присваивается значение 10+15, 4, или 25, 4.



Для большей лаконичности можно использовать оператор +=, например:

```
X += Y
```

Этот код эквивалентен следующему:

```
X = X + Y
```

## Вычитание чисел

Для того чтобы от одного числа отнять другое, воспользуйтесь оператором вычитания ( - ), например:

```
Dim Прибыль, Доход, Расход As Integer
```

```
Доход = 2500
```

```
Расход = 1500
```

```
Прибыль = Доход - Расход
```

Переменной Прибыль будет присвоено значение 1000 (2500-1500).



Если от значения переменной нужно отнять число, а полученный результат присвоить той же переменной, воспользуйтесь оператором -=, например:

```
X -= Y
```

Данный код равнозначен такому коду:

```
X = X - Y
```

## Получение отрицательных чисел

Применив оператор вычитания к одной переменной, можно поменять ее значение на противоположное, т.е. положительное число сделать отрицательным и наоборот. Например:

```
Dim Дебет, Кредит As Integer
```

```
Дебет = 500
```

```
Кредит = - Дебет
```

В этом случае переменной Кредит будет присвоено значение -500.

## Умножение чисел

Если нужно одно число умножить на другое, воспользуйтесь оператором умножения ( \* ), как показано в следующем примере:

Dim Цена, Количество, Сумма As Single

Цена = 25

Количество = 8

Сумма = Цена \* Количество

Переменной Сумма будет присвоено значение 200 (25\*8).



Если значение переменной нужно умножить на определенное число, а полученный результат присвоить этой же переменной, воспользуйтесь оператором `*`, например:

`X *= Y`

Этот код эквивалентен коду

`X = X * Y`

## Операция деления

При необходимости разделить одно число на другое и получить точный результат, представленный числом с плавающей запятой (тип `Decimal`), используйте оператор деления, обозначаемый прямой косой чертой ( `/` ). Например:

Dim Игры, Баллы, СреднийБалл As Single

Игры = 162

Баллы = 104

СреднийБалл = Баллы / Игры

Переменной СреднийБалл будет присвоено значение 0,6419753 (104/162).



Если значение одной переменной нужно разделить на число и полученный результат присвоить той же переменной, воспользуйтесь оператором `/=`, например:

`X /= Y`

Этот код равнозначен коду

`X = X / Y`

В случае, если возвращаться должна только целая часть числа, полученного в результате деления (т.е. если нужно получить целочисленный результат), применяется оператор деления, представленный обратной косой чертой ( `\` ). Например:

Dim Грузоподъемность, ВесБлока, Принять As Integer

Грузоподъемность = 1900

ВесБлока = 72

Принять = Грузоподъемность \ ВесБлока

Вот как Visual Basic .NET будет интерпретировать эти команды



1. Первая строка говорит Visual Basic .NET: "Создай переменные Грузоподъемность, ВесБлока и Принять типа `Integer`".
2. Вторая строка говорит: "Присвой переменной Грузоподъемность значение 1900".
3. Третья строка дает указание: "Присвой переменной ВесБлока значение 72".
4. А четвертая строка говорит: "Присвой переменной Принять значение, равное целой части числа, полученного в результате деления значения переменной Грузоподъемность на значение переменной ВесБлока". Переменной Принять будет присвоено значение 26.

Чтобы лучше понять, какие при этом производятся действия, если в качестве оператора используется обратная косая черта, рассмотрим еще один пример.

Число1 = 2.5

Число2 = 1.5

Результат = Число1 \ Число2

Прежде чем выполнить операцию деления, Visual Basic .NET округляет исходные значения до ближайших целых чисел. (Если значение расположено посередине между двумя целыми числами, например между 1,5 и 2,5, оно округляется до большего целого числа.) В нашем примере значение переменной Число1 округляется до числа 3, а значение переменной Число2 — до числа 2. Результатом деления числа 3 на число 2 будет число 1,5, но поскольку оператор “обратная косая черта” возвращает только целую часть от полученного результата, переменной Результат будет присвоено значение 1.



Для большей лаконичности можете использовать оператор  $\backslash$  =, например:

$X \backslash = Y$

Этот код равнозначен следующему коду:

$X = X \backslash Y$

## Применение оператора Mod

Если нужно определить остаток от деления двух чисел, используйте оператор Mod. Рассмотрим это на следующем примере.

Dim Грузоподъемность, ВесБлока, Остаток As Integer

Грузоподъемность = 1900

ВесБлока = 72

Остаток = Грузоподъемность Mod ВесБлока



1. Первая строка говорит Visual Basic .NET: “Создай переменные Грузоподъемность, ВесБлока и Остаток типа Integer”.
2. Вторая строка: “Присвой переменной Грузоподъемность значение 1900”.
3. Третья строка: “Присвой переменной ВесБлока значение 72”.
4. Четвертая строка: “Присвой переменной Остаток значение, равное остатку от деления значения переменной Грузоподъемность на значение переменной ВесБлока”. Переменной Остаток будет присвоено значение 28.

## Возведение в степень

*Возведением в степень* называется умножение числа самого на себя определенное количество раз. Например, если число 2 умножить четыре раза само на себя, т.е.  $2 * 2 * 2 * 2$ , получим 2 в четвертой степени (на письме обозначается как  $2^4$ ).

Поскольку вы не можете набрать  $2^4$ , а набирать  $2 * 2 * 2 * 2$  не очень удобно, Visual Basic .NET располагает для таких случаев оператором возведения в степень (  $\wedge$  ). Чтобы получить нужный результат, достаточно набрать

$2 \wedge 4$



Для большей лаконичности можете использовать оператор  $\wedge$  =, например:

$X \wedge = Y$

Этот код равнозначен коду

$X = X \wedge Y$

## Конкатенация двух строк

*Конкатенацией* двух строк называется объединение их в одну строку. Для этого используется оператор & (амперсанта). Рассмотрим операцию объединения двух строк на следующем примере.

```
Dim Имя, Фамилия, ПолноеИмя As String
```

```
Имя = "Джон "
```

```
Фамилия = "Ленон"
```

```
ПолноеИмя = Имя & Фамилия
```



При объединении строк не забывайте в конце первой строки добавлять пробел или вставляйте пробел как отдельную строку между двумя другими строками. В противном случае буквы двух строк просто сольются в одно слово, например так: ДжонЛенон.

Вот как Visual Basic .NET интерпретирует приведенный выше код.



1. Первая строка говорит: "Создай три переменные типа String и назови их Имя, Фамилия и ПолноеИмя".
2. Вторая строка: "Присвой переменной Имя значение Джон". Обратите внимание, что присваиваемое значение заканчивается пробелом.
3. Третья строка: "Присвой переменной Фамилия значение Ленон".
4. А четвертая строка дает указание: "Объедини значения переменных Имя и Фамилия и присвой полученное значение переменной ПолноеИмя". В результате такой операции переменной ПолноеИмя будет присвоено значение Джон Ленон.



С целью конкатенации строк может использоваться как символ амперсанта (&), так и знак "плюс" (+). Но поскольку последний применяется и для сложения чисел, для обозначения операции объединения строк лучше его не использовать, поскольку в этом случае сложнее будет читать коды вашей программы.



Для большей лаконичности при написании кодов для объединения строк можно использовать оператор &=, например:

```
Имя &= Фамилия
```

Этот код эквивалентен следующему коду:

```
Имя = Имя & Фамилия
```

## Изменение типов переменных

Типы переменных, над которыми выполняются арифметические операции, должны быть одинаковыми (например, все переменные должны иметь тип Integer) или подобными (каковыми являются, в частности, тип Single и тип Double). Если, предположим, нужно сложить значения двух переменных, тип одной из которых Integer, а второй — Double, вначале измените тип одной из переменных.

Чтобы изменить тип переменной, нужно набрать код, подобный приведенному ниже:

ФункцияПреобразования (Аргумент)

В зависимости от типа, к которому нужно преобразовать данные, вместо слова Функция Преобразования подставляется название соответствующей функции. Вместо слова Аргумент подставляется число, переменная или выражение, тип которых нужно преобразовать. Например, если нужно преобразовать значение типа Single (хранится в переменной X) в значение типа Integer, наберите следующий код:

```
Dim Y As Integer
```

```
Y = CInt(X)
```

Список функций, изменяющих тип данных, приведен в табл. 17.2.

**Таблица 17.2. Функции преобразования типов данных**

Функция	Тип возвращаемых данных	Диапазон принимаемых аргументов
Cbool	Boolean	Любое допустимое числовое или символьное выражение
Cbyte	Byte	От 0 до 255
Cchar	Char	От 0 до 65535
Cdate	Date	Любое допустимое представление даты или времени
CDec	Decimal	+/-79228162514264337593543950335 без десятичной запятой; +/-7,9228162514264337593543950335 с 28 знаками после запятой;
CDbl	Double	От -1,79769313486231E+308 до -4,94065645841247E-324 (отрицательные числа) и от 4,94065645841247E-324 до 1,79769313486231E+308 (положительные числа)
CInt	Integer	От -2 147 483 648 до 2 147 483 647; дробные части чисел округляются
CLng	Long	от -9 223 372 036 854 775 808 до 9 223 372 036 854 775 807; дробные части чисел округляются
CObj	Object	Любое допустимое выражение
CShort	Short	От -32 768 до 32 767; дробные части чисел округляются
CSng	Single	От -3,402823E+38 до -1,401298E-45 (отрицательные числа) От 1,401298E-45 до 3,402823E+38 (положительные числа)
CStr	String	Любое допустимое выражение

## Логические операторы

Логические операторы предназначены для обработки значений True (Истина) и False (Ложь). В языке Visual Basic .NET значению True соответствует число 1, а значению False — число 0. В табл. 17.3 приведен список наиболее часто используемых логических операторов.

**Таблица 17.3. Логические операторы**

Оператор	Пример использования
And	Переменная1 And Переменная2
Or	Переменная1 Or Переменная2
Xor	Переменная1 Xor Переменная2
Not	Not Переменная

# Применение оператора Not

Оператор Not меняет одно логическое значение на противоположное, т.е. вместо значения False присваивает значение True и наоборот:

Переменная	Значение
Правильное_Решение	True
Not Правильное_Решение	False



Для большей ясности и выразительности кода некоторые программисты используют круглые скобки. При их наличии приведенный выше пример будет выглядеть так:  
Not (Правильное\_Решение)

# Применение оператора And

Оператор And сравнивает логические значения двух переменных и на основании результатов такого сравнения возвращает новое значение False или True. Рассмотрим следующий пример.

Поездка = Машина And Бензин

В каком случае переменная Поездка будет иметь значение True, а в каком False? Возможны четыре варианта.

Машина	Бензин	Поездка
True	True	True
True	False	False
False	True	False
False	False	False



Обратите внимание на следующее обстоятельство: переменная Поездка будет иметь значение True лишь при условии, что обе переменные —и Машина, и Бензин — будут иметь значение True.

# Применение оператора Or

Подобно оператору And, оператор Or сравнивает значения двух переменных и возвращает новое логическое значение. Рассмотрим это на следующем примере:

Осадки = Дождь Or Снег

В зависимости от значений, принимаемых переменными Дождь и Снег, возможны такие четыре варианта:

Дождь	Снег	Осадки
True	True	True
False	True	True
True	False	True
False	False	False



Оператор `Or` возвращает значение `True`, если хотя бы одна из переменных имеет значение `True`.

## Применение оператора `Xor`

Оператор `Xor`, равно как операторы `And` и `Or`, сравнивает значения двух переменных и возвращает новое логическое значение. Рассмотрим это на таком примере:

`Конфликт = Эксперт1 Xor Эксперт2`

Два эксперта принимают одно из двух возможных решений (и соответствующие им переменные принимают значение `True` либо `False`). Посмотрим, в каких случаях будет возникать конфликтная ситуация (переменная `Конфликт` будет принимать значение `True`).

Эксперт1	Эксперт2	Конфликт
True	False	True
False	True	True
True	True	False
False	False	False



Итак, оператор `Xor` возвращает значение `True`, если две переменные имеют разные логические значения.

## Операторы сравнения

Операторы *сравнения* сопоставляют два числовых значения и определяют, равны они или не равны, какое из них больше, а какое меньше. Эти же операторы могут использоваться для определения равенства или неравенства текстовых значений. Перечень операторов сравнения приведен в табл. 17.4.

**Таблица 17.4. Операторы сравнения**

Оператор	Определяет
<code>=</code>	Равенство
<code>&lt;&gt;</code>	Неравенство
<code>&lt;</code>	Меньше чем
<code>&lt;=</code>	Меньше или равно
<code>&gt;</code>	Больше чем
<code>&gt;=</code>	Больше или равно

## Сравнение числовых и текстовых значений

Операторы сравнения сравнивают между собой два значения и в зависимости от результатов сравнения возвращают логическое значение `False` или `True`. Проиллюстрируем сказанное следующим примером:

```
Dim Доход, Расход As Integer
Dim Рентабельность As Boolean
Доход = 120
Расход = 140
Рентабельность = (Доход >= Расход)
```

Вот как Visual Basic .NET интерпретирует эти коды.



1. Первая строка говорит: "Создай переменные Доход и Расход типа Integer".
2. Вторая строка: "Создай переменную Рентабельность типа Boolean".
3. Третья строка: "Присвой переменной Доход значение 120".
4. Четвертая строка: "Присвой переменной Расход значение 140".
5. А пятая строка говорит: "Сравни значение переменной Доход со значением переменной Расход. Если значение переменное Доход будет больше или равным значению переменной Расход, присвой переменной Рентабельность значение True (Истина); в противном случае присвой переменной Рентабельность значение False (Ложь)." В нашем примере значение переменной Доход (120) меньше значения переменной Расход (140), поэтому переменной Рентабельность будет присвоено значение False.

Как видите, процесс сравнения числовых значений очевиден и прост. Со строковыми переменными все несколько сложнее. Сопоставляя строки, Visual Basic .NET определяет значение кода ANSI для каждого сравниваемого символа.



На самом нижнем (машинном) уровне компьютер способен различать только два числа: ноль и единицу. Поскольку компьютер на самом деле "понимает" только числовую информацию (и не различает букв и знаков), была придумана нехитрая система, связывающая с каждой буквой, знаком или символом определенное число. Так, букве "а" соответствует число 97, букве "А"— число 65, а восклицательному знаку (!)— число 33. Чтобы не было путаницы и все компьютеры использовали для представления букв и знаков одни и те же числа, Американский Национальный институт стандартизации (American National Standards Institute, ANSI) определил общую систему кодов для представления всех текстовых символов.

## Использование для сравнения строк операторов = и <>

Две строки равны только в том случае, если они абсолютно идентичны. Рассмотрим это на следующих примерах.

Операция	Возвращаемый результат
V = "а"	True
"а" = "А"	False
"а" = "aa"	False

Если строки хоть чем-то различаются, они не являются равными друг другу. Следовательно, если проверяется неравенство строк, то оно будет неверно только при условии, что строки абсолютно одинаковы, например:

Операция	Возвращаемый результат
"а" <> "А"	True
"Образ" <> 'Образ'	False





Сравнивая строки, Visual Basic .NET воспринимает прописные и строчные буквы как совершенно разные знаки.

## Использование для сравнения строк операторов $>$ , $>=$ , $<$ и $<=$

Сравнивая строки, Visual Basic .NET определяет и сопоставляет значения кодов ANSI для первых букв в каждой строке. Если они совпадают, сравниваются коды вторых букв и т.д. Та строка, чей сравниваемый символ имеет больший код, считается большей.

Например, букве "A" соответствует код ANSI, равный числу 65, а букве "a" соответствует код ANSI, равный числу 97. С учетом этого рассмотрим такой пример:

```
Flag = ("Air" < "aardvark")
```

Поскольку первой букве строки Air соответствует меньшее значение кода ANSI, чем первой букве строки aardvark, Visual Basic .NET расценивает строку Air как меньшую, чем строка aardvark. Поэтому сравнение считается корректным и переменной Flag будет присвоено значение True.

Теперь рассмотрим другой пример:

```
Flag = ("air" < "aardvark")
```

Переменной Flag будет присвоено значение False, поскольку строка air больше, чем строка aardvark. Почему? Первые буквы строк совпадают, следовательно, совпадают и значения их кодов ANSI. Поэтому Visual Basic .NET переходит ко вторым буквам. Код, соответствующий букве i, больше, чем код, соответствующий букве a, вот почему Visual Basic .NET решает, что строка air больше строки aardvark.

И последний пример:

```
Flag = ("air" < "airplane")
```

Теперь переменной Flag будет присвоено значение True. Первые три буквы у обеих строк одинаковы, поэтому Visual Basic .NET переходит к четвертой паре. Поскольку в первой строке четвертой буквы нет, а во второй таковая есть, вторая строка считается большей, и операция сравнения воспринимается как правильная (возвращается значение True).

### Тест на проверку полученных вами знаний

1. Чем различаются принципы использования операторов косая черта (/) и обратная косая черта (\)?
  - а. Косая черта делит одно число на другое, а обратная косая черта снова возвращает все на свои места.
  - б. Обратная косая черта создана для левшей, в то время как прямой косой чертой пользуются в основном правши.
  - в. Косая черта возвращает результат с плавающей запятой (тип Decimal), а обратная косая черта возвращает целое число (тип Integer).
  - г. Ничем не различаются, но если набрать их вместе (//), получится эффект, который называется "хижина дяди Тома".
2. Истинно или ложно выражение:  
"aeroplane" < "airplane"
  - а. Ложно. Слово aeroplane больше, поскольку оно на одну букву длиннее.
  - б. Истинно, поскольку aeroplane является устаревшим названием летательных аппаратов.
  - в. Слова равнозначны, поскольку обозначают, по сути, одно и то же.
  - г. Истинно, поскольку второй букве в слове aeroplane соответствует меньший код ANSI, чем второй букве в слове airplane.

# Приоритет операторов

Что произойдет, если в одной строке будут использованы сразу несколько операторов, например:

```
Dim Mess As Single
```

```
Mess = 4 / 7 + 9 * 2
```

Значение, которое будет присвоено переменной Mess, равно 18,57143. Почему получено именно такое значение? Потому что вначале вычисляются те операторы, которые имеют больший *приоритет*.

Если в одном выражении используется несколько операторов, Visual Basic .NET выполняет все операции поочередно, в соответствии с приоритетом каждого оператора. В табл. 17.5 все рассмотренные нами операторы расположены в порядке уменьшения их приоритета. Те из них, у кого приоритет больше, расположены выше остальных. Следовательно, приоритет оператора умножения (\*) выше, чем приоритет оператора равенства (=).

**Таблица 17.5. Приоритет операторов**

Оператор	Тип оператора
Возведение в степень (^)	Арифметический
Отрицание (-)	Арифметический
Умножение (*) и деление (/)	Арифметический
Деление без остатка (\)	Арифметический
Остаток от деления (mod)	Арифметический
Сложение (+) и вычитание (-)	Арифметический
Конкатенация строк (&)	Арифметический
Равенство (=)	Сравнения
Неравенство (<)	Сравнения
Больше чем (>) и меньше чем (<)	Сравнения
Больше или равно (>=)	Сравнения
Меньше или равно (<=)	Сравнения
Not	Логический
And	Логический
Or	Логический
Xor	Логический

Вы спросите, как же Visual Basic .NET вычисляет значение для переменной Mess в приведенном ниже примере?

```
Dim Mess As Single
```

```
Mess -= 4 / 7 + 9 * 2
```

Чтобы помочь вам в этом разобраться, рассмотрим данный процесс пошагово.



1. Вначале Visual Basic .NET смотрит на используемые в выражении операторы и определяет приоритет каждого из них.
2. Поскольку операторы деления и умножения имеют одинаковый приоритет, обрабатывается вначале тот из них, который расположен левее. Таким образом, первым вычисляется выражение  $4/7$ , а его результат ( $0,57143$ ) подставляется в общее выражение:

$$\text{Mess} = 0,57143 + 9 * 2$$

3. Далее Visual Basic .NET определяет, что приоритет оператора умножения выше, чем оператора сложения, поэтому следующим вычисляется выражение  $9 * 2$  и его результат ( $18$ ) также подставляется в общее выражение:

$$\text{Mess} = 0,57143 + 18$$

4. Последней выполняется операция сложения, и переменной Mess присваивается значение  $18,57143$ .

Не исключено, что из каких-то соображений вы решите, что вначале должна быть выполнена операция сложения. Чтобы задать порядок производимых операций, воспользуйтесь круглыми скобками. Те операции, которые заключены в круглые скобки, будут выполнены первыми, вне зависимости от их приоритета. Например, добавим скобки в рассмотренное выше выражение:

$$\text{Mess} = 4 / (7 + 9) * 2$$

Вот какими будут действия Visual Basic .NET в этом случае.



1. Расположение скобок означает, что вначале должна быть выполнена операция сложения, в результате чего исходное выражение принимает следующий вид:

$$\text{Mess} = 4 / 16 * 2$$

2. Поскольку операторы деления и умножения имеют одинаковый приоритет, обрабатывается вначале тот из них, который расположен левее. Левее расположен оператор деления. Определяется, что  $4/16$  равно  $0,25$ , и исходное выражение выглядит уже так:

$$\text{Mess} = 0.25 * 2$$

3. Последней выполняется операция умножения, и переменной Mess присваивается значение  $0,5$ .



Если ваше выражение состоит из нескольких операторов, то, воспользовавшись круглыми скобками, **ВЫ** сможете облегчить его чтение и получить гарантию того, что Visual Basic .NET выполнит операции в требуемом порядке.

# Обработка текстовой информации

*В этой главе...*

- Определение длины строки
- Изменение регистра
- Удаление лишних пробелов
- > Поиск текстовых значений
- > Преобразование строк в коды ASCII

**В**аша программа может обрабатывать текстовую информацию не менее эффективно, чем числовые данные. Любое текстовое значение называется *строкой* и представляет собой произвольный набор букв, цифр и символов.



Если вы хотите присвоить переменной текстовое значение, обязательно возьмите его в кавычки, например, "как эту строку". Кавычки дадут понять программе, что это значение является именно строкой. Чтобы присвоить переменной текстовое значение 123-4567, наберите код

```
Dim Phone As String
```

```
Phone = "123-4567"
```

Сохранив какую-либо информацию как строку, вы получаете доступ к огромным возможностям Visual Basic .NET по обработке, изменению и манипулированию текстовыми данными.

## Определение длины строки

Длина строки определяется количеством символов (включая пробелы), из которых она состоит. Для того чтобы установить длину строки, нужно воспользоваться командой `Len`:

```
Переменная = Len("Строка")
```

Например:

```
Dim Строка As String
```

```
Dim Длина As Integer
```

```
Строка = "Раз дза три"
```

```
Длина = Len(Строка)
```

В этом примере переменной `Длина` будет присвоено значение 11 (строка состоит из девяти букв и двух пробелов).

## Изменение регистра

Если вам не нравится, как выглядит строка, можете изменить регистр букв, например, сделать все буквы строчными или, наоборот, прописными. Для этих целей в Visual Basic .NET предусмотрены три команды: `LCase`, `UCase` и `StrConv`.

## Как сделать ПРОПИСНЫЕ буквы строчными

Чтобы все буквы строки стали строчными (нижний регистр), наберите следующую команду BASIC:

```
LCase ("Строка")
```

Например:

```
Dim Строка, строчные As String
Строка = "ЕСТЬ Ж ЖИЗНЬ НА МАРСЕ?"
строчные = LCase("Строка")
```

Переменной строчные будет присвоено значение  
есть ли жизнь на марсе?

Обратите внимание, что операция изменения регистра касается только букв. (Нельзя же сделать, например, вопросительный знак большим или маленьким.)

## Как сделать строчные буквы ПРОПИСНЫМИ

Чтобы все буквы строки стали прописными (верхний регистр), наберите команду BASIC, приведенную ниже:

```
UCase("Строка")
```

Например:

```
Dim Строка, ПРОПИСНЫЕ As String
Строка = "хочу быть большой"
ПРОПИСНЫЕ = LCase("Строка")
```

Переменной ПРОПИСНЫЕ будет присвоено значение  
ХОЧУ БЫТЬ БОЛЬШОЙ

## Как сделать Первые Буквы Всех Слов Прописными

Visual Basic .NET может не только делать все буквы строки большими или маленькими, но и преобразовывать строку к специальному виду, при котором первые буквы всех слов становятся прописными, а все остальные строчными.

Для выполнения такой операции наберите следующий код BASIC:

```
StrConv("Строка", VbStrConv.ProperCase)
```

Например:

```
Dim Строка, ProperCase As String
Строка = "хочу быть большой"
ProperCase = StrConv(Строка, VbStrConv.ProperCase)
```

Переменной ProperCase будет присвоено значение  
Хочу Быть Большой

## *Удаление лишних пробелов*

Текстовая информация не всегда изначально имеет корректное представление. Иногда строки могут содержать лишние символы, которые вы хотели бы удалить. Например, очень часто в начале и в конце строк присутствуют лишние пробелы:

" Лишние пробелы в начале строки"

"Лишние пробелы в конце строки "

## Удаление пробелов в начале строки

Если строка начинается с некоторого количества пробелов, наличие которых вовсе не обязательно, удалить их можно с помощью команды `LTrim`:

```
LTrim("ИсходнаяСтрока")
```

Например:

```
Dim ПолноеИмя As String
ПолноеИмя = " Мартин Лютер"
ПолноеИмя = LTrim(ПолноеИмя)
```

В результате выполнения этих кодов переменной `ПолноеИмя` будет присвоено значение "Мартин Лютер".

## Удаление пробелов в конце строки

Если строка заканчивается несколькими пробелами, наличие которых не обязательно, удалить их можно с помощью команды `RTrim`:

```
RTrim("ИсходнаяСтрока")
```

Например:

```
Dim ПолноеИмя As String
ПолноеИмя = "Альберт "
ПолноеИмя = RTrim(ПолноеИмя)
ПолноеИмя = ПолноеИмя & " " & "Эйнштейн"
```

В этом примере после применения команды `RTrim` в конце строки удаляются пробелы и переменной `ПолноеИмя` присваивается значение "Альберт". Затем к этому значению добавляется один пробел и слово "Эйнштейн", в результате чего в переменной `ПолноеИмя` уже содержится строка "Альберт Эйнштейн".

## Удаление пробелов в начале и в конце строки

Если лишние пробелы имеются и в начале, и в конце строки, примените сразу обе рассмотренные выше команды:

```
LTrim(RTrim("ИсходнаяСтрока"))
```

По сути, этот код говорит компьютеру: "Сперва удали лишние пробелы в начале строки, а затем удали лишние пробелы в конце строки".



Вы можете упростить ситуацию, заменив две команды одной командой `Trim`:

```
Trim("ИсходнаяСтрока")
```

Например:

```
Dim ПолноеИмя As String
ПолноеИмя = " Мартин Лютер "
ПолноеИмя = Trim(ПолноеИмя)
ПолноеИмя = ПолноеИмя & " " & "Кинг"
```

Команда Trim удаляет лишние пробелы в начале и в конце строки, а затем, следующим кодом, к значению переменной добавляется один пробел и слово "Кинг", в результате чего мы получаем строку "Мартин Лютер Кинг".

## Возвращение символов строки

У нас имеется возможность удалить либо вернуть часть символов, расположенных в начале, в конце или посередине существующей строки. Для этого следует воспользоваться командой Left, Right или Mid.

### Возвращение первых символов строки

Иногда так бывает, что строки содержат больше информации, чем это необходимо. Например, переменная ПолноеИмя может содержать в себе имя и фамилию человека, тогда как вам нужно, скажем, только имя. Чтобы вернуть лишь некоторое количество символов из начала строки, воспользуйтесь следующей командой BASIC:

```
Microsoft.VisualBasic.Left(ПолноеИмя, N)
```



Вас, наверное, удивляет наличие слов Microsoft.VisualBasic перед командой Left. Таким образом дается ссылка на отдельный файл классов, содержащий коды BASIC, которые делают рабочей команду Left. Более подробную информацию о файлах классов и объектно-ориентированном программировании вы найдете в седьмой части этой книги.

Приведенная выше команда обращается к компьютеру: "Видишь переменную ПолноеИмя? Верни первые N символов из этой строки". Проиллюстрируем сказанное на примере.

```
Dim ПолноеИмя, Имя As String
ПолноеИмя = "Рита Н. Брокович"
Имя = Microsoft.VisualBasic.Left(ПолноеИмя, 4)
```

Переменной Имя будет присвоено значение Рита.

### Возвращение последних символов строки

Чтобы получить определенное количество символов начиная с правого края строки, воспользуйтесь такой командой BASIC:

```
Microsoft.VisualBasic.Right(ПолноеИмя, N)
```

Эта команда дает компьютеру указание: "Видишь переменную ПолноеИмя? Верни последние N символов этой строки". Приведем пример:

```
Dim ПолноеИмя, Фамилия As String
ПолноеИмя = "Рита Н. Брокович"
Фамилия = Microsoft.VisualBasic.Right(ПолноеИмя, 8)
```

В результате переменной Фамилия будет присвоено значение Брокович.

### Возвращение символов, находящихся внутри строки

Если нужный текст расположен где-то внутри строки, для его возвращения следует воспользоваться командой:

```
Microsoft.VisualBasic.Mid(ПолноеИмя, X, Y)
```

Данная команда говорит компьютеру: "Видишь переменную ПолноеИмя? Отсчитай X символов от левого края строки и начиная с этой позиции верни следующие Y символов". Рассмотрим это на примере:

```
Dim ПолноеИмя, Инициал As String
```

```
ПолноеИмя = "Рита Н. Врокович"
```

```
Инициал = Microsoft.VisualBasic.Mid(ПолноеИмя, 6, 2)
```

Переменной Инициал будет присвоено значение Н. (включая точку).

## *Поиск и замена отдельных слов*

При необходимости найти имя, номер или какое-нибудь слово внутри одной большой строки, воспользуйтесь командами BASIC для поиска текстовой информации. В Visual Basic .NET имеется две такие команды: InStr и InStrRev.

### Поиск одной строки внутри другой

Если одна строка расположена где-то внутри другой строки (т.е. является ее частью), найти ее позицию можно с помощью команды BASIC InStr:

```
InStr("ИсходнаяСтрока", "ИскомаяСтрока")
```

Указанная команда возвращает номер позиции внутри исходной строки, начиная с которого следуют символы искомой строки. Например:

```
Dim ПолноеИмя As String
```

```
Dim Позиция As Integer
```

```
ПолноеИмя = "Иван Федорович Крузенштерн"
```

```
Позиция = InStr(ПолноеИмя, "Федорович")
```

В данном случае переменной Позиция будет присвоено значение 6.

Если строка, которую вы ищете, в исходной строке не будет найдена, команда InStr вернет значение 0.



Для Visual Basic .NET символы верхнего и нижнего регистров воспринимаются как совершенно разные. Поэтому при поиске вхождения одной строки в другую обращайте внимание на регистр. Например, следующая команда возвращает нулевое значение:

```
InStr("Иван Федорович Крузенштерн", _ "ФЕДОРОВИЧ")
```

Строка Федорович вовсе не соответствует строке ФЕДОРОВИЧ, и, возвращая нулевое значение, Visual Basic .NET как бы говорит: "Извини дружище, но такого текста в исходной строке нет".

### Поиск текста по шаблону

Поиск текста по шаблону подразумевает, что Visual Basic .NET может сравнивать текстовые значения с заранее заданным образцом и определять, соответствуют ли они ему. Для выполнения данной операции используется оператор Like:

```
Dim Flag As Boolean
```

```
Dim Строка As String
```



Строка = "мышь"

Flag = Строка Like "мышь"



1. Первая строка говорит Visual Basic .NET о необходимости создать переменную типа Boolean и назвать ее именем Flag.
2. Вторая строка дает указание Visual Basic .NET создать переменную типа String и назвать ее Строка.
3. В третьей строке переменной Строка присваивается значение мышь.
4. Четвертая строка приказывает Visual Basic .NET: "Сравни значение переменной Строка со значением мышь, а результат сравнения присвой переменной Flag". Поскольку в данном случае значение переменной Строка и значение мышь совпадают, переменной Flag будет присвоено значение True.

## Применение группового символа "?"

В предыдущем примере в качестве шаблона была использована строка мышь. Можно немного усложнить задачу и расширить круг поиска за счет замены одного из символов знаком вопроса (?). Знак вопроса в шаблоне означает, что вместо него в этой позиции может стоять любой другой символ. Например:

Flag = Строка Like "мыш?"

Теперь переменная Flag будет принимать значение True каждый раз, когда значением переменной Строка будет текст, состоящий из четырех символов, первыми тремя из которых будут мыш. Такому шаблону, в частности, будут удовлетворять строки *мыша*, *мышы*, *мыш5*, *мыш\** и т.д.



Для Visual Basic .NET имеет принципиальное значение, в каком регистре набран текст. Вот почему буква "М" — это не одно и то же, что и буква "м". Следовательно, сравнение указанного выше шаблона со строкой *Мышь* будет возвращать переменной Flag значение False.

Переменной Flag будет также присваиваться значение False и в том случае, если с шаблоном будет сравниваться строка, состоящая из менее чем четырех или более чем четырех символов, вне зависимости от того, будут ли в ней присутствовать буквы мыш.

Знаком вопроса можно также заменить два и более символов:

Flag = Строка Like "м???"

Теперь переменной Flag значение True будет присваиваться каждый раз, когда с шаблоном будет сравниваться строка, состоящая из четырех символов и начинающаяся с буквы "м".

## Групповой символ "звездочка"

Групповой символ "вопросительный знак" может заменять собой только один текстовый символ. Но если нужно обозначить произвольное количество любых символов, применяется групповой символ "звездочка" (\*):

Flag ← Строка Like "м\*ь"

В данном случае переменной Flag будет присваиваться значение True каждый раз, когда с шаблоном будет сравниваться строка, начинающаяся с буквы м, заканчивающаяся мягким знаком и состоящая из любого количества символов, в том числе и строки *мышь*, *моль*, *магистраль*, *мь*, *м354ь* и др.

При необходимости несколько групповых символов можно комбинировать для составления более гибких шаблонов, например:

Flag = Строка Like "м?ш\*"

Этому шаблону удовлетворяют строки произвольной длины, первой буквой которых является "м", а третьей — буква "ш": *мышь*, *мышка*, *мушкет*, *мнша657*.

## Применение группового символа "#"

Вопросительный знак и "звездочка" могут заменять собой любую букву, цифру или знак. В отличие от них групповой символ "диез" (#) заменяет собой только цифры (от 0 до 9). Например:

Flag := Строка Like "м#"

В этом случае переменной Flag будет присваиваться значение True каждый раз, когда с шаблоном будет сравниваться строка, начинающаяся с буквы "м" и заканчивающаяся любой цифрой: *м3*, *м0*, *м9* и т.д.

## Применение диапазонов

В Visual Basic .NET для обозначения символов, принадлежащих определенному ряду, используются *диапазоны*. Например, чтобы обозначить все буквы от A до Z, наберите следующий код:

Flag := Строка Like "[A — Z]"

В этом случае переменной Flag будет присваиваться значение True каждый раз, когда с шаблоном будет сравниваться строка, состоящая из одной прописной буквы латинского алфавита: *A*, *B*, ..., *Z*.

Диапазоны можно использовать в комбинации с любыми групповыми символами:

Flag := Строка Like "[K — Z]##.\*"

В этом случае переменной Flag будет присваиваться значение True каждый раз, когда с шаблоном будет сравниваться строка, начинающаяся на букву F, за которой следует буква из диапазона [K — Z], две цифры, точка и далее произвольное количество любых символов, например: *FM95.6*, *FZ12.yhbbhu*, *FU15.snap* и т.п.



Диапазон заменяет собой только один символ.

В диапазоне также может быть представлен ряд символов, которые не должны встречаться в строке в определенной позиции: [!a — z]. Такой код говорит Visual Basic .NET: "В этой позиции может быть любой символ, кроме строчных букв латинского алфавита". Например:

Flag ---- Строка Like "FM[!a — z]1\*"

В этом случае переменной Flag будет присваиваться значение True каждый раз, когда с шаблоном будет сравниваться строка, начинающаяся с букв FM, за которыми следует любой символ, кроме строчной буквы латинского алфавита, далее 1 и произвольное количество любых других символов: *FM.105.B*, *FM/101*, *FMD123* и т.п.

## Замена части строки другой строкой

Если у вас возникло желание создать с помощью Visual Basic .NET собственный текстовый процессор (с возможностями поиска и замены текста), рекомендуем воспользоваться такой командой:

Mid("ИсходнаяСтрока", Позиция) = "НоваяСтрока"

Эта команда дает указание компьютеру: "В исходной строке найди символ, расположенный в заданной позиции (считая от начала строки) и начиная с данного места замени последующие символы текстом новой строки".

Операция замены части строки другой строкой требует определенной сноровки и аккуратности. Обратимся к примеру:

ПолноеИмя = "Иван Иванович Сусанин"

Mid(ПолноеИмя, 6) = "Степанович"

Вот как Visual Basic .NET интерпретирует этот код.



1. В первой строке дается указание присвоить переменной ПолноеИмя значение Иван Иванович Сусанин.
2. Вторая строка говорит Visual Basic .NET: "В строке, представленной переменной ПолноеИмя, найди шестой от начала строки символ и начиная с него замени последующие символы строкой Степанович".

Вот что в результате этого выйдет:

Иван Иванович Сусанин

(исходная строка)

•

(шестой символ от начала строки)

Иван СтепановичСусанин

(новая строка)



Получив указание заменить часть строки другой строкой, Visual Basic .NET поймет все буквально и без разбора сотрет все исходные символы, оказавшиеся в пределах новой строки.

### Тест на проверку полученных вами знаний

1. Как **дать** понять программе, что вводимая информация должна быть воспринята как строка?
  - а. Нужно в конце строки поставить тире и указать автора.
  - б. Если набрать текст без грамматических ошибок, то он будет воспринят как строка.
  - в. Нужно создать отдельный файл и указать в нем перечень всех строк, которые могут быть потом набраны.
  - г. Текстовая информация должна быть заключена в кавычки.
2. Что делает следующий код BASIC: Поиск = InStr(\_\_\_\_ "ИсходнаяСтрока", \_\_\_\_ "ИскомаяСтрока")
  - а. Код заменяет исходную строку искомой, и ищет, чего бы еще заменить в тепле программы.
  - б. Код сравнивает исходную строку с искомой, и если он находит десять различий, компьютер издает звуковой сигнал.
  - в. Если искомая строка является частью исходной, переменной Поиск присваивается число, обозначающее позицию символа в исходной строке, начиная с которого далее следует текст искомой строки.
  - г. Выполняет все вышеперечисленные действия.

## Преобразование строк и чисел

Может возникнуть необходимость преобразовать строки в числовые значения, с тем чтобы иметь возможность использовать их для проведения вычислений. С другой стороны, преобразование чисел в строки позволяет обрабатывать их как текстовую информацию. При необходимости можно также преобразовать строки в коды ASCII или ANSI.

## Преобразование строк в числа

Можно ли использовать текстовое поле для получения от пользователей информации об их среднем заработке? К сожалению, текстовые поля сохраняют получаемые данные как значения свойства **Text**, которые изначально имеют тип **String** (т.е. значениями будут строки, а не числа). Чтобы преобразовать текст в число, нужно воспользоваться одной из функций преобразования, например такой:

`CDbl(Строка)`

или такой:

`CInt(Строка)`



Список функций преобразования строк в числа не ограничивается функциями `CDbl` и `CInt`. Есть другие функции, которые преобразовывают строки в значения различных числовых типов. Например, функция `CStr` преобразовывает текст в числовое значение типа **Single**. Выбор функции зависит от типа данных, к которому нужно преобразовать текстовое значение.

Первая команда говорит компьютеру: "Преобразуй текстовое значение переменной **Строка** в числовое значение типа **Double**".

Вторая команда требует иного: "Преобразуй текстовое значение переменной **Строка** в числовое значение типа **Integer**".

Проиллюстрируем сказанное таким примером:

```
Dim ЧисловоеЗначение As Double
```

```
ЧисловоеЗначение = CDbl(txtДоход.Text)
```

Вот как Visual Basic .NET интерпретирует данный код.



1. Первая строка требует: "Создай переменную типа **Double** и присвой ей имя **ЧисловоеЗначение**".
2. Вторая строка приказывает: "Возьми данные, сохраненные как значение свойства **Text** объекта, именуемого **txtДоход**, преобразуй их в числовое значение типа **Double**, а затем присвой это значение переменной **ЧисловоеЗначение**".

Таким образом, если пользователь наберет в текстовом поле **txtДоход** число 1000 (оно автоматически будет воспринято как строка), переменной **ЧисловоеЗначение** будет присвоено значение 1000.



Если пользователь наберет что-то вроде "Мой доход составляет 1000" или "1000 долларов", Visual Basic .NET выдаст сообщение об ошибке, поскольку он не будет знать, как преобразовать слова в числа.

## Преобразование чисел в строки

Допустим, у вас возникло желание (или необходимость) преобразовать числа в строки, с тем чтобы иметь возможность обрабатывать значения как текстовую информацию. Для этого вы должны воспользоваться следующей командой BASIC:

`CStr(Число)`

Такая команда дает указание компьютеру: "Возьми число, сохраненное как значение переменной **Число**, и преобразуй его в строку".

Для Visual Basic .NET числа и строки — это совершенно разные понятия, например:

10 ' Это число

"10" ' Это строка

Вот примеры преобразования чисел в строки:

```
CStr(10) ' Это строка " 10"  
CStr(10.5) ' Это строка " 10.5"  
CStr( - 10) ' Это строка "-10"
```



Когда Visual Basic .NET преобразовывает число в строку, в начало строки добавляется пробел, если число положительное; если число отрицательное, строка начинается со знака "минус".



Функция преобразования CStr становится просто незаменимой, если на экране нужно отобразить число, используя для этой цели свойство Text текстового поля или надписи.

## Преобразование строк в коды ASCII

Если вы серьезно занимаетесь программированием, вам наверняка придется иметь дело с кодами ASCII, поэтому рекомендуем приобрести копию таблицы с таковыми и повесить ее в рамочке где-нибудь рядом с компьютером.

Коды ASCII — это числа, используемые компьютером для представления всех основных символов (поскольку компьютер в конечном итоге имеет дело только с числами). Например, букве "Л" (латинский алфавит) соответствует код ASCII, равный числу 65, а букве "а" (также латинский алфавит) — код ASCII, равный числу 97.

Если нужно узнать код ASCII, соответствующий определенной букве, необходимо воспользоваться следующей командой BASIC:

```
Asc("Буква")
```

Например:

```
X = Asc ("А") ' X = 65
```

```
X = Asc ("а") ' X = 97
```

## Преобразование кодов ANSI в строки

Microsoft Windows не использует коды ASCII — вместо них применяются коды ANSI. Впрочем, они практически полностью идентичны.

Чтобы применить коды ANSI на практике, воспользуйтесь такой командой BASIC:

```
Dim X As String
```

```
X = Chr(Символ)
```

Коды ANSI применяются в основном для набора специальных управляющих символов, обеспечивающих, в частности, переход на новую строку и возврат каретки.

Приведенные ниже команды демонстрируют применение некоторых кодов ANSI.

```
НоваяСтрока = Chr(10)
```

```
НоваяСтраница = Chr(12)
```

```
ВозвратКаретки = Chr(13)
```

Используя такие команды для управления строками, вы всегда будете уверены, что текст на экране отображается именно так, как вы хотите, что, конечно же, повышает качество создаваемой вами программы.

# Определение констант и использование комментариев

*В этой главе...*

- Создание констант
- Область видимости констант
- Виды комментариев и способы их использования

**Константа** — это постоянное значение, т.е. такое, которое никогда не меняется в процессе выполнения программы. Константами могут быть строки, числа и даты.

Чем полезны константы? Существует несколько причин, побуждающих использовать их вне зависимости от конечных целей, ради которых создается программа.

Предположим, нам нужно написать программу, определяющую размеры начисляемой заработной платы исходя из некоторой фиксированной ставки. Если эта ставка равна, скажем, \$50, вам нужно будет набрать число 50 во всех формулах программы, где используется значение этой ставки.

Однако число 50 само по себе ничего не значит. Более того, если вдруг размер фиксированной ставки поменяется и будет равен уже не \$50, а, например, \$55, вам придется во всем тексте программы менять число 50 на 55. Если вы случайно пропустите хотя бы одно вхождение числа 50 и не замените его числом 55, вся программа будет работать некорректно.

Чтобы не иметь проблем, подобных описанной, нужно использовать константы. Константа -- это просто слово, заменяющее собой некоторое значение. Она не только может объяснять посредством своего имени смысл представляемого значения, но и позволяет легко и быстро заменить это значение во всем тексте программы.

## Наименование констант

Имена констант должны удовлетворять нескольким критериям, в том числе:

- ✓ начинаться с буквы;
- ✓ состоять не более чем из 40 символов;
- ✓ включать только буквы, цифры и символы подчеркивания ( \_ ); использование знаков пунктуации и пробелов не допускается;
- ✓ не совпадать с зарезервированными в Visual Basic .NET ключевыми словами.



Чтобы в тексте программы названия констант было легко отличить от названий переменных, присваивайте константам имена, состоящие из прописных букв, например: СТАВКА, ЛИМИТ, ДАТА\_ВЫПУСКА.

Чтобы по имени константы можно было определить, к какому типу данных относится ее значение, начинайте имена с трехбуквенных префиксов, указывающих на тип данных:

intСТАВКА, decЛИМИТ и т.д.

## Вычисление значений констант

Константы представляют только фиксированные значения. Однако эти значения могут быть вычислены исходя из значений других констант. Например:

```
Const intСТАВКА As Integer = 65  
Const sngМИН_СТАВКА As Single = intСТАВКА / 2
```

Этим кодом константе `intСТАВКА` присваивается значение 65, а константе `sngМИН_СТАВКА` — в два раза меньшее, т.е. равное числу 32,5.

## Использование констант

Объявленную константу можно использовать как любое другое значение. Проиллюстрируем сказанное на примере.

```
Dim Премия As Single  
Const sngМИН_СТАВКА As Single = 35  
Премия = sngМИН_СТАВКА * 10
```

Вот как данный код будет интерпретирован Visual Basic .NET.



1. Первой строкой создается переменная `Премия` типа `Single`.
2. Второй строкой создается константа `sngМИН_СТАВКА` типа `Single`, и ей присваивается значение 35.
3. Третья строка говорит компьютеру: "Возьми значение константы `sngМИН_СТАВКА`, умножь его на десять и присвой полученное значение переменной `Премия`". В данном случае константа `sngМИН_СТАВКА` имеет значение, равное числу 35, и оно умножается на число 10. Следовательно, переменной `Премия` присваивается значение 350.



Присвоенное константе значение далее по тексту программы уже не может быть изменено (на то она и "константа", чтобы иметь постоянное значение).

## Определение области видимости констант

Все константы в зависимости от границ, в пределах которых они могут быть использованы, делятся на три группы:

- ✓ локальные (Private);
- ✓ константы модуля (Module);
- ✓ глобальные (Public).

### Локальные константы

Локальные константы могут быть использованы только в пределах процедуры, где они были объявлены. Чтобы создать локальную константу, объявите ее внутри какой-либо процедуры, например процедуры обработки событий:

```
Private Sub Button1_Click(ByVal sender As System._  
    Object, ByVal e As System.EventArgs)_  
    Handles Button1.Click
```

```
Const intSPEED_LIMIT As Integer = 55
End Sub
```

Сам факт объявления константы внутри процедуры подразумевает ее локальный характер, однако вы также можете указать, что создается именно локальная константа:

```
Private Sub Button1_Click(ByVal sender As System._
    Object, ByVal e As System.EventArgs) _
    Handles Button1.Click

    Private Const intSPEED_LIMIT As Integer = 55
End Sub
```

Каждая локальная константа может быть использована лишь в пределах своей процедуры. Но как быть, если константу нужно применить сразу в нескольких процедурах? В этом случае нужно создать константу модуля.

## Константы модуля

*Константы модуля* могут быть использованы всеми процедурами, сохраненными в одном и том же файле. Чтобы создать константу модуля в файле формы, объявите ее сразу после строки `Public Class Form1`:

```
Public Class Form1
    Const intSPEED_LIMIT As Integer = 55
```



`Form1` — это автоматически генерируемое имя создаваемой формы. Если вы присвоите форме какое-либо другое имя, оно будет отображаться в кодах вместо имени `Form1`.



Если вы хотите создать константу модуля в файле модуля, объявите ее сразу после строки `Module Module1`, где `Module1` — это название файла модуля (у каждого файла оно свое). Например:

```
Module Module1
    Const intSPEED_LIMIT As Integer = 55
```

Итак, константы модуля могут быть использованы во всех процедурах, сохраненных в том файле, где была объявлена эта константа. Если же вам нужна константа, которую могла бы использовать любая процедура вашей программы, нужно создать глобальную константу.



Чтобы по имени константы можно было определить область ее видимости, начните его с буквы “m”:

```
Const mintSPEED_LIMIT As Integer = 55
```

## Глобальные константы

*Глобальные константы* являются, наверное, наиболее удобными в использовании, поскольку могут быть применены в любой процедуре вашей программы. Однако опытные программисты создают и используют такие константы лишь в том случае, когда это действительно необходимо. А объявление глобальных констант без особой необходимости является плохой практикой.



Необоснованное создание глобальных констант нежелательно, поскольку их изменение может отразиться на работе всей программы. Кроме того, если опытные программисты поймут вас за созданием ненужных глобальных констант, они объявят вас “чайником” и ни за что не примут в свой элитарный клуб (шутка).



Глобальные константы должны быть объявлены в файле с расширением .BAS (файл модуля или формы). Для того чтобы определить создаваемую константу как глобальную, добавьте в начало строки слово **Public**:

```
Public Const intSPEED_LIMIT As Integer = 55
```



Чтобы по имени константы можно было определить, что она является глобальной, начните его с буквы “g”:

```
Public Const gintSPEED_LIMIT As Integer = 55
```

## использование комментариев

Пока вы пишете коды своей программы, все вам в них кажется понятным и само собой разумеющимся. Однако если вы вернетесь к ним через пару лет (предположим, чтобы как-то их модифицировать), то вряд ли вспомните, зачем была написана та или иная команда и как это все работает.

Поэтому не скупитесь на комментарии к своей программе. *Комментарии* — это короткие пояснения, которыми программисты сопровождают коды программ, объясняя таким образом, что делает та или иная команда или что должно произойти после ее выполнения.



Комментарии никак не влияют на ход выполнения программы и, с точки зрения компьютера, они совершенно не нужны. Но для вас их помощь может быть просто необходимой.

### Создание комментариев

В Visual Basic .NET комментарии предваряются символом апострофа ('). Так, следующая строка является комментарием:

```
' В этой строке нет ничего, кроме комментария
```

В процессе выполнения кодов программы Visual Basic .NET игнорирует все, что отображается справа от апострофа.

Если апостроф вас из каких-то соображений не устраивает, вместо него можно использовать команду **REM** (от слова **REMark** — комментарий). Например:

```
REM В этой строке нет ничего, кроме комментария
```

Как и в случае с апострофом, компьютер проигнорирует все, что набрано справа от команды **REM**.



Комментарии могут быть созданы либо в отдельной строке, либо как продолжение строки с реально работающими кодами BASIC.

Так, следующая строка содержит в себе формулу, к которой добавлен комментарий:

```
X = Y * 2 ' Определение максимального значения
```

Комментарии могут занимать и несколько строк:

```
Y = 200 ' Количество опозданий на работу
```

```
X = Y * 2
```

```
' X содержит максимально допустимое количество  
' опозданий на работу, после которых начальство  
' начинает по-настоящему нервничать
```

Не забывайте каждую новую строку, содержащую комментарий, начинать с апострофа.

## Комментарии как объяснения

Основная цель использования комментариев— сделать коды программы более легкими для понимания. По этой причине многие программисты сопровождают комментариями каждую написанную ими процедуру.

Такие комментарии объясняют, какие данные использует процедура, что она с ними делает и какой результат должен быть получен. Любой человек, прочитав подобный комментарий, может понять суть выполняемых процедурой действий, не вникая в работу каждой отдельной команды BASIC. Как вы думаете, для чего написан этот код:

```
A = SQR(B ^ 2 + C ^ 2)
```

Трудно понять, но если бы он начинался с комментария, все сразу стало бы понятным:

```
' Следующая формула является следствием теоремы  
' Пифагора: квадрат гипотенузы равен сумме квадратов  
' двух других сторон треугольника. В данном случае  
' гипотенуза представлена переменной A, другие  
' стороны треугольника — переменными B и C.
```

```
A = SQR(B ^ 2 + C ^ 2)
```

Если разработкой программы занимаются несколько человек, комментарии могут содержать информацию о времени, когда в процедуру были внесены последние изменения, или, скажем, имя программиста, который это сделал. (Таким образом, если что-то вдруг перестанет работать, будет понятно, кого нужно за это "отблагодарить".) Например:

```
' Программист: Василий Нечухаев  
' Последние изменения: 05/03/02
```

```
A = SQR(B ^ 2 + C ^ 2)
```



Если комментарий будет слишком сжатым, он вряд ли принесет вам пользу. Скорее, наоборот, вы потратите немало времени, пытаясь вспомнить, что именно хотели этим сказать. С другой стороны, не увлекайтесь написанием слишком длинных речей. Человек, который потом будет просматривать коды этой программы, не должен засыпать за чтением ваших литературных очерков. Помните, что комментарии должны давать сжатую информацию и в объеме, достаточном для понимания сути выполняемых программой действий.

## Комментарии как средство улучшения читаемости кодов

Если программа состоит из большого количества кодов BASIC, можете разбить ее на отдельные фрагменты, используя в качестве разделителей комментарии и пустые строки. Применяв этот прием, вы сделаете коды программы более удобочитаемыми и легкими для восприятия. Например:

```
Const dblПРОЦЕНТНАЯ_СТАВКА As Double = .055  
' Процентная ставка составляет 5,5%  
Dim Msg As String ' Переменная, значение которой  
                   ' будет отображено на экране  
Dim Баланс, Процент As Decimal  
  
Баланс = 500  
Процент = Баланс * dblПРОЦЕНТНАЯ_СТАВКА
```

' Начисленные проценты ложатся на счет

Баланс = Баланс + Процент

' На экране отображается сообщение с информацией

' о состоянии счета

Msg = "На Вашем счету: " & Баланс

MsgBox (Msg, Information, "Состояние счета")

Кажется, все просто и понятно (это благодаря пустым строкам и комментариям).

Попробуем убрать пустые строки и комментарии, оставив только реально работающие коды BASIC:

```
Const dblПРОЦЕНТНАЯ_СТАВКА As Double = .055
```

```
Dim Msg As String
```

```
Dim Баланс, Процент As Decimal
```

```
Баланс = 500
```

```
Процент =- Баланс * dblПРОЦЕНТНАЯ_СТАВКА
```

```
Баланс = Баланс + Процент
```

```
Msg = "На Вашем счету: " & Баланс
```

```
MsgBox (Msg, Information, "Состояние счета")
```

Теперь код выглядит компактней, но понять, что он делает, уже намного сложнее, и на чтение такого короткого фрагмента вы потратите намного больше времени и усилий.

## Комментарии как временная дезактивация кодов

Комментарии могут быть использованы не только как вспомогательные элементы для улучшения читаемости написанных вами кодов, но и как средство временной дезактивации отдельных команд BASIC.

Например, написав программу, вы можете обнаружить, что какая-то команда работает не так, как вам того хотелось бы. Проверить, как программа будет работать без данной команды, вы можете одним из двух способов:

| ✓ удалив команду;

| ✓ превратив ее в комментарий.

Если вы удалите команду, а потом решите, что все-таки она была нужна, вам придется набирать ее заново. Если же вы просто "переведете команду в разряд комментариев", то для восстановления ее дееспособности достаточно будет просто удалить стоящий перед ней апостроф.

Рассмотрим пример, содержащий довольно-таки большую формулу:

```
Private Sub Button1_Click(ByVal sender As System._  
    Object, ByVal e As System.EventArgs)_  
    Handles Button1.Click
```

```
    X = 3.14 * 650 - (909 / 34.56) + 89.323
```

```
End Sub
```

Если вы просто сотрете вторую строку, то повторный ее набор займет немало времени. Поэтому имеет смысл сделать ее просто комментарием, добавив апостроф:

```
Private Sub Button1_Click(ByVal sender As System._  
    Object, ByVal e As System.EventArgs)_  
    Handles Button1.Click
```

```
    ' X = 3.14 * 650 - (909 / 34.56) + 89.323
```

```
End Sub
```

Как вы помните, компьютер игнорирует все, что набрано после апострофа. Поэтому для него приведенный выше код выглядит следующим образом:

```
Private Sub Button1_Click(ByVal sender As System, _  
    Object, ByVal e As System.EventArgs) _  
    Handles Button1.Click  
  
End Sub
```

Итак, добавляя перед командой апостроф, вы деактивируете ее, превращая всю последующую часть строки в комментарий. Удалив таковой, вы снова делаете команду рабочей.



Если вы хотите перевести в разряд комментариев сразу несколько строк, но вас не устраивает перспектива набора апострофа перед каждой из них, можете пойти другим путем.

1. **Выделите фрагмент кода, который должен быть превращен в комментарий.**
2. **Выберите команду View⇒Toolbars⇒Text Editor (Вид⇒Панели инструментов⇒Текстовый редактор).**

На экране появится панель Text Editor (рис. 19.1).



Рис. 19.1. На этой панели есть кнопка, позволяющая быстро превращать в комментарии выделенные фрагменты кода в программе

3. **Щелкните на кнопке Comment out the selected text (Сделать выделенный фрагмент комментарием).**

Visual Basic .NET автоматически добавит апострофы в начало каждой выделенной на первом шаге строки.



Чтобы превращенные в комментарии коды "вернуть в строй", выполните те же действия, но теперь на третьем шаге щелкните на кнопке **Uncomment the selected text** (Вернуть выделенный фрагмент к прежнему виду).

### Тест на проверку полученных вами знаний

1. Для чего коды программы нужно сопровождать комментариями?
  - а. Чтобы дать краткое описание выполняемых командами действий.
  - б. Это позволяет поупражняться а написании литературных очерков и рассказов.
  - в. Для того чтобы продемонстрировать, что я умею писать не только на Visual Basic .NET, но и на "человеческом" языке.
  - г. Таким образом можно объяснить другим людям, чего же я на самом деле хочу от этой программы.
2. Какой бы комментарий вы добавили к данной главе?
  - а. Ура! Наконец-то она закончилась и на сегодня можно забросить эту книжку на полку.
  - б. Простая глава. Жаль, что не все главы книги такие короткие и легкие.
  - в. Ознакомившись с данной главой, я понял, что писать комментарии - это мое призвание. Может быть, податься в комментаторы?
  - г. Все понятно, кроме одного: если в Visual Basic .NET можно писать простой текст, то почему нельзя писать программы а Microsoft Word?

# Создание структур данных

В этой главе...

- Массивы данных
- Структуры данных
- Коллекции данных

Одна переменная может содержать одно значение, но как быть, если вам нужно сохранить информацию о нескольких связанных между собой объектах. Предположим, нужно составить список всех знакомых, которые должны вам деньги. Создавать для каждого из них переменную--- это долгое и неинтересное занятие (тем более что они этого не заслуживают), а настоящие программисты, как известно, не любят выполнять рутинную работу.

Идеальным вариантом было бы создание переменной, которая могла бы содержать любое количество связанных между собой значений. К счастью, Visual Basic .NET располагает такой волшебной переменной, способной запоминать одновременно множество значений. Правда, называется она не волшебной переменной, а просто *структурой данных*. Visual Basic .NET различает три вида структур данных: массивы, структуры и коллекции.

Создание структуры данных — по способ, с помощью которого компьютер временно организует и сохраняет данные на период выполнения программы. Как только компьютер выключается или программа завершает свою работу, вся информация, содержащаяся в структуре данных, теряется (конечно, если вы предварительно не написали специальных кодов BASIC для ее сохранения где-нибудь на жестком диске).

## Создание массивов

Простейшая структура данных, которую вы можете создать, называется *массивом*. Массив можно представить как цепочку звеньев, каждое из которых содержит одно значение. Звено цепочки называется элементом массива. Каждому элементу массива соответствует свой уникальный номер, называемый *индексом* (рис. 20.1).



Рис. 20.1. Массив состоит из набора элементов, каждый из которых содержит одно значение

Чтобы создать массив, вам нужно определить:

- ✓ его имя;
- ✓ количество элементов (размер массива);
- ✓ тип данных, которые будут храниться в массиве.

Для этого наберите следующий код BASIC:

```
Dim ИмяМассива(Размер) As ТипДанных
```



Если вместо слова Dim набрать слово Public, будет создан массив, областью видимости которого станет вся программа.

Таким образом, чтобы создать массив, именуемый МойМассив, состоящий из четырех элементов и способный хранить значения типа Integer, наберите такой код:

```
Dim МойМассив(3) As Integer
```



Чтобы массив состоял из четырех элементов, его размер должен быть равен числу 3. Почему именно 3? Потому что отсчет элементов начинается с нулевого номера. Первому элементу соответствует нулевой номер, второму элементу — номер 1 и т.д.



В предыдущих версиях Visual Basic можно было указать первый и последний номера элементов массива:

```
Dim МойМассив(1 To 3) As Integer
```

В Visual Basic .NET нумерация всегда (подчеркиваем, всегда) начинается с нулевого номера. Поэтому, если вы наберете такой код, Visual Basic .NET воспримет его как ошибку.

## Присвоение значений элементам массива

Элементы только что созданного массива не содержат никаких данных. Чтобы сохранить в массиве какое-нибудь значение, нужно указать, какому элементу оно должно быть присвоено.

Предположим, вы создали массив, который может содержать в себе до пяти строк:

```
Dim Сотрудник(4) As String
```

Если вы хотите первому элементу массива присвоить значение Данила Быстрое, наберите такой код:

```
Сотрудник(0) = "Данила Быстров"
```



Первому элементу массива соответствует нулевой номер.

Некоторые программисты, создав массив, предпочитают сразу же инициализировать его, присвоив каждому элементу нулевое значение или пустую строку. Это можно сделать используя цикл For-Next (будет рассмотрен в главе 25), например:



```
Dim Сотрудник(4) As String
Dim I As Integer
For I = 0 To 4
    Сотрудник(I) = ""
Next I
```

Как следует из сказанного выше, массив, размер которого равен четырем, состоит из пяти элементов, первый из которых имеет нулевой порядковый номер, а пятый, соответственно, четвертый порядковый номер (рис. 20.2).

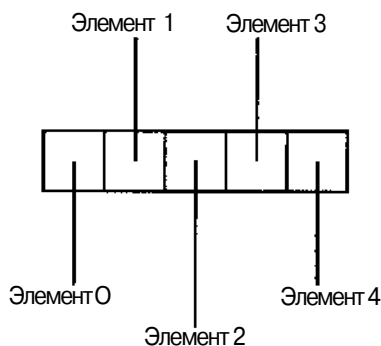


Рис. 20.2. Порядок нумерации элементов массива в Visual Basic .NET

Значения элементов массива можно присваивать другим переменным, например:

```
Dim Сотрудник(4) As String
Dim Брат As String
Сотрудник(3) = "Данила Быстров"
Брат = Сотрудник(3)
```



1. Первой строкой создается массив Сотрудник, состоящий из пяти текстовых значений.
2. Второй строкой создается текстовая переменная, именуемая Брат.
3. Третьей строкой четвертому элементу массива (которому соответствует третий порядковый номер) присваивается значение Данила Быстров.
4. В четвертой строке переменной Брат присваивается значение элемента массива Сотрудник, которому соответствует третий порядковый номер. Ранее этому элементу было присвоено значение Данила Быстров, поэтому теперь переменная Брат также будет содержать значение Данила Быстров.



Обычно элементы массива содержат значения, относящиеся к одному типу данных, например, только строки или только целые числа. Если же необходимо, чтобы в массиве содержались данные разных типов, при создании массива вместо названия одного из типов данных наберите слово `Object`:

```
Dim МойМассив As Object
```

Элементы такого массива могут содержать значения разных типов:

```
МойМассив(0) = "Спрут"
МойМассив(1) = 56
МойМассив(2) = 3.1415
```

## Создание многомерных массивов

Когда вы создаете массив и указываете одним числом его размер, вы создаете одномерный массив — он представляет собой один сплошной ряд элементов. Но Visual Basic .NET позволяет создавать также и многомерные массивы, которые могут иметь до шестидесяти измерений. Необходимость создавать огромные массивы, характеризующиеся десятками измерений, возникает очень редко, но вот двухмерными массивами пользуются довольно часто. Их можно представить в виде сетки, которая показана на рис. 20.3.

(0,0)	(0,1)	(0,2)	(0,3)	(0,4)
(1,0)	(1,1)	(1,2)	(1,3)	(1,4)
(2,0)	(2,1)	(2,2)	(2,3)	(2,4)

Рис. 20.3. Двухмерный массив может быть представлен в виде сетки

Порядок создания двухмерного массива тот же, что и одномерного, с той лишь разницей, что, указывая его размер, нужно добавить еще одно число — для дополнительного измерения. Таким образом, размер двухмерного массива определяется двумя числами:

`Dim ДвухММассив (Строки, Столбцы) As ТипДанных`

Размер трехмерного массива будет определяться тремя числами и т.д.:

`Dim ТрехММассив (X, Y, Z) As ТипДанных`



При создании массивов, в том числе и многомерных, для хранения значения каждого элемента выделяется оперативная память (даже если это нулевые значения или пустые строки). Таким образом, создавая большой массив, вы резко уменьшаете объем свободной памяти, что может негативно отразиться на работе вашей программы. Поэтому создавать многомерные массивы следует лишь по мере необходимости.

## Изменение размеров массива

После того как массив будет создан, вы в любой момент сможете увеличить или уменьшить его размер. Предположим, вы уже создали массив из пяти элементов, значениями которых являются строки:

`Dim МойМассив(4) As String`

Если вдруг появится необходимость добавить несколько новых элементов, воспользуйтесь командой `ReDim`:

`ReDim МойМассив(7) As String`

Эта команда добавляет три ( $7 - 4 = 3$ ) новых элемента к ранее созданному массиву `МойМассив`. Этой же командой можно удалить лишние элементы:

`ReDim МойМассив(2) As String`



С помощью команды `ReDim` можно также изменить и размер многомерного массива. Однако она не позволяет добавлять или удалять измерения многомерного массива.

При использовании команды `ReDim` для добавления или удаления элементов массива все данные, которые уже были сохранены в этом массиве, теряются. Если вы хотите, чтобы прежняя информация осталась на своих местах при добавлении новых элементов (в случае удаления элементов такая возможность отсутствует), вместе с командой `ReDim` используйте команду `Preserve`:

`ReDim Preserve МойМассив(25) As String`



Уменьшая по мере необходимости размеры массивов, вы сокращаете объем памяти, который требуется для выполнения вашей программы.



Если вы используете комбинацию команд `ReDim Preserve` для изменения размеров многомерного массива, повлиять можно только на размер последнего измерения. Допустим, был создан двухмерный массив:

```
Dim МойМассив(2, 5) As String
```

Теперь, если вы хотите увеличить данный массив, сохранив прежние значения, изменить можно только размер последнего измерения:

```
ReDim Preserve МойМассив(2, 17) As String
```



Если вы попытаетесь применить указанную комбинацию команд для увеличения размеров другого измерения (не последнего), Visual Basic .NET воспримет это как ошибку и откажется выполнять ваше указание.

## Создание структур

А что если вам нужно сохранить различные связанные между собой данные в одном месте? Обычная переменная может хранить в каждый конкретный момент времени только одно значение. Массив может хранить цепочки связанных между собой значений, но вам нужна именно одна переменная с несколькими значениями одновременно. К счастью, в Visual Basic .NET это можно сделать путем создания структуры данных, называемой (здесь не обойтись без тавтологии) *структурой*.

## Структуры и переменные

Структура, по сути, объединяет в себе несколько переменных вне зависимости от их типов данных. Чтобы создать структуру, нужно набрать такой код BASIC:

```
Structure ИмяСтруктуры
    ОбъявлениеПеременной1
    ОбъявлениеПеременной2
End Structure
```

Например, при необходимости создать структуру, содержащую имя, адрес и код клиента, наберите код:

```
Structure Клиент
    Public Имя As String
    Public Адрес As String
    Public Код As Integer
End Structure
```

Структуры относятся к пользовательским типам данных, поскольку после создания любой из них нужно создать еще и переменную, которая будет представлять эту структуру:

```
Переменная As ИмяСтруктуры
```

Чтобы создать переменную для представления структуры Клиент, наберите код:

```
Семенов As Клиент
```

Эта строка говорит Visual Basic .NET: “Создай переменную Семенов, которая будет представлять структуру Клиент. Данная структура содержит переменные Имя, Адрес и Код”.



Чтобы получить возможность использовать структуру, нужно пройти два этапа:

- ✓ определить структуру;
- ✓ объявить переменную для представления этой структуры.

## Сохранение данных

После того как будет создана структура и объявлена переменная, которая должна ее представлять, остается только присвоить этой переменной значения. Чтобы сделать это, нужно указать переменную, представляющую структуру, а через точку — ту переменную структуры, которой будет присвоено значение:

ИмяПеременной.ПеременнаяСтруктуры = НовоеЗначение

Допустим, вы создаете следующую структуру:

```
Structure Коты
    Public Имя As String
    Public Возраст As Integer
    Public Мальчик As Boolean
End Structure
```

Не забывайте, что структура создается в два этапа: вначале появляется сама структура, а затем объявляется переменная, которая будет эту структуру представлять. Поэтому следующим шагом должно быть объявление такой переменной:

```
Structure Коты
    Public Имя As String
    Public Возраст As Integer
    Public Мальчик As Boolean
End Structure
```

```
Dim МойКот As Коты
```

После того как будет объявлена переменная, представляющая структуру, можно приступить к сохранению данных. Для этого необходимо указать имя переменной, представляющей структуру, и имя переменной структуры, которой должно быть присвоено значение:

```
Structure Коты
    Public Имя As String
    Public Возраст As Integer
    Public Мальчик As Boolean
End Structure
```

```
Dim МойКот As Коты
```

```
МойКот.Имя = "Васька"
МойКот.Возраст = 3
МойКот.Мальчик = True
```

Вот как Visual Basic .NET интерпретирует эти коды.



1. В первых пяти строках создается структура Коты, состоящая из трех переменных: Имя (тип String), Возраст (тип Integer) и Мальчик (тип Boolean).
2. В шестой строке объявляется о создании переменной МойКот, которая будет

представлять структуру Коты.

3. В седьмой строке той части переменной МойКот, которой соответствует название Имя, присваивается значение Вася́ка.
4. В восьмой строке части переменной МойКот, которой соответствует название Возраст, присваивается значение 3.
5. А в девятой строке части переменной МойКот, которой соответствует название Мальчик, присваивается значение True.



Относитесь к структуре как к одной переменной, которая, тем не менее, может содержать сразу несколько значений.

## Комбинирование структур и массивов

Структуры сами по себе не очень-то полезны. Поэтому многие программисты используют их в комбинации с массивами. Этот прием позволяет создавать массивы, содержащие не просто связанные, но и сгруппированные (а ЗНАЧИТ, и правильно организованные) данные.

Чтобы создать массив, состоящий из структур, нужно просто создать структуру, а затем объявить массив, типом данных которого будет созданная структура (это означает, что каждый элемент массива будет представлять собой структуру):

```
Structure Коты
Public Имя As String
Public Возраст As Integer
Public Мальчик As Boolean
End Structure
```

```
Dim МоиКоты(3) As Коты
```

Как будет организован такой массив, показано на рис. 20.4.

Определение  
структуры

```
Structure Коты
Имя As String
Возраст As Integer
Мальчик As Boolean
End Structure
```

Объявление массива,  
состоящего из структур

```
Dim МоиКоты(3) As Коты
```

Имя = "Вася́ка" Возраст = 3 Мальчик = True	Имя = "Тишка" Возраст = 5 Мальчик = True	Имя = "Мурка" Возраст = 2 Мальчик = False	Имя = "Лева" Возраст = 1 Мальчик = True
МоиКоты(0)	МоиКоты(1)	МоиКоты(2)	МоиКоты(3)

Рис. 20.4. Создав массив структур, можно правильно и эффективно организовать связанные данные

# Коллекции данных

*Коллекция* — это специальная структура данных, работающая как супер-массив, который может расширяться до любых пределов и содержать в себе данные любых типов, начиная от строк и чисел и заканчивая другими массивами и структурами.

Одна из проблем обычных массивов заключается в том, что если они имеют большие размеры, то для поиска сохраненных данных иногда приходится просто просматривать поочередно все их элементы. Один из вариантов решения этой проблемы — назначение специальных "ключей" для каждого сохраненного элемента данных. Каждый ключ представляет собой строку и является уникальным, что позволяет по его значению быстро найти нужный элемент коллекции.

Чтобы создать коллекцию, нужно только объявить переменную, которая будет ее представлять:

```
Dim Поставщики As New Collection()
```

Эта строка говорит Visual Basic .NET: "Создай переменную, которая будет представлять коллекцию данных, и назови ее Поставщики".

## Добавление информации в коллекцию данных

Теперь, когда коллекция у нас уже имеется, можно приступить к созданию ее элементов. Для этого нужно указать само значение и уникальный ключ, по которому позднее можно будет найти это значение в коллекции данных. И данные, и ключ должны быть строками (тип String).

Код, добавляющий значение в коллекцию данных, должен выглядеть следующим образом:

```
ИмяКоллекции.Add (Значение, Ключ)
```

Вместо слова ИмяКоллекции наберите имя переменной, которая представляет эту коллекцию, вместо слова Ключ — уникальное (не совпадающее с другими ключами) текстовое значение, а вместо слова Значение — то значение, которое должно быть присвоено этому элементу (им может быть число, строка или другая структура данных).

Так, если нужно добавить к коллекции Поставщики название "Батлер и партнеры" и поставьте ему в соответствие ключ "13", наберите следующий код BASIC:

```
Dim Поставщики As New Collection()  
Поставщики.Add ("Батлер к партнеры", "13")
```



Если вам не нравится каждый раз брать ключ в кавычки, можете воспользоваться командой ToString и преобразовать числовое значение в строку:

```
Dim Поставщики As New Collection()  
Поставщики.Add ("Батлер и партнеры", 13.ToString)
```

## Определение количества элементов коллекции

Если вы уже что-то сохраняли в своей коллекции, то рано или поздно захотите проверить количество созданных элементов. Для этого достаточно набрать такой код BASIC:

```
ИмяКоллекции.Count
```

Возвращаемое этим кодом число в точности соответствует текущему количеству элементов вашей коллекции. Чтобы запомнить его, создайте переменную и присвойте ей это значение:

```
Dim X As Integer  
X = ИмяКоллекции.Count
```

Это же число можно использовать как часть другого кода BASIC, например в цикле For-Next (он будет описан в главе 25):

```
Dim X As Integer
For X = 0 to ИмяКоллекции.Count
    ' Здесь может быть любой код
Next X
```

## Чтение информации, сохраненной в коллекции

Для того чтобы найти какое-нибудь значение, сохраненное в коллекции, нужно воспользоваться такой командой BASIC:

```
ИмяКоллекции.Item(X)
```

Вместо буквы X наберите уникальный ключ, который соответствует нужному элементу. Так, если необходимо найти элемент коллекции Поставщики, которому соответствует ключ "13", наберите:

```
Поставщики.Item("13")
```

Этим кодом будет возвращено значение элемента, сохраненного вместе с ключом "13".

Указанное значение потом можно будет присвоить какой-нибудь другой переменной или использовать для проведения вычислений в других кодах BASIC:

```
Dim НаПечать As String
Dim Поставщики As New Collection()
Поставщики.Add ("Батлер и партнеры", "13")
НаПечать = Поставщики.Item("13")
```

В данном случае переменной НаПечать будет присвоено значение Батлер и партнеры.

## Удаление данных из коллекции

Удалить данные из коллекции еще проще, чем записать их туда. Для этого нужно всего лишь набрать такую команду BASIC:

```
ИмяКоллекции.Remove(X)
```

Как вы уже могли догадаться, вместо буквы X нужно набрать ключ элемента, подлежащего удалению. Если из коллекции Поставщики нужно удалить элемент, которому соответствует ключ "13", наберите:

```
Поставщики.Remove("13")
```



# Борьба с ошибками

В этой главе...

- Классификация ошибок
- > Стратегия поиска ошибок
- Ловушки для ошибок
- Ликвидация пойманных ошибок

**Д**аже если вы напишете тысячи программ, все равно последующие ваши программы не будут застрахованы от ошибок. Можно неправильно написать слово или, скажем, забыть набрать какую-то команду. И как бы аккуратно и внимательно вы ни старались писать коды программы, вряд ли она будет работать так, как вам того хотелось бы. А то, что заставляет программу работать некорректно (или вообще не работать), называется *ошибкой*.

Каждая созданная программа содержит ошибки, включая WordPerfect, Linux, Paint Shop Pro и Microsoft Windows XP. Основное различие между ошибками в вашей программе и ошибками в коммерческих продуктах заключается в том, что вам за исправление своих ошибок никто платить не станет. А исправлять их придется в любом случае.

Но не стоит расстраиваться. Многие ошибки вовсе не критичны. Они не могут остановить работу программы или испортить исходные данные; максимум, на что они способны, — это немного замедлить выполнение программы, отобразить на экране не тот цвет или оттенок и т.п.

Но существуют и более опасные ошибки. Так, говорят, что из-за программной ошибки недавно взорвался один из спутников NASA. Причиной взрыва послужила какая-то одна неправильно набранная команда.

Ничто не совершенно, поэтому каждая программа может содержать в себе ошибки. Даже программисты-профессионалы с учеными степенями регулярно пишут несовершенные программы.

Ошибки в программах — это жизненный факт. Они так же неизбежны, как тараканы на кухне. Вы никогда от них полностью не избавитесь, но постараться уничтожить их как можно больше вы просто обязаны.

## Классификация ошибок

Чтобы ошибку ликвидировать, ее нужно сначала обнаружить. Если программа маленькая, наподобие той, которая отображает на экране окно Hello, world!, у ошибок мало мест, где можно спрятаться. Но если это большая программа, ошибки могут быть повсюду, и найти их будет не легче, чем иголку в стоге сена.

Врага нужно знать в лицо, поэтому, чтобы облегчить процесс поиска, программисты делят ошибки на три категории:

- ✓ синтаксические ошибки;
- ✓ рабочие ошибки;
- ✓ логические ошибки.

## Синтаксические ошибки

*Синтаксические* ошибки обычно появляются при вводе кода, т.е. когда вы неправильно набираете какую-нибудь команду или ключевое слово. Например, если вы наберете Integer вместо Integer, Visual Basic .NET не зная, что означает слово Integer, даже не попытается продолжить выполнение программы.

Когда Visual Basic .NET сталкивается с такой ошибкой, он выделяет неправильно набранное слово, с тем чтобы вам легче было найти эту ошибку и устранить ее. Как только слово будет исправлено, программа снова станет рабочей.

Если программа содержит хотя бы одну синтаксическую ошибку, она работать не будет. Поэтому, как только ваша программа начнет функционировать, это будет означать, что синтаксических ошибок в ней точно нет. Теперь вас могут беспокоить только рабочие и логические ошибки.

## Рабочие ошибки

*Рабочие* ошибки более хитрые и менее очевидные, чем синтаксические, поскольку могут быть обнаружены только в процессе выполнения программы, когда она сталкивается с данными, которые не способна обработать. Программа может содержать массу таких ошибок, но вы даже не будете подозревать об их существовании до тех пор, пока она не пройдет испытания в так называемых полевых условиях.

Попробуем провести аналогию с повседневной жизнью. Представьте, что вы заходите в ресторан Mc'Donalds и на вопрос официанта "Что будете заказывать?" с серьезным видом отвечаете: "Два билета до Хельсинки, пожалуйста". Вряд ли найдется официант, который сразу сможет адекватно отреагировать на ваши слова. Но поскольку это человек, а не компьютер, через некоторое время он что-нибудь да ответит. Компьютер же просто прекратит работу (зависнет или остановит выполнение программы).

Вернемся опять к кодам BASIC и рассмотрим такой пример. Вот обычная формула, вычисляющая какой-то результат:

СреднийБал = СуммаБалов / КоличествоПопыток

Этот код работает корректно до тех пор, пока переменная КоличествоПопыток не принимает нулевое значение. Но поскольку делить на ноль нельзя, программа прекращает свою работу.

Чтобы обнаружить все рабочие ошибки, вам нужно протестировать работу программы при любых мыслимых ситуациях: начиная с того, что кто-то нажимает не ту клавишу, и заканчивая тем, что какой-то идиот в поле Возраст набирает отрицательное значение.

Поскольку количество возможных глупостей стремится к бесконечности (вспомните законы Мэрфи), теперь можно понять, почему каждая программа, тем более большая, содержит в себе ошибки. (Интересно, как вы теперь будете себя чувствовать, находясь в самолете, который управляется бортовым компьютером?)

## Логические ошибки

Самыми хитрыми и сложными с точки зрения устранения являются *логические* ошибки. Они случаются, если компьютер получает неправильную команду или если таковая входит в противоречие с другими командами. "Как же так? — удивитесь вы. — Кто мог дать неправильную команду, если кроме меня никто эту программу не писал?" Верите вы этому или нет, но набрать ошибочную команду очень даже легко.

Те, у кого есть дети-подростки, знают, что если попросить их убрать в своей комнате или купить что-нибудь к ужину, то они сделают это, но совсем не так, как вы это себе представляли. Вместо того чтобы убрать грязную одежду в стиральную машину, а разные бумажки выбросить в мусорное ведро, они затолкают это все под кровать или просто перенесут в другую комнату. А о том, что покупается к ужину, лучше вообще не вспоминать.



В любом случае дети выполняют ваши инструкции, но проблема в том, что эти инструкции были недостаточно точны. Если оставить им возможность для свободной интерпретации полученных указаний, они обязательно этим воспользуются. Компьютер поступит точно так же.

Поскольку вы уверены, что дали компьютеру правильные инструкции, вы не можете ответить на вопрос, почему он некорректно работает. А это значит, что теперь нужно искать те команды, которые оказались для него недостаточно точными. Если программа большая, скорее всего, вам придется просматривать в поисках ошибки все коды. Строка за строкой. (И скажите теперь, разве программирование не одно из самых приятных занятий?)

## Стратегия охоты за ошибками

Обычно процесс поиска и ликвидации ошибок проходит четыре стадии.

1. **Определение факта наличия ошибок в программе.**
2. **Поиск ошибок.**
3. **Поиск причин возникновения ошибок.**
4. **Ликвидация ошибок.**



Самое неприятное в этом процессе то, что, найдя и удалив одну ошибку, вы тем самым можете породить две или три новые. Теперь придется заново повторить весь процесс в поиске только что внесенных ошибок. Желаем удачи!

## А есть ли в программе ошибки?

Лучший способ определить, есть ли в программе ошибки, — это дать попользоваться ею посторонним людям. (В мире коммерческих разработок программного обеспечения таких посторонних людей обычно называют *покупателями*.)



Чем больше вы найдете людей, согласных протестировать вашу программу, тем больше вероятность того, что будут найдены даже самые скрытые ошибки. В числе этих ошибок могут быть как те, из-за которых зависает компьютер, так и те, которые просто возвращают недостаточно точный результат по причине, скажем, неправильного округления десятичных дробей.

После того как наличие ошибок в программе будет установлено, останется отследить и удалить их. (Программисты, которые с оптимизмом смотрят на вещи, называют найденные ошибки *незадокументированными возможностями*.)

## Поиск ошибок

Во всем процессе ликвидации ошибок этап их поиска считается самым трудоемким. Наиболее простой способ (он же и наиболее рутинный) обнаружения места в программе, где засела ошибка, — это запустить программу и шаг за шагом следить за процессом ее выполнения. В момент, когда ошибка себя проявит, вы сможете точно определить место, где она прячется.

Если программа маленькая, то такой путь вполне приемлем. Если же программа большая, вы рискуете окончательно испортить свою нервную систему.

Есть альтернативный вариант. Исследуйте пошагово лишь ту часть программы, которая, по-вашему, может содержать ошибку. Например, если программа, выводит на печать не то, что нужно, ошибка, надо полагать, содержится в тех кодах программы, которые сообщают компьютеру, что должно быть напечатано.

## Источник возникновения ошибки

Когда вы определили место, где прячется ошибка, нужно найти ее первоисточник (именно тот код, из-за которого происходит сбой в работе программы).

Предположим, программа должна выводить на печать ваше имя, но принтер почему-то печатает информацию о вашем семейном положении. С точки зрения компьютера процесс печати проходит корректно — печатаются те данные, относительно которых была получена команда вывода на печать.

Если вы сами немного подумаете, то поймете, что ошибка, вероятнее всего, кроется в коде, сохраняющем информацию о вашем имени, или в коде, извлекающем эту информацию из памяти для вывода на печать.

## Ликвидация ошибок

После того как будет найден код, являющийся непосредственным виновником возникновения ошибки, останется только его исправить. Но прежде чем это сделать, хорошенько подумайте! Иногда исправление одной ошибки провоцирует возникновение дюжины других.

Чтобы пояснить данную мысль, проведем следующую аналогию. Представьте, что в вашем доме начала протекать труба. Можно разрушить стену и полностью заменить трубу. Проблема с водопроводом будет решена, но, ломая стену, вы наверняка повредите перекрытия и по потолку пойдут трещины. Можно заменить потолок, положив вместо него бетонные плиты. Но это даст дополнительную нагрузку на фундамент, он просядет, и трещины пойдут уже по стенам. Теперь, наверное, легче будет просто построить новый дом. Итак, начав с починки водопровода, можно закончить списанием дома под снос.

С программными ошибками может происходить то же самое. Иногда легче просто переписать часть кодов программы, чем пытаться выловить из них все ошибки.

Лучший способ борьбы с ошибками — не делать их вовсе. Конечно, это примерно то же самое, что сказать: "Чтобы не испытывать недостатка в деньгах, нужно всегда их иметь".

Поскольку от ошибок не застрахована ни одна программа, вы можете лишь попытаться свести их количество к минимуму. Вот несколько советов, которые помогут вам в этом.



- ✓ Разбейте программу на много маленьких подпрограмм, каждая из которых будет решать какую-то одну задачу. Чем меньше программа, тем легче найти и удалить ошибки. На этом принципе основано объектно-ориентированное программирование, речь о котором пойдет в седьмой части книги.
- ✓ Тестируйте программу после каждого внесения в нее изменений. Если программа до изменения каких-то двух строк работала корректно, то понятно, что ошибка "сидит" именно в этих строках.
- ✓ Делайте перерывы на отдых. Если программа не работает и вы никак не можете понять в чем дело, оставьте ее на некоторое время и займитесь чем-нибудь другим. Когда вы вернетесь к ней снова, решение проблемы может прийти на ум само собой, и вы даже будете удивлены, как можно было сразу этого не понимать.

## Ловушки для ошибок

Программа должна состоять из абсолютно точных инструкций, указывающих компьютеру, что нужно делать в тот или иной момент. Если же компьютер попадает в ситуацию, когда не знает, что делать дальше, или если он сталкивается с проблемой, решить которую не в состоянии, происходит сбой в работе. Это может выражаться в том, что программа отказывает-

ся дальше функционировать (она "зависает"), или, в худшем случае, в том, что происходит сбой в работе всего компьютера.

Чтобы избежать подобных проблем, Visual Basic .NET предлагает вам возможность создания программных ловушек для ошибок, приводящих к серьезным сбоям в работе программы. Специальный кол говорит компьютеру: "Вот тебе инструкции на тот случай, если возникнет ситуация, с которой ты не сможешь справиться". Инструкции могут быть самыми простыми, например как те, что указывают на необходимость отобразить на экране сообщение. И, конечно же, это лучший вариант, чем перезагрузка компьютера.

Простейшая ловушка для ошибок выглядит следующим образом:

```
Try
' Код, работу которого нужно протестировать
Catch
' Инструкции на случай возникновения ошибок
End Try
```

Рассмотрим это на примере:

```
Try
    Dim intX As Integer
    irtX = 9 / CInt(TextBox1.Text)
Catch
    MsgBox ("Деление на ноль!")
End Try
```



1. Visual Basic .NET выполняет коды, расположенные между командами Try (первая строка) и Catch (четвертая строка). В данном случае во второй строке создается переменная intX типа Integer, а в третьей строке число 9 делится на число, которое пользователь набрал в текстовом поле TextBox1, а полученный результат присваивается переменной intX.
2. Коды, расположенные между командами Catch (четвертая строка) и End Try (шестая строка), выполняются только в том случае, если обнаружится ошибка при выполнении кодов, расположенных между командами Try (первая строка) и Catch (четвертая строка). В нашем примере при возникновении ошибки на экране отображается окно с сообщением "Деление на ноль!". Вы, наверное, уже догадались, что ошибка может возникнуть, если пользователь наберет в текстовом поле TextBox1 число 0.
3. Шестая строка (End Try) сообщает Visual Basic .NET, что на этом коды ловушки для ошибок заканчиваются.



Если ошибка произойдет при выполнении кодов, расположенных за пределами строк Try-Catch, данная ловушка компьютеру ничем не поможет.

Ловушки для ошибок сами по себе очень удобны и полезны, поскольку не позволяют компьютеру попасть в безвыходное положение и зависнуть. Благодаря этому вам при отладке программы намного реже придется тянуться к кнопке RESET.

# Средства Visual Basic .NET для отслеживания и удаления ошибок

В Visual Basic .NET есть два основных метода, позволяющих выявить и удалить ошибки: пошаговое выполнение программы и просмотр текущих значений.

- ✓ При пошаговом выполнении программы вы построчно отслеживаете производимые компьютером действия. Пока программа работает правильно, вы переходите от одной строки к другой. Как только в работе происходит сбой или переменная принимает некорректное значение, вы точно можете сказать, в какой строке содержится ошибка.
- ✓ Просматривая текущие значения переменных, вы можете видеть, какие данные используются на каждом шаге выполнения программы. Как только переменная принимает неправильное значение, вы сразу можете локализовать ошибку.

Обычно эти два метода используются совместно. Выполняя пошагово программу и просматривая принимаемые переменными значения, вы рано или поздно обнаружите ошибку.

## Пошаговое выполнение программы

Если вы совершенно не представляете, где могла спрятаться ошибка, придется пошагово выполнять программу, начиная с самой первой строки. При этом можно использовать одну из трех перечисленных далее команд.

- ✓ **Debug⇒Step Into** (Отладка⇒Пошаговое выполнение) или клавиша <F11> — каждый раз при выборе этой команды программа выполняет коды одной строки и переходит на другую строку. Коды всех процедур также выполняются пошагово.
- ✓ **Debug⇒Step Over** (Отладка⇒Пошаговое выполнение с перешагиванием через процедуры) или клавиша <F10> — коды программы также выполняются пошагово, но отдельные процедуры выполняются целиком. Если вы уверены, что данная процедура не содержит ошибок, выберите эту команду, и вам не придется выполнять пошагово все ее коды.
- ✓ **Debug⇒Step Out** (Отладка⇒Выход из процедуры) или комбинация клавиш <Shift+F10> — если вы начали пошаговое выполнение процедуры и решили, что в ней ошибок нет, эта команда позволяет быстро завершить ее.



Эти три команды можно использовать в любой последовательности. Вначале выберите команду **Step Into**, чтобы начать пошаговое выполнение программы. Затем, если вы вошли в какую-то процедуру и не хотите дальше пошагово выполнять ее, выберите команду **Step Out**. И, наконец, используйте команду **Step Over** при необходимости "перешагнуть" через процедуру, в правильности выполнения которой вы не сомневаетесь.

Вот как следует использовать команды **Step Into** и **Step Over** на практике.

1. Выберите команду **Debug⇒Step Into** (или нажмите клавишу <F11>) либо команду **Debug⇒Step Over** (или нажмите клавишу <F10>).

Visual Basic .NET откроет окно с кодами вашей программы, в котором слева от выполняемой в данный момент строки будет отображаться желтая стрелка (рис. 21.1).

2. Повторяйте первый шаг для каждой строки, точность выполнения которой вы хотите проверить. Если вы хотите завершить процедуру без пошагового выполнения оставшихся кодов, выберите команду **Debug⇒Step Out**.
3. Для того чтобы прервать процесс пошагового выполнения программы, выберите команду **Debug⇒Stop Debugging** (Отладка⇒Остановить отладку) или нажмите комбинацию клавиш **<Shift+F5>**.

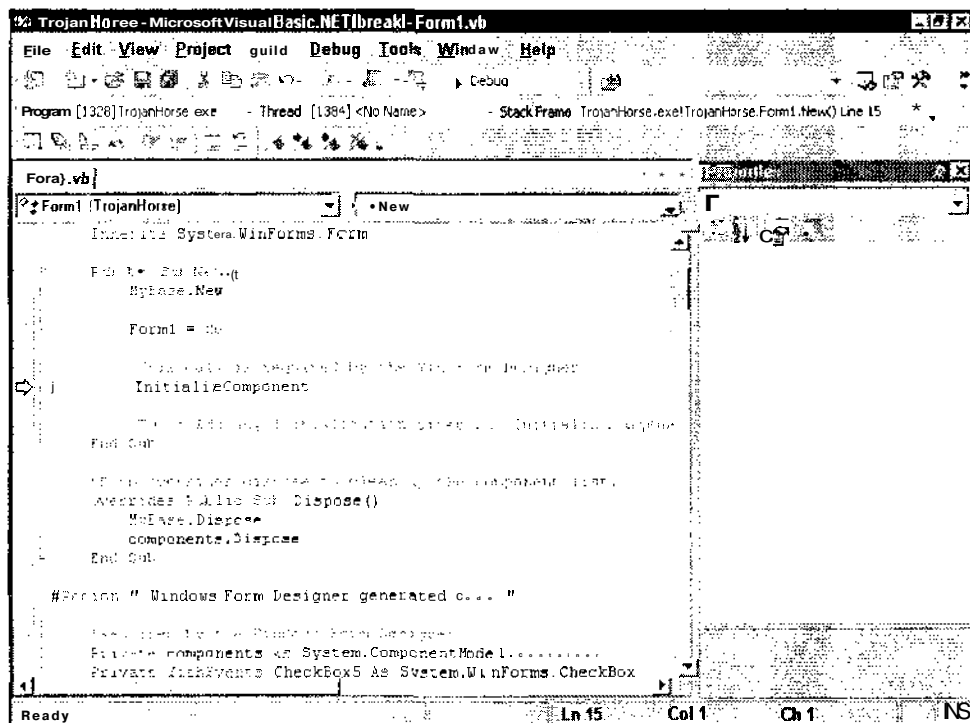


Рис. 21.1. VisualBasic.NET пошагово выполняет коды вашей программы



Если на экране отображается пользовательский интерфейс вашей программы, то, чтобы прервать процесс пошагового выполнения ее кодов, можно выбрать команду **Exit** — в таком случае завершается выполнение программы в целом.

## Определение точек останова

Команды **Step Into** и **Step Over** приступают к пошаговому выполнению программы, начиная с самой первой строки. Если программа невелика, то в этом нет ничего плохого. Но если программа большая, у вас не хватит терпения начинать каждый раз сначала (тем более что в этом нет никакой необходимости).

Чтобы избежать пошагового выполнения части кодов, которые, как вы надеетесь, работают правильно, используйте точку останова. *Точка останова* говорит компьютеру: “Выполни все коды до этой точки, а затем жди команду **Step Into** или **Step Over** для дальнейшего пошагового выполнения программы”.

Точка останова создается следующим образом.

1. Откройте окно кодов путем нажатия клавиши <F7>, выбора команды **View⇒Code (Вид⇒Коды)** или щелчка в окне Solution Explorer на кнопке View Code.
2. Щелкните на строке, где должна быть создана точка останова.
3. Нажмите клавишу <F9> (или щелкните правой кнопкой мыши и выберите в открывшемся меню команду Insert Breakpoint).

Слева от строки, выбранной в качестве точки останова, отобразится большая красная точка.

После того как точка останова будет создана, нажмите клавишу <F5> (или выберите команду **Debug⇒Start**), чтобы сразу выполнить все коды от начала программы до строки с точкой останова. Для дальнейшего пошагового выполнения программы можно использовать команды **Step Into** (клавиша <F11>) и **Step Over** (клавиша <F10>).

Чтобы удалить точку останова, повторите те же три шага, щелкнув на строке, к которой таковая относится.



Для того чтобы сразу удалить все точки останова, созданные в вашей программе, нажмите комбинацию клавиш <Ctrl+Shift+F9> или выберите команду **Debug⇒Clear All Breakpoints (Отладка⇒Удалить все точки останова)**.

## Просмотр значений переменных

Пошаговое выполнение программы может быть по-настоящему полезным, если вы при этом видите, какие изменения происходят с данными. Чтобы иметь возможность отслеживать текущие значения переменных, воспользуйтесь окном Watch (Просмотр).

Это окно говорит компьютеру: "Вот переменные, которые меня интересуют. В течение всего процесса пошагового выполнения программы отображай здесь их текущие значения".

Чтобы получить возможность пользоваться окном Watch, выполните такие действия.

1. Выберите команду **Debug⇒Step Into** (ей соответствует клавиша <F11>) или команду **Debug⇒Step Over** (клавиша <F10>).  
Visual Basic .NET откроет окно с кодами вашей программы.
2. Щелкните правой кнопкой мыши на переменной, значения которой вас интересуют.  
Откроется контекстное меню.
3. Выберите команду **Add Watch (Добавить для просмотра)**.
4. Выберите команду **Debug⇒Windows⇒Watch (Отладка⇒Окна⇒Просмотр)** или нажмите комбинацию клавиш <Ctrl+Alt+W>, чтобы отобразить на экране окно Watch (рис. 21.2).

При пошаговом выполнении программы в окне Watch постоянно будут отображаться текущие значения переменных, выбранных для просмотра.

5. Выберите команду **Debug⇒Step Into** (клавиша <F11>) или команду **Debug⇒Step Over** (клавиша <F10>).

В момент, когда очередная выполняемая строка изменит значение переменной, выбранной для просмотра на шаге 2, в окне Watch сразу же отобразится ее новое значение.

6. Выберите команду **Debug⇒Stop Debugging** или нажмите комбинацию клавиш <Shift+F5>, чтобы остановить пошаговое выполнение программы.

Остановив пошаговое выполнение программы вы можете вернуться к редактированию кодов.

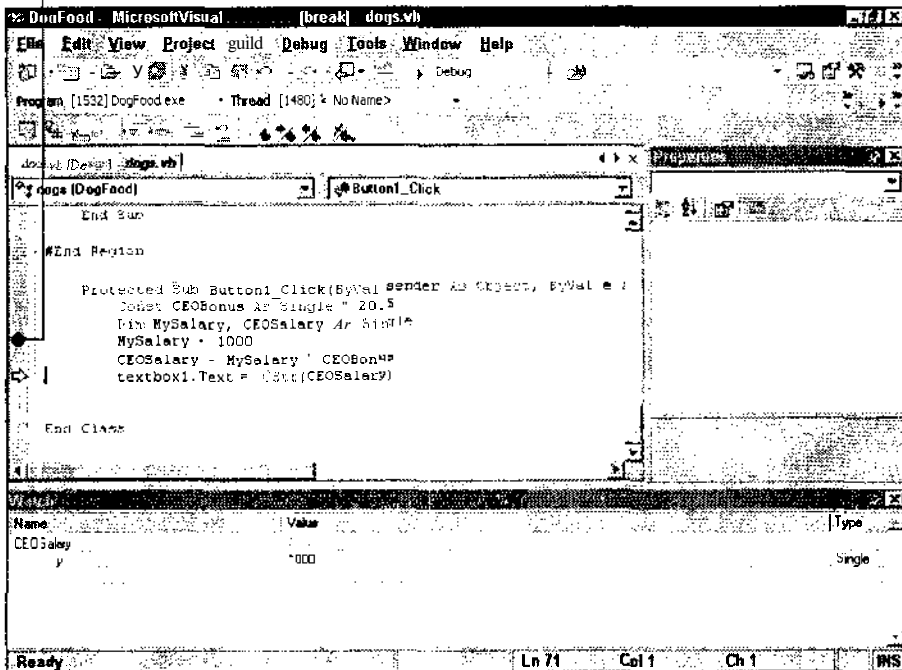


Рис.21.2. Окно Watch отображает текущие значения выбранных для просмотра переменных

## Тест на проверку полученных вами знаний

1. Что по-вашему является ошибкой?
  - а. Решение заняться программированием.
  - б. Тот факт, что команды Visual Basic .NET обозначаются английскими словами. С русскими словами все было бы намного проще.
  - в. Разные глупые действия, совершаемые неопытными пользователями.
  - г. Все, что заставляет программу работать не так как нужно или вообще не работать.
2. Как Visual Basic .NET помогает "отлавливать" ошибки?
  - а. Каждая ошибка выделяется определенным цветом и, когда вы на нее смотрите, начинает дрожать.
  - б. Когда вы сообщаете Visual Basic .NET, что намерены заняться "ловлей" ошибок, включается динамичная музыка, которая добавляет вам злости и решительности.
  - в. Позволяет выполнить программу пошагово и проследить за принимаемыми переменными значениями.
  - г. Не дает им размножаться.





## Часть V

# Создание разветвлений и циклов



### *В этой части...*

Любая программа представляет собой набор инструкций для компьютера, объясняющих, что ему делать дальше. Простейшие программы выглядят подобно переменной команд, подлежащих последовательному выполнению.

Но слепое выполнение полученных инструкций — это далеко не то, что способны делать серьезные программы. Большинство программ получают информацию от пользователя и сами решают, что с ней делать дальше.

Итак, данная часть книги посвящена вопросам создания программ, которые могут принимать решения самостоятельно. Кроме того, вы узнаете, как дать компьютеру инструкции, которые должны быть выполнены несколько раз подряд. Такие инструкции называются циклами. Они фактически говорят компьютеру: "Видишь этот набор команд? Их нужно выполнять снова и снова, пока я не разрешу тебе остановиться".

Ваша программа должна самостоятельно принимать решения и многократно повторять нужные команды. Лишь "живые" и динамичные программы в состоянии заставить компьютер делать что-то по-настоящему полезное.

# Условные операторы If-Then

*В этой главе...*

- > Использование логических значений
- Условные операторы If-Then и If-Then-End If
- Условные операторы If-Then-Else и If-Then-ElseIf

**С**амые примитивные программы последовательно выполняют некий набор инструкций, а затем останавливаются. Разумеется, программы, которые каждый раз выполняют одни и те же действия, не могут представлять особого интереса, поэтому нужно научить их изменять свое поведение в зависимости от поступающих извне данных. Этими данными могут быть щелчок мыши или нажатие клавиши, информация, введенная в текстовое поле формы или **взятая** из файла базы данных.

После того как данные получены, программа должна решить, как на них реагировать. Большинство решений принимается в форме Если-То (If-Then), например: "Если пользователь щелкнул на кнопке Печать, то нужно вывести открытый документ на печать". Если программу научить таким образом реагировать на поступление внешних данных и самостоятельно принимать решения о своих последующих действиях, она начнет поддерживать диалог с пользователем и будет выглядеть даже более **сообразительной**, чем это есть на самом деле.

## Логические значения

Чтобы иметь основание для принятия решения, нужно получить данные. Если данные получены, нужно решить, что делать дальше. Принятое программой решение зависит от выполнения или невыполнения *условия*, чему соответствуют значения True (Истина) и False (Ложь). Именно эти два значения называются *логическими*.

В Visual Basic .NET логические значения могут быть представлены:

- ✓ отдельными переменными;
- ✓ выражениями.

## Присвоение логических значений переменным

Логическое значение может быть присвоено только переменной, имеющей тип Boolean. Это означает, что она может принимать лишь одно из двух значений: True или False. Вот пример кода, посредством которого объявляются логические переменные:

```
Dim Flag, CheckMe As Boolean
```

Теперь эти переменные могут принимать только одно из двух логических значений:

```
Flag = True
```

```
CheckMe = False
```

Проверить значение логической переменной можно двумя способами. Первый из них — проверить, соответствует ли присвоенное значение значению True:

If Flag = True Then

Второй способ (более быстрый) позволяет сделать то же самое без необходимости набора = True:

If Flag Then



Проверить, соответствует ли значение переменной значению True, необязательно поскольку Visual Basic .NET в любом случае определяет, какое из двух значений присвоено переменной.

Чтобы проверить, присвоено ли переменной значение False, можно набрать

If Flag = False Then

Или можно убрать = False, а перед названием переменной набрать слово Not:

If Not Flag Then

## Логические значения выражений

Логические значения могут соответствовать не только отдельным переменным, но и группам переменных, собранных в отдельное выражение. В простейших выражениях сравнивается значение одной переменной с некоторым заданным значением, например:

Возраст > 21

Если переменная Возраст будет иметь значение, превышающее число 21, выражению будет соответствовать логическое значение True (Истина). Если же значение переменной будет равно или меньше числа 21, выражению будет соответствовать логическое значение False (Ложь).

Выражение которому соответствует одно из логических значений, может быть использовано в качестве условия оператора If-Then:

If Возраст > 21 Then

Вместо заданного постоянного значения в выражении можно использовать значение другой переменной — в таком случае сравниваться уже будут значения двух переменных:

If Возраст > ОграничениеПоВозрасту Then

Какое логическое значение будет соответствовать выражению Возраст > ОграничениеПоВозрасту, зависит от значений двух переменных: Возраст и ОграничениеПоВозрасту.

Сравнивать можно не только числа, но и строки. В приведенном ниже примере имя, набранное в текстовом поле txtName (оно сохраняется как значение свойства Text), сравнивается со строкой Степан Разин. Если набранное имя совпадает с этой строкой, выражение принимает значение True, если не совпадает — False:

If (txtName.Text = "Степан Разин") Then



В приведенном примере наличие круглых скобок вовсе не обязательно. Они просто помогают визуально выделить проверяемое выражение.

Теперь, когда вы знаете, что из себя представляют логические значения и как они работают, можно приступить к изучению работы условных операторов.



Если вы забыли, как обработать сразу несколько логических значений, вернитесь к разделу "Логические операторы" главы 17.

## Условный оператор *If-Then*

В Visual Basic .NET самым простым средством, с помощью которого программа может сделать какой-то выбор, является условный оператор *If-Then*. Он проверяет, соответствует ли значению или выражению логическое значение *True*. Если соответствует, программа выполняет определенную инструкцию, а если нет — эта инструкция игнорируется.

Вот как условный оператор *If-Then* выглядит в терминологии кодов BASIC:

If ЛогическоеЗначение Then Инструкция

Продемонстрируем это на примере:

If Количество > 25 Then txtNote.Text = "Заполнен!"

Visual Basic .NET интерпретирует данный код следующим образом.



1. Эта команда говорит компьютеру: "Проверь значение переменной Количество и, если оно больше числа 25, присвой свойству Text текстового поля txtNote значение Заполнен!".
2. Потом она, команда, добавляет: "Если значение переменной Количество меньше или равно числу 25, пропусти эту инструкцию и переходи к выполнению кодов следующей строки программы."

Рассмотрим другой пример:

If Красный Or Желтый Then Сообщение = "Стоп"

Вот как это воспримет Visual Basic .NET.



1. Эта команда говорит: "Проверь значения переменных Красный и Желтый. Если хотя бы одно из них равно значению *True*, присвой переменной Сообщение значение Стоп".
2. Затем она сообщает: "Если значения переменных Красный и Желтый совпадают со значением *False*, пропусти эту инструкцию и переходи к выполнению кодов следующей строки программы."

Условный оператор *If-Then* проверяет истинность или ложность условия, а затем выполняет или не выполняет какую-то одну инструкцию. Но что делать, если в зависимости от условия должно быть выполнено или не выполнено сразу несколько инструкций? В таком случае нужно воспользоваться разновидностью оператора *If-Then*, а именно оператором *If-Then-End If*.

## Оператор *If-Then-End If*

Этот оператор проверяет, соответствует ли выражению или переменной логическое значение *True*, и если это так, программа выполняет заданный набор инструкций. Синтаксис оператора *If-Then-End If* выглядит следующим образом:

If Условие Then

Инструкция1

Инструкция2

End If

Посредством данного кода программа говорит компьютеру: "Проверь, соответствует ли переменной или выражению Условие логическое значение True, и если соответствует, выполни все инструкции, указанные между словами Then и End If".

Рассмотрим пример:

```
If ПравильныйОтвет = True Then
    ЗадачаРешена = True
    txtMessage.Text = "Отличный результат!"
End If
```

Вот как Visual Basic .NET интерпретирует эти коды.



1. Первая строка дает указание: "Проверь значение переменной Правильный-Ответ. Если оно совпадает со значением True, переходи к выполнению следующей строки, а если не совпадает, переходи к строке End If".
2. Вторая строка: "Присвой переменной ЗадачаРешена значение True".
3. Третья строка: "Присвой свойству Text объекта txtMessage значение Отличный результат".
4. Наконец, четвертая строка сообщает: "На этом выполнение оператора If-Then-End If заканчивается".

## Оператор If-Then-Else

Условные операторы If-Then и If-Then-End If выполняют какие-либо инструкции только в том случае, если переменной или выражению соответствует логическое значение True (т.е. проверяемое условие выполняется). А как быть, если нужно, чтобы при невыполнении условия также происходили определенные действия?

В этом случае вам нужно воспользоваться оператором If-Then-Else. Выглядит он следующим образом:

```
If Условие Then
    Инструкции1
Else
    Инструкции2
End If
```

Оператор говорит компьютеру: "Если переменной или выражению Условие соответствует значение True, выполни набор команд Инструкции1. В противном случае, если переменной или выражению Условие соответствует значение False, выполни набор команд Инструкции2".

Вообще говоря, можно ограничиться использованием обычного оператора If-Then и набрать примерно такой код:

```
If Курс > 15 Then txtBox1.Text = "Проливайте акции!"
If Курс <= 15 Then txtBox1.Text = "Подождите еще."
```

Данный код работает корректно, но все же выглядит несколько "топорно" и непрофессионально. Чтобы поправить такое положение, воспользуйтесь оператором If-Then-Else:

```
If Курс > 15 Then
    txtBox1.Text = "Продавайте акции!"
```

```
Else
  txtBox1.Text = "Подождите еще."
End If
```

Выражение `Курс > 15` можно заменить противоположным (`Курс <= 15`) и набрать такой код:

```
If Курс <= 15 Then
  txtBox1.Text = "Подождите еще."
Else
  txtBox1.Text = "Продавайте акции!"
End If
```

Оба кода с операторами `If-Then-Else` выполняют одни и те же действия, и не имеет значения, какой из них вы наберете.



Между строками `If-Then` и `Else`, а также `Else` и `End If` можно ввести любое количество инструкций.

Единственный возможный недостаток условного оператора `If-Then-Else` заключается в том, что при невыполнении условия второй набор инструкций реализуется автоматически. Если же вы хотите, чтобы ваше условие проверялось для каждого набора инструкций, воспользуйтесь оператором `If-Then-ElseIf`.

## Оператор *If-Then-ElseIf*

Синтаксис этого оператора таков:

```
If Условие1 Then
  Инструкции1
ElseIf Условие2 Then
  Инструкции2
End If
```

Этот код дает компьютеру такие указания: “Если выражению или переменной `Условие1` соответствует значение `True`, выполни набор команд `Инструкции1`. Если выражению или переменной `Условие1` соответствует значение `False`, проверь, какое значение соответствует переменной или выражению `Условие2`. Если `True` — выполни набор команд `Инструкции2`, если `False` — не делай ничего, т.е. заканчивай выполнение данного оператора”.



При использовании оператора `If-Then-Else` в любом случае будет выполнен какой-то набор команд. Если же вы используете оператор `If-Then-ElseIf` и ни одному из проверяемых условий не соответствует значение `True`, никакие команды выполняться не будут.

Рассмотрим такой пример:

```
If Курс > 15 Then
  txtBox1.Text = "Продавайте акции!"
ElseIf Курс > 10 Then
  txtBox1.Text = "Подождите еще."
End If
```

Вот что произойдет, если переменная `Курс` будет иметь значение 12.



1. Visual Basic .NET проверит первое условие и установит, что ему соответствует логическое значение False (так как выражение  $12 > 15$  неправильно).
2. Затем Visual Basic .NET проверит второе условие. Ему будет соответствовать значение True ( $12 > 10$ ), а потому будет выполнена вторая инструкция и свойству Text объекта `textBox1` будет присвоено текстовое значение Подождите еще.

А теперь посмотрим что произойдет, если переменная Курс получит значение 5.



1. Visual Basic .NET проверит первое условие и определит, что оно не выполняется (так как выражение  $5 > 15$  не соответствует действительности).
2. Затем Visual Basic .NET проверит второе условие, которое тоже не выполняется (поскольку 5 не может быть больше 10), а потому вторая инструкция также будет проигнорирована.
3. Далее следует строка `End If`, свидетельствующая об окончании работы этого условного оператора. Таким образом, ни одна из инструкций не будет выполнена.

Нужно отметить, что возможности оператора `If-Then-ElseIf` не ограничиваются проверкой только двух условий. Об этом читайте в следующем разделе.

## Если вариантов должно быть много

Если необходимо учесть сразу несколько возможных вариантов, просто добавьте еще несколько строк `ElseIf`:

```
If Условие1 Then
    Инструкции1
ElseIf Условие2 Then
    Инструкции2
ElseIf Условие3 Then
    Инструкции3
End If
```

Этот код говорит компьютеру: "Если выражению или переменной `Условие1` соответствует значение True, выполни набор команд `Инструкции1`!. Если выражению или переменной `Условие1` соответствует значение False, проверь, какое значение соответствует переменной или выражению `Условие2`. Если True — выполни набор команд `Инструкции2`, если False — проверяй `Условие3`. Если ему соответствует значение True, выполняй набор команд `Инструкции3`. Но если ни одно из условий не выполняется, ничего не делай и переходи к следующим кодам программы."

На общее количество строк `ElseIf`, использованных одним и тем же оператором, ограничений не существует, поэтому их можно добавить столько, сколько нужно для учета всех возможных вариантов. Но не усложняйте себе жизнь и не занимайтесь построением слишком больших логических конструкций (если не выполняется условие номер 27, тогда...) Помните; все гениальное — просто.

## Если должен быть выбран хотя бы один вариант

Даже если вы учтете, используя оператор `If-Then-ElseIf`, целое множество возможных вариантов, в соответствии с которыми будут выполняться те или иные действия, все рав-



но может случиться так, что ни одно из действий выполнено не будет (если не будет выполняться ни одно из предложенных условий). Если же вы хотите быть уверены, что хоть какие-то инструкции обязательно будут выполнены, наберите их последними и поставьте перед ними ключевое слово `Else`:

```
If Условие1 Then
    Инструкции1
ElseIf Условие2 Then
    Инструкции2
Else
    Инструкции3
End If
```

Этот код говорит компьютеру: “Если выражению или переменной `Условие1` соответствует значение `True`, выполни набор команд `Инструкции1`!. Если выражению или переменной `Условие1` соответствует значение `False`, проверь, какое значение соответствует переменной или выражению `Условие2`. Если `True` — выполни набор команд `Инструкции2`, если `False` — набор команд `Инструкции3`”.

## Использование вложенных операторов *If-Then*

Если какой-либо действия должны быть предприняты только в случае выполнения сразу нескольких условий, можно вложить несколько операторов `If-Then` один в другой и последовательно проверить каждое из условий. Например:

```
If Возраст > 15 Then
    If ID > 120 Then
        txtMessage.Text = "Вы нам подходите"
    End If
Else
    txtMessage.Text = "Извини, малыш, эта работа пока_
                        не для тебя"
End If
```

Посмотрим, что произойдет, если переменная `Возраст` будет иметь значение `18`, а переменная `ID` — значение `147`.



1. Visual Basic .NET проверит условие первой строки, определит, что оно выполняется ( $18 > 15$ ), и перейдет ко второй строке.
2. Второе условие также будет выполняться ( $147 > 120$ ), поэтому свойству `Text` объекта `txtMessage` будет присвоено текстовое значение `Вы нам подходите`.

Теперь рассмотрим ситуацию, когда значению переменной `Возраст` будет соответствовать число `18`, а переменной `ID` — число `110`.



1. Visual Basic .NET проверит условие первой строки, определит, что оно выполняется ( $18 > 15$ ), и перейдет ко второй строке.
2. Второе условие выполняться не будет (выражению  $110 > 120$  соответствует логическое значение `False`), поэтому ничего не произойдет и свойству `Text` объекта `txtMessage` никакое новое значение присвоено не будет.

И наконец, рассмотрим ситуацию, когда переменная **Возраст** будет иметь значение 14, а переменная **ID** — значение 125.



1. Visual Basic .NET проверит условие первой строки, определит, что оно не выполняется (13 меньше 15) и перейдет к строке с ключевым словом **Else**.
2. Свойству **Text** объекта **txtMessage** будет присвоено текстовое значение **Извини, малыш, эта работа пока не для тебя, и выполнение условного оператора на этом будет закончено**. Обратите внимание, что в случае невыполнения первого условия, второе условие даже не проверяется.



Очень внимательно относитесь к вложенным друг в друга условным операторам, поскольку набранный код может работать не совсем так, как вы себе это представляете. Например, вы наверняка могли бы упустить из виду второй рассмотренный вариант, при котором свойству **Text** объекта **txtMessage** не присваивается никакое значение.

### Тест на проверку полученных вами знаний

#### 1. Что такое логические значения?

- а. Это такие значения, которые поддаются нормальной логике.
- б. Если вы можете понять, что означает то или иное значение, значит, оно логично.
- в. Если компьютер может "понять", что вы для него набрали, значение считается логическим.
- г. Это значения **True** и **False**, которые используются при проверке условий условными операторами.

#### 2. Какой из предложенных вариантов идентичен коду

**If Проблемы = False Then?**

- а. **If Проблемы Then "А у кого их •не бывает?"**
- б. **If Проблемы Then "Продайте их кому-нибудь"**
- в. **If Not Проблемы Then**
- г. **If Проблемы = False And True Then "Ребята, не создавайте таким кодом проблему для компьютера!"**

# Оператор выбора Select Case

*В этой главе...*

- > Знакомство с оператором Select Case
- > Как добиться того, чтобы программа выполнила хотя бы одну из возможных инструкций
- > Использование вложенных условных операторов

**О**сновным недостатком рассмотренных в предыдущей главе условных операторов If-Then-ElseIf является их громоздкая структура и сложность для написания, чтения и понимания. Попробуйте понять смысл, скажем, этих инструкций:

```
If Оценка = 5 Then
    TxtReply.Text = "Отлично"
ElseIf Оценка = 4 Then
    TxtReply.Text = "Хорошо, но можно лучше"
ElseIf Оценка = 3 Then
    TxtReply.Text = "Вы способны на большее, _попробуйте еще раз"
ElseIf Оценка = 2 Then
    TxtReply.Text = "Нет, сегодня не Ваш день"
End If
```

Вы спросите, есть ли какая-то альтернатива этой конструкции и можно ли вместо оператора If-Then-ElseIf использовать нечто более понятное и наглядное?

Конечно, альтернатива есть, и не одна. Можно, в частности забросить Visual Basic .NET и попробовать изучить какой-нибудь другой язык программирования. Но разумней будет просто воспользоваться оператором выбора Select Case.

## *использование оператора выбора Select Case*

Оператор Select Case выглядит следующим образом:

```
Select Case ИмяПеременной
    Case X
        Инструкции1
    Case Y
        Инструкции2
    Case Z
        Инструкции3
End Select
```

Данный оператор говорит Visual Basic .NET: "Проверь значение переменной ИмяПеременной. Если оно совпадает со значением X, выполни набор команд Инструкции!, если со значением Y — набор команд Инструкции2, если со значением Z — набор команд Инструкции3".

Если в рассмотренном в начале главы примере заменить условный оператор If-Then-ElseIf оператором выбора Select Case, код программы будет выглядеть намного проще:

```
Select Case Оценка
Case 5
    TxtReply.Text = "Отлично"
Case 4
    TxtReply.Text = "Хорошо, но можно лучше"
Case 3
    TxtReply.Text = "Вы способны на большее, _
    попробуйте еще раз"
Case 2
    TxtReply.Text = "Нет, сегодня не Ваш день"
End Select
```

Обратите внимание, что здесь уже не нужно каждый раз повторять такие слова, как ElseIf и Then.

В операторе Select Case нет ограничения на количество строк Case, поэтому вы можете создать их столько, сколько нужно для учета всех возможных вариантов.

## *использование оператора Select Case с операторами сравнения*



Обычно оператор Select Case проверяет точное совпадение значения переменной с несколькими заданными значениями. Но если использовать его совместно с операторами сравнения, такими как <, <= и о, можно проверять принадлежность значения переменной к разным диапазонам значений.

Чтобы оператор Select Case и операторы сравнения можно было использовать вместе, перед каждым оператором сравнения нужно добавить ключевое слово is:

```
Select Case Показатель
Case is > 20
    TxtReadMe.Text = "Хороший результат"
Case is > 10
    TxtReadMe.Text = "Попробуйте еще раз"
End Select
```

Этому коду эквивалентен код с использованием оператора If-Then:

```
If Показатель > 20 Then
    TxtReply.Text = "Хороший результат"
Elseif Показатель > 10 Then
    TxtReply.Text = "Попробуйте еще раз"
End If
```

## *Если хотя бы один вариант должен быть выбран*

Может случиться так, что значение переменной не будет совпадать ни с одним из предложенных значений или что оно не будет относиться ни к одному из предложенных диапазо-

нов. Но как быть, если нужно, чтобы хотя бы какой-то один набор инструкций был выполнен в любом случае. Тогда, как и при использовании оператора If-Then-ElseIf, следует ввести ключевое слово Else:

```
Select Case Показатель
Case 1
    Инструкции1
Case 2
    Инструкции2
Case 3
    Инструкции3
Case Else
    Инструкции4
End Select
```



Этот код говорит компьютеру: “Если значение переменной Показатель совпадает с числом 1, выполни набор команд Инструкции1, если с числом 2 — набор команд Инструкции2, а если с числом 3 — набор команд Инструкции3. Если значение переменной Показатель не совпадает ни с одним из этих трех чисел, выполни последний набор команд”.

## Использование вложенных условных операторов

Маленькие дети очень любят играть с матрешками. Когда они открывают одну матрешку, внутри находят другую, во второй — еще одну, потом еще одну. И так они доходят до самой маленькой, которая уже не открывается.

Оператор Select Case обычно включает в себя несколько наборов простых инструкций:

```
Select Case ID
Case 123
    ChkFrank.Value = True
Case 124
    ChkBob.Value = True
Case 125
    ChkMartha.Value = True
End Select
```

Это простейший вариант. Но он, подобно матрешкам, может содержать несколько вложенных друг в друга операторов Select Case и If-Then. Рассмотрим такой пример:

```
Select Case ID
Case 120
    Select Case Возраст
    Case is <= 9
        txtРезультат.Text = "У Вас очень умный ребенок"
    End Select
End Select
```

Вот как этот код будет воспринят компьютером.



1. Первая строка говорит: "Проверь значение переменной ID и переходи на вторую строку".
2. Вторая строка: "Если значение переменной ID совпадает с числом 120, переходи на третью строку. Если переменная ID имеет любое другое значение (119, 121 или 34), переходи сразу на седьмую строку".
3. Третья строка: "Проверь значение переменной Возраст и переходи на четвертую строку".
4. Четвертая строка: "Если значение переменной Возраст равно или меньше числа 9, переходи на пятую строку, в противном случае переходи сразу на шестую строку".
5. Пятая строка: "Присвой свойству Text объекта txtРезультат значение У Вас очень умный ребенок".
6. Шестая строка: "Этой строкой заканчивается выполнение вложенного оператора Select Case".
7. Седьмая строка: "Этой строкой заканчивается выполнение первого оператора Select Case".

Visual Basic .NET предоставляет полную свободу в использовании условных операторов и операторов выбора, позволяя применять операторы If-Then внутри операторов Select Case, а операторы Select Case — **внутри** операторов If-Then.



Теоретически использовать можно любое количество вложенных друг в друга операторов, но чем меньше вы их задействуете, тем проще будут коды вашей программы для чтения и понимания. В общем случае считается, что если кто-то использует больше трех вложенных друг в друга операторов, то, скорее всего, этот человек просто не понимает, что делает.



В случае применения нескольких вложенных операторов добавляйте отступы, чтобы легче было понять, где каждый из операторов начинается и заканчивается. Ведь разобраться в кодах, набранных без отступов, очень непросто:

Select Case Баллы

Case 1200

If Имя = "Джон" Then

txtReview.Text = "Да, брат, сегодня ты не в форме"

ElseIf Имя = "Джессика" Then

txtReview.Text = "Поздравляю! Ты делаешь успехи"

End If

End Select

А вот те же коды, но с отступами в начале строк:

Select Case Баллы

Case 1200

If Имя = "Джон" Then

txtReview.Text = "Да, брат, сегодня ты не в форме"

ElseIf Имя = "Джессика" Then

txtReview.Text = "Поздравляю! Ты делаешь успехи"

End If

End Select

Для компьютера оба эти фрагмента абсолютно одинаковы. Но с точки зрения программиста, как и любого другого человека, второй вариант намного удобней и понятней.

### Тест на проверку полученных вами знаний

1. Существует ли ограничение на количество вложенных друг в друга операторов If-Then и Select Case?
  - а. Да. Программистам, не имеющим двухлетнего стажа работы, запрещается вкладывать более 200 операторов в день (Минздрав запрещает).
  - б. Рекомендуется использовать не более 300 вложенных друг в друга операторов, так как замечено, что попытки написать код с большим числом вложенных операторов превращаются в пагубную привычку.
  - в. Нет. Их можно вкладывать хоть до двух часов ночи.
  - г. Такого ограничения не существует, но если используется слишком много вложенных операторов, код программы становится трудно читаемым и малопонятным.
2. Что нужно сделать, чтобы коды вложенных друг в друга операторов стали легче для чтения и понимания?
  - а. Нужно вообще отказаться от использования вложенных операторов, тогда все будет предельно просто и понятно.
  - б. Все команды следует набрать на русском языке, и картина несколько прояснится.
  - в. Нужно использовать минимальное количество вложенных операторов и добавлять отступы для выделения каждого из них.
  - г. Необходимо составить краткий конспект этой главы и добавить его к коду программы в качестве комментария.





# Создание циклов

*В этой главе...*

- Циклы, которые могут не выполняться
- Циклы, выполняющие хотя бы одну итерацию
- Сравнение свойств разных циклов

**И**ногда необходимо, чтобы программа выполнила несколько раз подряд один и тот же набор инструкций. Писать один и тот же код несколько раз подряд неразумно. Ведь можно же написать набор необходимых инструкций один раз, а затем дать указание компьютеру выполнить его нужное количество раз.

Код, содержащий инструкции для многократного повторения, называется *циклом*. В Visual Basic .NET различают циклы трех типов:

- ✓ циклы, которые могут не выполняться;
- ✓ циклы, набор инструкций которых выполняется по крайней мере один раз;
- ✓ циклы, которые выполняются фиксированное количество раз.

В этой главе рассматриваются циклы первых двух типов. Циклы третьего типа, выполняющиеся фиксированное количество раз, описаны в главе 25.

## *Циклы, которые могут не выполняться*

Циклы заставляют компьютер выполнять одни и те же инструкции снова и снова. Как же определить, сколько именно должно быть повторений? На помощь опять приходят логические значения (о них было рассказано в главе 22).

Visual Basic .NET перед выполнением каждой итерации проверяет условие и продолжает выполнение цикла до тех пор, пока ему соответствует логическое значение True, или, наоборот, до тех пор, пока ему соответствует логическое значение False. Если нужно выполнять цикл, пока условию соответствует значение True, используется цикл Do-While (Выполнять-Пока). Если же нужно выполнять цикл, пока условию соответствует значение False, используется цикл Do-Until (Выполнять-ПокаНе).

Если условию цикла Do-While сразу же соответствует логическое значение False, набор его команд выполняться не будет и программа перейдет к выполнению следующих кодов. Точно так же, если условию цикла Do-Until сразу же соответствует логическое значение True, набор команд этого цикла ни разу выполнен не будет.

## Циклы Do-While

Эти циклы продолжают выполняться до тех пор, пока условию соответствует логическое значение True. Их код выглядит так:

Do While Условие  
' Набор инструкций  
Loop

Как только Visual Basic .NET видит такой код, он проверяет, какое логическое значение соответствует переменной или выражению, выполняющим роль условия, и если условию соответствует значение True, выполняется набор инструкций.

Рассмотрим это на примере:

```
Dim Счетчик As Integer
Счетчик = 0
Do While Счетчик <> 5
    Счетчик = Счетчик + 1
    txtСчетчик.Text = CStr(Счетчик)
Loop
```

Вот как Visual Basic .NET будет выполнять этот код.



1. В первой строке создается переменная типа Integer и ей присваивается имя Счетчик.
2. Во второй строке переменной Счетчик присваивается нулевое значение.
3. Третья строка говорит Visual Basic .NET: "Пока значение переменной Счетчик не будет равно числу 5, выполняй снова и снова инструкции, набранные между строками Do While и Loop".
4. В четвертой строке к значению переменной Счетчик прибавляется число 1.
5. Пятая строка говорит: "Возьми значение переменной Счетчик, преобразуй его в текстовое значение и присвой свойству Text объекта txtСчетчик".
6. Шестая строка говорит: "На этом список инструкций заканчивается. Возвращайся к третьей строке и снова проверяй условие".

На каждой итерации (итерация — одноразовое выполнение набора команд) значение переменной Счетчик увеличивается на единицу. В тот момент, когда оно станет равным 5, условие Счетчик <> 5 примет значение False и выполнение цикла завершится.



Если по какой-то причине условие постоянно будет выполняться (и ему все время будет соответствовать значение True), инструкции цикла будут повторяться снова и снова, в результате чего программа "зависнет". Чтобы этого не происходило, всегда добавляйте к набору команд цикла по крайней мере одну инструкцию, которая рано или поздно сможет сделать так, чтобы условие перестало выполняться.

## Циклы Do-Until

Эти циклы проверяют условие и, если оно ложно, выполняют один и тот же набор команд, пока условие не станет истинным. Выглядят они так:

```
Do Until Условие
    ' Набор инструкций
Loop
```

Если Visual Basic .NET видит такой код, он проверяет условие и, если оно ложно, выполняет заданный набор инструкций, а затем снова проверяет условие. Как только условие становится истинным, выполнение цикла прекращается.

Рассмотрим такой пример:

```
Dim Счетчик as Integer
Счетчик = 0
Do Until Счетчик > 4
    Счетчик = Счетчик + 1
Loop
```

Вот как этот код будет интерпретирован Visual Basic .NET.



1. В первой строке создается переменная типа Integer и ей присваивается имя Счетчик.
2. Во второй строке переменной Счетчик присваивается нулевое значение.
3. Третья строка говорит: "Это первая строка цикла Do-Until. До тех пор пока условие Счетчик > 4 ложно, продолжай выполнять инструкции, набранные между этой строкой и строкой Loop. Как только условие станет истинным (это произойдет когда переменная Счетчик примет значение 5), заканчивай выполнение этого цикла и переходи к следующим кодам программы".
4. В четвертой строке значение переменной Счетчик увеличивается на единицу.
- 5\* Пятая строка говорит: "На этом набор инструкций цикла заканчивается. Возвращайся опять к проверке условия".

На каждой итерации значение переменной Счетчик увеличивается на единицу. Как только оно станет равным 5, условие примет истинное значение и выполнение цикла прекратится.



Чтобы цикл не превратился в бесконечный и программа не зависла, убедитесь, что по крайней мере одна из команд цикла может через какое-то количество итераций сделать условие истинным.

## *Циклы, выполняющие не менее одной итерации*

Иногда возникает необходимость, чтобы набор инструкций цикла выполнялся хотя бы один раз. Циклы Do-While и Do-Until для решения такой задачи не подойдут, поскольку они вначале проверяют условие, а затем могут выполнить или не выполнить набор инструкций. В Visual Basic .NET есть еще два типа циклов, которые предназначены для решения именно таких задач. Они вначале выполняют одну итерацию, затем проверяют условие и, в зависимости от того, выполняется оно или нет, переходят к новой итерации или завершают свою работу.

### **Циклы Do-Loop Until**

Эти циклы продолжают выполнение итераций до тех пор, пока проверяемое условие не станет истинным. Выглядят они так:

Do

' Набор инструкций

Loop Until Условие

Такой код говорит Visual Basic .NET: "Выполни набор инструкций, затем проверь условие. Если оно истинно, переходи к следующим кодам программы, а если ложно, начинай все сначала".

Рассмотрим следующий пример:

```
Dim Счетчик As Integer
```

```
Счетчик = 0
```

Do

```
Счетчик = Счетчик + 1
```

```
Loop Until Счетчик > 4
```

Вот что этот код означает для Visual Basic .NET.



1. В первой строке создается переменная типа `Integer` и ей присваивается имя `Счетчик`.
2. Во второй строке переменной `Счетчик` присваивается нулевое значение.
3. Третья строка говорит: "Отсюда начинается выполнение инструкций цикла `Do-Loop Until`".
4. В четвертой строке значение переменной `Счетчик` увеличивается на единицу.
5. Пятая строка говорит Visual Basic .NET: "На этом выполнение набора инструкций цикла `Do-Loop Until` завершается. Проверь, какое логическое значение соответствует условию, представленному выражением `Счетчик > 4`. Если `False`, переходи к строке, с которой начинается этот цикл, и снова выполняй набор его инструкций, если `True` — переходи к реализации следующих кодов программы".

При каждом повторном выполнении набора инструкций цикла значение переменной `Счетчик` увеличивается на единицу. Когда оно станет равным 5, условие будет считаться истинным, а выполнение цикла остановится.



Усоедитесь, что повторные итерации рано или поздно приведут к тому, что проверяемое условие станет истинным. В противном случае цикл будет выполняться до бесконечности и программа просто зависнет.

## Циклы Do-Loop While

Подобные циклы нередко можно встретить в повседневной жизни. Суть их в том, что они повторяются до тех пор, пока выполняется какое-то условие. Например, если ребенку насыпать в тарелку его любимую кашу и дать ложку, то он будет опускать ложку в тарелку, набирать кашу и возвращать ложку себе в рот. Этот "цикл" будет повторяться до тех пор, пока в тарелке будет оставаться каша. Как только каша закончится, выполнение цикла прекратится. Ребенок, конечно, может потребовать добавку, но это будет уже другой цикл.

В кодах BASIC цикл `Do-Loop While` выглядит так:

Do

' Набор инструкций

Loop While Условие

Код говорит Visual Basic .NET: "Выполни набор инструкций и проверь, какое логическое значение соответствует выражению или переменной, выступающим в качестве условия. Если это True, переходи к строке Do и снова выполняй набор инструкций, а если False — заканчивай выполнение данного цикла и переходи к следующим кодам программы".

Проиллюстрируем сказанное таким примером:

```
Dim Счетчик As Integer
Счетчик := 0
Do
    Счетчик = Счетчик + 1
Loop While Счетчик < 5
```

Вот как эти коды будут восприняты Visual Basic .NET.



1. В первой строке создается переменная типа Integer и ей присваивается имя Счетчик.
2. Во второй строке переменной Счетчик присваивается нулевое значение.
3. Третья строка говорит: "Отсюда начинается выполнение инструкций цикла Do-Loop While".
4. В четвертой строке значение переменной Счетчик увеличивается на единицу.
5. Пятая строка говорит Visual Basic .NET: "На этом выполнение набора инструкций цикла Do-Loop While завершается. Проверь, какое логическое значение соответствует условию, представленному выражением Счетчик < 5. Если таковым является True, переходи к строке, с которой начинается этот цикл, и снова выполняй набор его инструкций. Но если это значение False, переходи к выполнению следующих кодов программы".

При каждом повторном выполнении набора инструкций цикла значение переменной Счетчик увеличивается на единицу. Когда оно станет равным 5, условие перестанет выполняться и цикл будет остановлен.



Убедитесь, что повторные итерации рано или поздно приведут к тому, что проверяемое условие станет ложным. В противном случае программа зависнет.

## Какой цикл лучше?

Хотя некоторые циклы выглядят очень похожими друг на друга, выполняемые ими действия заметно различаются. Чтобы не запутаться, имеет смысл выбрать из них один-два и всегда стараться при написании программы использовать только их. В таком случае вам будет легче писать программу, а другим программистам будет легче читать ее коды.

Если выполнение инструкций цикла при определенных условиях может быть пропущено, выберите для себя один из этих двух циклов:

Do While Условие  
' Набор инструкций  
Loop

Do Until Условие  
' Набор инструкций  
Loop

Но если набор инструкций должен быть выполнен хотя бы один раз, выберите для себя один из этих двух циклов:

Do

' Набор инструкций

Loop Until Условие



Инструкции любого цикла должны содержать в себе команды, которые через какое-то количество итераций должны изменять проверяемое условие и заканчивать таким образом выполнение цикла.

Do

' Набор инструкций

Loop While Условие

### Тест на проверку полученных вами знаний

1. Чем циклы Do-Until отличаются от циклов Do-While?
  - а. Циклы **Do-Until** более правильные, поскольку предпочитают истинные условия, в то время как циклам Do-While больше нравятся ложные условия.
  - б. Ничем не отличаются, разве что пишутся по-разному.
  - в. Цикл Do-While прекращает выполнение итераций, как только проверяемое **условие** становится ложным, а цикл Do-Until прекращает свою работу сразу после того, как проверяемое условие становится истинным.
  - г. **А по-моему, они одинаковые.**
2. Почему цикл Do-While может ни разу не **выполнить** набор своих инструкций?
  - а. Потому что вы просто забудете написать эти инструкции.
  - б. Это недоработка Visual Basic .NET – иногда цикл **"забывает"** выполнять свои инструкции.
  - в. Потому, что условие проверяется до выполнения первой итерации, а если ему сразу соответствует значение False, Visual Basic .NET переходит к выполнению следующих кодов программы.
  - г. Свой вариант.

# Циклы, которые умеют считать

*В этой главе...*

- Цикл For-Next
- Прямой и обратный порядок отсчета
- Ключевое слово Step

**Ц**иклы, рассмотренные в главе 24, выполняются некоторое количество раз — до тех пор, пока проверяемое условие не станет истинным или ложным. Но довольно часто возникают ситуации, когда вы заранее можете точно сказать, сколько именно должно быть выполнено итераций. Конечно, используя циклы Do-While и Do-Until, также можно было бы создать код, который выполнял бы определенный набор инструкций заданное количество раз. Но зачем заново изобретать велосипед, если можно просто воспользоваться циклом For-Next, специально предназначенным для подобных целей.

Выглядит цикл For-Next следующим образом:

```
Dim Счетчик As Integer
For Счетчик = Начало To Конец Step X
    НаборИнструкций
Next Счетчик
```

Здесь словом Счетчик обозначена переменная, содержащая числовое значение. Начало — это значение, присваиваемое переменной Счетчик перед началом выполнения первой итерации. Конец — это значение переменной Счетчик, при котором набор инструкций выполняется последний раз. Буквой X обозначено число, которое добавляется к значению переменной Счетчик при выполнении каждой итерации (другими словами, X — это шаг, с которым изменяется значение переменной Счетчик). Если ключевое слово Step и число X опущены, шаг принимается равным единице.

## Как работает цикл For-Next

Если вам нужно, чтобы набор инструкций был выполнен ровно три раза, наберите такой код:

```
Dim X As Integer
For X = 1 To 3
    ' Набор инструкций
Next X
```

Вот как работает этот код.



1. Первой строкой создается переменная типа Integer и ей присваивается имя X.
2. Вторая строка говорит компьютеру: "Присвой переменной X значение 1. Выполняй этот цикл до тех пор, пока переменная X будет принимать значения 1, 2 или 3. Если переменная X примет какое-нибудь другое значение (например, 4), прекрати выполнение данного цикла."

3. В третьей строке вы можете набрать одну или несколько инструкций, которые будут выполняться на каждой итерации. В числе этих инструкций могут быть также и другие циклы For-Next.
4. Четвертая строка говорит: "Присвой переменной X следующее значение (поскольку в данном случае шаг установлен по умолчанию, т.е. равен единице, следующее значение будет на единицу больше предыдущего) и снова переходи ко второй строке".

Итак, следующей строкой определяется цикл, набор инструкций которого будет выполнен три раза:

```
For X = 1 To 3
```

Поскольку шаг, с которым изменяется значение переменной X, по умолчанию равен единице, количество итераций в конечном счете определяется разностью между числом, которое присваивается переменной X перед выполнением цикла, и числом, после присвоения которого переменной X выполняется последняя итерация.

Для большей ясности продемонстрируем это на примере:

```
For 1209 To 1211
' Набор инструкций
Next X
```

Хотя здесь значения переменной X совсем другие, чем в первом примере, набор инструкций также будет выполнен ровно три раза:

- первая итерация, X = 1209;
- вторая итерация, X = 1210;
- третья итерация, X = 1211.



Применять такие нестандартные числа имеет смысл в том случае, если они как-то соотносятся с данными, используемыми вашей программой. Например, они могут быть порядковыми номерами клиентов из вашей базы данных:

```
For КодКлиента = 1250 To 1290
' На каждой итерации обрабатывается информация по
' клиенту, чей код соответствует присвоенному в
' данный момент значению переменной КодКлиента
Next КодКлиента
```

В этом цикле значение переменной КодКлиента используется не только для отсчета количества итераций, но и для ссылки на информацию в базе данных о конкретном клиенте.



Если значения переменной будут использоваться только для отсчета количества итераций, начинайте таковой с числа 1, чтобы легче было понять, сколько именно итераций будет выполнено. Например, если инструкции должны повторяться пять раз, наберите код

```
For X = 1 To 5
' Набор инструкций
Next X
```

И только если эти значения имеют какой-то дополнительный смысл в отношении информации, обрабатываемой в процессе выполнения цикла, используйте коды наподобие следующих:



```

For ИмяПеременной = 3213 To 3218
    ' Набор инструкций
Next ИмяПеременной

```

## Прямой и обратный порядок отсчета

Обычно при выполнении циклов For-Next значение переменной, которая используется в качестве счетчика, увеличивается на единицу после каждой итерации. Но шаг можно изменить и увеличивать значение сразу на 5, 10 или, скажем, на 27 единиц. Чтобы сделать это, нужно в коды добавить ключевое слово Step и указать величину шага, например:

```

Dim Счетчик As Integer
For Счетчик = start To end Step шаг
    ' Набор инструкций
Next Счетчик

```

Ключевое слово Step говорит компьютеру: "Вместо того чтобы изменять значение переменной Счетчик с шагом 1, изменяй ее на число, указанное после меня". Если вы хотите, чтобы шаг равнялся, предположим, числу 16, наберите такой код:

```

Dim X As Integer
For X = 0 To 32 Step 16
    ' Набор инструкций
Next X

```

В результате будет выполнено три итерации:

- первая итерация,  $X = 0$ ;
- вторая итерация,  $X = 16$ ;
- третья итерация,  $X = 32$ .

При желании можно создать цикл, который будет отсчитывать итерации в обратном порядке. Например, чтобы отсчитать три итерации в обратном порядке, наберите такой код:

```

Dim X As Integer
For X = 3 To 1 Step -1
    ' Набор инструкций
Next X

```

Вот как программа будет выполнять его.



1. Первой строкой создается переменная X типа Integer.
2. Вторая строка говорит компьютеру: "Присвой переменной X значение 3 и отсчитывай итерации в обратном порядке до числа 1 с шагом -1".
3. В третьей строке содержится набор инструкций, которые выполняются на каждой итерации.
4. Четвертая строка говорит: "Уменьши значение переменной X на единицу и, если оно еще не равно числу 1, снова выполняй набор инструкций".

Хотя для Visual Basic .NET совершенно безразлично, с каким шагом и в каком порядке вы производите отсчет итераций, старайтесь использовать для этого простейшие варианты, чтобы вам и другим людям, которые будут смотреть коды вашей программы, легче было понять, сколько именно итераций будет выполнено.



Обратный порядок отсчета, или нестандартный шаг, следует использовать лишь в тех случаях, если значения счетчика как-то учитываются при выполнении инструкций на каждой итерации (т.е. если это действительно необходимо). В противном случае вы просто сделаете коды программы более сложными для понимания.

В заключение этого раздела рассмотрим еще один пример. Как вы думаете, что произойдет, если будет набран такой код:

```
Dim J As Integer
For J = 1 To 7 Step 5
    ' Набор инструкций
Next J
```

Этот код будет выполняться следующим образом.



1. Перед выполнением первой итерации переменной J присваивается значение 1.
2. Перед выполнением второй итерации переменной J присваивается значение 6 (шаг равен числу 5;  $1 + 5 = 6$ ).
3. После того как выполнение второй итерации завершится, переменной J будет присвоено значение 11 ( $6 + 5 = 11$ ). Поскольку число 11 не принадлежит к диапазону  $J = 1 \text{ To } 7$ , третьей итерации не будет, и выполнение цикла останавливается.

## *Переменная-счетчик и ее использование*

Если переменная, применяемая для отсчета количества итераций, не была ранее объявлена, цикл For-Next создает ее самостоятельно. Так, приведенным ниже кодом создается переменная XXL, значение которой изменяется с шагом 10:

```
For XXL = 1 To 50 Step 10
    ' Набор инструкций
Next XXL
```

Следующим кодом создается переменная KLM, значение которой изменяется с шагом 1,5:

```
For KLM = 1 To 7 Step 1.5
    ' Набор инструкций
Next KLM
```



В циклах For-Next обычно задается шаг, равный целому числу (например, 1, 2, 5 или 28). В этом случае легче определить количество выполняемых итераций. В первом примере значение счетчика изменяется с шагом 10, поэтому нетрудно понять, что всего будет выполнено пять итераций.

Во втором примере в качестве шага указано число 1,5. Поскольку это число дробное, человеку определить количество итераций уже сложнее (компьютеру все равно, он и не такое может посчитать). Во втором примере, как и в первом, тоже будет выполнено пять итераций, хотя это уже и не так очевидно. Поэтому, чтобы не усложнять без надобности коды программы, избегайте применения в качестве шага дробных чисел.



Создавая циклы For-Next, *никогда* (повторяем, *никогда!*) не изменяйте в процессе выполнения итерации значение счетчика. Это может привести к созданию бесконечного цикла и к зависанию программы. Проиллюстрируем сказанное на таком примере:

```
For X = 1 To 5
  X = 3
Next X
```

Вот к чему приведет выполнение этого кода.



1. В первой строке переменной X присваивается значение 1 и указывается, что цикл будет выполняться до тех пор, пока ей, переменной X, не будет присвоено значение, превышающее 5.
2. Во второй строке переменной X присваивается значение 3.
3. В третьей строке значение переменной X увеличивается на единицу (оно становится равным 4), и цикл переходит на следующую итерацию.

На каждой итерации переменной X будет присваиваться значение 3, по завершении итерации оно будет увеличиваться на единицу, и поскольку число 4 меньше числа 5 (значение, при котором выполнение цикла останавливается), цикл будет повторяться снова и снова и без внешнего вмешательства никогда не завершится. Поэтому при написании инструкций, предназначенных для выполнения на каждой итерации, убедитесь, что среди них нет тех, которые самостоятельно изменяют значение счетчики.

## *Когда For-Next лучше, чем другие циклы*

Цикл For-Next разработан специально для таких случаев, когда количество итераций заранее известно.

В подобных ситуациях можно, конечно, использовать и другие типы циклов. Например, ниже приведены два цикла, выполняющие ровно по шесть итераций:

```
Dim X As Integer
```

```
X = 0
```

```
Do While X < 6
```

```
  X = X + 1
```

```
  ' Набор инструкций
```

```
Loop
```

```
Dim X As Integer
```

```
For X = 1 To 6
```

```
  ' Набор инструкций
```

```
Next X
```

Как видите, в данном случае цикл For-Next намного проще и понятней, чем цикл Do-While. Ранее уже было отмечено, что существует огромное множество способов написания кодов, корректно выполняющих одну и ту же задачу (наверное, их даже больше, чем способов написания неправильных кодов, которые не смогут выполнить задачу как надо). Но почти всегда самый простой способ является одновременно и самым лучшим.

## Тест на проверку полученных вами знаний

1. Сколько итераций выполнит такой цикл:

```
For ID = 15 To 1 Step -1
```

Набор инструкций

```
Next ID
```

- а. Пятнадцать итераций.
- б. Тут без калькулятора не обойдешься.
- в. Чего гадать? Надо набрать этот код и проследить за его выполнением.
- г. Да он вообще не будет выполняться! Он ведь набран на бумаге, а не на компьютере.

2. Какой цикл лучше: For-Next или Do-While?

- а. Если заранее известно, сколько должно быть выполнено итераций, лучше использовать цикл For-Next.
- б. For-Next лучше, потому что он не похож на остальные циклы.
- в. Лучше Do-While. For-Next, наверное, настолько сложный, что под него выделена отдельная глава.
- г. Все циклы плохие, не стоит на них заикливаться.

# Вложенные циклы

*В этой главе...*

V Создание вложенных циклов

➤ Советы по использованию вложенных циклов

➤ Быстрое завершение циклов

**И**ногда для решения особенно сложных или каких-то специфических задач приходится создавать один цикл внутри другого. Цикл, который выполняется на каждой итерации другого цикла, называется *вложенным*.

## Как работают вложенные циклы

Если вы создаете один цикл внутри другого, итерация внешнего цикла не будет выполненной, пока не будут выполнены все итерации вложенного цикла. Проиллюстрируем сказанное на таком примере:

```
Dim J As Integer, Имя As String
Do While Имя = "Майкл"
    For J = 1 To 5
        ' Набор инструкций
    Next J
Loop
```

Вот как этот код будет воспринят компьютером.



1. В первой строке создается переменная `J` типа `Integer` и переменная `Имя` типа `String`.
2. Вторая строка говорит компьютеру: "Сравни значение переменной `Имя` со строкой `Майкл`. Если они совпадают, переходи на третью строку. Если не совпадают — заканчивай выполнение данного цикла".
3. Третья строка дает указание: "Присвой переменной `J` значение 1, с каждой итерацией увеличивай его на единицу и продолжай выполнять набор инструкций цикла `For-Next` до тех пор, пока переменная `J` не примет значение больше 5".
4. Четвертая строка содержит набор инструкций, которые выполняются на каждой итерации цикла `For-Next`.
5. Пятая строка говорит: "Увеличь значение переменной `J` на единицу и снова переходи к третьей строке".
6. Шестая строка сообщает: "На этом выполнение очередной итерации цикла `Do-While` заканчивается. Переходи на вторую строку и снова сравнивай значение переменной `Имя` со строкой `Майкл`". (Обратите внимание: если среди инструкций цикла `Do-While` нет команды, которая могла бы изменить значение переменной `Имя`, цикл превратится в бесконечный и программа зависнет.)



Выполнение одной итерации внешнего цикла не закончится до тех пор, пока вложенный цикл не выполнит все свои итерации. При выполнении следующей итерации внешнего цикла процесс выполнения вложенного цикла начинается заново.

## Советы по использованию вложенных циклов



Visual Basic .NET позволяет вкладывать друг в друга любое количество циклов. Для вас как программиста главное — не запутаться в их кодах и видеть “общую картину”. Помочь в этом могут отступы, которые обозначают границы каждого цикла. Посмотрите на код, набранный без отступов:

```
Do While Имя = "Сэм"  
Do  
For K = 20 To 50 Step 10  
Do  
Do Until ID = 13  
  \ Набор инструкций  
Loop  
Loop While Возраст > 21  
Next K  
Loop Until Код = 1234  
Loop
```

А теперь посмотрите на те же команды, оформленные с применением отступов:

```
Do While Имя = "Сэм"  
Do  
  For K = 20 To 50 Step 10  
  Do  
    Do Until ID = 13  
      \ Набор инструкций  
    Loop  
  Loop While Возраст > 21  
Next K  
Loop Until Код = 1234  
Loop
```

Для компьютера оба кода идентичны, но для человека намного предпочтительней второй вариант. Если начало и окончание циклов помечено посредством отступов, код становится намного проще для чтения и понимания. Редактор кодов Visual Basic .NET может автоматически добавлять отступы, но если вы хотите лично контролировать данный процесс, можете делать это вручную.



Если вы создаете вложенные циклы, убедитесь, что команды вложенных циклов не изменяют значения переменных, используемых в качестве счетчиков для внешних циклов. В противном случае может быть создан один большой бесконечный цикл, и в поисках ошибки вам придется просматривать коды всех внешних и вложенных циклов.

Другой проблемой при создании вложенных циклов может стать выход одного цикла за пределы другого, например:

```

For K = 1 To 4
  For J = 2 To 10 Step 2
Next K
Next C

```

На самом деле это уже превращается в попытку (надеюсь, непреднамеренную) создать не вложенные циклы, а пересекающиеся. Разумеется, такой код нелогичен и работать не будет. К счастью, Visual Basic .NET автоматически отлавливает подобные ошибки, а их исправление не составляет большого труда.

## *Быстрое завершение циклов*

Циклы, которые начинаются с ключевого слова Do, продолжают выполнять итерации до тех пор, пока соответствующее проверяемому условию логическое значение не изменится на противоположное. Цикл For-Next выполняет заранее заданное количество итераций. Но что делать, если в процессе выполнения очередной итерации возникает необходимость немедленно прекратить выполнение всего цикла? Специально для таких случаев существует команда Exit (Выход).

Чтобы немедленно завершить выполнение цикла, начинающегося со слова Do, наберите команду Exit Do:

```

Dim X As Integer
X = 0
Do While X < 6
  X = X + 1
  If X = 4 Then Exit Do
Loop

```

Цикл Do-While должен выполняться до тех пор, пока значение переменной X будет меньше 6. Но на самом деле он закончится еще раньше, поскольку по достижении переменной X значения, равного 4, вступит в действие команда Exit Do и цикл будет прерван.

Для немедленной остановки цикла For-Next используется команда Exit For:

```

For Q = 1 To 100
  If Q = 50 Then Exit For
Next Q

```

Изначально этот цикл должен выполнить 100 итераций, но поскольку при значении Q, равном 50, запускается команда Exit For, произведено будет всего 50 итераций.

Используйте команды немедленного выхода из циклов осторожно и только в тех случаях, когда это действительно необходимо. Сразу же проверьте, не будут ли такие выходы преждевременными и не приведет ли это к возникновению ошибок, чтобы при отладке программы вы не ломали себе голову над вопросом “Ну почему она работает неправильно?”



При использовании команды Exit для выхода из вложенного цикла завершается не весь цикл, а именно этот вложенный цикл, а далее продолжается выполнение итерации внешнего цикла.

### Тест на проверку полученных вами знаний

1. Сколько циклов может быть вложено друг в друга?
  - а. Их вообще нельзя вкладывать. Это же не условные операторы.
  - б. Теоретически количество вложенных циклов ничем не ограничено. Однако не забывайте, что чем больше циклов вы вложите друг в друга, тем сложнее будет потом понять, что каждый из них делает.
  - в. Количество вложенных циклов ограничивается терпением и трудолюбием каждого отдельно взятого программиста.
  - г. Ничем. Если бы за каждый вложенный цикл платили деньги, это был бы хороший бизнес.
2. Что нужно сделать, чтобы вложенные друг в друга циклы стали легче для чтения и понимания?
  - а. Нужно выучить на память эту главу, и тогда картина начнет проясняться.
  - б. О! Нужны годы практики, чтобы научиться наконец читать и понимать коды вложенных **циклов**.
  - в. Нужно снабдить их обильными комментариями - тогда и "чайнику" все будет понятно.
  - г. Нужно добавить отступы, чтобы лучше было видно, где каждый из циклов начинается и где заканчивается.



## Часть VI

# Создание подпрограмм



Это лаборатория Альтаир-14, где мы организовали совместную работу самых нестандартных личностей

### *В этой части...*

Наконец пришло время узнать, как разбить одну большую программу Visual Basic .NET на много маленьких подпрограмм, чтобы она стала проще для написания, чтения, понимания и внесения изменений. Вместо того, чтобы писать одну огромную монолитную программу (похожую на каменное изваяние, вытесанное с огромным трудом из цельной скалы), намного разумней будет написать ряд маленьких подпрограмм и соединить их в одно целое (подобно тому как из маленьких кирпичиков складывается большое здание).

Разделив программу на несколько **подпрограмм**, вы без труда сможете протестировать каждую из **них**, проследить за ее выполнением. Решить маленькую задачу просто. Решить последовательно много маленьких задач тоже несложно, это лишь вопрос времени. Вот почему разработка подпрограмм — это путь, которым идут все опытные **программисты**, создающие серьезные программные продукты.

# Общие процедуры

*В этой главе...*

- Создание общих процедур
- Присвоение имен общим процедурам
- Использование общих процедур

**П**роцедуры — это небольшие подпрограммы, из которых состоит одна большая программа (аналогично тому, как из отдельных кирпичиков складывается здание). В Visual Basic .NET есть два вида процедур: процедуры обработки событий и общие процедуры.

*Процедуры обработки событий*, равно как и кнопки, флажки, переключатели, являются частью пользовательского интерфейса. Они сохраняются в файле формы и выполняются лишь при условии, что случается некоторое событие в отношении объекта, для которого они написаны (например, пользователь щелкает на кнопке или выбирает команду контекстного меню).

*Общие процедуры* не относятся к какому-то конкретному объекту пользовательского интерфейса. Они ничего не делают до тех пор, пока процедуры обработки событий (или другие общие процедуры) не прикажут им начать работу.

Итак, нужны ли процедуры обработки событий? Да. Они обеспечивают способность пользовательского интерфейса адекватно реагировать на действия пользователей. Нужны ли общие процедуры? Нет. Они создаются только из прихоти программистов.

Но вот какая возникает проблема. Если одна или несколько процедур обработки событий должны осуществлять одинаковые или подобные действия (что случается довольно часто), коды для их выполнения придется набирать по нескольку раз для каждой отдельной процедуры, а это уже делает работу программиста довольно рутинной и скучной. Более того, при необходимости модифицировать коды для одного какого-то выполняемого действия вносить изменения придется во все процедуры обработки событий, где такие коды встречаются.

Чтобы избавить себя от необходимости набирать одни и те же коды для разных процедур обработки событий, сохраните их в общей процедуре. То есть общая процедура позволяет сохранять коды, общие для разных объектов и разных событий, в одном месте и в одном экземпляре. Теперь, если нужно будет их модифицировать, достаточно внести изменения только один раз.

## *Что такое файлы модулей*

Любая программа, написанная на языке Visual Basic .NET, состоит не менее чем из одного файла формы, а каждый файл формы содержит в себе процедуры обработки событий, которые делают программу рабочей. Если же вы создадите общую процедуру, можете также сохранить ее в файле формы.

Правда, сохраняя общие [[процедуры и процедуры обработки событий в одном файле формы, вы делаете коды программы запутанными и сложными для понимания. Представьте, что было бы, если бы в один ящик своего стола вы складывали все свои тетрадки, дискеты, компакт-диски, носовые платки, журналы и еще много всякой всячины. И в первом и во втором случае процесс поиска нужного компонента или вещи превращается в нелегкую задачу. В качестве альтернативного варианта Visual Basic .NET предоставляет возможность сохранять общие процедуры отдельно, в файлах модулей.



Многие программисты предпочитают сохранять общие процедуры, выполняющие один класс задач, в отдельных файлах. Например, процедуры, отвечающие за печать документов, сохраняются в одном файле, а процедуры, отвечающие за их редактирование, — в другом. Таким образом, если вдруг возникнут ошибки при выводе документа на печать, нетрудно будет понять, что искать их нужно в том файле, где содержатся общие процедуры, отвечающие за данный процесс.

На рис. 27.1 показана структура файла, в котором сохранены сразу все процедуры, выполняемые программой. При возникновении какой-либо ошибки найти таковую во всем этом обилии кодов будет крайне тяжело. Но если вы разобьете подобные процедуры на разные группы и сохраните их в отдельных файлах, как это показано на рис. 27.2, коды уже не будут казаться такими сложными и запутанными, а процесс поиска ошибок потребует меньшего времени и усилий.

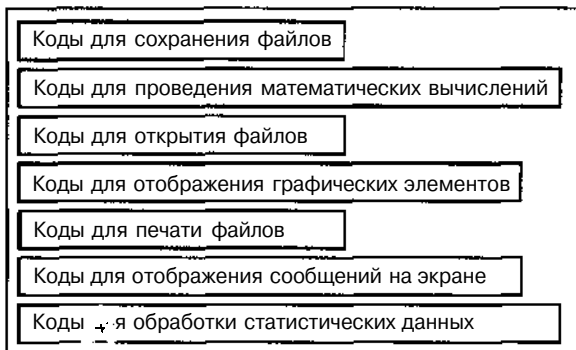


Рис. 27.1. Если все коды вашей программы будут сохранены в одном файле, модифицировать их и исправлять ошибки будет очень тяжело

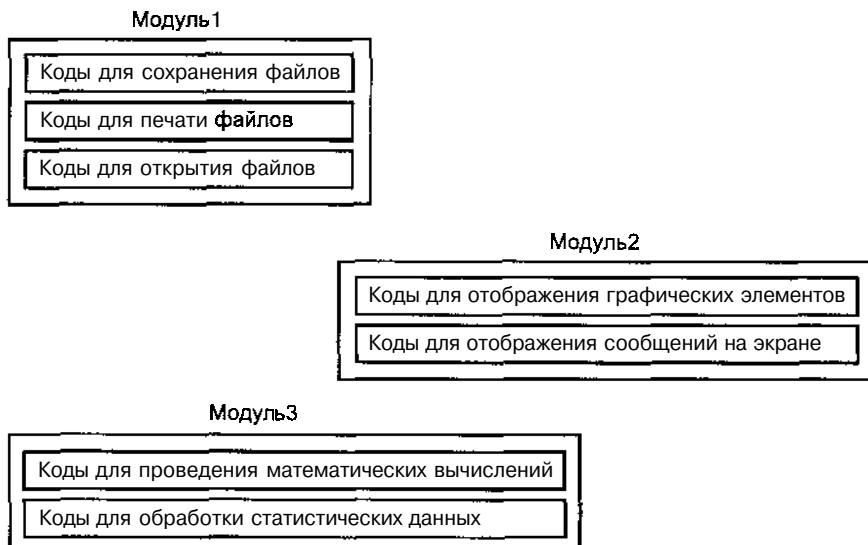


Рис. 27.2. Коды программы, разбитые по группам и сохраненные в отдельных файлах, более легки для понимания и внесения изменений



Программа Visual Basic .NET может состоять из любого количества файлов модулей или не иметь таковых вообще.

Для создания нового файла модуля выполните следующие действия.

1. Выберите команду **Project**⇒**AddModule** (**Проект**⇒**Добавить модуль**).

Откроется диалоговое окно **Add New Item** (Создание нового элемента), показанное на рис. 27.3.

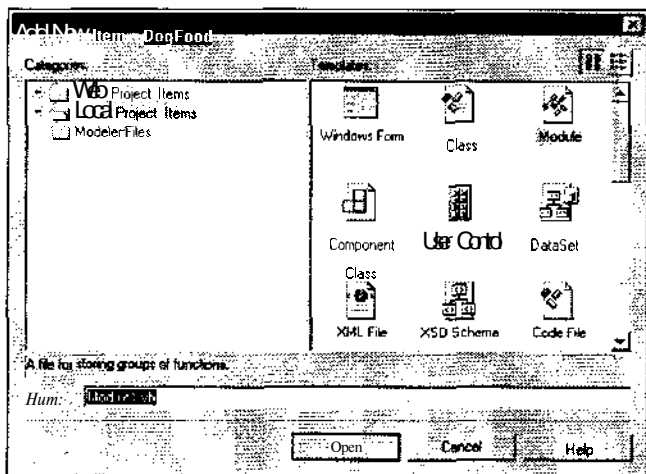


Рис. 27.3. Окно создания для программы нового файла модуля

2. Щелкните на значке **Module**, а затем — в текстовом поле **Name**.
3. Наберите имя нового файла модулей.  
Не забудьте указать расширение **.VB**.
4. Щелкните на кнопке **Open**.

Visual Basic .NET отобразит в окне **Solution Explorer** название только что созданного файла (если в данный момент это окно открыто), а в окне кодов откроет пустой файл модулей, который будет выглядеть так, как показано ниже (вместо слова **Module1** должно отображаться название файла, которое вы указали на третьем шаге):

```
Module Module1
```

```
End Module
```

## Создание общих процедур

Код общей процедуры представлен ниже:

```
Public Sub ИмяПроцедуры()  
    ' Инструкции  
End Sub
```

Обратите внимание, что этот код имеет пять составляющих:

- ✓ ключевое слово `Public` (или `Private`);
- ✓ ключевое слово `Sub`;
- ✓ название процедуры (ее имя);
- ✓ пара круглых скобок `()`;
- ✓ ключевые слова `End Sub`.

Слово `Public` указывает на тот факт, что процедура является глобальной. Если вы сохранили общую процедуру в файле модуля, вызывать ее может любая другая общая процедура или процедура обработки событий, сохраненная в файле модуля или в файле формы.

Если вы хотите ограничить область видимости процедуры теми процедурами, которые сохранены в том же файле, наберите вместо слова `Public` ключевое слово `Private`:

```
Private Sub ИмяПроцедуры()
```

Инструкции

```
End Sub
```



Если вы сохраняете общую процедуру в файле формы, вызывать ее могут только те процедуры, которые сохранены в этом же файле. Следовательно, процедуры, сохраненные в файле формы, не могут быть глобальными.

Ключевое слово `Sub` идентифицирует данный фрагмент кода как процедуру (если вместо слова `Sub` набрать слово `Function`, будет создана общая функция, которая описывается в главе 29). Имя процедуры является тем словом, которое используют другие общие процедуры и процедуры обработки событий для ее вызова. *Вызов* определенной процедуры означает, что к ней обращаются со словами: "Эй! Уважаемая, теперь твоя очередь что-то сделать."

Пара круглых скобок называется *списком аргументов* (он будет подробно рассмотрен в главе 28). Простейшие общие процедуры имеют пустые списки аргументов, о чем свидетельствует пустая пара круглых скобок.

Коды общей процедуры заканчиваются словами `End Sub`.

Создать и сохранить общую процедуру можно:

- ✓ во-первых, в файле формы (т.е. в том же файле, где хранится информация об объектах пользовательского интерфейса, таких как кнопки, раскрывающиеся меню и т.п.);
- ✓ во-вторых, в файле модулей.



Сохраняя коды общих процедур в отдельных файлах, вы можете создать целую библиотеку полезных процедур, которые затем будут использованы в других создаваемых вами программах Visual Basic .NET. Вообще, оптимальным вариантом является сохранение всех общих процедур в отдельных файлах модулей, а всех процедур обработки событий — в файлах форм. В таком случае все коды, относящиеся к работе пользовательского интерфейса, будут отделены от кодов, выполняющих вычисления и обработку данных.

## Сохранение общих процедур в файле формы



Сохраняя общие процедуры в файле формы, вы рискуете сделать коды своей программы сложными для понимания, а также для внесения в них изменений. Эта проблема становится особенно актуальной в больших программах.

При необходимости создать и сохранить общую процедуру в файле формы, вы должны выполнить такие действия.

1. Чтобы открыть окно кодов, в окне Solution Explorer щелкните на названии файла формы и нажмите клавишу <F7>, выберите команду View⇒Code или щелкните на значке View Code.
2. Прокрутите окно кодов таким образом, чтобы увидеть блок с надписью Windows Form Designer generated code (рис. 27.4).
3. Щелкните прямо под надписью Windows Form Designer generated code и наберите код общей процедуры.

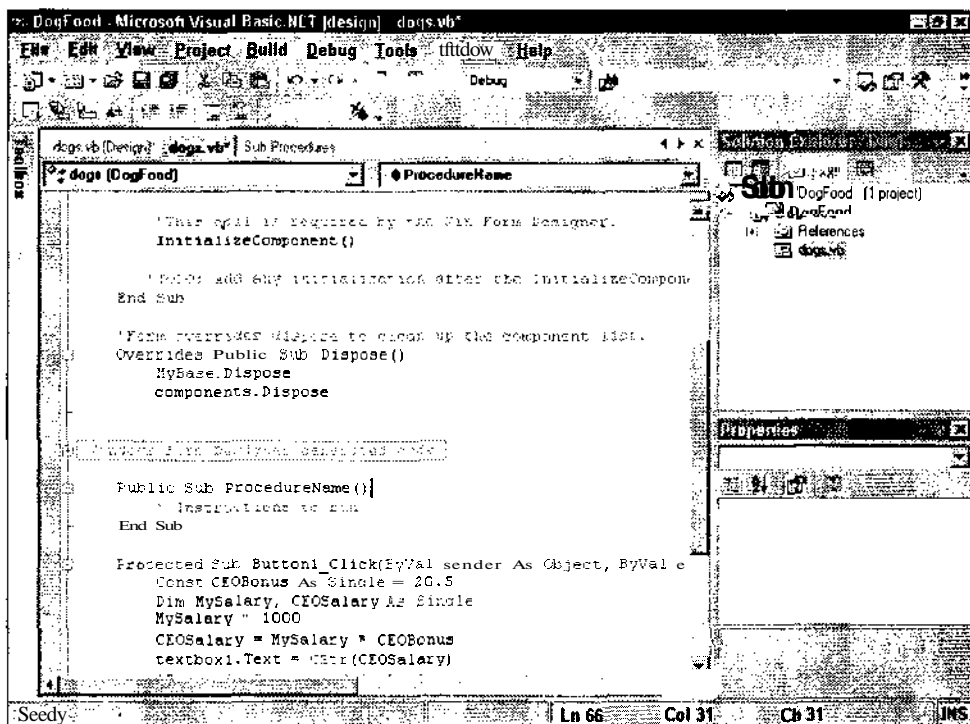


Рис. 27.4. Код общей процедуры можно набрать прямо под надписью *WindowsForm Designer generated code*

## Создание и сохранение общих процедур в файлах модулей



Если вы еще не создали файл модуля, вернитесь, в начало этой главы, к разделу “Что такое файлы модулей”. Здесь вы найдете описание последовательности действий, отвечающих за создание файла модуля.

Далее вам необходимо создать и сохранить общую процедуру в файле модуля.

1. Выберите команду View⇒Solution Explorer или нажмите комбинацию клавиш <Ctrl+Alt+L>.  
Откроется окно Solution Explorer.
2. Дважды щелкните на имени файла модуля, в котором хотите создать общую процедуру.

Visual Basic .NET откроет в окне кодов пустой файл модуля, который будет выглядеть следующим образом (вместо слова `Module1` должно отображаться действительное название файла):

```
Module Module1
```

```
End Module
```

### 3. Наберите коды процедуры в любом месте между строками `Module1` и `End Module`.



При создании глобальных общих процедур, которые могут быть вызваны любым кодом вашей программы, используйте ключевое слово `Public`. Если нужно создать общую процедуру, вызывать которую смогут только процедуры, сохраненные в этом же файле, наберите вместо `Public` ключевое слово `Private`.

## *Присвоение имен общим процедурам*

В отличие от процедур обработки событий, имена которых состоят из названия соответствующего объекта и названия события, общие процедуры могут быть названы любыми именами, которые, впрочем, должны:

- ✓ состоять не более чем из сорока символов;
- ✓ начинаться с буквы и включать только буквы, цифры и символы подчеркивания (`_`);
- ✓ не совпадать с зарезервированными в Visual Basic .NET ключевыми словами (такими как `End`, `Sub` и `Private`).

Процедурам рекомендуется давать такие имена, которые будут как-то описывать выполняемые ими действия, например:

ВычислениеСреднего

Поиск4кодов

СохранениеМассива

Коды названных такими именами процедур будут выглядеть так:

```
Public Sub ВычислениеСреднего ( )
```

```
End Sub
```

```
Public Sub Поиск4кодов ( )
```

```
End Sub
```

```
Public Sub СохранениеМассива ( )
```

```
End Sub
```

## *Вызов общих процедур*

Каждая общая процедура содержит в себе одну, несколько или множество инструкций. Если одной процедуре требуется, чтобы были выполнены коды, сохраненные в другой процедуре, она должна попытаться эту другую процедуру, воспользовавшись ее именем.



Вызвать для выполнения другую процедуру можно двумя способами. Во-первых, можно просто указать имя этой процедуры:

ИмяПроцедуры{>

А во-вторых, можно набрать ключевое слово Call и уже после него указать имя нужной процедуры:

Call ИмяПроцедуры()



Для Visual Basic .NET совершенно безразлично, каким образом вы будете вызывать другие процедуры, но чтобы коды были более понятными, выберите для себя какой-то один способ и придерживайтесь его постоянно.



Набрать лишь название процедуры проще, но если вы используете ключевое слово Call, то впоследствии будет легче найти в программе те коды, которые вызывают процедуры, сохраненные в других файлах.

Рассмотрим такой пример. Допустим, вы создали такую общую процедуру:

```
Public Sub Warning()  
    MsgBox ("Your computer will blow up in 3 seconds!"_  
        , , "Warning!")  
End Sub
```

Единственное, что способна делать данная процедура, — это отображать на экране диалоговое окно, показанное на рис. 27.5.

Чтобы запустить эту процедуру из другого места программы (например, из какой-нибудь процедуры обработки событий), достаточно набрать ее имя:

```
Public Sub cmdAlert_Click()  
    Warning()  
End Sub
```

Тот же результат можно получить путем ввода перед названием процедуры ключевого слова Call:

```
Public Sub cmdAlert_Click()  
    Call Warning()  
EndSub
```

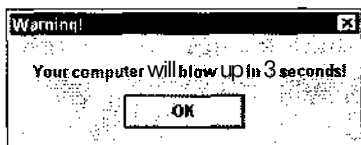


Рис. 27.5.Диалоговоеокно,открываемое  
привывозепроцедурыWarning()

Оба последних кода эквивалентны такому коду:

```
Public Sub cmdAlert_Click()  
    MsgBox ("Your computer will blow up in 3 seconds!"_  
        , , "Warning!")  
End Sub
```



В рассмотренном выше примере вызываемая процедура довольно проста, но в действительности почти все процедуры включают по две, три и более инструкций. Понятно, что имеет смысл набрать все эти инструкции по одному разу (создав отдельную общую процедуру) и вызывать их лишь при необходимости, указывая только имя процедуры.

### Тест на проверку полученных вами знаний

1. Для чего создаются общие процедуры?

- а. Для того чтобы программа **содержала** побольше кодов и ее можно было **подороже продать**.
- б. Многие программисты занимаются **этим**, желая создать видимость выполнения **какой-то работы**.
- в. Для того, чтобы **не набирать заново** одни и те же инструкции, а также для того, чтобы коды программы были **проще для понимания** и для **внесения в них изменений**.

2. Как можно вызвать общую процедуру?

- а. Открыть **коды** программы, найти нужную процедуру и нажать **клавишу <Enter>**.
- б. Надо присвоить процедуре номер, **подключить** к компьютеру модем и с другого **телефона** позвонить на этот номер (**вам** ответит **приятный женский голос**).
- в. Это можно сделать **двумя способами**: либо набрав ее **имя**, либо набрав **ключевое слово Call**, а после него — **опытати** имя процедуры.

# Использование аргументов

*В этой главе...*

- Передача аргументов процедуре
- Возможные проблемы при передаче аргументов
- Досрочный выход из процедуры

**К**огда процедура вызывает общую процедуру, она ссылается на ее имя. Если вызов происходит только по имени, управление просто передается вызываемой процедуре, без какой-либо сопроводительной информации. При этом вызывающая процедура как бы говорит вызываемой: “Эй! Теперь твоя очередь работать. Начинай!”

Но довольно часто вызываемой процедуре, для того чтобы приступить к работе, нужно иметь дополнительные сведения и инструкции. В частности, ей могут понадобиться специфические данные, на основании которых она сможет выполнять какие-то действия или использовать их для проведения расчетов.

Любая процедура, которая вызывает общую процедуру, может предоставить и необходимые для ее работы данные. Эти данные программисты называют *аргументами*. В таком случае вызывающая процедура говорит вызываемой: “Эй! Теперь твоя очередь работать. Возьми нужные тебе данные и начинай!”

## *Для него нужны аргументы*

*Аргументы* — это данные (числа, строки либо переменные, содержащие в себе числа или строки), которые нужны общей процедуре для выполнения ее работы. Воспользовавшись аргументами, можно создать простую, похожую на шаблон, процедуру, которая заменит собой несколько подобных процедур, использующих в своей работе заранее определенные значения.

Рассмотрим в качестве примера две общие процедуры. Одна из них отображает диалоговое окно с предупреждением:

```
Public Sub DisplayCaution ()  
    TxtReadMe.Text = "Внимание! Это последняя попытка"  
End Sub
```

Вторая отображает диалоговое окно с сообщением:

```
Public Sub DisplayMessage()  
    TxtReadMe.Text = "Поздравляем! Отличная попытка"  
End Sub
```

Любую из этих процедур можно вызвать, воспользовавшись одним из двух способов:

- ✓ `DisplayCaution()` или `Call DisplayCaution()`
- ✓ `DisplayMessage()` или `Call DisplayMessage()`

Однако создавать для каждого сообщения отдельную процедуру — это не лучший вариант. Вместо того чтобы писать несколько почти одинаковых процедур, можно написать лишь одну, заменив специфическую информацию аргументом. Например:

```
Public Sub Display( ByVal Сообщение As String)
```

```
    TxtReadMe.Text = Сообщение
```

```
2nd Sub
```

Такая процедура говорит компьютеру: "Создай текстовую переменную Сообщение и присваивай ей значения, которые будут передавать вызывающие меня процедуры. Каким бы ни было переданное значение, присвой его свойству **Text** объекта `txtReadMe`".

Чтобы вызвать процедуру, для работы которой нужны дополнительные данные, приведите таковые в круглых скобках. Например, чтобы вызвать процедуру `Display`, наберите в круглых скобках текст, который должен быть отображен на экране:

```
Display("Внимание! Это последняя попытка")
```

Или:

```
Display("Поздравляем! Отличная попытка")
```

Или:

```
Call Display("Внимание! Это последняя попытка")
```

Или:

```
Call Display("Поздравляем! Отличная попытка")
```

## *Передача процедуре значений аргументов*

Для вызова процедуры и передачи ей данных (называемых аргументами) предусмотрены два метода:

- ✓ ИмяПроцедуры (Аргумент)
- ✓ Call ИмяПроцедуры (Аргумент)

Допустим, вами была создана такая общая процедура:

```
Public Sub Display(ByVal Сообщение As String)
```

```
    TxtReadMe.Text = Сообщение
```

```
End Sub
```

Чтобы вызвать эту процедуру и отобразить на экране сообщение "Внимание! Это последняя попытка", можно набрать один из двух кодов:

- ✓ Display("Внимание! Это последняя попытка")
- ✓ Call Display("Внимание! Это последняя попытка")

Не имеет значения, какой из этих вариантов когда будет использоваться, но лучше остановиться на каком-то одном из них и применять его постоянно.

Теперь проследим за пошаговым выполнением вызываемой процедуры `Display`.



1. Код, посредством которого процедура вызывается, говорит компьютеру: "Найди процедуру, именуемую `Display`, и передай ей текстовое значение **Внимание ! Это последняя попытка**". В данном случае строка **Внимание ! Это последняя попытка** является аргументом.

2. Visual Basic .NET находит общую процедуру Display и дает ей указание присвоить переменной Сообщение передаваемое значение.
3. Затем процедура Display говорит компьютеру: "Присвой значение переменной Сообщение свойству Text объекта txtReadMe". Поскольку значением переменной Сообщение является строка Внимание! Это последняя попытка, свойству Text объекта txtReadMe будет присвоено это же значение.

Если для вызова процедуры вы наберете код

```
Display("Поздравляем! Отличная попытка")
```

свойству Text объекта txtReadMe будет присвоено значение Поздравляем! Отличная попытка

Таким образом, создавая процедуры, использующие для своей работы значения аргументов, вы можете избавиться от необходимости писать по многу подобных процедур, обрабатывающих заранее определенные данные.

## Получение значений аргументов

Чтобы создаваемая общая процедура смогла получать внешние данные (значения аргументов), нужно задать для нее список аргументов. По сути, этим списком определяется, сколько значений и каких типов может принять процедура.

Если создаваемой процедуре для работы дополнительные данные не нужны, список аргументов не создается, о чем свидетельствует пара пустых скобок:

```
Public Sub Самодостаточная()
```

```
End Sub
```

Чтобы вызвать такую процедуру, достаточно набрать один из двух кодов:

- ✓ Самодостаточная()
- ✓ Call Самодостаточная!

А чтобы создать процедуру с одним аргументом, необходимо набрать такой код:

```
Public Sub ОдинАргумент(ByVal Число As Integer)
```

```
End Sub
```

Данная процедура объявляет о создании аргумента, представляемого переменной Число типа Integer. Чтобы вызвать такую процедуру, нужно указать ее имя и значение аргумента:

- ✓ ОдинАргумент(4)
- ✓ Call ОдинАргумент(4)



Если при вызове подобной процедуры вы укажете значение не того типа (в нашем случае отличное от типа Integer), Visual Basic .NET воспримет это как ошибку и откажется продолжать выполнение программы.

## Аргументы-значения и аргументы-ссылки

Чтобы отдельные общие процедуры не могли повредить или удалить данные, используемые другими частями программы, Visual Basic .NET по умолчанию передает процедурам для

обработки только значения переменных, но не сами переменные. На тот факт, что процедуре будут передаваться только значения, указывает ключевое слово `ByVal`.

Если процедуре в качестве аргумента передается значение, создается отдельная копия исходных данных, и процедура работает только с этой копией. Таким образом, как бы процедура не изменяла полученные значения, на исходных данных это никак не отразится и их целостность не будет нарушена.



Если при создании списка аргументов вы забудете набрать слово `ByVal`, Visual Basic .NET добавит `ero` автоматически.



На самом деле у вас есть возможность выбора: передавать процедуре для обработки только значения переменных или сами переменные. Если процедуре передается значение переменной, она имеет дело лишь с копией исходных данных. Но если передаются сами переменные, процедура получает возможность изменять исходные данные. Во втором случае появляется вероятность возникновения ошибки, поскольку процедура может изменить данные, используемые другими частями программы. По умолчанию Visual Basic .NET передает процедуре только копии исходных данных (т.е. только значения переменных). Но если вам это действительно необходимо, можете позволить процедурам вносить изменения в исходные данные. Для этого вы должны вместо слова `ByVal` набрать ключевое слово `ByRef`:

```
Public Sub НазваниеПроцедуры(ByRef _  
    ИмяПеременной As Integer)
```

```
End Sub
```

## Объявление сразу нескольких аргументов

Процедура может использовать а своей работе сразу несколько аргументов. Но прежде в списке аргументов необходимо указать их имена и типы получаемых данных. Объявляя аргументы, отделяйте их друг от друга запятыми. Ниже приведен код процедуры, которая объявляет, что будет использовать значения сразу трех аргументов, а именно I, S и D:

```
Public Sub Yellow(ByVal I As Integer, _  
    ByVal S As String, ByVal D As Double)
```

```
End Sub
```

Вызвать эту процедуру и передать ей значения 30, Yes и 12.9 можно с помощью одного из двух следующих кодов:

```
✓ Yellow(30, "Yes", 12.9)  
✓ Call Yellow(30, "Yes", 12.9)
```

Теоретически никаких ограничений на количество используемых аргументов не существует, но чем больше вы их создаете, тем сложнее становится процедура и увеличивается вероятность того, что впоследствии вы не сможете вспомнить, зачем все эти аргументы нужны и что вообще делает процедура.

# *Возможные ошибки при передаче данных процедуре*

Наиболее часто при вызове процедуры и передаче ей данных возникают ошибки двух типов. Во-первых, количество значений, передаваемых процедуре, может не совпадать с количеством объявленных для нее аргументов. Во-вторых, передаваемые данные могут не соответствовать типу объявленных аргументов.

## **Неверное количество переменных**

Создавая процедуру и объявляя список аргументов, вы определяете точное количество значений, которые должны быть переданы процедуре при ее вызове. Если набрать код, вызывающий процедуру, и попытаться передать ей другое количество значений (неважно, большее или меньшее), это будет воспринято как ошибка и программа работать не станет. Например:

```
Public Sub Green(ByVal Число As Integer)
```

```
End Sub
```

Чтобы эта процедура работала правильно, ей нужно передать одно значение типа Integer. Ни один из приведенных ниже способов вызова процедуры Green не является корректным, поскольку в каждом случае передается ошибочное количество аргументов:

- ✓ Green()
- ✓ Green(9, "Stop!")
- ✓ Green("OK", "No", 334)

## **Неправильный тип данных**

Вызывая процедуру, не забывайте о том, что передаваемые значения должны соответствовать тому типу данных, который был объявлен для аргументов в момент ее создания. Рассмотрим в качестве примера такую процедуру:

```
Public Sub Green(ByVal Строка As String)
```

```
End Sub
```

Чтобы вызвать эту процедуру, нужно передать ей в качестве аргумента одно текстовое значение. Приведенные ниже способы вызова процедуры являются ошибочными, поскольку передаваемые значения — это числа, а не строки:

- ✓ Green(42)
- ✓ Green(1)
- ✓ Green(12.053)

## *Немедленный выход из процедуры*

Обычно процедуры выполняются до тех пор, пока не будут выполнены все их инструкции. Однако процедура может быть завершена и раньше.

Чтобы немедленно прервать выполнение какой-либо процедуры, наберите такой код:

```
Exit Sub
```

Проиллюстрируем это на примере следующей процедуры:

```
Public Sub ПрерванныйЦикл()
```

```
    X = 0
```

```
    Do
```

```
        X = X + 1
```

```
        If (X = 13) Then
```

```
            Exit Sub
```

```
        End If
```

```
    Loop Until X = 25
```

```
End Sub
```

Цикл `Do-Until` должен был выполнить 25 итераций, по окончании которых переменной `X` было бы присвоено значение 25. Однако в самом цикле содержится оператор `If-Then`, который при достижении переменной `X` значения 13 прерывает выполнение процедуры посредством команды `Exit Sub`.



## Создание функций

В этой главе...

- Вызов функций
- Использование аргументов
- Быстрое завершение функций

**Ф**ункции представляют собой миниатюрные подпрограммы, которые получают данные, производят вычисления и возвращают результат. В Visual Basic .NET имеется целый набор встроенных функций, обозначаемых разными загадочными именами. К их числу относятся, в частности, функция Sqr (принимает число и возвращает его квадратный корень) и функция LCase (принимает произвольную строку и возвращает этот же текст, набранный строчными буквами). Четыре из числа наиболее часто используемых функций описаны в табл. 29.1.

**Таблица 29.1. Наиболее часто используемые функции Visual Basic .NET**

Встроенная функция	Производимые действия
Abs (число)	Возвращает модуль числа
Date	Возвращает текущую дату
LCase (строка)	Преобразует символы в символы нижнего регистра
Sqr (число)	Возвращает квадратный корень числа

Но Visual Basic .NET не ограничивается использованием только встроенных функций — вам предоставляется возможность создавать собственные функции. Код обычной функции выглядит так:

```
Public Function ИмяФункции(СписокАргументов) As _
    ТипДанных
    ИмяФункции = Значение
End Function
```

Если функция сохраняется в файле модуля, то слово Public указывает на тот факт, что она может быть вызвана любой общей процедурой или процедурой обработки событий, сохраненной в каком-либо файле этой же программы. (Если сохранить функцию в файле модуля, но вместо слова Public ввести слово Private, вызвать ее смогут лишь те процедуры, которые сохранены в этом же файле.)



Если функция будет сохранена в файле формы, вызывать ее смогут только те об-щие процедуры или процедуры обработки событий, которые сохранены в этом же файле формы.

Ключевое слово Function указывает на то, что подпрограмма является функцией. Вме-сто слова ИмяФункции наберите любое допустимое в Visual Basic .NET имя, но желательно

такое, которое будет каким-то образом описывать выполняемые функцией действия. Список Аргументов может быть пустым, а может объявлять некоторое количество аргументов. Словом ТипДанных определяется, значение какого типа будет возвращаться функцией.

Приведенный выше фрагмент кода может быть набран несколько иначе:

```
Public Function ИмяФункции(СписокАргументов) As_  
    ТипДанных  
    Return Значение  
End Function
```

Этот код отличается от предыдущего тем, что вместо непосредственного присвоения имени функции возвращаемого значения использовано ключевое слово Return. Создавая свою функцию, вы можете выбрать любой из этих способов определения значения, которое должно быть возвращено.

## Создание функций

Создать и сохранить функцию можно:

- ✓ в файле модуля;
- ✓ в файле формы.

Если вы сохраняете функцию в файле формы, она может быть вызвана и использована только процедурой или функцией, сохраненной в этом же файле. Но если вы создаете и сохраняете функцию в файле модуля, она может быть вызвана любой процедурой или функцией, сохраненной в любом файле вашей программы.



Хорошим решением является сохранение всех функций в файлах модулей. В этом случае коды, отвечающие за работу пользовательского интерфейса, не будут смешиваться с кодами, выполняющими обработку данных и производящими вычисления.

## Создание функции в файле формы

Вот каким образом вы можете создать функцию в файле формы.

1. Чтобы открыть окно редактора кодов, в окне Solution Explorer щелкните на названии файла формы, затем нажмите клавишу <F7>, выберите команду View⇒Code или щелкните на значке View Code.
2. Прокрутите окно редактора кодов таким образом, чтобы был виден блок с надписью Windows Form Designer generated code.
3. Щелкните прямо под надписью Windows Form Designer generated code и наберите код функции.

## Создание и сохранение функции в файле модуля

(Если файл модуля еще не создан и вы не знаете, как это сделать, вернитесь к главе 27, к разделу "Что такое файлы модулей".)

Чтобы создать и сохранить функцию в файле модуля, выполните действия перечисленные ниже.

1. Выберите команду View⇒Solution Explorer или нажмите комбинацию клавиш <Ctrl+Alt+L>.

Откроется окно **Solution Explorer**.

2. **Дважды щелкните на названии файла модуля, в котором хотите сохранить создаваемую функцию.**

Visual Basic .NET в окне кодов откроет пустой файл модулей, который будет выглядеть так, как показано ниже (правда, вместо слова `Module1` должно отображаться действительное имя файла):

```
Module Module1
```

```
End Module
```

3. **Наберите коды функции в любом месте между строками `Module` и `End Module`.**

## *Возвращаемый результат*

Где-нибудь в кодах функции обязательно нужно либо набрать имя функции и сопоставить ему значение или выражение, либо ввести ключевое слово `Return`, после него набрать значение или выражение. Это значение или значение, вычисляемое выражением, и будет тем результатом, который должна вернуть функция. Например:

```
Public Function ЯрдыМетры(ByVal Ярды As Single) As_  
    Single  
    Const Коэффициент = 0.9  
    ЯрлыМетры = Ярды * Коэффициент  
End Function
```

Или, что то же самое:

```
Public Function ЯрдыМетры(ByVal Ярды As Single) As_  
    Single  
    Const Коэффициент = 0.9  
    Return Ярды * Коэффициент  
End Function
```



Если вы не присвоите имени функции никакого значения или не наберете ключевое слово `Return`, функция не вернет никакого результата, т.е. станет совершенно бесполезной.

Функция в качестве результата может возвращать данные любого типа, в том числе `Integer`, `String` и `Boolean`. Более подробную информацию об использовании различных типов данных вы можете найти в главе 15.



Существует три основных различия между функциями и процедурами.

- ✓ Каждая функция производит вычисления и возвращает одно (и только одно) значение. Процедуры же создаются для решения различных задач и выполнения определенных действий.
- ✓ Где-нибудь в кодах функции должно быть указано значение, которое она возвращает. Для процедуры ничего подобного делать не нужно.
- ✓ Для каждой функции нужно определить тип данных, которые она возвращает в качестве результата. Для процедуры, опять-таки, этого делать не нужно.

# вызов функций

Вызов функции отличается от вызова процедуры. Поскольку каждая функция возвращает в качестве результата какое-то значение, то чтобы вызвать таковую, нужно присвоить какой-либо переменной ее имя. Например:

```
Public Function ЯрдыМетры(ByVal Ярды As Single) As_  
    Single  
    Const Коэффициент = 0.9  
    ЯрдыМетры = Ярды * Коэффициент  
End Function
```

```
Private Sub Button1_Click(ByVal sender As_  
    System.Object, ByVal e As System.EventArgs)_  
    Handles Button1.Click  
  
    Dim Метры As Single  
    Метры = ЯрдыМетры(CSng(txtЯрды.Text))  
    txtМетры.Text = CStr(Метры)  
End Sub
```



Процедура обработки событий дает компьютеру указание: "Когда пользователь щелкнет не кнопке Button1, ты должен выполнить ряд действий."

1. Создай переменную Метры, которая будет принимать значения типа Single.
2. Возьми значение, сохраненное в свойстве Text объекта txtЯрды, преобразуй его к типу Single и используй в качестве аргумента функции ЯрдыМетры.
3. Функция ЯрдыМетры возьмет это значение, умножит его на коэффициент 0,9 и вернет полученное значение как свой результат. Возьми этот результат и присвой его переменной Метры.
4. Значение, сохраненное в переменной Метры, преобразуй к текстовому виду и присвой его свойству Text объекта txtМетры."

Сравните коды, вызывающие процедуру и функцию. Процедуру можно вызвать так:

```
ИмяПроцедуры(СписокАргументов)
```

либо так:

```
Call ИмяПроцедуры(СписокАргументов)
```

Чтобы вызвать функцию, нужно возвращаемое ею значение присвоить какой-нибудь переменной:

```
Переменная = ИмяФункции(СписокАргументов)
```



Если функция возвращает логическое значение, она может быть использована как условие оператора If-Then или других условных операторов:

```
If ИмяФункции(СписокАргументов) Then  
    ' коды BASIC  
End If
```

Поскольку функции возвращают какое-то одно значение, их можно использовать при вычислении математических выражений:

Переменная1 = ИмяФункции(Аргументы) + Переменная2

Например, процедура, вызывающая функцию ЯрдыМетры, может выглядеть так:

```
Private Sub cmdConvert_Click()  
    Dim Метры As Single, NewValue As Single  
    NewValue = (ЯрдыМетры(Метры) + 32) * 4  
End Sub
```

## *Тип данных принимаемых и возвращаемых значений*

Поскольку функция возвращает только одно значение, обязательно нужно указать какой-то один тип данных, которому это значение будет соответствовать.

Посмотрим еще раз на уже знакомую нам функцию:

```
Public Function ЯрдыМетры(ByVal Ярды As Single) As_  
    Single  
    Const Коэффициент = 0.9  
    ЯрдыМетры = Ярды * Коэффициент  
End Function
```

Первая строка кодов этой функции заканчивается указанием на тип данных Single, который и будет соответствовать возвращаемому результату. Функции могут возвращать значения различных типов, среди которых и такие:

- ✓ Integer;
- ✓ Long;
- ✓ Boolean;
- ✓ Single;
- ✓ Double;
- ✓ Decimal;
- ✓ String.

Неважно, какой именно тип данных возвращается функцией, но тип переменной, которой присваивается возвращаемое значение, обязательно должен с ним совпадать. Например:

```
Public Function ЯрдыМетры(ByVal Ярды As Single) As_  
    Const Коэффициент = 0.9  
    ЯрдыМетры = Ярды * Коэффициент  
End Function  
  
Private Sub Button1_Click(ByVal sender As_  
    System.Object, ByVal e As System.EventArgs)____  
    Handles Button1.Click
```

```

Dim Метры As Single
Метры = ЯрдыМетры(CSng(txtЯрды.Text))
txtМетры.Text = CStr(Метры)
End Sub

```

Функция ЯрдыМетры возвращает значение типа Single; переменная Метры, которой присваивается возвращаемое значение, также имеет тип Single, поэтому такой код является правильным.

Но допустим, что переменная Метры была объявлена как текстовая переменная:

```
Dim Метры As String
```

В таком случае строка

```
Метры = ЯрдыМетры(CSng(txtЯрды.Text))
```

будет содержать ошибку, поскольку тип возвращаемого функцией результата и тип переменной, которой этот результат присваивается, не совпадают.

## Тип принимаемых данных

Аргументы — это исходные данные (числа, строки либо переменные, содержащие числа или строки), которые нужны функции для нормальной работы.

При объявлении списка аргументов нужно указать не только их количество и названия, но и значения каких типов данных каждый из них будет представлять.

Каждый аргумент может представлять данные самых разных типов, в том числе и тех, которые возвращают функции (см. выше).

Ниже показана первая строка функции, которая использует в своей работе значения двух аргументов типа Integer:

```

Public Function Convert(ByVal F1 As Integer, ByVal_
                        F2 As Integer) As Single

```

Итак, при вызове такой функции ей необходимо передать два значения типа Integer, а возвращаемое значение присвоить переменной типа Single. Далее представлены процедура Button1\_Click, несколько раз вызывающая функцию Convert, и комментарии, описывающие допущенные ошибки:

```

Private Sub Button1_Click(ByVal sender As_
                        System.Object, ByVal e As System.EventArgs)_
Dim X, Y, Z As Integer
Dim A, B, C As String
Dim L, M, N As Single
    L = Convert(X, Y) ' В этой строке ошибок нет
    C = Convert(A, B) ' A и B не Integer, C не Single
    Z = Convert(M, N) ' M и N не Integer, Z не Single
    Y = Convert("OK", X) ' Y не Single, "OK" - строка
End Sub

```

## Ошибки при определении значений аргументов

Наиболее часто при вызове функций встречаются ошибки двух типов. Во-первых, количество передаваемых значений не всегда соответствует количеству объявленных аргументов.

Во-вторых, случаются попытки передать значения не тех типов данных, которые были объявлены для аргументов.

### Передача неверного числа аргументов

При объявлении списка аргументов определяется общее количество значений, которые должны быть переданы функции в момент ее вызова. Если при вызове функции вы попытаетесь передать ей большее или меньшее количество значений, это будет воспринято как ошибка и программа откажется работать. Рассмотрим в качестве примера такую функцию:

```
Public Function Flash(ByVal Set As String) As Single
```

```
End Function
```

Для нормальной работы этой функции ей требуется передать одно текстовое значение. Поэтому ни одна из предпринимаемых ниже попыток вызова функции не будет успешной, поскольку значение либо не передается вовсе, либо передается более чем одно значение:

```
✓ X = Flash
✓ X = Flash("Sun", "Moon")
✓ X = Flash("S", "M", "F", 12)
```

### Несоответствие типов данных

При вызове функции убедитесь, что ей передаются значения тех же типов, которые были объявлены для ее аргументов. Обратимся еще раз к знакомой нам функции:

```
Public Function Flash(ByVal Set As String) As Single
```

```
End Function
```

При вызове этой функции ей нужно передать одно значение текстового типа. Все предпринимаемые ниже попытки вызова функции ни к чему не приведут, поскольку передаваемые значения являются числами:

```
✓ X = Flash(14.2135)
✓ X = Flash(101)
✓ X = Flash(0.00101)
```

## Быстрое завершение работы функции

Обычно функции продолжают свою работу до тех пор, пока не будет выполнена последняя инструкция. Однако в случае необходимости выполнение функции можно прервать.

Для этого достаточно набрать такой код:

```
Exit Function
```



Прежде чем команда `Exit Function` будет приведена в действие, имени функции должно быть присвоено какое-то значение (т.е. должен быть определен возвращаемый функцией результат). В противном случае программа, скорее всего, будет работать некорректно.

## Тест на проверку полученных вами знаний

1. Когда лучше использовать функции, а когда процедуры?
  - а. Если нужно сделать программу более функциональной, создавайте функции.
  - б. Если вам надоели функции, используйте процедуры, и наоборот.
  - а. Остановитесь на чем-то одном. Например, функции используйте по пятницам, а процедуры - по субботам.
  - г. Процедуры используются для **выполнения** определенных задач (скажем, отображения сообщения на экране), в то время как функции предназначены для вычисления **какого-то** одного результата.
2. Какая строка вызывает функцию, а какая процедуру:
 

```
Dim A, B, C As Single
A = 3
B = 5
C = Black(A, B) 'Стр.4
FlashColour(C) 'Стр.5
```

  - а. В строке 4 переменной с присваивается значение, возвращаемое функцией Black, а в строке 5 вызывается процедура FlashColour.
  - б. Первая строка вызывает процедуру, потому что она самая длинная, а последняя строка вызывает функцию, потому что она не похожа на все остальные.
  - в. Вы можете выбрать звонок другу.
  - г. Вы можете выбрать подсказку зала.



## Часть VII

# Объектно-ориентированное программирование



## *В этой части...*

**Объектно-ориентированное программирование** — это именно то, что **поможет** вам быстро и **без** особых усилий создавать надежные и качественные **программы**. Секрет заключается в том, что методы **объектно-ориентированного программирования** позволяют правильно организовать **структуру программы**, а следовательно, **упрощают** процесс ее написания и поиск возможных ошибок.

В этой части дается **краткий** обзор принципов и приемов **объектно-ориентированного программирования**, достаточный для того, чтобы вы **смогли начать использовать** их при написании своих программ. Конечно, сам **факт применения** методов **объектно-ориентированного программирования** **не гарантирует** того, что вы сразу же станете профессиональным программистом, но ваши шансы написать эффективно работающую программу резко **возрастают**. А **создать функционирующую** программу не всегда удастся даже людям, **работающим** на крупные корпорации и правительства.

# Так что же это такое - объектно-ориентированное программирование?

*В этой главе...*

- Создание структурированных программ
- Объектно-ориентированное программирование как способ решения различных проблем

**Н**е отставая от новейших тенденций и достижений в области программирования, Visual Basic .NET предоставляет вам возможность для написания своих самых потрясающих программ использовать методы объектно-ориентированного программирования. Конечно, если вы не будете знать, что это такое и "с чем его едят", все эти потенциальные возможности останутся вами не замеченными и не использованными.

Поэтому, прежде чем сказать: "Мне это не нужно, я могу программировать и так", не поленитесь и прочтите следующие главы, чтобы иметь хотя бы общее представление о данном предмете.

Одна из причин, почему было создано такое направление, как объектно-ориентированное программирование, заключается в том, что несмотря на попытки многих учебных заведений подготовить хороших программистов, программирование остается настолько специфической сферой деятельности, что относится скорее к области искусства, чем к науке. Даже если вы закончите вуз и получите диплом специалиста по компьютерным наукам, это еще не означает, что вы сможете программировать лучше, чем какой-нибудь школьник-вундеркинд.

Поэтому, чтобы сделать процесс написания программ более простым и поддающимся обычной человеческой логике, постоянно разрабатываются новые техники и приемы программирования. Овладев ими, вы сможете программировать на качественно более высоком уровне.




## Принципы структурного программирования

На заре программирования почти никто не утруждал себя составлением каких-либо планов собственных программ. Программисты просто начинали писать коды программ, добавляя в них по мере необходимости все новые и новые команды. Создаваемые программы хотя и работали, но понять их коды было практически невозможно. Со временем эти коды даже стали называть *спагетти*, поскольку понять в них что к чему было так же сложно, как распутать настоящие спагетти, приготовленные в итальянском ресторане.

Но запутанные коды работающих программ — это еще не проблема. Настоящие проблемы начинались тогда, когда нужно было модифицировать эти коды, с тем чтобы как-то изменить существующую программу или добавить к ней новые возможности. Поскольку структура кодов не имела никакой логики и была в высшей степени запутанной, добавление новых кодов почти всегда сопровождалось появлением множества ошибок, в результате чего программы вовсе выходили из строя.

Еще более серьезные проблемы, связанные с отсутствием планирования при написании программ, случались из-за того, что программисты не могли удержать в своей голове множество разрозненных деталей, теряли их след, забывали о них, и хотя программы работали, это было не совсем то или совсем не то, что задумывалось изначально. Поскольку разобраться в написанных кодах и правильно их откорректировать было практически невозможно, большинство программистов просто начинали писать свои программы заново. Новые версии программ, опять-таки, были далеки от совершенства, процесс повторялся снова и снова, и в конечном результате некоторые программы так и не доводились до конца.

Одним из первых методов, разработанным для решения подобных проблем, а также для усовершенствования процесса создания больших программ, был метод, названный *структурным программированием*. В основу этого метода положено три идеи:

- ✓  делить большую программу на множество маленьких подпрограмм;
- ✓  объявлять переменные и типы принимаемых ими значений;
- ✓  использовать в маленьких программах только последовательные команды, команды ветвления и команды циклов.

## Разделение большой программы на множество подпрограмм

Написать одну большую программу, решающую сразу много задач, гораздо сложнее, чем написать несколько маленьких программ, каждая из которых предназначена для решения всего одной небольшой задачи. Поэтому, теоретически, можно написать много маленьких программ, занимающихся решением разных проблем, а затем объединить их для создания одной большой программы, решающей сразу множество задач.

Этот прием не только упрощает процесс написания больших программ, но и значительно облегчает процесс внесения в них изменений и поиска ошибок, поскольку теперь не нужно модифицировать программу в целом — достаточно определить соответствующую подпрограмму и работать только с ней.

Visual Basic .NET поддерживает такую практику, автоматически разделяя коды пользовательского интерфейса на отдельные процедуры обработки событий. Если же вы пишете коды BASIC для обработки данных и проведения вычислений, Visual Basic .NET также позволяет вам создавать небольшие общие процедуры и сохранять их в отдельных файлах модулей (о чем рассказывалось в главе 27).

## Объявление переменных

Изначально можно было создавать переменные в любом месте программы и присваивать им значения любых типов, благодаря чему разобраться, что делает программа, было практически невозможно. Поэтому и было решено, что все переменные должны объявляться заранее. В этом случае другие программисты могли понять, для чего в программе используется та или иная переменная.

Как средство борьбы с ошибками было принято еще одно решение: при объявлении переменных должен указываться тип данных, которые эти переменные могут содержать. Если вы попытаетесь присвоить переменной значение другого типа (не того, который был для нее объявлен), это сразу же будет воспринято как ошибка и выполнение программы прекратится. Этим способом предотвращается создание и распространение программ, неправильно сохраняющих данные. А как известно, некорректные данные приводят к сбоям в работе компьютеров, на которых они обрабатываются.

Сделав обязательным объявление переменных и типов принимаемых ими значений, Visual Basic .NET помогает вам создавать программы, понятные и простые для внесения в них изменений. (Информацию об использовании переменных вы найдете в главе 15.)

## Использование последовательных команд, команд ветвления и команд циклов

Самые большие проблемы в ранних программах возникали при использовании специальной команды, обозначаемой словом `GOTO`. Эта команда, по сути, говорила компьютеру: "Перейди в конец программы и выполни эту инструкцию, затем перейди на начало программы и выполни эту инструкцию, потом перейди к середине программы и сделай это, потом перейди в начало, затем перейди туда-то и т.д."

Из-за постоянных переходов из одного конца программы в другой практически невозможно было понять, что именно эта программа делает. Идея структурного программирования положила этому конец и ограничила перемещение по программе выполнением только трех видов команд: последовательных, ветвления и команд циклов.

При выполнении последовательных команд выполняется одна инструкция, затем вторая и т.д.

Если используется команда ветвления, компьютер выполняет один из возможных наборов инструкций. В Visual Basic .NET эти команды представлены операторами `If-Then` и `Select-Case` (они рассматривались в главах 22, 23).

Если выполняются команды циклов, компьютер несколько раз подряд повторяет один и тот же набор инструкций. Команды циклов, используемые в Visual Basic .NET, рассмотрены в главах 24, 25.

Как видите, ни одна из команд не может осуществлять переходы из одного конца программы в другой, а потому все последовательно выполняемые инструкции оказываются компактно собранными в отдельных частях программы, что значительно облегчает их понимание и дальнейшую модификацию.

## Объектно-ориентированное программирование

Хотя разработка и внедрение идей структурного программирования знаменовали собой большой шаг вперед, две крупные проблемы оставались нерешенными. Несмотря на то, что разные части программы становились изолированными друг от друга, каждая из них по-прежнему имела доступ к одним и тем же данным. Поэтому, если ошибка в одной части программы приводила к нарушению целостности данных, она не могла быть изолирована и в поисках приходилось просматривать коды всей программы.

Более того, если нужно было изменить источник данных (например, какой-нибудь файл, где хранились данные, перемещался из одного каталога в другой), приходилось просматривать коды всей программы и вносить изменения во все команды, которые как-то ссылались на эти данные. Если хотя бы одна такая команда пропусклась, возникала ошибка.

Вторая нерешенная проблема заключалась в том, что процесс написания небольших подпрограмм, как правило, занимал много времени.

Чтобы ускорить его, многие программисты копировали части уже работающих программ и вставляли их в свои программы. Это было достаточно удобно, поскольку стоило только внести небольшие коррективы — и фрагмент программы готов. Кроме того, если этот фрагмент мог быть использован в других частях программы, он копировался еще несколько раз. Теперь уже несложно было иметь по несколько вариантов одной и той же подпрограммы в кодах своей программы.

Но что если возникала необходимость внести в эту подпрограмму какие-то изменения? Приходилось опять-таки просматривать коды всей программы, находить эти скопированные фрагменты и изменять каждый из них по отдельности.

Для решения перечисленных проблем была разработана концепция объектно-ориентированного программирования (сокращенно ООП), объединяющая в себе принципы структурного программирования и некоторые новые идеи, которые не только позволяют быстрее и более простыми способами создавать надежные программы, но и значительно упрощают процесс их отладки. Двумя основными нововведениями ООП являются принцип инкапсуляции и принцип наследования.

## Инкапсуляция: разделение данных и команд

Как и структурное программирование, ООП рекомендует делить большие программы на несколько маленьких подпрограмм. Но в отличие от структурного программирования, просто разбивающего связанные между собой наборы команд на группы, объектно-ориентированное программирование изолирует как инструкции, так и обрабатываемые ими данные в отдельных подпрограммах. Чтобы лучше понять этот принцип, взгляните на рис. 30.1, демонстрирующий различие между обычными программами и программами, написанными с использованием приемов ООП.

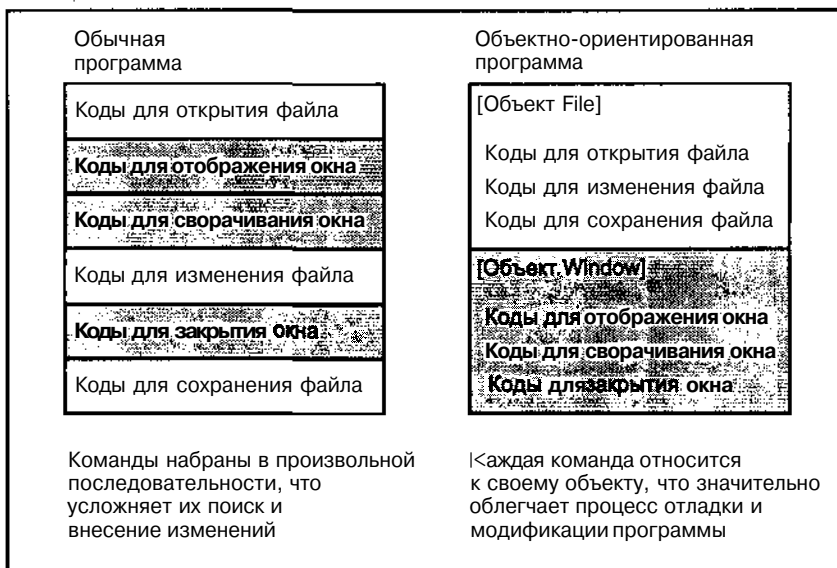


Рис. 30.1. Коды объектно-ориентированных программ легче для понимания и дальнейшей модификации

Выделяя команды и обрабатываемые ими данные в отдельный объект, вы изолируете разные части программы и не позволяете им вмешиваться в работу друг друга. Таким образом, большая программа создается не из множества подпрограмм, а из множества отдельных объектов.

## Наследование повторно используемых кодов

В структурном программировании при повторном использовании какого-либо набора кодов таковой необходимо было физически скопировать и вставить в другое место программы. В объектно-ориентированном программировании этого делать не нужно. Достаточно создать новый объект на базе существующего.

На самом деле новый объект не является обычной копией исходного объекта. Вместо того чтобы просто скопировать существующие коды, новый объект их *наследует*. Это означает,

что в новом объекте содержится ссылка на коды исходного объекта. Когда компьютер приступает к выполнению инструкций объекта, который наследует коды другого объекта, он получает такую команду: "Коды BASIC, которые должны быть выполнены, содержатся в другом объекте. Вот ссылка на эти коды, найди их и приступай к работе".

Главным преимуществом метода наследования является тот факт, что повторно используемые коды не нужно копировать. Теперь, если возникнет необходимость как-то их изменить, достаточно будет внести изменения только в исходный фрагмент, и не заниматься поиском в программе всех остальных мест, где эти коды были использованы.

## Перегрузка существующих кодов

Еще одним новшеством объектно-ориентированного программирования является возможность *перегрузки*. Она вытекает из принципа наследования кодов, при котором один объект ссылается на коды другого объекта. Разумеется, почти всегда к новому объекту добавляются какие-то дополнительные коды, а значит, такой объект будет состоять из двух наборов кодов: из тех, которые были наследованы из другого объекта, и из собственных.

Как бы там ни было, но объекты, которые наследуют коды друг друга, *похожи* между собой и являются, по сути, разными версиями одной и той же подпрограммы. Возможность *перегрузки* позволяет присваивать одинаковые имена подпрограмме, которая содержит исходные коды, и подпрограмме, которая их наследует.

В обычных условиях, если бы вы присвоили двум процедурам одинаковые имена, Visual Basic .NET воспринял бы это как ошибку и отказался выполнять вашу программу. Но в случае создания объектно-ориентированной программы вы можете присваивать различным подпрограммам, относящимся к разным объектам, одинаковые имена. Таким образом, отпадает необходимость присваивать "почти одинаковые" (различающиеся одной-двумя буквами) имена подпрограммам, которые, по сути, решают одни и те же задачи.

Предположим, вы создали объект, отображающий на экране фигурку самолета. Одна из подпрограмм этого объекта, имеющая название Полет, отвечает за его перемещение по экрану. А еще вам нужно создать объект, отображающий на экране фигурку воздушного шара. Этот объект можно было бы создавать "с нуля", но намного проще будет наследовать коды из уже существующего объекта, отображающего на экране фигурку самолета.

Конечно, путь преодолеваемый воздушным шаром, несколько отличается от траектории самолета, поэтому процедуры, отвечающие за их перемещение по экрану, будут немного разными. Благодаря возможности перезагрузки процедуру, перемещающую по экрану фигурку воздушного шара, также можно назвать именем Полет. Теперь, если коды объекта, изображающие на экране воздушный шар, вызывают процедуру Полет, запускается именно та процедура, которая принадлежит этому объекту (т.е. перемещающая воздушный шар). Если же процедуру Полет вызывает объект, изображающий на экране самолет, запускается именно та процедура, которая отвечает за перемещения самолета.

Таким образом, если ваша программа вызывает процедуру Полет, компьютер в первую очередь уточняет, фигурку какого именно объекта нужно отправить в путешествие по экрану, и уже в зависимости от этого запускается та или иная версия процедуры Полет.

Разрешая использовать одинаковые имена для подпрограмм, управляющих различными объектами, объектно-ориентированное программирование дает вам возможность присваивать одни и те же имена столько раз, сколько это необходимо.



Объектно-ориентированное программирование упрощает процесс создания больших программ благодаря тому, что не смешиваются данные, обрабатываемые разными группами команд, а также благодаря повторному использованию кодов без их предварительного копирования.

## Тест на проверку полученных вами знаний

1. Перечислите основные принципы структурного программирования.
  - а. Не бойся (ошибок).  
Не верь (чужим программам).  
Не проси (повышения зарплаты).
  - б. Разделение большой программы на несколько маленьких подпрограмм; объявление переменных; использование только последовательных команд, команд ветвления и команд **циклов**.
  - в. Основной принцип таков: создайте свою структуру из нескольких программистов — и они сами начнут писать для вас программы.
  - г. **Пожалуй, нужно еще раз прочитать эту главу.**
2. Какие новые возможности предоставляет объектно-ориентированное программирование?
  - а. Упрощает процесс отображения на экране самолетиков, воздушных шаров и других летательных аппаратов.
  - б. Не **позволяет** программе совершать ошибки и менять свою ориентацию.
  - в. Инкапсуляция команд и данных; наследование существующих кодов; перегрузка.
  - г. Можно указать в своем резюме, что вы программист со знанием ООП, и приступить к поиску более престижной работы.



# Объектно-ориентированное программирование на практике

*В этой главе...*

- Определение объектно-ориентированного программирования
- Создание объектов в Visual Basic .NET
- Использование объектов

**Д**лавная идея объектно-ориентированного программирования заключается в том, чтобы разделить программу на отдельные объекты, которые в дальнейшем будут взаимодействовать друг с другом. Например, если вы создаете игру, смысл которой в том, чтобы бегать по лабирингу и убивать всяких монстров, вам нужно создать две группы объектов, представляющих боевой арсенал и всех участвующих в резне персонажей.

Каждый из "оружейных" объектов будет представлять какой-то один вид оружия, который может быть использован игроками или монстрами (другими словами, все, что стреляет), а каждый из объектов второй группы будет представлять одного из игроков или одного из монстров (другими словами, все, что бегает).



Объектно-ориентированное программирование призвано облегчить процесс создания и модификации больших программ. Но само по себе оно не является гарантией быстрого создания эффективно работающих программных продуктов. Плохой программист, использующий ООП, по-прежнему будет писать более слабые программы, чем хороший программист, который преимуществами ООП не пользуется.

## *использование приемов ООП в Visual Basic .NET*

Теперь, когда вы имеете общее представление об объектах, пришло время узнать, как они создаются и применяются в Visual Basic .NET. Процесс создания объектов в Visual Basic .NET можно разделить на три этапа:

1. **Создание файла классов.**
2. **Написание колов BASIC в файле классов.**
3. **Объявление в программе переменных, которые представляют собой файлы классов.**

Если переменная представляет файл классов (который является как бы одним из типов данных, таким как Integer или String), она называется *объектом* — термином, взятым из концепции объектно-ориентированного программирования.

Файл классов является, по сути, шаблоном, который определяет, как должны вести себя объекты. После создания файла классов вам еще нужно объявить переменную — чтобы создать таким образом сам объект.



## Создание файла классов

Файл классов определяет характеристики и поведение отдельного объекта. Чтобы создать файл классов, выполните такие действия.

1. **Выберите команду Project⇒Add Class (Проект⇒Создать класс).**

Откроется диалоговое окно **Add New Item** (Создать новый элемент).

2. **Щелкните на значке Class, а затем — в поле Name.**

3. **Укажите имя создаваемого файла.**

В конце имени указывать расширение .VB необязательно — Visual Basic .NET сделает это автоматически.

4. **Щелкните на кнопке Open.**

Название только что созданного файла появится в окне Solution Explorer (если оно отображено на экране), а в окне кодов отобразятся коды пустого класса, которые будут выглядеть так, как показано ниже (вместо слова `Class1` будет указано имя, данное этому файлу на третьем шаге):

```
Public Class Class1

End Class
```

Файл, не содержащий кодов, никакой ценности не представляет. Теперь вам предстоит набрать коды BASIC между строками `Public Class Class1` и `End Class`.

## Определение объекта

После того как файл классов будет создан, приступайте к написанию кодов BASIC, определяющих, что из себя представляет новый объект. Обычно такие коды (они также называются модулем класса) состоят из трех частей:

- ✓ объявление переменных;
- ✓ объявление свойств (данных);
- ✓ методы, являющиеся процедурами BASIC (описаны в главе 27), которые манипулируют переменными и свойствами.

```
Public Class Class1
    ' Объявление переменных

    ' Объявление свойств

    ' Методы
End Class
```

Каждый объект инкапсулирует (отделяет) свои данные (свойства) от остальной части программы. Эти данные могут быть изменены лишь командами (методами), сохраненными в том же файле классов. Таким образом, другие команды программы никогда непосредственно не обращаются к данным, принадлежащим какому-то объекту. Различие между тем, как осуществляется доступ к данным в обычной и в объектно-ориентированной программах, показано на рис. 31.1.

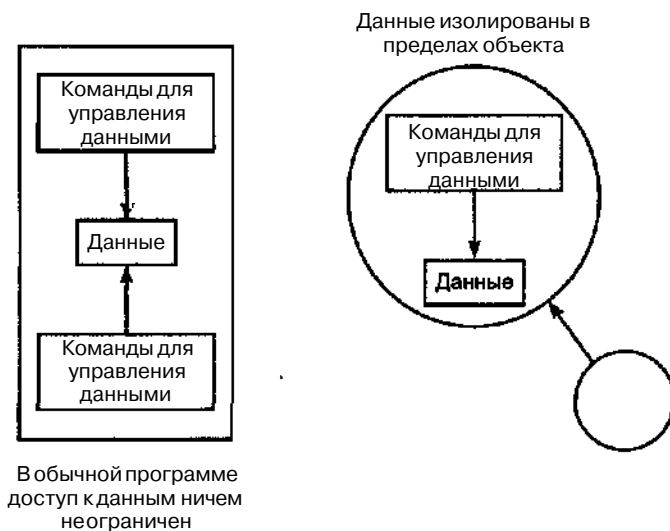


Рис.31.1. В объектно-ориентированной программе данные изолируются путем внедрения их в объект

Если возникает необходимость изменить способ обработки каких-либо данных, просматривать всю программу уже не нужно — достаточно изменить коды только одного объекта. Таким образом, объектно-ориентированное программирование помогает изолировать команды, которые имеют доступ к определенным данным, что значительно снижает вероятность возникновения ошибок при внесении изменений в коды программы.



Само по себе использование объектов не является единственно верным способом написания программы, но чем больше вы их создадите, тем понятнее станет ваша программа и тем проще будет в дальнейшем вносить в нее изменения.

## Объявление переменных

Объявление переменных в начале модуля класса является хорошей практикой (к тому же обязательной), поскольку позволяет в любой момент видеть, какие данные этим классом используются. Если нужно объявить переменную, доступ к которой могут получить команды только этого класса, объявите ее как локальную переменную:

```
Private Число As Integer
```



Переменные очень часто используются для временного хранения значений свойств объектов, о чем речь пойдет в следующем разделе данной главы.

Хотя делать это нежелательно, но переменную можно объявить и как глобальную, доступ к значениям которой могут получить любые команды вашей программы. Чтобы объявить глобальную переменную, замените слово `Private` ключевым словом `Public`:

```
Public Число As Integer
```



Если переменная будет объявлена как глобальная, любая часть программы (включая другие объекты) будет иметь возможность изменять хранимые ею данные. Следовательно, вероятность возникновения ошибок значительно возрастет. Поэтому не объявляйте глобальные переменные, если на это нет действительно веских причин.

## Объявление свойств объектов

Свойства объектов представляют те данные, доступ к которым имеют все коды вашей программы. Любая часть программы может передать свойствам объектов новые значения и извлечь хранимые ими данные. Коды объявления свойств выглядят приблизительно так:

```
Dim. ИмяПеременной As ТипДанных
```

```
Public Property НазваниеСвойства() As ТипДанных
```

```
Get
```

```
    НазваниеСвойства = ИмяПеременной
```

```
End Get
```

```
Set (ByVal Значение As ТипДанных)
```

```
    ИмяПеременной = Value
```

```
End Set
```

```
End Property
```

Чтобы объявить свойство объекта, нужно создать локальную переменную (ИмяПеременной), в которой будет храниться значение свойства. Вместо имени ИмяПеременной можно присвоить любое другое имя.

Тип этой переменной (ТипДанных) должен совпадать с типом данных, объявленным для свойства. Например, если объявлено, что свойство будет содержать значения типа String, переменная также должна иметь тип String.

И, наконец, необходимо указать имя, посредством которого другие коды программы смогут иметь доступ к значениям свойств объектов (НазваниеСвойства).

Например, если нужно объявить свойство объекта, именуемое словом Вектор и содержащее значения типа Integer, это можно сделать так:

```
Dim ЗначВек As Integer
```

```
Public Property Вектор() As Integer
```

```
Get
```

```
    Вектор = ЗначВек
```

```
End Get
```

```
Got
```

```
    ЗначВек = Value
```

```
End Set
```

```
End Property
```

Вот как эти коды будут восприняты Visual Basic .NET.



1. В первой строке объявляется о создании переменной ЗначЗек, которая может содержать значения типа Integer.
2. Вторая строка говорит: "Создай свойство Вектор, которое будет представлять данные типа Integer".
3. Третья, четвертая и пятая строки дают указание: "Когда другая часть программы хочет извлечь (Get) данные, представляемые свойством Вектор, передай ей значение переменной ЗначВек".
4. Шестая, седьмая и восьмая строки говорят: "Когда другая часть программы хочет передать (Set) новое значение свойству Вектор, сохрани его как значение локальной переменной ЗначВек".
5. А девятая строка сообщает: "На этом объявление свойства завершается".

## Создание методов

После того как все необходимые переменные и свойства объекта объявлены, можно приступить к написанию методов (другими словами, процедур и функций), которые будут что-то делать с данными, сохраненными в этом объекте.

Методы — это те же процедуры и функции, процесс создания которых был рассмотрен в главах 27–29. Главным их отличием от таковых является то, что вместо слова `Private` при объявлении процедуры или функции нужно использовать слово `Public`, например:

```
Public Function ИмяФункции () As Integer
    ' Набор инструкций
End Function
```

Затем, если нужно запустить в действие метод объекта, вы просто указываете имя объекта и название метода:

```
ra_Object.Move
```

Этот код приказывает Visual Basic .NET: "Найди объект, именуемый `m_Object`, и запусти процедуру, названную `Move`". Пока эта процедура будет выполняться, основная программа не будет знать, что именно она делает, поскольку процедура изолирована внутри своего объекта.

## Прежде чем приступить к написанию кодов

Хотя вас, наверное, одолевает желание немедленно приступить к написанию кодов BASIC для создания модуля классов, вначале все же потратьте немного времени и продумайте общую его структуру. Итак, какая структура является оптимальной? Неизвестно. (Сейчас вы, наверное, подумали: "А стоило ли покупать эту книгу?")

Дело в том, что выбор структуры модуля класса зависит от того, для каких целей этот класс создается. Являясь оптимальным для одной программы, вариант структуры модуля может быть совершенно неприемлемым для другой. Но все же можно выделить несколько общих рекомендаций, которые работают практически во всех случаях.



- ✓ Чтобы решить, какими свойствами должны обладать модули классов, определите основные блоки данных, которыми должна оперировать программа. Если вы хотите написать программу, обрабатывающую данные о сотрудниках, модуль класса должен иметь такие свойства, как имя, адрес, телефон, возраст, стаж и т.п. Если вы создаете видеоигру, где по экрану ползают разные жучки, а игроки безжалостно их отстреливают, модуль класса должен содержать такие свойства, как координаты X и Y для определения позиции жучка на экране.
- ✓ Перед тем как создавать методы в модулях классов, определите, какие основные действия должна выполнять программа в отношении данных, хранящихся в отдельных объектах. Например, если модуль класса содержит личные данные сотрудников, нужно создать методы, позволяющие основной программе извлекать, сортировать и выводить их на печать. Если же модуль класса содержит информацию о координатах ползающих по экрану насекомых, нужно создать методы, позволяющие основной программе заниматься перемещением, отображением и удалением с экрана всей этой живности.
- ✓ После того как вы определите для себя весь перечень свойств и методов, которые должны быть созданы, можете приступить к непосредственному воплощению своих идей в кодах BASIC.

# использование модулей классов в *fttoifaa*tax Visual Basic .NET

После того как вы пройдете через все перипетии процесса создания модулей классов, в основной программе нужно будет написать коды BASIC, использующие модули классов для создания реальных объектов. Как только объекты будут готовы, можно будет использовать методы объектов для сохранения и извлечения необходимой информации. Доступ к информации осуществляется через свойства объектов.

## Создание объектов

Модуль класса— это еще не объект. Он является лишь средством, с помощью которого объекты создаются. Согласно терминологии объектно-ориентированного программирования, создание объектов называется также *созданием экземпляров классов*.

Чтобы создать экземпляр класса, нужно создать объект, который будет представлять модуль класса. Для этого используется ключевое слово New:

Dim ИмяОбъекта As New НазваниеКласса

Вот что эта строка означает для Visual Basic .NET.



1. Слово Dim говорит: "Сейчас будет объявлено о создании объекта".
2. Словом ИмяОбъекта обозначается имя объекта.
3. Слово New дает указание: "Создай новый объект, который будет представлять модуль класса, именуемый НазваниеКласса".



НазваниеКласса — это то имя, которое вы дали файлу класса в момент его создания. Если вы сами не укажете это имя, Visual Basic .NET присвоит ему автоматически генерируемое имя наподобие Class1.

## Использование объектов

После того как объект создан, остается только использовать его для:

- ✓ сохранения значений в свойствах объектов;
- ✓ извлечения данных через свойства объектов;
- ✓ манипулирования данными объектов с помощью методов объектов.

Чтобы передать объекту данные, необходимо написать такой код:

ИмяОбъекта.Свойство = Значение

Извлечь информацию, хранящуюся в объекте, вам поможет код:

ИмяПеременной = ИмяОбъекта.Свойство

А чтобы запустить метод объекта, наберите следующее:

ИмяОбъекта.Метод

## Как это выглядит на практике

Лучший способ понять что-то — применить его, это "что-то", на практике. Если то, о чем вы прочитали в данной главе, кажется вам слишком сложным, попробуйте вместе с нами соз-

дать программу, отображающую на экране диалоговое окно, показанное на рис. 31.2. Сама по себе эта программа достаточно проста, Она призвана продемонстрировать, как можно создавать модули классов, затем создавать объекты для представления данных, вызывать методы объектов для их обработки, и, наконец, извлекать обработанные данные.

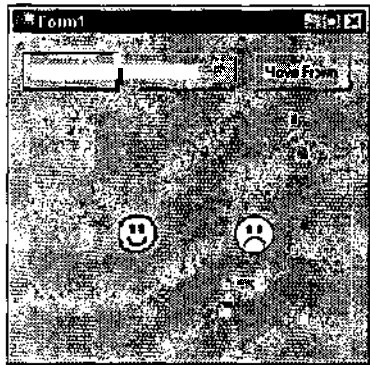


Рис. 31.2. Такое диалоговое окно будет отображать создаваемая программа

В данной программе создаются файлы классов для представления местоположения улыбающегося и грустного лиц. Каждый раз, когда пользователь щелкает на кнопке Move Smiley (Переместить улыбающееся лицо) или Move Frown (Переместить фустное лицо), эти картинки начинают перемещаться. Ниже в таблице приведен список объектов пользовательского интерфейса, которые нужно создать для этой программы, и список значений, которые нужно присвоить их свойствам.



Если вам по-прежнему трудно понять, что такое объектно-ориентированное программирование и почему его нужно использовать, запомните хотя бы одно: его главная цель — помочь вам правильно организовать свои программы и снизить вероятность возникновения ошибок.

Объект	Свойство	Значение
PictureBox1	Image	FASE02
	Name	picSmiley
	SizeMode	StretchImage
PictureBox2	Image	FASE04
	Name	picFrown
	SizeMode	StretchImage
Button 1	Text	Move Smiley
	Name	btnSmiley
Button2	Text	Exit
	Name	btnExit
Button3	Text	Move Frown
	Name	btnFrown

Дважды щелкните на каждой из трех кнопок формы и создайте такие процедуры обработки событий:

```
Private Sub btnSmile_Click(ByVal sender As System._
    Object, ByVal e As System.EventArgs)_
    Handles btnSmile.Click
    Dim Smile As New Class1()
    Smile.xspot = picSmile.Location.X
    Smile.Move()
    picSmile.Left = Smile.xspot
End Sub

Private Sub btnFrown_Click(ByVal sender As System._
    Object, ByVal e As System.EventArgs)_
    Handles btnFrown.Click
    Dim Frown As New Class1()
    Frown.yspot = picFrown.Location.Y
    Frown.Move().
    picFrown.Top = Frown.yspot
End Sub

Private Sub btnExit_Click(ByVal sender As System._
    Object, ByVal e As System.EventArgs)_
    Handles btnExit.Click
    Me.Close()
End Sub
```

Создайте отдельный файл класса и наберите следующее:

```
Public Class Class1
    Dim x, y As Integer
    Const Mar = 10

    Public Property xspot() As Integer
        Get
            xspot = x
        End Get
        Set(ByVal Value As Integer)
            x = Value
        End Set
    End Property

    Public Property yspot() As Integer
        Get
            yspot = y
        End Get
        Set (ByVal Value As Integer)
            y = Value
        End Set
    End Property

    Public Sub Move()
        x = x + Mar
        y = y + Mar
    End Sub
End Class
```



# Наследование и перегрузка

*В этой главе*

- Преимущества наследования кодов
- Наследование элементов пользовательского интерфейса
- Наследование кодов
- > Перегрузка методов и свойств

**Л** программисты, как и все нормальные люди, — существа достаточно ленивые. Вместо того чтобы написать новую подпрограмму, они, скорее, скопируют уже созданную кем-то работающую процедуру и вставят ее в свою программу. Копированием существующих подпрограмм, во-первых, значительно экономится время, во-вторых, снижается вероятность возникновения ошибок.

Однако ранее повторное использование одних и тех же кодов означало создание их отдельных копий и физическое копирование в разные части программы. Очень скоро таких копий накапливалось в программе довольно много, и если требовалось внести какие-то изменения в их работу, приходилось кропотливо просматривать коды всей программы.

Более того, создание отдельных физических копий кодов означало, что вы или кто-то другой могли случайно (или преднамеренно) внести в некоторые из них изменения, нарушив таким образом работу программы. Итак, мы подходим к одному из наиболее существенных преимуществ объектно-ориентированного программирования — к *наследованию кодов*, которое позволяет повторно использовать фрагменты кодов без необходимости создания их отдельных копий.

## *Что есть хорошего в наследовании кодов?*

Один файл классов определяет один вид объектов. Используя один и тот же файл классов, можно создать любое количество подобных объектов, но все они будут иметь одинаковые характеристики. Предположим, что нужно создать два объекта, характеристики которых должны быть почти одинаковыми, но все же немного различаться.

Можно создать второй файл классов для второго объекта, но этот путь связан с временными затратами и с повышением вероятности возникновения ошибок. (Дело в том, что написание любых кодов связано с возможностью возникновения ошибок, поэтому всегда старайтесь быть законными.)

Вместе этого объектно-ориентированное программирование позволяет вам наследовать коды, что означает возможность создания нового объекта на базе *существующего*. Так, вместо создания нового файла классов для второго объекта, можно наследовать первый объект и добавить коды BASIC для определения различающихся характеристик (известных как свойства и методы).

Таким образом, новый объект будет обладать всеми характеристиками старого объекта и иметь *собственные*, которыми вы его “снабдите”. Обратите внимание, что если коды первого объекта работают правильно, то вам остается только без ошибок написать дополнительные коды, чтобы коды второго объекта также работали корректно.



Наследуя правильно работающие коды, вы уменьшаете вероятность возникновения ошибок.

## Наследование кодов для создания новых форм

Простейший пример наследования — это наследование готовых форм. При этом, по сути, создается новая копия уже существующей формы. Если вы вносите какие-то изменения в оригинальную форму, Visual Basic .NET автоматически вносит те же изменения в форму, созданную путем наследования.

Наследуя одну форму, вы создаете ее точную копию, которая содержит те же элементы пользовательского интерфейса (в частности кнопки и раскрывающиеся меню) и использует те же процедуры обработки событий.

Чтобы наследовать форму, вначале, разумеется, нужно ее создать. А наследуется исходная форма следующим образом.

1. Выберите команду **Project**⇒**Add Inherited Form** (**Проект**⇒**Наследовать форму**).

Откроется диалоговое окно **Add New Item**, показанное на рис. 32.1.

2. Щелкните на значке **Inherited Form**.

3. Щелкните в поле **Name** (**Имя**) и наберите имя новой формы.

Какое бы имя вы ни набрали, не указывайте расширение, отличное от **.VB**. В противном случае Visual Basic .NET может просто не распознать файл создаваемой формы как часть вашей программы.

4. Щелкните на кнопке **Open** (**Открыть**).

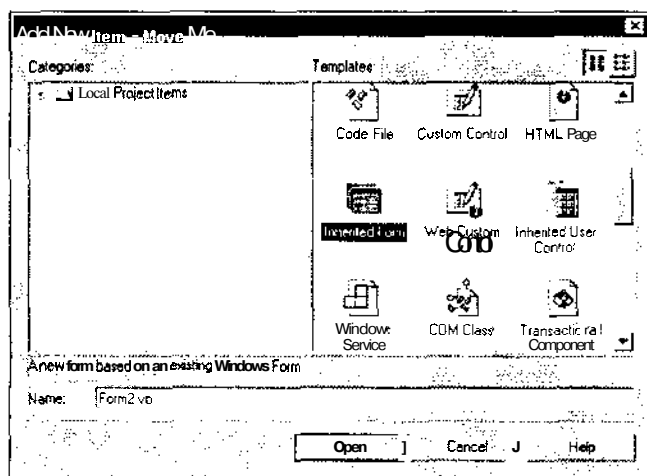


Рис. 32.1. Диалоговое окно **AddNewItem**

Откроется диалоговое окно **Inheritance Picker** (Источник наследования), показанное на рис. 32.2.

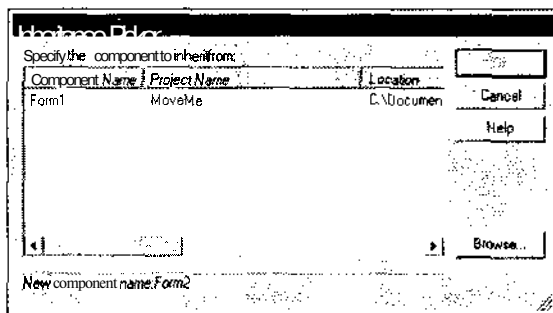


Рис. 32.2. Диалоговое окно *Inheritance Picker*

5. Выберите форму, которую вы хотите наследовать, и щелкните на кнопке ОК.

Visual Basic .NET создаст НОВУЮ форму и отобразит ее имя в окне Solution Explorer.



Если вы внесете какие-то изменения в исходную форму (выбранную на пятом шаге), выполните команду **Build⇒Rebuild All** и все эти изменения будут перенесены и на другие формы, созданные из данной путем наследования.

## Наследование рабочих кодов программы

Наследование форм (и сопутствующих им процедур обработки событий)— еще более простая задача, чем обычное создание пользовательского интерфейса. Но возможности Visual Basic .NET этим не ограничиваются, и вы можете наследовать другие коды BASIC, сохраненные в файлах классов.

Чтобы наследовать коды, нужно создать отдельный файл классов и вставить в него строку с ключевым словом **Inherits** и с именем файла классов, коды которого вы хотите наследовать. Например:

```
Public Class Class1
    Inherits ИмяКласса
End Class
```

Вот как Visual Basic .NET воспримет этот код:

1. Первой строкой начинаются коды файла классов, названного именем Class1.
2. Вторая строка говорит, что нужно наследовать коды BASIC из класса ИмяКласса.
3. Третья строка указывает на окончание кодов класса Class1.

Для того чтобы продемонстрировать, как работают наследованные коды, рассмотрим следующий пример. Предположим, вы создали файл классов MainClass, который **выглядит** так;

```
Public Class MainClass
    Private mWay As Integer

    Public Property Direction() As Integer
        Get
            Direction = mWay
        End Get

        Set (ByVal Value As Integer)
            mWay = Value
        End Set
    End Property
End Class
```

```

        End Set
    End Property
End Class

```

Если вам нужно создать новый файл классов (именуемый CopyClass), который будет наследовать коды файла MainClass, коды нового файла могут выглядеть так:

```

Public Class CopyClass
    Inherits MainClass
End Class

```

Хотя новый файл классов может выглядеть пустым, на самом деле его коды абсолютно равнозначны следующим кодам:

```

Public Class CopyClass
    Private mWay As Integer

    Public Property Direction() As Integer
        Get
            Direction = mWay
        End Get

        Set (ByVal Value As Integer)
            mWay = Value
        End Set
    End Property
End Class

```

## Перегрузка свойств и методов

При наследовании кодов BASIC существующего объекта для создания нового могут возникнуть некоторые проблемы. Что если исходный объект использует для обработки своих данных метод с тем же именем, которое вы хотите присвоить методу, работающему с данными второго объекта?

Возможным решением этой проблемы может быть присвоение методам всех объектов разных имен. Разумеется, такой путь неудобен, поэтому Visual Basic .NET предлагает вам другое решение: так называемую *перегрузку (overriding)*.

Смысл перегрузки состоит в том, что свойства и методы объектов, созданных путем наследования, могут быть модифицированы без внесения изменений в свойства и методы исходного объекта.

Например, вы можете создать объект, использующий метод KillHardDisk, а затем путем наследования создать новый объект, который, разумеется, также будет иметь в своем арсенале метод KillHardDisk с теми же кодами. Однако, воспользовавшись приемом перегрузки, в новом объекте можно оставить метод с тем же именем, но полностью изменить его коды.



Если нужно запустить какой-то метод для обработки данных объекта, укажите имя этого объекта и имя самого метода:

ИмяОбъекта.ИмяМетода

Так, при необходимости вызвать метод KillHardDisk для обработки данных объекта, именуемого Virus, наберите код

```
Virus.KillHardDisk
```

Если вы наследуете коды объекта `Virus` для создания нового объекта `Crash`, а затем захотите для обработки данных нового объекта вызвать метод `KillHardDisk`, наберите код `Crash.KillHardDisk`

Таким образом, прием перегрузки позволяет использовать одни и те же имена для обозначения разных методов исходных и наследованных объектов.

Чтобы перегрузить какой-либо метод или свойство, выполните такие действия.

1. В окне `Solution Explorer` щелкните на названии файла классов, коды которого должны быть наследованы для создания новых классов.
2. Нажмите клавишу `<F7>`, выберите команду `View⇌Code` или щелкните на значке `View Code` в окне `Solution Explorer`.  
`Visual Basic .NET` отобразит на экране коды `BASIC` выбранного файла.
3. Для каждого свойства или метода, коды которого могут быть перегружены, замените слово `Public` ключевым словом `Overridable`.

Допустим, коды выбранного файла выглядят так:

```
Public Class CopyClass

    ' Здесь объявляются переменные

    Public Property Свойства() As String
        ' Здесь объявляются свойства
    End Property

    Public Sub Метод()
        ' Коды BASIC, обрабатывающие данные
    End Sub

End Class
```

Для свойств и методов замените слово `Public` словом `Overridable`:

```
Public Class CopyClass

    ' Здесь объявляются переменные

    Overridable Property Свойства() As String
        ' Здесь объявляются свойства
    End Property

    Overridable Sub Метод()
        ' Коды BASIC, обрабатывающие данные
    End Sub

End Class
```

Коды тех свойств и методов, для которых вы оставите слово `Public`, в новых объектах не могут быть изменены.

4. Выберите команду `Project⇌Add Class`.  
Откроется диалоговое окно `Add New Item`.
5. Наберите имя нового файла классов в поле `Name` и щелкните на кнопке `Open`.  
`Visual Basic .NET` отобразит показанные ниже коды пустого класса, где вместо слова `ClassName` будет фигурировать указанное вами имя:

```
Public Class ClassName
```

```
End Class
```

6. Наберите слово **Inherits**, а затем укажите название класса, коды которого должны быть наследованы.

Например, если нужно наследовать коды класса, названного именем **Class1**, наберите

```
Public Class ClassName  
    Inherits Class1  
End Class
```

7. Наберите коды свойств и методов, которые должны быть перегружены, но вместо слова **Public** используйте слово **Overrides**.

Слово **Overrides** используется при написании свойств и методов, коды которых будут отличаться от кодов свойств и методов исходного файла с теми же именами:

```
Public Class ClassName  
    Inherits Class1  
  
    Overrides Property Свойства() As String  
        ' Здесь объявляются свойства  
    End Property  
  
    Overrides Sub Метод()  
        ' Коды BASIC, обрабатывающие данные  
    End Sub  
End Class
```

Таким образом, при создании нового класса вам нужно будет написать коды только тех свойств и методов, которые должны отличаться от свойств и методов исходного класса.



Если приемы наследования и перегрузки кодов кажутся вам не очень понятными, не спешите сразу же отказываться от их применения. Помните, что они действительно могут помочь вам быстрее и более простым способом создавать корректно работающие программы, позволяя повторно использовать либо коды свойств и методов других объектов, либо только имена этих свойств и методов.

## *Практическое применение приемов наследования и перегрузки*

Чтобы лучше понять изложенный в этой главе материал, попробуйте набрать приведенные ниже коды и проследить за их выполнением. В примере используются форма и класс, созданные в главе 31.

Измените коды класса **Class1.vb**, заменив для метода слово **Public** словом **Overridable**:

```
Public Class Class1  
    Dim x, y As Integer  
    Const Max = 10  
  
    Public Property xspot{> As Integer
```

```

        Get
            xspot = x
        End Get
        Set: (ByVal Value As Integer)
            x = Value
        End Set
    End Property

    Public Property yspot() As Integer
        Get
            yspot = y
        End Get
        Set (ByVal Value As Integer)
            y = Value
        End Set
    End Property

    Overridable Sub Move()
        x = x + Шаг
        y = y + Шаг
    End Sub
End Class

```

Затем создайте новый файл классов, назовите его CopyClass и наберите для него код, приведенный ниже:

```

Public Class CopyClass
    Inherits Class1

    Overrides Sub Move()
        MsgBox("Метод Move перегружен")
    End Sub
End Class

```

Наконец, измените коды в файле формы таким образом, чтобы был использован как оригинальный файл классов (Class1), так и наследованный (CopyClass):

```

Public Class Form1
    Inherits System.Windows.Forms.Form

    Private Sub btnSmile_Click(ByVal sender As _           System.
Object, ByVal e As System.EventArgs)
        Handles btnSmile.Click
        Dim Smile As New Class1()
        Smile.xspot = picSmile.Location.X
        Smile.Move()
        picSmile.Left = Smile.xspot
    End Sub

    Private Sub btnFrown_Click(ByVal sender As
System.Object, ByVal e As System.EventArgs)
        Handles btnFrown.Click
        Dim Frown As New CopyClass()
        Frown.yspot = picFrown.Location.Y
        Frown.Move()
    End Sub

```

```

        picFrown.Top = Frown.yspot
End Sub

Private sub btnExit_Click(ByVal sender As
System.Object, ByVal e As System.EventArgs) _
    Handles btnExit.Click
    Me.Close()
End sub
End class

```

Если теперь вы щелкнете на кнопке формы **Move Smiley**, улыбающееся лицо, как и прежде, будет перемещаться по экрану. Если же вы щелкнете на кнопке **Move Frown**, Visual Basic .NET обратится к наследованному классу `CopyClass`, коды которого были перегружены, и на экране появится сообщение "Метод Move перегружен".

### Тест на проверку полученных вами знаний

1. Какое основное преимущество метода наследования?
  - а. Можно составить завещание на свои авторские права относительно создаваемой программы.
  - б. Способствует распространению законов гражданского кодекса на принципы программирования.
  - в. Обеспечивает возможность получения юридических навыков в процессе создания программы.
  - г. Основное преимущество этого метода заключается в возможности повторного использования проверенных рабочих кодов.
2. Что означает термин "перегрузка кодов"?
  - а. Если вы пытаетесь вложить слишком много смысла в набираемые коды, они становятся перегруженными.
  - б. Перегрузка кодов -- это действия программы, которые приводят к перезагрузке компьютера.
  - в. Если написанная вами программа заставляет так **напряженно "думать"** компьютер, что над ним появляется серый дымок, говорят, что он перегружен вашими кодами.
  - г. Это возможность редактировать коды созданных путем наследования свойств и методов объектов без внесения изменений в коды свойств и методов исходного объекта.



## Часть VIII

# Горячие десятки



Нас прислали почистить коды

## *В этой части*

Если вы дошли до этих **строк**, поздравляем вас с получением базовых знаний по языку программирования Visual Basic .NET (если **вы**, конечно, не принадлежите к категории людей, которые любят начинать читать книга с конца). В заключение хотелось бы дать несколько полезных советов и **рекомендаций**, призванных помочь вам в кратчайшие сроки достичь вершин мастерства в написании программ на Visual Basic .NET.

Вы можете принять предлагаемые советы или оставить их без внимания, но в любом случае помните, что **нет** предела совершенству, а потому старайтесь не пропускать никакой новой информации о программировании вообще и о языке Visual Basic .NET в частности.

# Десятка полезнейших советов, которые вы вряд ли найдете в каком-нибудь другом месте

### *В этой главе*

- Читайте периодику
- Посещайте Web-страницы Visual Basic .NET
- Принимайте участие в технических конференциях
- Освойте C#
- Пишите программы для операционных систем Macintosh, Linux, Palm OS и PocketPC

**Т**еперь, когда настоящая книга почти прочитана, вы наверняка думаете о том, что делать дальше, и как наиболее эффективно организовать процесс дальнейшего освоения языка Visual Basic .NET.

Ответ на данный вопрос вы найдете, прочитав последнюю часть книги, в которой содержатся общие рекомендации относительно того, как стать знатоком и специалистом по использованию одного из самых мощных и перспективных на сегодняшний день языков программирования, каковым является Visual Basic .NET.

## *Читайте специализированные периодические издания*

Если вы кроме Visual Basic .NET успели немного изучить и английский язык, найдите возможность просматривать специализированный журнал *Visual Basic Programmer's Journal*. В нем обсуждаются различные вопросы, связанные с Visual Basic .NET, дается обзор создаваемых надстроек, приводятся листинги кодов, которые вы можете использовать при создании своих программ.

Ни один из других компьютерных журналов не содержит так много полезной информации о языке Visual Basic .NET. Более подробные сведения об этом издании вы можете получить по адресу:

- ✓ *Visual Basic Programmer's Journal*, Fawcette Technical Publications, 209 Hamilton Avenue, Palo Alto, CA 94301-2500; Tel: 650-833-7100; Fax: 650-853-0230; [www.vbpj.com](http://www.vbpj.com).

# Подпишитесь на информационные бюллетени *Visual Basic Developer*

Корпорация Pinnacle Publishing, Inc. выпускает ежемесячные информационные бюллетени *Visual Basic Developer*, где вы найдете работающие подпрограммы Visual Basic .NET, которые можно копировать и использовать при создании собственных программ. Конечно, эта услуга стоит денег (около \$149 в год), но если вы сможете убедить свое начальство в полезности научных инвестиций, по вашему карману эта сумма не ударит.

Более подробно об информационных бюллетенях *Visual Basic Developer* вы можете узнать по адресу:

- ✓ *Visual Basic Developer*, Pinnacle Publishing, Inc., 1000 Holcomb Woods Parkway, 200, Suite 280, Roswell, GA 30076; Tel: 770-992-9401; Fax: 770-993-4323; [www.pinpub.com](http://www.pinpub.com).

## Ищите информацию в Internet

Если у вас есть доступ к Internet, почаще посещайте Web-страницы, посвященные Visual Basic .NET. Число таких Web-страниц постоянно растет, поэтому здесь указаны адреса лишь некоторых из них:

- ✓ Carl and Gary's Visual Basic Home Page: [www.cgvb.com](http://www.cgvb.com)
- ✓ Planet Source Code's Visual Basic section: [www.planet-source-code.com/vb](http://www.planet-source-code.com/vb)
- ✓ Visual Basic Explorer Home Page: [www.vbexplorer.com](http://www.vbexplorer.com)
- ✓ VB Tips & Tricks Home Page: [www.vbtt.com](http://www.vbtt.com)



Намного больший список адресов вы можете получить, набрав в своем Web-браузере в качестве параметра поиска слова **Visual Basic .NET**.

## Посещайте технические конференции

Периодически, по несколько раз в год, где-нибудь в Штатах, Европе или Азии компания Microsoft совместно с журналом *Visual Basic Programmer's Journal* проводит технические конференции, носящие официальное название *Visual Basic Technical Summit*. Здесь можно принять участие в обсуждении различных вопросов, связанных с языком Visual Basic .NET, послушать, о чем говорят представители компании Microsoft, купить по дешевке надстройки для Visual Basic .NET и просто познакомиться со многими людьми, так или иначе связанными с программированием на Visual Basic .NET. Для получения более полной информации об этих технических конференциях, обращайтесь по адресу:

- ✓ *Visual Basic Programmer's Journal*, Fawcette Technical Publications, 209 Hamilton Avenue, Palo Alto, CA 94301-2500; Tel: 650-833-7100; Fax: 650-853-0230; [www.vbpj.com](http://www.vbpj.com).

## *Покупайте программные продукты через посредников*

Не покупайте Visual Basic .NET или любые надстройки для этого приложения непосредственно у производителя. Большинство фирм, выпускающих программное обеспечение, без зазрения совести устанавливают цены на свои продукты ничуть не ниже, чем в розничной сети. Поэтому покупать что-либо у них настолько же неразумно, как платить полную цену за поддержанный автомобиль.

Если хотите сэкономить деньги, обратитесь к дилерам, распространяющим программное обеспечение по почте. Они предоставляют солидные скидки (до пятидесяти процентов), и вообще с ними всегда можно договориться.

К числу наиболее известных дилеров принадлежат такие компании, как Programmer's Paradise, Provantage и VBxtras. Все они специализируются на распространении программного обеспечения, и у них можно приобрести приложение Visual Basic .NET и все сопутствующие ему надстройки.

Более подробные сведения вы сможете получить по следующим адресам:

- ✓ Programmer's Paradise, 1157 Shrewsberry Avenue, Shrewsberry, N.I07702; Tel: 732-389-8950; Fax: 732-389-0010; [www.programmersparadise.com](http://www.programmersparadise.com)
- ✓ Provantage, 7249 Whipple Avenue NW, North Can/on, OH 30339; Tel: 330-494-8715; Fax: 330-494-5260; [www.provantage.com](http://www.provantage.com)
- ✓ VBxtras, 1905 Power Ferry, Suite 100, Atlanta, GA30339; Tel: 770-952-6356; Fax: 770-952-6388; [www.vbxtras.com](http://www.vbxtras.com)

## *Комбинируйте Visual Basic .NET с C# и другими языками программирования*

Одним из самых больших преимуществ среды .NET является то обстоятельство, что вы можете писать одну и ту же программу, используя сразу несколько языков программирования. Следовательно, можно объединить усилия нескольких программистов, пишущих программы на разных языках, и использовать среду .NET для создания одной большой программы.

Если вы уже освоили другой язык программирования или просто хотите повысить свою квалификацию, попробуйте написать программу сразу на нескольких языках. Например, язык Visual Basic .NET можно использовать для создания пользовательского интерфейса, а C# — для написания работающих кодов программы, которые будут проводить какие-то вычисления.

Возможно, вас испугает перспектива применения нескольких языков одновременно, но в этом есть свои неоспоримые преимущества. Допустим, если вы уже имеете работающую программу, написанную на языке C или COBOL, можете скопировать ее в среду .NET, а затем использовать язык Visual Basic .NET для создания подходящего пользовательского интерфейса.

Предоставляя возможность комбинировать Visual Basic .NET с другими языками программирования, компания Microsoft надеется, что вы сумеете извлечь из этого максимальную выгоду, вплоть до того, что научитесь создавать такие программные продукты, которые будут конкурировать даже с программным обеспечением, выпускаемым самой Microsoft.

## Принимайте участие в разработке открытых проектов Visual Basic .NET

Одним из наиболее эффективных способов достижения мастерства в использовании какого-либо языка программирования является непосредственное изучение реально существующих и действующих программ и участие в их разработке. Хотя работающие программы Visual Basic .NET можно найти в сети Internet, но лучше самому принять участие в одном из открытых проектов, в котором такие программы создаются.

Открытые проекты содержат в себе работающие коды, которые каждый желающий может изучить и проверить в действии. При этом основная идея заключается в том, что чем большее количество людей принимает участие в проекте, тем более совершенным он становится.

Чтобы найти открытый проект, в котором используется Visual Basic .NET, посетите Web-страницу SourceForge (<http://sourceforge.net>). Там вы сможете найти не только сам проект, к которому можно присоединиться, но и выбрать категорию такового (например, открытая видеоигра, мультимедиапрограмма или коммуникационные утилиты). Какой бы проект вы ни выбрали, им наверняка уже кто-нибудь занимался, так что вы сможете как изучать работающие коды, так и добавлять к ним что-то свое.

## Приобретите программу для создания справочной системы

Любая хорошая Windows-программа имеет работающую в режиме on-line справочную систему, позволяющую неопытным пользователям быстро находить ответы на свои вопросы с помощью указателей и гиперссылок, а также избавляющую от утомительного и неэффективного штудирования многостраничных справочных пособий. Если вы намерены создать с помощью Visual Basic .NET серьезную программу, то должны позаботиться и об аналогичной справочной системе.

В создании собственной справочной системы нет ничего сложного, за исключением того, что этот процесс занимает много времени и превращается в рутинную работу. К счастью, можно приобрести специальную программу, значительно упрощающую эту задачу. Двумя наиболее популярными программами, создающими справочные системы для других программ, являются RoboHelp и ForeHelp. Они делают процесс создания экранных подсказок таким же простым, как процесс набора обычных текстовых документов. Каждая созданная с помощью этих программ справочная система может быть тут же протестирована и проверена на предмет того, как она будет выглядеть после присоединения к вашей программе.

В мире современного программного обеспечения наличие эффективной справочной системы является необходимым условием для продвижения какого-либо продукта на рынке, поэтому уделите данному вопросу должное внимание.

Для получения более подробной информации о программах RoboHelp и ForeHelp обращайтесь:

- ✓ floboHdp, eHelp Corporation. 7777 Fay Avenue, La Jolla, CA 92037; Tel: 858-459-6365; Fax: 858-459-6366; [www.ehelp.com](http://www.ehelp.com)
- ✓ ForeHelp, Inc., 4710 North Mesa Boulevard, Boulder, CO 80305; Tel: 303-499-9181; Fax: 303-494-5446; [www.ff.com](http://www.ff.com)

# *Создавайте инсталляционные программы*

После того как программа создана, встает вопрос о ее распространении. Любую программу можно просто скопировать на дискету (или дискеты) в надежде на то, что будущие пользователи смогут ее правильно установить на своих компьютерах. Но лучше все же доверить этот процесс специальным инсталляционным программам.

Инсталляционная программа берет на себя функции проводника и шаг за шагом контролирует процесс копирования программного продукта на диск другого компьютера. Используя специальную инсталляционную программу, можно не только оказать услугу будущим покупателям вашего продукта, но и сопровождать его процесс установки отображением логотипа своей компании и рекламной информации, различными визуальными и звуковыми эффектами.

Помочь создать инсталляционные программы для любых программ, написанных на языке Visual Basic .NET, могут такие приложения, как InstallShield и Wise. Более подробную информацию о них можно получить по адресам:

- ✓ Wise Installer. Wise Solutions, 5880 North Can/on Center Road, Suite 450, Canton, MI 48187; Tel: 734-456-2100; Fax: 734-456-2345; [www.wisesolutions.com](http://www.wisesolutions.com)
- ✓ InstallShield, InstallShield Corporation, 900 National Parkway, Suite 125 Schaumburg, IL 60173-5108; Tel: 847-240-9111; Fax: 847-619-0788; [www.installshield.com](http://www.installshield.com)

## *Пишите Basic-программы для Macintosh, Linux, Palm OS и PocketPC*

Первая версия Visual Basic, выпущенная компанией Microsoft в 1991 году, имела большой успех у программистов, поскольку позволяла быстрее и более простым способом создавать качественные приложения. В связи с этим представители компании Microsoft пообещали в скором времени адаптировать язык Visual Basic для других операционных систем.

После выпуска приложения Visual Basic для MS-DOS, которому, правда, была уготована весьма недолгая жизнь, компания Microsoft забыла о своем обещании, сделав исключение лишь для платформы Windows CE (известна также как операционная система PocketPC). Если Microsoft когда-нибудь адаптирует среду .NET и для других операционных систем, в частности для Linux и Macintosh, вы, надо полагать, сможете писать на Visual Basic .NET программы, которые будут одновременно работать на компьютерах с разными операционными системами.

Но пока это лишь предположение, не имеющее практического воплощения, вам придется довольствоваться написанием на Visual Basic .NET программ, которые работают только в среде Windows. Если же вы захотите написать на данном языке программу для Macintosh, ничего у вас из этого не выйдет.

Однако вы можете использовать клон Visual Basic для Macintosh, называемый REALBasic. Как и Visual Basic, REALBasic позволяет вначале нарисовать объекты пользовательского интерфейса и определить их свойства, а затем написать коды программы, которые будут обрабатывать данные.

REALBasic во многом похож на Visual Basic. Он даже может конвертировать коды программ, написанных с помощью более ранних версий языка Visual Basic, в коды REALBasic. Более того, на языке REALBasic можно написать программы для Macintosh, а затем перекомпилировать их для запуска в среде Windows. Другими словами, одни и те же коды можно использовать для создания программ для Windows и для Macintosh.

Подобные клоны Visual Basic имеются и для систем Palm OS и PocketPC, используемых в карманных компьютерах. С помощью языка NSBasic писать программы можно с применением ноутбука или настольного компьютера, а затем их можно протестировать, используя программу эмуляции Palm OS или PocketPC. Это позволяет убедиться, что после загрузки в карманный компьютер новые программы будут работать без ошибок.

В заключение отметим, что существует еще язык XBasic, являющийся открытым BASIC-компилятором, с помощью которого можно создавать программы для Windows и Linux. Вы можете не только изучать коды, делающие XBasic функционирующим компилятором, но и использовать их для написания своих программ, работающих сразу под Windows и Linux.

Еще несколько адресов, по которым мы советовали бы вам обратиться:

- ✓ REALBasic, *Real Software, Inc., PMB 220, 3300 Bee Caves Road, Suite 650, Austin, TX 78746; Tel: 512-263-1233; Fax: 512-263-1441; [www.realbasic.com](http://www.realbasic.com)*
- ✓ NSBasic, *NS Basic Corporation, 77 Hill Crescent, Toronto, Canada M1M 1J3; Tel: 416-264-5999; Fax: 416-264-5888; [www.nsbasic.com](http://www.nsbasic.com)*
- ✓ XBasic, [www.xbasic.org](http://www.xbasic.org)



# Советы по использованию интерфейса Visual Basic .NET

*В этой главе*

- Использование окна **Properties**
- > Окно Solution Explorer
- Настройка панели Toolbox
- Использование списка Class View

**К**оличество одновременно отображаемой компьютером информации ограничено размерами монитора, поэтому Visual Basic .NET использует различные окна для демонстрации именно тех данных, которые необходимы в каждый отдельный момент времени. Поскольку вы уже знакомы с этими окнами и в общих чертах знаете, для чего каждое из них предназначено, остается только дать вам несколько советов относительно того, как лучше их применять и как сделать процесс программирования на Visual Basic .NET максимально легким и приятным занятием.

## *использование окна Properties*

В окне Properties (F4) можно просмотреть и изменить свойства объектов пользовательского интерфейса, которые определяют способ отображения таковых, т.е. объектов, на экране и реакцию на некоторые действия пользователей. Чтобы помочь вам быстрее найти нужные свойства, информация в окне Properties может быть отсортирована либо в алфавитном порядке, либо по категориям.

Чтобы отсортировать названия свойств по алфавиту, щелкните в окне Properties на кнопке Alphabetic (По алфавиту), а для того чтобы разбить свойства по категориям, щелкните в окне Properties на кнопке Categorized (По категориям).

Окно Properties может одновременно отображать свойства лишь одного объекта. Если вам нужно просмотреть свойства другого объекта, щелкните на нем или выберите его название в списке Object. Чтобы открыть список с названиями всех объектов созданного вами пользовательского интерфейса, нажмите кнопку со стрелкой списка Object, затем найдите название объекта, свойства которого нужно изменить, и щелкните на нем. Вы увидите значения всех свойств данного объекта в окне Properties.

## *Окно Solution Explorer*

В окне Solution Explorer отображается список всех файлов, из которых состоит открытый на данный момент проект. С помощью этого окна можно открывать файлы для редактирования кодов или изменения элементов пользовательского интерфейса, исключать файлы из проекта или удалять их физически. Для этого достаточно щелкнуть правой кнопкой мыши на названии нужного файла и в открывшемся меню выбрать одну из команд:

- ✓ View Code — для редактирования кодов Basic;
- ✓ View Designer — для изменения пользовательского интерфейса;
- ✓ Exclude From Project — для исключения файла из проекта без его физического удаления;
- ✓ Delete — для физического удаления файла.

## *Настройка панели Toolbox*

На вкладке Windows **Forms** панели Toolbox (Ctrl+Alt+X) представлены все типы объектов, которые могут применяться при создании пользовательского интерфейса (это кнопки, переключатели, текстовые поля и т.п.). Перечень объектов очень большой, поэтому для удобства вы можете создать отдельную вкладку и поместить на ней только те из них, которые используются наиболее часто.

## *Создание новой вкладки на панели Toolbox*

При необходимости создать новую вкладку на панели **Toolbox** вы должны выполнить следующие действия.

1. Щелкните правой кнопкой мыши на любой вкладке панели **Toolbox** (например, Windows **Forms**).  
Откроется контекстное меню.
2. Выберите команду **Add Tab (Создать вкладку)**.  
На панели **Toolbox** появится новая пустая вкладка.
3. Укажите имя для новой вкладки и нажмите клавишу <Enter>.  
На панели **Toolbox** будет создана новая пустая вкладка с указанным вами именем.

## *Добавление объектов в новую вкладку панели Toolbox*

Для того чтобы добавить объекты в созданную вами вкладку панели **Toolbox**, необходимо выполнить такие действия.

1. Щелкните правой кнопкой мыши на названии нужного объекта.  
Откроется контекстное меню.
2. Выберите команду **Copy**.
3. Щелкните на вкладке, к которой хотите добавить выбранный на первом таге объект.
4. Щелкните правой кнопкой мыши и задайте команду **Paste**.  
Название скопированного объекта появится на этой вкладке.

# Удаление вкладок панели Toolbox

А вот как можно удалить ставшую ненужной вкладку из панели **Toolbox**.

1. Щелкните правой кнопкой мыши на вкладке, подлежащий удалению.

Откроется контекстное меню.

2. Выберите команду **Delete Tab (Удалить вкладку)**.

Если эта вкладка содержит названия каких-то объектов, откроется диалоговое окно с вопросом, действительно ли вы хотите удалить ее.

3. Щелкните на кнопке **Yes**.

Visual Basic .NET удалит вкладку из панели **Toolbox**.

## использование окна *Class View*

В окне **Class View** (Ctrl+Alt+C) содержится список всех переменных, методов и свойств, используемых в открытом на данный момент файле классов (рис. 34.1). С помощью этого окна можно быстро переходить к нужному фрагменту кодов указанного файла.

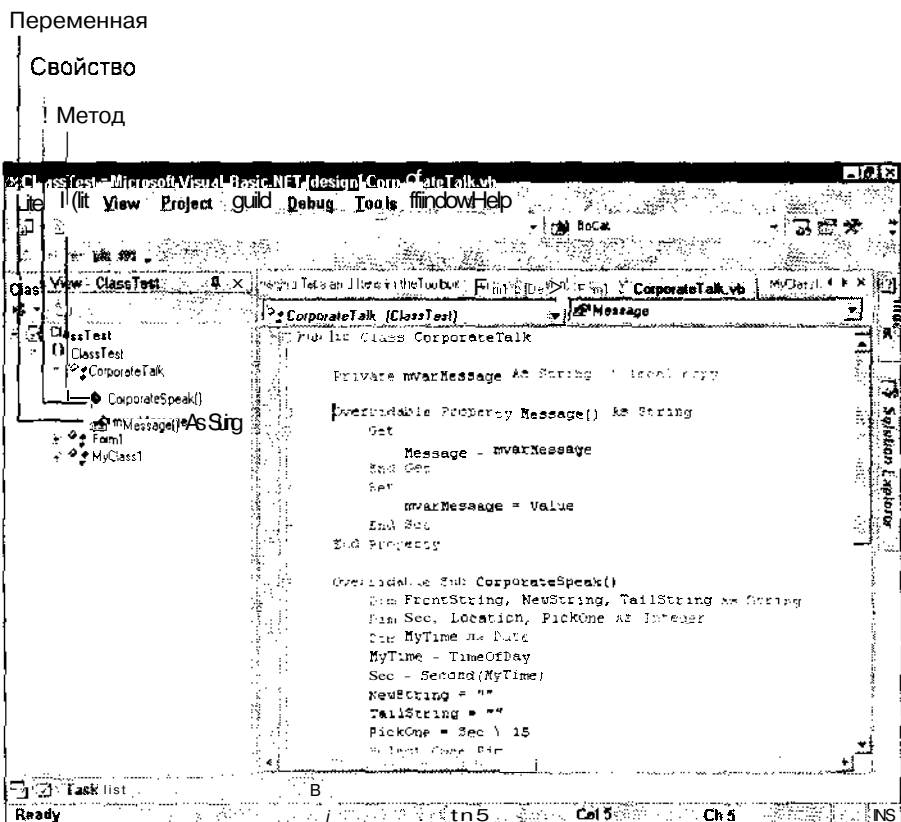


Рис. 34.1. Использование окна *Class View*

Чтобы найти в окне кодов нужную переменную, метод или свойство, выполните действия, перечисленные **ниже**.

1. Выберите команду View⇒Class View или нажмите комбинацию клавиш <Ctrl+Alt+C>, в результате чего будет открыто окно Class View.
2. Дважды щелкните на названии переменной, метода или свойства, т.е. того элемента файла классов, который должен быть изменен или работу которого вы хотите проверить.

Visual Basic .NET перейдет к тому фрагменту кодов, который определяет работу выбранного элемента.

# Предметный указатель

## А

Аргументы, 275, 286  
    значения, 278  
    ссылки, 278

## В

Вкладка  
    Code, 35  
    Design, 35  
Возведение в степень, 186  
Вызов процедуры, 276  
    функций, 284

## Д

Данные, 167  
Диалоговые окна, 132; 139  
    Color, 147  
    Font, 148  
    OpenFile, 144  
    PageSetup, 151  
    Print, 150  
    SaveFile, 147  
    стандартные, 143  
Диапазоны, 201

## З

Значения переменных, 168

## И

Индекс, 213  
Инсталляционная программа, 319  
Инструмент  
    Button, 91  
    CheckBox, 92  
    ColorDialog, 147  
    ComboBox, 106  
    ContextMenu, 136  
    FontDialog, 148  
    GroupBox, 94

Label, 98  
ListBox, 106  
MainMenu, 122  
OpenFileDialog, 144  
PageSetupDialog, 151  
PrintDialog, 150  
PrintDocument, 150; 151  
RadioButton, 92  
SaveFileDialog, 147  
TextBox, 98

## К

Категория  
    Appearance, 43  
    Behavior, 43  
    Data, 43  
    Design, 43  
    Focus, 43  
    Layout, 43  
Ключевые слова, 47  
    выделение цветом, 161  
Кнопки, 91  
Коды, 27; 47  
    временная деактивация, 211  
    ANSI, 204  
    ASCII, 204  
Коллекция, 220  
Команды, 25; 27; 47  
    ветвления, 293  
    последовательные, 293  
    циклов, 293  
Комбинации клавиш, 126  
Комментарии, 209  
Конкатенация строк, 187  
Константы  
    глобальные, 208  
    локальные, 207  
    модуля, 208

## Л

Логические значения, 235

## **М**

Массив, 213

многомерный, 215

одномерный, 215

Меню

Edit, 121

File, 121

Help, 122

Tools, 122

Window, 121

выпадающие, 137

динамически расширяемые, 133

контекстное, 136

Методы, 135; 301

Модуль класса, 298

## **Н**

Надпись, 97

Наследование кодов, 305

форм, 306

## **О**

Область видимости переменных, 174

Общие процедуры

присвоение имен, 272

Объект

DomainUpDown, 178

Label, 97

TextBox, 97

Объектно-ориентированное программирование, 294

Объекты, 39; 63

выделение, 74

выделение серым, 76

выравнивание, 114

выравнивание по центру, 116

закрепление, 72

изменение свойств, 66

копирование, 73

определение размеров, 70

определение свойств, 42

отмена отображения, 77

прикрепление к стороне формы, 71

присвоение имен, 67

размещение на экране, 70

рисование, 40; 64

удаление, 73

фиксирование местоположения, 116

Объекты, 27; 297

Объявление переменных, 299

Окно

Add New Item, 79; 269

Font, 98

Inheritance Picker, 306

New Project, 32

Project Location, 33

Properties, 35; 321

Property Pages, 88

Solution Explorer, 35; 321

Start Page, 31; 34

String Collection Editor, 108

Watch, 230

закрепленное, 36

плавающее, 36

скрытое, 36

Окончание выполнения программы, 52

Оператор выбора, 243

Операторы, 183

арифметические, 183

логические, 188

приоритет, 193

сравнения, 190

Открывающиеся меню, 34

Открытые проекты, 318

Отображение текста, 68

Ошибки, 223

логические, 224

рабочие, 224

синтаксические, 224

## **П**

Панели инструментов, 34

Панель Toolbox, 35; 39

Перегрузка, 308

кодов, 295

Переключатели, 92; 178

Переменные, 167

глобальные, 176

- объявление. 168
- определение области видимости, 174
- присвоение значений, 172
- присвоение имен, 171
- Пиктограммы, 139
- Подменю . 131
- Поиск текста по шаблону, 199
- Поле со списком, 178
- Поля со списком, 105; 106
- Пользовательский интерфейс, 26; 39; 48; 63; 164; 177
  - Visual Basic .NET, 31
- Принцип инкапсуляции, 294
  - наследования, 294
- Программа, 47
- Проект, 31; 32
- Процедуры
  - общие, 267
  - обработки событий, 155; 267
  - создание, 50
  - управления событиями, 48

## Р

- Разделительные линии, 125
- Раскрывающиеся меню, 121
- Расширенное текстовое поле, 178
- Редактор кодов, 155
  - разделение окна, 159
- Режим
  - кодов, 50
  - конструктора, 49

## С

- Свойства, 27; 168
  - изменение, 43
- Свойство
  - Anchor, 72
  - BackColor, 81; 95; 99
  - BackgroundImage, 82
  - BorderStyle, 100; 141
  - CheckAlign, 93
  - Checked, 93; 128; 178
  - ContextMenu, 137
  - Dock, 71

- Document, 150; 152
- DocumentName, 151
- DropDownStile, 108
- Enabled, 76; 129
- Filename, 146; 147
- Filter, 146
- FilterIndex, 146
- Font, 69; 96; 98; 111; 149
- ForeColor, 99
- FormBorderStyle, 84
- GridSize Height, 115
- GridSize Width, 115
- Height, 70
- Items, 108
- Location, 71
- Locked, 118
- MaxLenght, 103
- MinimizeBox. 85
- Multiline, 101
- Name, 43; 66; 80; 125
- PassChar, 103
- RadioCheck, 129
- ScrollBars, 102
- SelectedIndices, 180
- SelectedItem, 180
- SelectionMode, 180
- Shortcut, 127
- ShowShortcut, 128
- Size, 70
- Sorted, 110
- StartPosition, 86
- TabIndex, 74
- TabStop, 76
- Text, 66; 96; 177
- Text Align, 94; 100
- Value, 179
- Visible. 77; 130; 134
- Width, 70
- WindowState, 85
- Wordwrap, 101
- X. 70
- Y. 70

- Сворачивание и разворачивание кодов, 156

- Символы, 167
  - групповые, 200

- объявления типов, 170
- унифицированные, 170
- Событие, 27
  - Click, 49
- События, 155; 163
  - генерируемые системой, 157
  - инициируемые пользователем, 157
- Списки, 105
- Список: Class Name, 162
- Method Name, 162
  - аргументов, 270
  - объектов, 50
  - системы, 318
- Среда .NET, 28
- Строки, 167; 195
- Структура, 217
- Структурное программирование, 292
- Структуры данных, 213

## Т

- Текстовое поле, 97; 178
- Типы данных, 169

## Ф

- Файл классов, 298

- Файлы модулей, 267
- Файлы модулей и классов, 48
- Фильтры, 145
- Флажки, 92; 178
- Форма, 27
- Формы, 35; 39; 63; 79
  - открытие, 89
  - отображение, 80
  - отображение на экране, 84
  - присвоение имен, 68
  - присвоение имени, 79
  - создание, 64; 79
  - удаление, 87
- Функции, 281

## Ц

- Циклы, 249
- Циклы вложенные, 261

## Ч

- Числа, 167

## Э

- Экземпляры классов, 302



*Научно-популярное издание*

Уоллес Вонг

## Visual Basic .NET для "чайников"

*В издании использованы карикатуры  
американского художника Рича Теннанта*

Верстка *О.В. Мицутина*  
Художественный редактор *Е.П. Дынник*  
Корректор *О.В. Мицутина*

Издательский дом "Вильямс",  
101509, Москва, ул. Лесная, д. 43, стр. 336.  
Изд. лип. ЛР № 090230 от 23.06.99  
Госкомитета РФ по печати.

Подписано в печать 24.05.02. Формат 70х100/16.  
Гарнитура Times. Печать офсетная.  
Усл. печ. л. 21,15. Уч.-изд. л. 16,40.  
Тираж 4000 экз. Заказ № 530.

Отпечатано с диапозитивов в ФГУП "Печатный двор"  
Министерства РФ по делам печати,  
телерадиовещания и средств массовых  
коммуникаций.  
197110, Санкт-Петербург, Чкаловский пр., 15.



COMPUTER  
BOOK  
SERIES

# Visual Basic .NET для "чайников"



СЕРИЯ  
КОМПЬЮТЕРНЫХ  
КНИГ ОТ  
ИЗДАТЕЛЬСТВА

Шпаргалка

Объекты пользовательского интерфейса которые сохраняют данные, полученные от пользователя

Объект	Тип данных	Свойство объекта
Checkbox	Boolean	Checked
Checked list box (выделенный элемент)	String	Text
Checked list box (выбранный элемент)	Boolean	Checked
Color dialog box	System.Drawing.Color	Color
Combo box (единичные элементы)	String	Text
Combo box (множественные элементы)	Collection	SelectedItem.Item
DateTimePicker	Long	Value
DomainUpDown	String	Text
Font dialog box	System.Drawing.Color	Color
	Boolean	Font.Bold
	Boolean	Font.Italic
	Boolean	Font.StrikeThru
	Boolean	Font.Underline
	String	Font.Name
	Single	Font.Size
List box (единичные элементы)	String	Text
List box (множественные элементы)	Collection	SelectedItem.Item
NumericUpDown	Decimal	Value
Open dialog box	String	Filename
Radio button	Boolean	Checked
RichTextBox	String	Text
Save As dialog box	String	Filename
Scroll bars	Integer	Value
Text box	String	Text
TrackBar	Integer	Value

Символы объявления типов данных

ТИП данных	Символ	Пример	Ему соответствует
Decimal	@	Dim Дебет@	Dim Дебет As Decimal
Double	#	Dim Дробь#	Dim Дробь As Double
Integer	%	Dim Счет%	Dim Счет As Integer
Long	&	Dim Макс&	Dim Макс As Long
Single	!	Dim Мин!	Dim Мин As Single
String	\$	Dim Имя\$	Dim Имя As String



COMPUTER  
BOOK  
SERIES

# Visual Basic.NET

## для чайников



СЕРИЯ  
КОМПЬЮТЕРНОГО  
КНИЖНОГО  
ДИАЛЕКТА

Мини-книжка

### Клавиши быстрого доступа

Клавиши	Выполняемые действия
<Ctrl+B>	Создать точку останова
<Ctrl+N>	Создать новый файл
<Ctrl+O>	Открыть файл
<Ctrl+P>	Печать
<Ctrl+S>	Сохранить выделенные элементы
<Ctrl+Shift+S>	Сохранить все
<F4>	Отобразить окно Properties
<F7>	Показать коды выделенного в данный момент объекта
<Shift+F7>	Показать объект, к которому относится данная процедура обработки событий
<F10>	Пошаговое выполнение программы с перешагиванием через процедуры
<F11>	Пошаговое выполнение программы
<Ctrl+Alt+>	Отображение окна Solution Explorer
<Ctrl+Alt+X>	Отображение панели Toolbox
<Ctrl+Shift+A>	Создать новый элемент
<Ctrl+F5>	Запуск программы без ее отладки
<F5>	Запуск созданной вами программы

### Соглашения о присвоении имен объектам пользовательского интерфейса

Объект	Префикс	Пример
Button	btn	btnКнопка
Check box	chk	chkФлажок
Combo box	cbo	cboПолеСоСписком
Form	frm	frmФорма
Horizontal scroll bar	hsb	hsbГорПолосаПрокр
Image	img	imgКартинка
Label	lbl	lblНадпись
List box	lst	lstСписок
Menu	mnu	mnuМеню
Radio button	rad	radПереключатель
Picture box	pic	picРисунок
Text box	txt	txtТекстовоеПоле
Vertical scroll bar	vsb	vsbВертПолосаПрокр

### Объявление и вызов функций

```
Public Function ИмяФункции  
(СписокАргументов) As  
ТипДанных  
    ИмяФункции = Значение  
End Function  
или  
Public Function ИмяФункции  
(СписокАргументов) As  
ТипДанных  
    Return Значение  
End Function  
ИмяПеременной = ИмяФункции  
(СписокАргументов)
```

### Объявление и вызов процедур

```
Public Sub ИмяПроцедуры:  
(СписокАргументов)  
    Список инструкций  
End Sub  
ИмяПроцедуры (СписокАргументов)  
или  
Call ИмяПроцедуры(СписокАргументов)
```

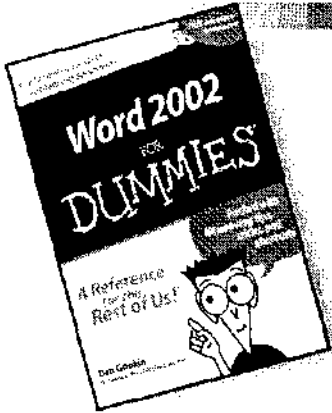


## Access 2002 для "чайников"

В продаже

**М**icrosoft Access — самая популярная система управления базами данных для персональных компьютеров. В книге охвачено множество вопросов, касающихся проектирования, поддержки и использования баз данных Access 2002. Начав изучение с самого понятия баз данных в целом и Access 2002 в частности, вы получите представление о концепции баз данных, узнаете о возможностях Access 2002, а затем научитесь создавать сложные таблицы, запросы, отчеты и формы. Вы узнаете, как размещать данные в Internet или локальной сети вашей компании, как решать сложные задачи по выборке информации с помощью запросов Access, и многое другое! Джон Кауфельд, автор многих книг по Access, эксперт по базам данных, непринужденно и доходчиво объясняет возможности Access 2002.

Книга рассчитана на пользователей с различным уровнем подготовки.

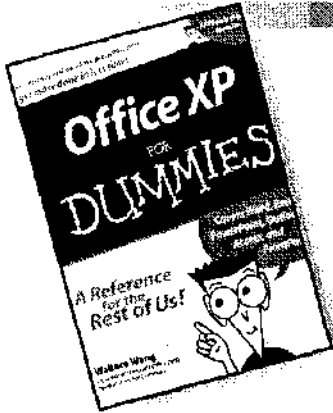


## Word 2002 для "чайников"<sup>1</sup>

Плановая дата выхода —  
2-й квартал 2002 г.

**Н**овая книга Дэна Гукина поможет вам поближе познакомиться с Word 2002 — новой версией самого популярного в мире текстового процессора. Вы узнаете, как создавать и редактировать документы, вставлять в них таблицы, картинки и сноски, как конвертировать файлы из одного формата в другой и распечатывать готовый материал. Кроме того, в книге описаны новые возможности Word, например опыт в распознавании речи.

Книга написана увлекательно и интересно, насыщена фактическим материалом и рассчитана на пользователей всех уровней подготовки.



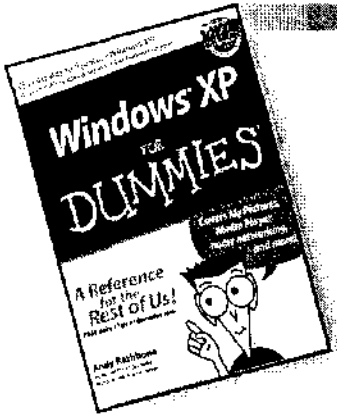
## Office XP для "чайников"

Плановая дата выхода —  
2-й квартал 2002 г.

**Д**анная книга — незаменимое руководство для всех, кто хочет освоить передовые компьютерные технологии в области делопроизводства. Эта книга поможет вам освоить все программы самого популярного в мире пакета офисных программ Microsoft Office XP. Хотя она не сделает вас специалистом по использованию какой-либо одной программы, в ней вы узнаете основные принципы работы каждой программы данного пакета.

Автор книги Уоллес Вонг с юмором и без излишнего углубления в технические подробности дает самые необходимые навыки для эффективной работы с Office XP.

Книга рассчитана на пользователей с различным уровнем подготовки.



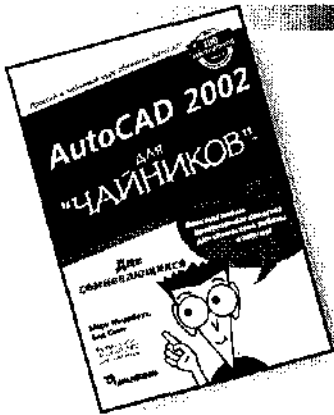
## Windows XP для "чайников"

Б продаже

**П**еред вами замечательная книга о самой последней редакции операционной системы от компании Microsoft, предназначенной для домашних пользователей, — Windows XP.

Написанная известным автором Энди Ратбоном, эта книга поможет вам сделать первые шаги в освоении новой операционной системы. Из материала книги вы узнаете, что представляет собой Windows XP и на что она способна. Здесь вы встретите описания компонентов Windows XP среди которых Проигрыватель Windows Media, Internet Explorer 6.0, Outlook Express 6.0, Мастер новых подключений и др.

Книга рассчитана на пользователей с различным уровнем подготовки. Легкий и доступный стиль изложения поможет даже начинающим быстро освоить Windows XP.



## AutoCAD 2002 для “чайников”

В продаже

Эта книга будет полезна тем, кто работает или только начинает работать с одной из наиболее популярных и мощнейших систем автоматизированного проектирования — программой AutoCAD 2002. На протяжении своего развития AutoCAD становится все более изощренной и сложной в применении, в соответствии с совершенствованием процессов проектирования и черчения. В этой книге вы найдете определения фундаментальных основ и практические, пошаговые руководства выполнения специфических задач автоматизированного проектирования — от установки параметров нового чертежа, до особенностей работы с компоновками в пространстве листа и тайнств процесса качественной печати.