Виктор Зиборов

Visual Basic 2010
НА ПРИМЕРАХ

Санкт-Петербург
«БХВ-Петербург»
2010

УДК 681.3.06
ББК 32.973.26-018.2
З-59

Зиборов В. В.

З-59 Visual Basic 2010 на примерах. — СПб.: БХВ-Петербург, 2010. — 336 с.: ил. + CD-ROM

ISBN 978-5-9775-0402-7

Рассмотрено более сотни типичных примеров, встречающихся в практике реального программирования для платформы .NET Framework в среде Microsoft Visual Basic 2010: работа с экранной формой и элементами управления, обработка событий мыши и клавиатуры, чтение/запись текстовых и бинарных файлов, редактирование графических данных, управление буфером обмена, ввод/вывод табличных данных, решение системы уравнений, использование функций MS Word, MS Excel и AutoCAD, обработка баз данных с использованием технологии ADO.NET, разработка веб-приложений, создание веб-служб и многое другое. Материал располагается по принципу от простого к сложному, что позволяет использовать книгу одновременно как справочник для опытных и как пособие для начинающих программистов. Компакт-диск содержит исходные коды примеров, приведенных в книге.

Для программистов

УДК 681.3.06
ББК 32.973.26-018.2

Группа подготовки издания:

Главный редактор	*Екатерина Кондукова*
Зам. главного редактора	*Игорь Шишигин*
Зав. редакцией	*Григорий Добин*
Редактор	*Анна Кузьмина*
Компьютерная верстка	*Ольги Сергиенко*
Корректор	*Зинаида Дмитриева*
Дизайн серии	*Игоря Цырульникова*
Оформление обложки	*Елены Беляевой*
Зав. производством	*Николай Тверских*

Лицензия ИД № 02429 от 24.07.00. Подписано в печать 31.03.10.
Формат 70×100 1/16. Печать офсетная. Усл. печ. л. 27,09.
Тираж 2000 экз. Заказ №
"БХВ-Петербург", 190005, Санкт-Петербург, Измайловский пр., 29.

Санитарно-эпидемиологическое заключение на продукцию
№ 77.99.60.953.Д.005770.05.09 от 26.05.2009 г. выдано Федеральной службой
по надзору в сфере защиты прав потребителей и благополучия человека.

Отпечатано с готовых диапозитивов
в ГУП "Типография "Наука"
199034, Санкт-Петербург, 9 линия, 12

ISBN 978-5-9775-0402-7

Оглавление

Введение 9

Глава 1. Простейшие программы с экранной формой и элементами управления 11

Пример 1. Форма, кнопка, метка и диалоговое окно 11
Пример 2. Событие *MouseHover* 16
Пример 3. Ввод и вывод в консольном приложении 20
Пример 4. Проверка типа данных: функция *IsNumeric* 23
Пример 5. Ввод данных через текстовое поле *TextBox* 26
Пример 6. Ввод пароля в текстовое поле и изменение шрифта 29
Пример 7. Управление стилем шрифта с помощью элемента управления *CheckBox* 31
Пример 8. Побитовый оператор *Xor* 33
Пример 9. Вкладки *TabControl* и переключатели *RadioButton* 35
Пример 10. Свойство *Visible* и всплывающая подсказка *ToolTip* 38
Пример 11. Калькулятор на основе использования комбинированного списка *ComboBox* 40
Пример 12. Ссылка на другие ресурсы *LinkLabel* 43
Пример 13. Греческие буквы, математические операторы. Символы Unicode 45

Глава 2. Инициирование и обработка событий мыши и клавиатуры 51

Пример 14. Инициирование события в создаваемом классе 51
Пример 15. Создание элемента управления *Button* программным способом и подключение события для него 53
Пример 16. Координаты курсора мыши относительно экрана и элемента управления 55
Пример 17. Ассоциация нескольких элементов управления с одним событием с помощью *Handles* 57
Пример 18. Калькулятор 60

Пример 19. Обработка событий клавиатуры 64
Пример 20. Контроль вводимых пользователем числовых данных при обработке события нажатия клавиши 67

Глава 3. Чтение, запись текстовых и бинарных файлов, текстовый редактор 69

Пример 21. Чтение/запись текстового файла в кодировке Unicode. Блоки *Try...Catch* 69
Пример 22. Чтение/запись текстового файла в кодировке Windows 1251 73
Пример 23. Простой текстовый редактор. Открытие и сохранение файла. Событие формы *Closing* 74
Пример 24. Простой RTF-редактор 80
Пример 25. Печать текстового документа 84
Пример 26. Чтение/запись бинарных файлов с использованием потока данных 89

Глава 4. Редактирование графических данных 93

Пример 27. Простейший вывод отображения графического файла в форму 93
Пример 28. Рисование в форме указателем мыши 96
Пример 29. Рисование в форме графических примитивов (фигур) 99
Пример 30. Выбор цвета с использованием *ListBox* 102
Пример 31. Печать графических примитивов 105
Пример 32. Печать BMP-файла 106

Глава 5. Управление буфером обмена с данными в текстовом и графическом форматах 109

Пример 33. Буфер обмена с данными в текстовом формате 109
Пример 34. Элемент управления *PictureBox*. Буфер обмена с растровыми данными 111
Пример 35. Имитация нажатия комбинации клавиш <Alt>+<PrintScreen> 114
Пример 36. Запись содержимого буфера обмена в BMP-файл 116
Пример 37. Использование таймера *Timer* 117
Пример 38. Запись в файлы текущих состояний экрана с интервалом 5 секунд 119

Глава 6. Ввод и вывод табличных данных. Решение системы уравнений 123

Пример 39. Формирование таблицы. Функция *String.Format* 123
Пример 40. Форматирование *Double*-переменных в виде таблицы. Вывод таблицы на печать. Поток *StringReader* 126
Пример 41. Вывод таблицы в Internet Explorer 130
Пример 42. Формирование таблицы с помощью элемента управления *DataGridView* 134
Пример 43. Табличный ввод данных. *DataGridView*. *DataTable*. *DataSet*. Файл XML 136
Пример 44. Решение системы линейных уравнений. Ввод коэффициентов через *DataGridView* 140

Глава 7. Элемент управления *WebBrowser* 145

Пример 45. Отображение HTML-таблиц 145
Пример 46. Отображение Flash-файлов 147
Пример 47. Отображение Web-страницы и ее HTML-кода 148

Глава 8. Использование функций MS Word, MS Excel, AutoCAD 151

Пример 48. Проверка правописания в текстовом поле с помощью обращения к MS Word 151
Пример 49. Вывод таблицы средствами MS Word 154
Пример 50. Обращение к функциям MS Excel из Visual Basic 2010 156
Пример 51. Использование финансовой функции MS Excel 158
Пример 52. Решение системы уравнений с помощью функций MS Excel 161
Пример 53. Построение диаграммы средствами MS Excel 165
Пример 54. Управление функциями AutoCAD из программы на VB2010 167

Глава 9. Обработка баз данных с использованием технологии ADO.NET 171

Пример 55. Создание базы данных SQL Server 171
Пример 56. Отображение таблицы базы данных SQL Server в экранной форме 173
Пример 57. Создание базы данных в среде MS Access 175
Пример 58. Редактирование таблицы базы данных MS Access в среде Visual Studio без написания программного кода 176
Пример 59. Отображение таблицы базы данных MS Access в экранной форме 178
Пример 60. Чтение всех записей из таблицы БД MS Access на консоль с помощью объектов *Command* и *DataReader* 180
Пример 61. Создание БД MS Access в программном коде 182
Пример 62. Запись структуры таблицы в пустую БД MS Access. Программная реализация подключения к БД 185
Пример 63. Добавление записей в таблицу базы данных MS Access 187
Пример 64. Чтение всех записей из таблицы БД с помощью объектов *Command*, *DataReader* и элемента управления *DataGridView* 189
Пример 65. Чтение данных из БД в сетку данных *DataGridView* с использованием объектов *Command*, *Adapter* и *DataSet* 191
Пример 66. Обновление записей в таблице базы данных MS Access 194
Пример 67. Удаление записей из таблицы БД с использованием SQL-запроса и объекта *Command* 197

Глава 10. Другие задачи, решаемые с помощью *Windows Forms Application* 199

Пример 68. Проверка вводимых данных с помощью регулярных выражений 199
Пример 69. Управление прозрачностью формы 202
Пример 70. Время по Гринвичу в полупрозрачной форме 203

Пример 71. Создание ссылки на процесс, работающий в фоновом режиме, в форме значка в области уведомлений 207
Пример 72. Нестандартная форма. Перемещение формы мышью 210
Пример 73. Проигрыватель Windows Media Player 11 212
Пример 74. Программирование контекстной справки. Стандартные кнопки в форме 216
Пример 75. Создание инсталляционного пакета для распространения программы 218

Глава 11. Программирование Web-ориентированных приложений на Visual Basic 2010 221

Создание Web-страницы на языке HTML. Интернет-технологии 221
Web-хостинг на платформах UNIX и Windows 223
Клиент-серверное взаимодействие на основе технологии ASP.NET 223
Отладка активного Web-приложения 224
Пример 76. Создание простейшей активной Web-страницы на VB2010 225
Пример 77. Проверка введенных пользователем числовых данных с помощью валидаторов 228
Пример 78. Проверка достоверности ввода имени, адреса e-mail, URL-адреса и пароля с помощью валидаторов 231
Пример 79. Регистрация и аутентификация пользователя с помощью базы данных Access 236
Пример 80. Таблица с переменным числом ячеек, управляемая двумя раскрывающимися списками 245
Пример 81. Организация раскрывающегося меню гиперссылок с помощью *DropDownList* 248
Пример 82. Передача данных между Web-страницами через параметры гиперссылки 250
Пример 83. Передача данных формы на другую Web-страницу методами класса *Request* 254
Пример 84. Передача значений элементов управления на другую Web-страницу с помощью объекта *PreviousPage* 258
Пример 85. Отображение табличных данных в Web-форме с помощью элемента управления *GridView* 261
Пример 86. Отображение в Web-форме хэш-таблицы 263
Пример 87. Чтение/запись текстового файла Web-приложением 266
Пример 88. Программирование счетчика посещений сайта с использованием базы данных и объекта *Session* 270
Пример 89. Чтение/запись cookie-файлов 275
Пример 90. Вывод изображения в Web-форму 279
Пример 91. Формирование изображения методами класса *Graphics* и вывод его в Web-форму 284
Пример 92. Гостевая книга 286
Пример 93. Отображение времени в Web-форме с использованием технологии AJAX 292

Глава 12. Создание Web-служб и их клиентов 295

О Web-службах 295
ПРИМЕР 94. Клиентское приложение, потребляющее сервис Web-службы "Прогноз погоды" 297
ПРИМЕР 95. Создание простейшей Web-службы 301
ПРИМЕР 96. Создание Windows-приложения, потребителя сервиса Web-службы 304
ПРИМЕР 97. Web-служба "Торговая рекомендация на рынке Forex" 306
ПРИМЕР 98. Клиентское приложение, потребляющее сервис Web-службы "Торговая рекомендация на рынке Forex" 309
ПРИМЕР 99. Клиентское Web-приложение, потребляющее сервис Web-службы "Морфер" 311
ПРИМЕР 100. Получение данных от Web-службы Центрального банка РФ Web-приложением 313
ПРИМЕР 101. Получение данных от Web-службы Национального банка Республики Беларусь Windows-приложением 315

Приложение. Описание компакт-диска 319

Предметный указатель 327

Введение

Система разработки программного обеспечения Microsoft Visual Basic 2010 является хорошим средством *быстрой разработки программ* для ускоренного создания приложений для Microsoft Windows и Интернета. Целью книги является популяризация программирования. Автор стремился показать, как малой кровью можно написать, почти сконструировать, как в детском конструкторе, довольно-таки функционально сложные приложения. Для реализации этой цели автор выбрал форму демонстрации *"на примерах"* решения задач от самых простых, элементарных до более сложных.

Рассмотрены примеры программ с экранной формой и элементами управления в форме, такими как текстовое поле, метка, кнопка и др. Написанные программы *управляются событиями*, в частности событиями мыши и клавиатуры. Поскольку большинство существующих программ взаимодействует с дисковой памятью, в книге приведены примеры чтения и записи файлов в долговременную память. Описаны решения типичных задач, которые встречаются в практике программирования, в частности работа с *графикой* и *буфером обмена*. Рассмотрены манипуляции табличными данными, использование элемента управления **WebBrowser** для отображения различных данных. Приведены примеры программирования с помощью функций (методов) объектных библиотек систем Microsoft Office и AutoCAD. Разобраны вопросы обработки баз данных SQL Server и MS Access с применением технологии ADO.NET, программирования Web-ориентированных приложений, а также использование и разработка Web-сервисов.

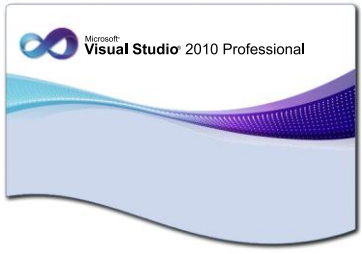
Несколько слов об особенностях книги. Спросите у любого программиста, как он работает (творит...) над очередной поставленной задачей. Он вам скажет, что всю задачу он мысленно *разбивает на фрагменты* и вспоминает, в каких ситуациях он *уже решал подобную задачу*. Далее он просто копирует фрагменты отлаженного программного кода и вставляет их в новую задачу. Сборник таких фрагментов (101 пример) содержит данная книга. Автор пы-

тался выделить наиболее типичные, актуальные задачи и решить их, с одной стороны, максимально эффективно, а с другой — кратко и выразительно. Вместе с книгой читателю предлагается компакт-диск с рассмотренными в ней примерами.

Самая серьезная проблема в проектировании больших программ — это *сложность, запутанность текстов*. Из-за запутанности программ имеются ошибки, нестыковки и проч. Как следствие — страдает производительность процесса создания программ и их сопровождение. Решение этой проблемы состоит в структуризации программ. Появление объектно-ориентированного программирования связано в большой степени со структуризацией программирования. Мероприятия для обеспечения большей структуризации — это проектирование программы как иерархической структуры, отдельные процедуры, входящие в программу, не должны быть слишком длинными, налагается запрет на использование операторов перехода `goto` и проч. Современные системы программирования разрешают в названиях переменных использовать *русские буквы*. Между тем современные программисты, как правило, *не используют* данную возможность, хотя когда вдруг в среде англоязычного текста появляются русские слова, это *вносит большую выразительность* в текст программы. Программный код начинает от этого лучше читаться, восприниматься человеком (транслятору, компилятору — все равно).

Данная книга предназначена для начинающих программистов, программистов среднего уровня, а также для программистов, имеющих навыки программирования на других языках и желающих ускоренными темпами освоить новый для себя язык Visual Basic 2010. Как пользоваться книгой? Эффективно пользоваться ею можно, последовательно решая примеры, в порядке их следования в книге, поскольку они расположены в последовательности *от простого к более сложному*, и тем самым постепенно совершенствуя свои навыки программирования на Visual Basic. А для программистов среднего уровня можно посоветовать искать выборочно именно те примеры, которые необходимы им при программировании их текущих задач.

Надеюсь, что читатель получит одновременно удовольствие и пользу от использования этой книги в своей работе и творчестве. Свои впечатления о данной книге присылайте по адресу **ziborov@ukr.net**, я с удовольствием их почитаю.



ГЛАВА 1

Простейшие программы с экранной формой и элементами управления

Пример 1. Форма, кнопка, метка и диалоговое окно

После инсталляции системы программирования Visual Studio 2010, включающей в себя Visual Basic 2010 (далее VB2010), загрузочный модуль системы devenv.exe будет, скорее всего, расположен в папке: C:\Program Files\Microsoft Visual Studio 10.0\Common7\IDE.

Целесообразно создать ярлык на рабочем столе для запуска VB2010. После запуска увидим начальный пользовательский интерфейс, показанный на рис. 1.1.

Чтобы запрограммировать какую-либо задачу, необходимо в пункте меню **File** выполнить команду **New Project**. В появившемся окне **New Project** в левой колонке находится список инсталлированных шаблонов (**Installed Templates**). Среди них — шаблоны языков программирования, встроенных в Visual Studio, в том числе Visual Basic, Visual C#, Visual C++, Visual F# и др. Нам нужен язык Visual Basic. В средней колонке выберем шаблон (Templates) **Windows Forms Application** и щелкнем на кнопке **OK**. В результате увидим окно, представленное на рис. 1.2.

В этом окне изображена *экранная форма* — **Form1**, в которой программисты располагают различные компоненты графического интерфейса пользователя или, как их иначе называют, элементы управления. Это поля для ввода текста **TextBox**, командные кнопки **Button**, строчки текста в форме — метки **Label**, которые не могут быть отредактированы пользователем, и прочие элементы управления. Причем здесь используется самое современное так называемое

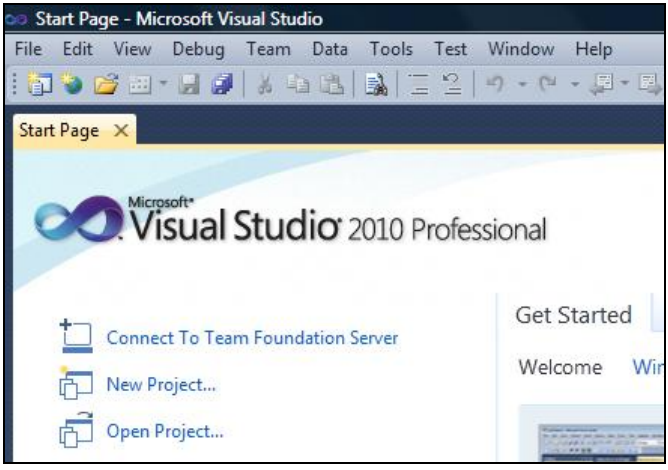


Рис. 1.1. Фрагмент стартовой страницы системы Visual Studio 2010

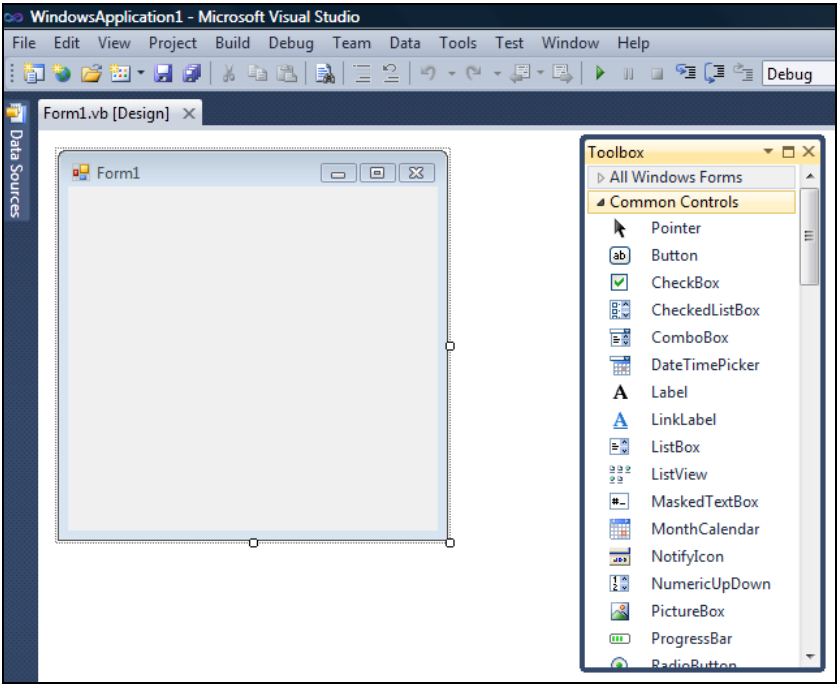


Рис. 1.2. Окно для проектирования пользовательского интерфейса

визуальное программирование, предполагающее простое перетаскивание мышью из панели элементов **Toolbox**, где расположены всевозможные элементы управления, в форму. Таким образом, стараются свести к минимуму непосредственное написание программного кода.

Ваша первая программа будет отображать такую экранную форму, в которой будет что-либо написано, например, "Microsoft Visual Basic 2010", также в форме будет расположена командная кнопка с надписью "Нажми меня". При нажатии кнопки появится диалоговое окно с сообщением "Всем привет!".

Написать такую программку — вопрос 2—3 минут. Но вначале я хотел бы буквально двумя словами объяснить современный объектно-ориентированный подход к программированию. Подход заключается в том, что в программе все, что может быть названо именем существительным, называют *объектом*. Так в нашей программе мы имеем четыре объекта: форму **Form**, надпись на форме **Label**, кнопку **Button** и диалоговое окно `MessageBox` с текстом "Всем привет!" (окно с приветом). Итак, добавьте метку и кнопку на форму примерно так, как показано на рис. 1.3.

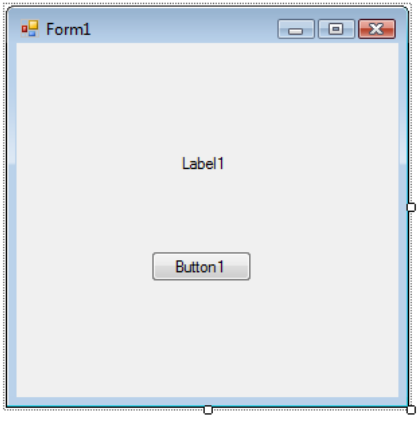


Рис. 1.3. Форма первого проекта

Любой такой объект можно создавать самому, а можно пользоваться готовыми объектами. В данной задаче мы пользуемся готовыми визуальными объектами, которые можно перетаскивать мышью из панели элементов управления **Toolbox**. В этой задаче нам нужно знать, что каждый объект имеет свойства (properties). Например, свойствами кнопки являются (рис. 1.4): имя кнопки (`Name`) — `Button1`, надпись на кнопке (`Text`), расположение кнопки (`Location`) в системе координат формы `X`, `Y`, размер кнопки `Size` и т. д. Свойств много, их можно увидеть, если щелкнуть правой кнопкой мыши в пределах формы и выбрать в контекстном меню команду **Properties**.

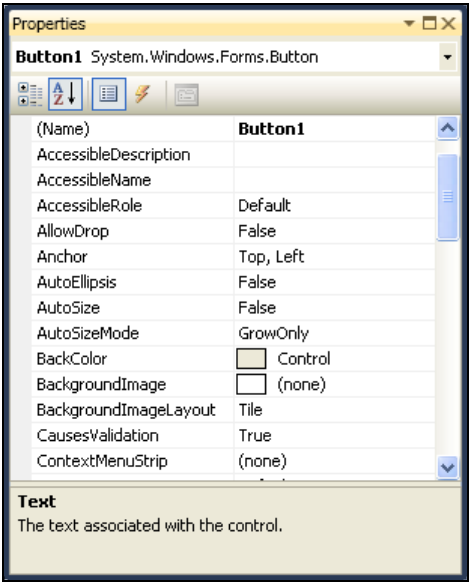


Рис. 1.4. Свойства кнопки `Button1`

Указывая мышью на все другие элементы управления в форме, можно просмотреть их свойства: формы `Form1` и надписи в форме — метки `Label1`.

Вернемся к нашей задаче. Для объекта `Label1` выберем свойство `Text` и напишем напротив этого поля "Microsoft Visual Basic 2010" (вместо текста `Label1`). Для объекта `Button1` также в свойстве `Text` напишем "Нажми меня".

Кроме того, что объекты имеют свойства, следует знать, что объекты обрабатываются событиями. Событием, например, является щелчок на кнопке, щелчок в пределах формы, загрузка (`Load`) формы в оперативную память при старте программы и проч. В нашей задаче единственным событием, которым мы управляем, является щелчок по командной кнопке. Напомню, что после щелчка на кнопке должно появиться диалоговое окно, в котором написано: "Всем привет!".

Чтобы обработать это событие, необходимо написать всего одну строчку программного кода. Перейдем на вкладку для написания кода: щелчок правой кнопкой мыши в пределах формы, затем выбор команды **View Code**. Слева вверху мы видим раскрывающийся список, где перечислены объекты, которые присутствуют в данном проекте. Здесь мы выберем `Button1` (командную кнопку). Справа вверху находится раскрывающийся список, в котором перечислены все события для кнопки; выбираем событие `Click`.

При этом управляющая среда VB2010 генерирует две строчки программного кода (рис. 1.5).

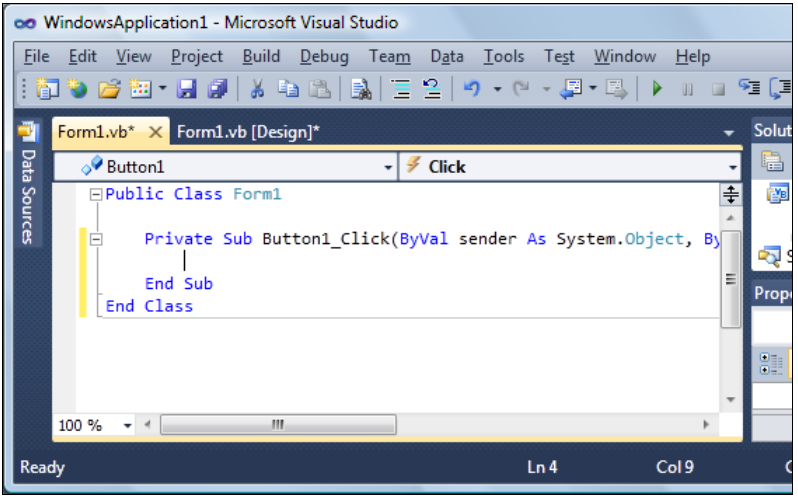


Рис. 1.5. Вкладка программного кода

Таким образом, система VB2010 написала начало процедуры `Sub` обработки события `Button1_Click` и конец процедуры `End Sub`. Эти две строчки называют *пустым обработчиком события*. Заполним этот обработчик. Для этого между этими строчками пишем:

```
MessageBox.Show("Всем привет!")
```

Здесь вызывается метод (программа) `Show` объекта `MessageBox` с текстом "Всем привет!". Таким образом, я здесь "нечаянно" проговорился о том, что объекты кроме свойств имеют также и *методы*, т. е. программы, которые обрабатывают объекты.

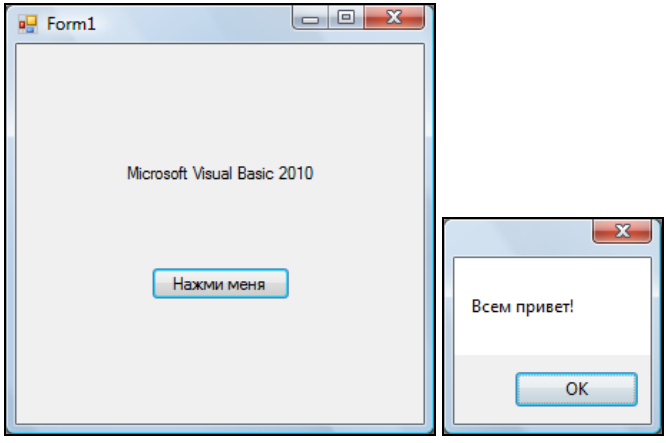


Рис. 1.6. Работающая программа

В этих трех строчках мы написали *процедуру обработки события* нажатия кнопки (Click) Button1. Теперь нажмем клавишу <F5> и проверим работоспособность программы (рис. 1.6).

Пример 2. Событие *MouseHover*

Немного усложним предыдущую задачу. Добавим еще одну обработку события MouseHover мыши для объекта Label1. Событие MouseHover наступает тогда, когда пользователь указателем мыши "зависает" над каким-либо объектом, причем именно "зависает", а не просто перемещает мышь над объектом (от англ. *hover* — реять, парить). Есть еще событие MouseEnter (Войти), когда указатель мыши *входит в пределы* области элемента управления (в данном случае метки Label1).

Переключимся на вкладку программного кода **Form1.vb**. Вы видите, что у нас две вкладки: **Form1.vb** и **Form1.vb [Design]**, т. е. вкладка программного кода и вкладка визуального проекта программы. Переключаться между ними можно мышью или нажатием комбинации клавиш <Ctrl>+<Tab>.

Итак, запрограммируем событие MouseHover объекта Label1. Для этого в окне редактора кода (где мы сейчас находимся) в левом верхнем раскрывающемся списке выбираем объект Label1, а в правом — событие MouseHover. При этом среда VB2010 генерирует две строки программного кода (пустой обработчик):

```
Private Sub Label1_MouseHover(Параметры процедуры...)
End Sub
```

Между этими двумя строчками вставляем вызов диалогового окна:

```
MessageBox.Show("Событие Hover!")
```

Теперь проверим возможности программы: нажимаем клавишу <F5>, "зависаем" указателем мыши над Label1, щелкаем на кнопке Button1. Все работает!

А теперь я буду немного противоречить сам себе. Я говорил про визуальную технику программирования, направленную на минимизацию написания программного кода. А сейчас хочу сказать про *наглядность*, *оперативность*, *технологичность* работы программиста. Посмотрите на свойства каждого объекта в окне **Properties**. Вы видите, как много строчек. Если вы меняете какое-либо свойство, то оно будет выделено жирным шрифтом. Удобно! Но все-таки еще более удобно свойства объектов *назначать* (устанавливать) *в программном коде*. Почему?

Каждый программист имеет в своем арсенале множество уже отлаженных фрагментов, которые он использует в своей очередной новой программе. Программисту стоит лишь вспомнить, где он программировал ту или иную

ситуацию. Программа, которую написал программист, имеет свойство быстро забываться. Если вы посмотрите на строчки кода, которые писали три месяца назад, то будете ощущать, что многое забыли; если прошел год, то вы смотрите на написанную вами программу, как на чужую. Поэтому при написании программ на первое место выходит *понятность, ясность, очевидность* написанного программного кода. Для этого каждая система программирования имеет какие-либо средства. Кроме того, сам программист придерживается некоторых правил, помогающих ему работать *производительно* и *эффективно*.

Назначать свойства объектов в программном коде удобно при обработке события `Form1_Load`, т. е. события загрузки формы в оперативную память при старте программы. На вкладке программного кода слева вверху выбираем `(Form1 Events)`, а справа — событие `Load`. А можно еще быстрей, просто сделать двойной щелчок в пределах формы на вкладке **Form1.vb [Design]**. При этом управляющая среда генерирует две строчки:

```
Private Sub Form1_Load()
End Sub
```

Между этими двумя строчками обычно вставляют свойства различных объектов и даже часто пишут много строчек программного кода. Здесь мы назначим свойству `Text` объекта `Label1` значение "Microsoft Visual Basic 2010":

```
Label1.Text = "Microsoft Visual Basic 2010"
```

Аналогично для объекта `Button1`:

```
Button1.Text = "Нажми меня!"
```

Совершенно необязательно писать каждую букву приведенных команд. Например, для первой строчки достаточно написать "la" (даже с маленькой буквы), уже это вызовет выпадающее меню, где вы сможете выбрать нужные для данного контекста ключевые слова. Это очень мощное и полезное современное средство редактирования программного кода! Если вы от VB2010 перешли в другую систему программирования, в которой отсутствует данная функция, то будете ощущать сильный дискомфорт.

Вы написали название объекта `Label1`, поставили точку. Теперь вы видите выпадающее меню, где можете выбрать либо нужное свойство объекта, либо метод (т. е. подпрограмму). В данном случае вы выберите свойство `Text`.

Как видите, не следует пугаться слишком длинных ключевых слов, длинных названий объектов, свойств, методов, имен переменных. Система подсказок современных систем программирования значительно облегчает всю нетворческую работу. Вот почему в современных программах можно наблюдать такие длинные имена ключевых слов, имен переменных и проч. Потому что

на первое место выходит ясность, прозрачность программирования, а громоздкость программ компенсируется системой подсказок.

Далее хотелось бы, чтобы слева вверху формы на синем фоне (в так называемой строке заголовка) была не надпись **Form1**, а что-либо осмысленное. Например, слово "Приветствие". Для этого ниже присваиваем эту строку свойству `Text` формы. Поскольку мы изменяем свойство объекта `Form1` внутри подпрограммы обработки события, связанного с формой, следует к форме обращаться через ссылку `Me`: `Me.Text = "Приветствие"` или `MyBase.Text = "Приветствие"`.

После написания последней строчки кода мы должны увидеть на экране программный код, показанный в листинге 1.1.

Листинг 1.1. Программирование событий

```
Public Class Form1

  Private Sub Form1_Load(ByVal sender As Object,
                         ByVal e As System.EventArgs) Handles Me.Load
    Label1.Text = "Microsoft Visual Basic 2010"
    Button1.Text = "Нажми меня!"
    ' Здесь объект Form1 ссылается на себя — Me
    Me.Text = "Приветствие"
  End Sub

  Private Sub Button1_Click(ByVal sender As Object,
                    ByVal e As System.EventArgs) Handles Button1.Click
    MessageBox.Show("Всем привет!")
  End Sub

  Private Sub Label1_MouseHover(ByVal sender As Object,
                ByVal e As System.EventArgs) Handles Label1.MouseHover
    ' Событие Hover, когда указатель мыши "завис" над меткой Label1
    MessageBox.Show("Событие Hover!")
  End Sub
End Class
```

Комментарии, поясняющие работу программы, в окне редактора кода будут выделены зеленым цветом, чтобы в тексте выразительно отделять его от прочих элементов программы. В VB комментарий пишут после одиночной кавычки (`'`) или после ключевого слова `REM` (от англ. *remark* — примечание). Уважаемые читатели, даже если вам кажется весьма очевидным то, что вы пишете в программном коде, *напишите комментарий*. Как показывает опыт,

даже очень очевидный замысел программиста забывается удивительно быстро. Человеческая память отметает все, что по оценкам организма считается ненужным.

Посмотрим еще раз на текст программы. Обратите внимание на то, что для улучшения читаемости программы второй параметр процедуры обработки события загрузки формы `Form1_Load` был перенесен на другую строку. При этом использован символ продолжения (_) для обозначения того, что остаток строки *переносится на следующую строку*. Замечу, что для обозначения переноса строки символ продолжения можно не использовать, если при переносе не разделять параметр процедуры, как это сделано, например, в процедуре обработки события щелчок на кнопке. Здесь запятая после первого параметра означает, что завершение строки нужно искать на следующей строке. В VB2010 существуют также ситуации, когда строку программного кода *можно разрывать*, а символ переноса (_) ставить не обязательно. Эти ситуации с примерами приведены по адресу **http://msdn.microsoft.com/en-us/library/865x40k4(VS.100).aspx**. В дальнейшем мы будем ориентироваться на эти допущения в написании кода. Но если вы захотите использовать код в предыдущих версиях VB, вам придется использовать символ переноса.

На рис. 1.7 приведена работа программы.

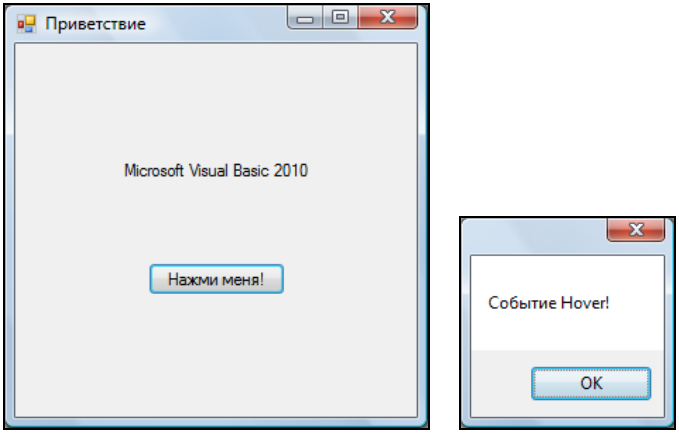


Рис. 1.7. Работа программы

Обычно в редакторах программного кода используется моноширинный шрифт, поскольку все символы такого шрифта имеют одинаковую ширину, в том числе точка и прописная русская буква "Ш". По умолчанию в редакторе программного кода VB2010 задан шрифт Consolas. Однако если пользователь привык к шрифту Courier New, то настройку шрифта можно изменить, выбрав меню **Tools | Options | Environment | Fonts and Colors**.

Теперь закроем проект (**File | Close Project**). Система предложит нам сохранить проект, сохраним его под именем **Hover**. Теперь программный код этой программы можно посмотреть, открыв решение Hover.sln, в папке Hover.

Пример 3. Ввод и вывод в консольном приложении

Иногда, например для научных расчетов, требуется организовать какой-нибудь самый простой ввод данных, выполнить, может быть, весьма сложную математическую обработку введенных данных и оперативно вывести на экран результат вычислений. Такая же ситуация возникает тогда, когда большая программа отлаживается по частям. И для отладки вычислительной части совершенно не важен сервис при вводе данных.

Можно по-разному организовать такую программу, в том числе программируя так называемое *консольное приложение* (от англ. *console* — пульт управления). Под консолью обычно подразумевают экран компьютера и клавиатуру.

Для примера напишем консольное приложение, которое приглашает пользователя ввести два числа, складывает их и выводит результат вычислений в диалоговое окно.

Для этого запускаем VB2010, далее создаем новый проект (**New Project**), выбираем шаблон **Console Application**. После двойного щелчка на этом шаблоне попадаем сразу на вкладку программного кода (рис. 1.8).

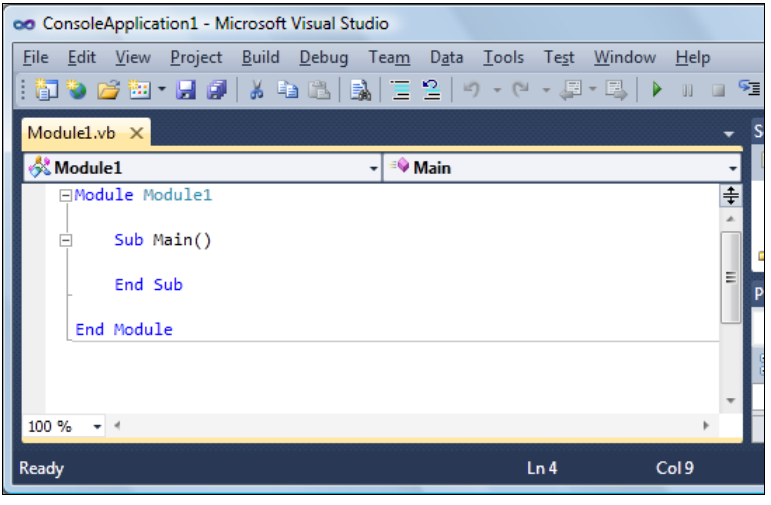


Рис 1.8. Вкладка программного кода

Как видите, здесь управляющая среда VB2010 приготовила четыре строки кода. Между `Sub Main()` и `End Sub` мы можем вставлять собственный программный код. `Sub Main()` — это стартовая точка, с которой начинается выполнение программы.

В программе будут участвовать три переменные: `X`, `Y`, `Z`. `X` и `Y` вводятся пользователем, далее `Z = X + Y` и `Z` выводится в диалоговое окно со словами "Сумма =". Эти переменные следует объявить как `Single`. Тип данных `Single` применяется тогда, когда число, записанное в переменную, может иметь целую и дробную части. Переменная типа `Single` занимает 4 байта.

Объявление этих переменных на VB2010 может иметь такой вид:

```
Dim X, Y, Z As Single
```

При этом в отличие от Visual Basic версии 6 (VB6), все три переменные будут иметь тип `Single`.

Далее используем объект `Console` для ввода `X`:

```
Console.WriteLine("Введите первое слагаемое:")
X = Console.ReadLine()
```

Аналогично для `Y`:

```
Console.WriteLine("Введите второе слагаемое:")
Y = Console.ReadLine()
```

Далее вычисление `Z`:

```
Z = X + Y
```

Заметьте, каждую команду (оператор) традиционно пишут с новой строчки. Если вы видите целесообразность написать несколько операторов в одной строке, то после каждого оператора надо ввести двоеточие (`:`).

После вычисления суммы необходимо вывести `Z` из оперативной памяти на экран. Для этого воспользуемся форматированным выводом в фигурных скобках метода `WriteLine` объекта `Console`:

```
Console.WriteLine("{0} + {1} = {2}", X, Y, Z)
```

Программа написана. Нажмите клавишу <F5>, чтобы увидеть результат.

Вы видите, что все окна появляются на фоне черного окна. Вообще говоря, консольное приложение задумывалось, чтобы программировать в этом черном окне. Вы можете попробовать, если вам это любопытно. Здесь ввод данных организуют через функцию `Console.WriteLine` (листинг 1.2).

Листинг 1.2. Ввод и вывод данных в консольном приложении

```
Module Module1
  Sub Main()
    Dim X, Y, Z As Single

    Console.Title = "Складываю два числа:"
    Console.WriteLine("Введите первое слагаемое:")
    X = Console.ReadLine()
    Console.WriteLine("Введите второе слагаемое:")
    Y = Console.ReadLine()

    Z = X + Y

    Console.WriteLine("{0} + {1} = {2}", X, Y, Z)

    Console.Read() ' Чтобы остановить и увидеть все, что на консоли
  End Sub
End Module
```

Такое программирование напоминает период конца 1980-х годов, когда появились первые персональные компьютеры с очень слабой производительностью и небольшой памятью. Поэтому рекомендую пользоваться в консольном приложении не объектом `Console`, а функциями `InputBox` и `MsgBox`.

Для вывода данных Visual Basic, начиная с версии VB.NET, имеет удобное средство `MessageBox.Show`, однако в консольном приложении его вызвать нельзя. Альтернативой `MessageBox.Show` может быть функция `MsgBox`, хорошо знакомая лицам, программировавшим на VB6:

```
MsgBox("Сумма = " & Z)
```

Функция `MsgBox` осталась в VB2010 и может применяться наряду с `MessageBox.Show`. Допустимо использовать `MsgBox` и в консольном приложении. Заменив в консольном приложении все методы объекта `Console` функциями `InputBox` и `MsgBox`, получим такой программный код, который очень напоминает VB6 (листинг 1.3).

Листинг 1.3. Ввод и вывод данных в стиле VB6

```
Module Module1
  Sub Main()
    Dim X, Y, Z As Single
```

```
    X = InputBox("Введите первое слагаемое:")
    Y = InputBox("Введите второе слагаемое:")

    Z = X + Y
    MsgBox("Сумма = " & Z)
  End Sub
End Module
```

Кстати, в консольном приложении вместо `MsgBox` можно все-таки использовать `MessageBox.Show`. Для этого в пункте меню **Project** выбираем команду **Add Reference**, на вкладке **.NET** выбираем строку **System.Windows.Forms**, а затем в программном коде перед `Module` вставляем строку `Imports System.Windows.Forms`. Ключевое слово `Imports` используется для импортирования пространства имен, которое содержит класс `MessageBox`.

При организации научных расчетов или в ситуации, когда необходимо отладить расчетную часть большой программы, когда сервис при вводе данных вообще не имеет значения, можно просто присваивать значения переменным при их объявлении. Очень технологичным является вариант, когда данные записываются в текстовый файл с помощью, например, Блокнота (notepad.exe), а в программе предусмотрено чтение текстового файла в оперативную память.

Пример 4. Проверка типа данных: функция *IsNumeric*

В предыдущей программе при вводе данных пользователь может ошибочно вводить символы вместо чисел, поэтому целесообразно в этом случае предлагать пользователю ввести данные еще раз. При этом можно воспользоваться проверкой типа введенных данных с помощью функции `IsNumeric(X)` и вечным циклом `Do` *<тело цикла>* `Loop`. В программе это выглядит таким образом:

```
Do ' Вечный цикл, пока пользователь не введет именно число:
  X = InputBox("Введите первое слагаемое", "Суммирование")
  If IsNumeric(X) = True Then Exit Do
Loop
```

В этом фрагменте операторы, заключенные между `Do` и `Loop`, будут выполняться до тех пор, пока функция `IsNumeric(X)` не вернет значение `True` (Истина), т. е. введенное значение `X` является числом. Только в этом случае произойдет выход из цикла `Exit Do`. Кроме того, обе вводимые переменные следует объявить как `String`, т. е. как строковые переменные:

```
Dim X, Y As String
```

Заметим, что при введении чисел, имеющих целую и дробную части в качестве разделителя, следует использовать *запятую*, а не точку[1]. Иначе функция `IsNumeric` будет воспринимать введенные символы как строку, а не число.

Немножко изменим приведенный выше вечный цикл `Do...Loop`:

```
Do
  X = InputBox("Введите первое слагаемое", "Суммирование")
  If IsNumeric(X) Then Exit Do
Loop
```

Здесь проверку на выход из цикла мы написали более компактно, словами эту проверку можно описать так: "Если `X` является числом, то выйти из цикла".

Кроме того, обратите внимание на `InputBox`. Здесь у этой функции появился еще один параметр после запятой — "Суммирование". Этот текст будет в заголовке окна ввода. Аналогично организуем ввод `Y`. Законченный программный код представлен в листинге 1.4.

Листинг 1.4. Программный код с проверкой типа

```
Module Module1
  ' Эта программа проверяет, числовые ли данные ввел пользователь,
  ' а затем складывает два введенных числа.
  Sub Main()
    Dim X, Y As String

    Do ' Вечный цикл, пока пользователь не введет именно число
      X = InputBox("Введите первое слагаемое", "Суммирование")
      If IsNumeric(X) = True Then Exit Do   ' Проверка типа
    Loop
    Do
      Y = InputBox("Введите второе слагаемое", "Суммирование")
      If IsNumeric(Y) = True Then Exit Do
    Loop

    Dim Z As Single = Val(X) + Val(Y)
    Z = Convert.ToSingle(X) + Convert.ToSingle(Y)

    ' Возможности конвертирования в VB.NET и выше, в т. ч. VB 2010
    ' Z = CType(X, Single) + CType(Y, Single)
    ' Z = Single.Parse(X) + Single.Parse(Y)
```

[1] В общем случае десятичный разделитель задается в Панели управления. Если вы работаете в русифицированной версии Windows, то по умолчанию у вас в качестве такого разделителя действительно задана запятая. — *Ред.*

```
    ' Конвертирование в VB6: Z = CSng(X) + CSng(Y)

    MsgBox("Сумма = " & Z, , "Результат:")
  End Sub
End Module
```

В тексте программы после ввода двух чисел в строковые переменные `X` и `Y` объявляем переменную `Z` как `Single`, в переменную `Z` будет копироваться сумма введенных чисел. Однако чтобы сложить эти два числа, их необходимо привести (преобразовать) также к типу `Single`. Для преобразования строковых переменных в VB6 имелись удобные названия функций преобразования: `CDbl(X)`, `CSng(X)`, `CInt(X)`. Они *конвертируют* (от англ. *convert* — преобразовать) строковую переменную `X` соответственно в переменную типа `Double`, `Single` и `Integer`. В VB6 имелась также функция `Val(X)`, однако она корректно конвертирует строковую переменную в числовой тип данных, если в качестве разделителя целой и дробной частей применяют десятичную точку (не запятую).

Начиная с VB.NET, кроме этих функций в Visual Basic имеется класс `Convert`, который включает в себя функции преобразования типов. Поэтому используем в нашей программе наиболее современную возможность преобразования `Convert.ToSingle(X)`. Конвертировать переменные одного типа в переменные другого типа можно также и другими функциями, такими как `CType`, `Parse`, конкретное использование этих функций приведено в тексте программы в комментарии (см. листинг 1.4).

Для вывода результата сложения на экран используем функцию `MsgBox`, здесь она получила еще один параметр для более привлекательного дизайна. Попробуем запустить эту программу, нажав клавишу <F5>, фрагмент работы этой программы показан на рис. 1.9.

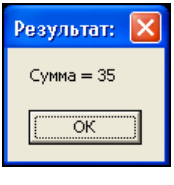


Рис 1.9. Фрагмент работы программы

Если появились ошибки, то работу программы следует проверить отладчиком — клавиши <F8> или <F11>. В этом случае управление останавливается на каждом операторе, и вы можете проверить значение каждой переменной, наводя указатель мыши на переменные. Можно выполнить программу до определенной программистом точки *(точки останова)*, используя, например,

клавишу <F9> или оператор `Stop`, и в этой точке проверить значения необходимых переменных.

Текст этой программы можно посмотреть, открыв решение conso_input.sln в папке conso_input.

Пример 5. Ввод данных через текстовое поле *TextBox*

При работе с формой чаще всего ввод данных организуют через элемент управления текстовое поле **TextBox**. Напишем типичную программу, которая вводит через текстовое поле число, при нажатии командной кнопки извлекает из него квадратный корень и выводит результат на метку **Label**. В случае ввода не числа сообщает пользователю об этом, очищая текстовое поле. Есть еще одна кнопка **Очистка** для обнуления текстового поля и метки.

Для этого запускаем VB2010, выбираем пункт меню **File | New Project**, затем — шаблон **Windows Forms Application** и щелкаем на кнопке **OK**. Далее из панели элементов управления **Toolbox** в форму указателем мыши перетаскиваем текстовое поле `TextBox1`, метку `Label1` и две командные кнопки `Button1` и `Button2`. Таким образом, в форме будут находиться четыре элемента управления.

Двойной щелчок в пределах проектируемой формы, и мы попадаем на вкладку программного кода в обработку события `Form1_Load` — события загрузки формы. Здесь задаем свойствам формы (к форме обращаемся посредством ссылки `Me`), кнопкам `Button1` и `Button2`, текстовому полю `TextBox1`, метке `Label1` следующие значения:

```
Me.Text = "Извлечение квадратного корня"
Button1.Text = "Извлечь корень"
Button2.Text = "Очистка"
TextBox1.Clear()    ' Очистка текстового поля
Label1.Text = ""
Label1.TextAlign = ContentAlignment.MiddleCenter
```

Последняя строка означает выравнивание текста, записанного в `Label1.Text`, по центру и на середине метки.

Нажмите клавишу <F5> для выявления возможных опечаток, т. е. синтаксических ошибок и предварительного просмотра дизайна будущей программы.

Далее программируем событие `Button1_Click` — щелчок мышью на кнопке **Извлечь корень**. Создать пустой обработчик этого события удобно, дважды щелкнув мышью на этой кнопке. Между двумя появившимися строчками

программируем диагностику правильности вводимых данных, конвертирование строковой переменной в переменную типа `Single` и непосредственное извлечение корня (листинг 1.5).

Листинг 1.5. Извлечение корня

```
' Программа вводит через текстовое поле число, при нажатии
' командной кнопки извлекает из него квадратный корень и выводит
' результат на метку Label1. В случае ввода не числа сообщает
' пользователю об этом, очищая текстовое поле. Есть еще одна кнопка
' Очистка для обнуления текстового поля и метки.
Public Class Form1

  Private Sub Form1_Load(ByVal sender As System.Object,
              ByVal e As System.EventArgs) Handles MyBase.Load
    Me.Text = "Извлечение квадратного корня"
    Button1.Text = "Извлечь корень"
    Button2.Text = "Очистка"
    TextBox1.Clear()    ' Очистка текстового поля
    Label1.Text = ""
    Label1.TextAlign = ContentAlignment.MiddleCenter
  End Sub

  Private Sub Button1_Click(ByVal sender As System.Object,
                  ByVal e As System.EventArgs) Handles Button1.Click

     ' Обработка события "щелчок на кнопке" Извлечь корень
     If Not IsNumeric(TextBox1.Text) Then
        MessageBox.Show("Следует вводить числа", "Ошибка",
                        MessageBoxButtons.OK, MessageBoxIcon.Error)
        TextBox1.Clear()    ' Очистка текстового поля
        TextBox1.Focus()    ' Установить фокус на текстовом поле
        Exit Sub
     End If

     Dim X, Y As Single
     ' Преобразование из строковой переменной в Single
     X = Convert.ToSingle(TextBox1.Text)
     Y = Math.Sqrt(X)
     Label1.Text = "Корень из " + X.ToString + " равен " + Y.ToString
  End Sub

  Private Sub Button2_Click(ByVal sender As System.Object,
                ByVal e As System.EventArgs) Handles Button2.Click
```

```
    ' Обработка события "щелчок на кнопке" Очистка:
    Label1.Text = ""
    TextBox1.Clear()  ' Очистка текстового поля
    TextBox1.Focus()
  End Sub
End Class
```

Здесь при обработке события "щелчок мышью на кнопке" **Извлечь корень** проводится проверка, введено ли число в текстовом поле. Проверка осуществляется с помощью функции `IsNumeric`: если введено не число (например, введены буквы), то выводится диалоговое окно с текстом "Следует вводить числа" (рис. 1.10).

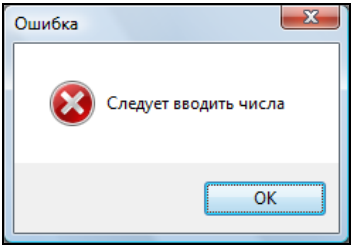


Рис. 1.10. Диагностика правильности вводимых данных

Далее, поскольку ввод неправильный, текстовое поле очищается — `TextBox1.Clear()`, а фокус передается опять на текстовое поле для ввода числа (т. е. курсор будет находиться в текстовом поле).

Оператор `Exit Sub` означает выход из программы обработки события `Button1_Click`.

Если пользователь ввел число, то управление не пойдет на ветку `If...Then`, а будет выполняться следующий оператор `Dim` — объявление переменных `X` и `Y`. Обычно все объявления делают в начале подпрограммы, но удобно объявить переменную там, где она впервые используется.

Далее функция `Convert.ToSingle` конвертирует строковую переменную `TextBox1.Text` в число `X`, из которого уже можно извлекать квадратный корень `Math.Sqrt(X)`. Математические функции VB2010 являются методами класса `Math`. Их можно увидеть, набрав `Math` и поставив точку (`.`). В выпадающем списке вы увидите множество математических функций: `Abs`, `Sin`, `Cos`, `Min` и т. д. и два свойства — две константы `E = 2,71...` (основание натуральных логарифмов) и `PI = 3,14...` (число диаметров, уложенных вдоль окружности).

Последней строчкой обработки события `Button1_Click` является присваивание переменной `Label1.Text` длинного текста. Здесь символ `+` (можно также `&` — амперсанд) означает "сцепить" переменные в одну строку.

Нажав клавишу <F5>, проверяем, как работает программа. Заметьте, что при вводе в текстовое поле числа, имеющего целую и дробную части, в качестве разделителя необходимо *ставить запятую*, а не точку. Иначе, введенное в текстовое поле не будет восприниматься как число.

Аналогично обработаем событие `Button2_Click` — очистка текстового поля и метки, а также передача фокуса опять на текстовое поле. Результат работающей программы представлен на рис. 1.11.

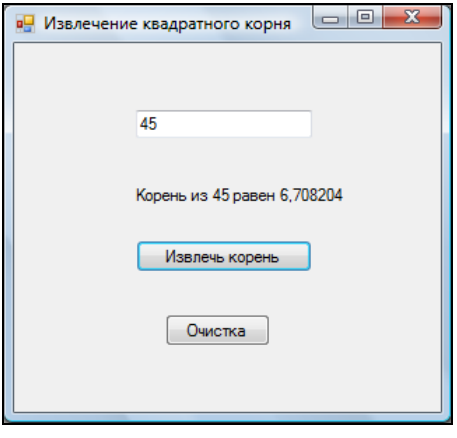


Рис. 1.11. Извлечение квадратного корня

При необходимости используйте отладчик (клавиша <F11>) для *пошагового выполнения программы* и выяснения всех промежуточных значений переменных путем "зависания" указателя мыши над переменными.

Текст этой программы можно посмотреть, открыв решение sqrt.sln в папке sqrt.

Пример 6. Ввод пароля в текстовое поле и изменение шрифта

Это очень маленькая программа для ввода пароля в текстовое поле, причем при вводе вместо вводимых символов некто, "находящийся за спиной пользователя", увидит только звездочки. Программа состоит из формы `Form1`, текстового поля `TextBox1`, метки `Label1`, куда для демонстрации возможностей мы будет копировать пароль (паспорт, т. е. секретные слова), и командной кнопки `Button1` — **Покажи паспорт**.

Перемещаем в форму все названные элементы управления. Текст программы приведен в листинге 1.6.

Листинг 1.6. Ввод пароля

```
' Программа для ввода пароля в текстовое поле, причем при вводе вместо
' вводимых символов некто, "находящийся за спиной пользователя",
' увидит только звездочки
Public Class Form1

  Private Sub Form1_Load(ByVal sender As System.Object,
                         ByVal e As System.EventArgs) Handles MyBase.Load
    Me.Text = "Введи пароль"
    TextBox1.Text = Nothing
    TextBox1.PasswordChar = "*"
    TextBox1.Font = New System.Drawing.Font("Courier New", 9.0!)
    Label1.Text = ""
    Label1.Font = New System.Drawing.Font("Courier New", 9.0!)
    Button1.Text = "Покажи паспорт"
  End Sub

  Private Sub Button1_Click(ByVal sender As System.Object,
                       ByVal e As System.EventArgs) Handles Button1.Click
    ' Обработка события "щелчок на кнопке"
    Label1.Text = TextBox1.Text
  End Sub
End Class
```

Обрабатываем два события. Первое событие — загрузка формы `Form1_Load`. Здесь очищаем текстовое поле и делаем его "защищенным от посторонних глаз" с помощью свойства `TextBox1.PasswordChar`, каждый введенный пользователем символ маскируется символом звездочки (`*`). Далее мы хотели бы для большей выразительности и читабельности программы, чтобы вводимые звездочки и результирующий текст имели одинаковую длину. Все символы шрифта Courier New имеют одинаковую ширину, поэтому его называют моноширинным шрифтом. Кстати, используя именно этот шрифт, удобно программировать таблицу благодаря одинаковой ширине букв этого шрифта. Еще одним широко используемым моноширинным шрифтом является шрифт Consola. Задаем шрифт, используя свойство `Font` обоих объектов: `TextBox1` и `Label1`. Число `9.0` означает размер шрифта.

Осталось обработать событие `Button1_Click` — щелчок на кнопке. Здесь — банальное присваивание текста из поля тексту метки. Программа написана,