ВИКТОР ЗИБОРОВ

Visual C# 2012

НА ПРИМЕРАХ

БЫСТРАЯ И ЭФФЕКТИВНАЯ РАЗРАБОТКА ПРИЛОЖЕНИЙ ДЛЯ ОБРАБОТКИ ТЕКСТОВЫХ, ТАБЛИЧНЫХ И ГРАФИЧЕСКИХ ДАННЫХ

ИСПОЛЬЗОВАНИЕ ФУНКЦИЙ MS WORD, MS EXCEL, AUTOCAD И MATLAB, А ТАКЖЕ СОЗДАНИЕ PDF-ФАЙЛОВ

ОБРАБОТКА БАЗ ДАННЫХ С ИСПОЛЬЗОВАНИЕМ ТЕХНОЛОГИИ ADO.NET И LINQ

СОЗДАНИЕ ИНТЕРАКТИВНЫХ ВЕБ-ПРИЛОЖЕНИЙ

СОЗДАНИЕ ВЕБ-СЛУЖБ И ИХ КЛИЕНТОВ С ПОМОЩЬЮ ТЕХНОЛОГИЙ WEB SERVICE И WCF SERVICE

РАЗРАБОТКА WPF-ПРИЛОЖЕНИЙ

БОЛЕЕ 140 ТИПИЧНЫХ ПРИМЕРОВ

+ Материалы на www.bhv.ru

bhv®

Виктор Зиборов

Visual C# 2012

НА ПРИМЕРАХ

Санкт-Петербург
«БХВ-Петербург»
2013

УДК 004.4
ББК 32.973.26-018.2
З-59

Зиборов В. В.

З-59 Visual C# 2012 на примерах. — СПб.: БХВ-Петербург, 2013. — 480 с.: ил.

ISBN 978-5-9775-0888-9

Рассмотрено более 140 типичных примеров, встречающихся в практике реального программирования для платформы .NET Framework в среде Microsoft Visual C# 2012: обработка событий мыши и клавиатуры, чтение/запись файлов, редактирование графических данных, управление буфером обмена, ввод/вывод данных, использование функций MS Word, MS Excel, AutoCAD и MATLAB, а также создание PDF-файлов, использование технологий LINQ и ADO.NET при работе с базами данных, разработка интерактивных веб-приложений, создание веб-служб с помощью технологий Web Service и WCF Service, разработка WPF-приложений 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. На сайте издательства находятся примеры из книги.

Для программистов

УДК 004.4
ББК 32.973.26-018.2

Группа подготовки издания:

Главный редактор	*Екатерина Кондукова*
Зам. главного редактора	*Игорь Шишигин*
Зав. редакцией	*Екатерина Капалыгина*
Редактор	*Григорий Добин*
Компьютерная верстка	*Ольги Сергиенко*
Корректор	*Зинаида Дмитриева*
Дизайн серии	*Инны Тачиной*
Оформление обложки	*Марины Дамбиевой*

Подписано в печать 30.11.12.
Формат 70×100$^{1}/_{16}$. Печать офсетная. Усл. печ. л. 38,7.
Тираж 1500 экз. Заказ №
"БХВ-Петербург", 191036, Санкт-Петербург, Гончарная ул., 20.

Первая Академическая типография "Наука"
199034, Санкт-Петербург, 9 линия, 12/28

ISBN 978-5-9775-0888-9

Оглавление

Предисловие

Система разработки программного обеспечения Microsoft Visual Studio 12 является мировым лидером на рынке программного обеспечения. Используя эту систему, можно "малой кровью" и очень быстро написать, почти сконструировать, *как в детском конструкторе*, довольно-таки функционально сложные как настольные приложения (в виде EXE-файлов), так и приложения, исполняемые в браузере. В центре системы Visual Studio 12 находится среда программирования (платформа) .NET Framework — это встроенный компонент Windows, который поддерживает создание и выполнение приложений нового поколения и веб-служб. Основными компонентами .NET Framework являются общеязыковая среда выполнения (CLR) и *библиотека классов* .NET Framework, включающая ADO.NET, ASP.NET, Windows Forms и Windows Presentation Foundation (WPF). Платформа .NET Framework предоставляет среду управляемого выполнения, возможности упрощения разработки и развертывания, а также возможности интеграции со многими языками программирования.

Среда разработки программного обеспечения Visual Studio 12 включает в себя языки программирования Visual Basic, Visual C#, Visual C++ и Visual F#. Используя эти языки программирования, можно подключаться к библиотекам классов и тем самым *иметь все преимущества* ускоренной разработки приложений. В этой среде программный код пишется проще, легче читается, а конечный продукт получается очень быстро.

Существенный положительный эффект достигается при групповой разработке какого-либо проекта. Используя Visual Studio, над одним проектом могут работать программисты на C#, на Visual Basic и на C++, при этом среда .NET обеспечит совместимость программных частей, написанных на разных языках.

Цель книги — популяризация программирования. Для реализации этой цели автор выбрал форму демонстрации *на примерах* решения задач от самых простых, элементарных, до более сложных.

Так, 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 *самых типичных задач*, которые встречаются в практике программирования, в частности работа с графикой и буфером обмена. Приведено несколько подходов

к выводу диаграмм (графиков). Рассмотрены манипуляции табличными данными, в том числе организация связанных таблиц. Показан принцип использования элемента управления `WebBrowser` для отображения различных данных, а также для программного заполнения веб-форм. Обсуждены примеры программирования с применением функций (методов) объектных библиотек систем MS Excel, MS Word, AutoCAD и MATLAB. Представлено несколько выразительных примеров создания PDF-файла. Разобраны вопросы обработки баз данных SQL Server и MS Access с помощью технологии ADO.NET. Рассмотрены методы обработки различных источников данных с использованием технологии LINQ. Представлено много различных авторских оригинальных решений задач программирования, которых читатель не сможет найти в Интернете. Приведены примеры программирования *веб-ориентированных приложений,* а также использования и разработки *веб-служб* (как Web Service, так и WCF Service). Новейшая технология WPF представлена несколькими показательными примерами.

Несколько слов об особенностях книги. Спросите у любого программиста, *как он работает* (творит...) над очередной поставленной ему задачей. Он вам скажет, что всю задачу он мысленно *разбивает на фрагменты* и вспоминает, в каких ранее решенных им задачах он *уже сталкивался с подобной ситуацией*. Далее он просто копирует фрагменты отлаженного программного кода и *вставляет их в новую задачу*. Сборник таких фрагментов (более 140 примеров) и содержит эта книга. Автор пытался выделить *наиболее типичные*, актуальные задачи и решить их, с одной стороны, максимально эффективно, а с другой стороны, *кратко* и *выразительно*.

> ***Внимание!***
>
> Электронный архив с примерами, рассмотренными в книге *(см. приложение)*, можно скачать с FTP-сервера издательства по ссылке **ftp://ftp.bhv.ru/9785977508889.zip**, а также со страницы книги на сайте **www.bhv.ru**.

Очень удобно читать книгу и пытаться запускать, трассировать отладчиком системы программирования, изменять, исследовать те или иные примеры из скачанного архива.

Самая серьезная проблема в программировании больших, сложных проектов — это *запутанность текстов*. Из-за нее в программах появляются ошибки, нестыковки и проч. Как следствие — страдает производительность процесса создания программ *и их сопровождение*. Решение этой проблемы состоит в *структуризации* программ. Переход к объектно-ориентированному программированию связан 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*, отказ от использования операторов перехода `goto` и проч. Кроме того, современные системы программирования разрешают в названиях переменных, методов, свойств, событий, классов, объектов *применять русские буквы (символы кириллицы)*. Между тем современные программисты, как правило, *не используют* такую возможность, хотя когда вдруг в среде англоязычного текста появляются русские слова, это *вносит большую выразительность* в текст программы, и тем самым большую структуризацию. Программный код начинает от этого лучше читаться, *восприниматься человеком* (транслятору, компилятору — все равно).

Эта книга предназначена для начинающих программистов, программистов среднего уровня, а также для программистов, имеющих навыки разработки *на других языках* и

желающих ускоренными темпами освоить новый для себя язык *MS C#*. Как пользоваться книгой? Для начинающих эффективнее всего — это последовательно решать примеры в порядке их представления в книге, поскольку примеры расположены *от простых к более сложным*. И тем самым постепенно совершенствовать свои навыки разработки на изучаемом языке программирования. Программистам среднего уровня можно посоветовать искать *выборочно* именно те примеры, которые соответствуют вопросам, возникшим у них при программировании текущих задач. Если вы программируете на Visual Basic или C++/CLI, то также можете получить положительный эффект от чтения этой книги, поскольку и C#, и C++/CLI, и Visual Basic "питаются" *все той же средой* .NET, следовательно, названия соответствующих классов, методов, свойств и событий являются одинаковыми, а программные коды различных языков отличаются всего лишь синтаксисом.

Надеюсь, что изучая эту книгу, читатель получит одновременно интеллектуальное *удовольствие* и *пользу* от примененения полученных знаний в своей работе и творчестве. Свои впечатления о книге присылайте по адресу **ziborov@ukr.net**, я с удовольствием с ними ознакомлюсь.

ВВЕДЕНИЕ

Что такое "хороший стиль программирования"?

Отвечая на этот вопрос одной строчкой, можно сказать, что хороший стиль программирования подразумевает наличие *структуры* у написанной программы. Соответственно, если программа содержит много строк программного кода в одной процедуре, не разделена на смысловые блоки, напоминает "спагетти", т. е. имеет множество переходов управления, содержит мало комментариев и, как следствие, становится запутанной, громоздкой, *трудно модифицируемой и управляемой*, то это и есть "плохой стиль программирования".

Отсюда появилось понятие *структурное программирование*. Появилось оно еще в 60-е годы прошлого века, когда весь мир переживал кризис разработки программного обеспечения. То есть создание больших автоматизированных систем затягивалось, программисты не укладывались в сроки, а в готовых программах обнаруживалось множество ошибок. С этими двумя проблемами (низкая производительность разработки программ и ошибки в них) программисты борются во все времена. В 1968 году голландец Дийкстра назвал (предположил) причину — в больших программах часто нет четкой математической структуры, они *неоправданно сложны*. Причем эту "неоправданную" сложность вносит команда `goto`. Программу с множеством `goto` называют "спагетти". Для сравнения, представьте, что вы читаете роман: сначала две страницы назад, потом четыре страницы вперед и т. д. Если программа содержит много операторов `goto`, то человеку проследить передачу управления (последовательность управления) весьма затруднительно (это только компьютеру все равно). Вместо операторов `goto` Дийкстра предложил использовать всего три типа управляющих структур. Первый тип — это простая последовательность *(следование)*, когда операторы выполняются друг за другом слева направо и сверху вниз. Второй тип — это альтернатива *(ветвление)*, выбор по условию (`if – else`), множественный выбор (`switch – case`). И третий тип управляющей структуры — это *цикл*, т. е. повторение одного или нескольких операторов до выполнения условия выхода из цикла.

Так вот, основная идея Дийкстры заключается в том, что, используя эти три структуры (отсюда и название "структурное программирование"), можно обходиться без операторов `goto`. Он утверждал, что отсутствие наглядности — это и есть основной источник ошибок. Сначала программисты лишь посмеивались над идеями Дийкстры, тем более, что он был всего лишь теоретиком программирования. Однако смеяться над чем-то новым — весьма опрометчиво. Один русский прославленный генерал, когда ему впервые показали только что изобретенный пулемет, снисходительно похлопал изобретателя по

плечу и сказал, что если бы в реальном бою нужно было убить одного человека несколько раз, то его изобретение было бы очень кстати. Так что не следует спешить высмеивать новое, непонятное и непривычное.

Так случилось и с идеями Дийкстры — фирма IBM весьма успешно применила принципы структурного программирования для создания базы данных газеты "Нью-Йорк таймс". И это стало весомым аргументом в пользу такого подхода. Со временем оператор `goto` программисты стали называть "позорным" и использовать его лишь в очень крайних случаях. И с тех пор концепция структурного программирования оказывает заметное влияние на теорию и практику программного обеспечения всех рангов.

Сегодня структурное программирование, как альтернатива "интуитивному", "рефлекторному" подходу, — это методология разработки программного обеспечения, в основе которой лежит не только три типа управляющих структур, но и многое другое, что помогает представлять программу в виде иерархической структуры и тем самым "спрятать сложность". Все тело программы стараются делить на логически целостные вычислительные блоки, которые оформляются в виде подпрограмм, функций и классов. Эти вычислительные блоки и формируют собой иерархическую структуру, причем наименее значимые "подробности" программы стараются спрятать в самые дальние ветви иерархии, чтобы они не мешали пониманию основной логики программы. Пряча те или иные смысловые блоки подальше от глаз, мы тем самым скрываем сложность программы для того, чтобы ее (программы) логика была максимально понятой. Причем "понятность", читабельность нужно обеспечить не только автору программы, но и другим разработчикам. Вполне справедливо утверждение, что любую, самую сложную, программу можно сделать доступной для понимания "широким слоям разработчиков", скрывая сложности в смысловых блоках.

Каждый смысловой вычислительный блок должен содержать как можно больше комментариев, а в его начале необходима преамбула, в которой указано, что этот блок делает, и какие технологии и приемы при этом используются. Концепция объектно-ориентированного программирования — это одна из ступеней к большей структуризации программ. Современные системы программирования стараются обеспечить максимальную читабельность наших программ — обратите, кстати, внимание на отступы и разноцветность программного кода в окнах системы, это придает программам бóльшую выразительность и понятность.

Следует всегда учитывать простой факт — суть программы, которую вы писали неделю назад, очень быстро забывается. А если вы посмотрите на программу, написанную вами же месяц назад? Вы удивитесь, но у вас создастся впечатление, что это чужая программа, написанная никак не вами. Не стоит удивляться, это особенность человеческого организма — отбрасывать все на его (организма) взгляд ненужное!

В ходе написания программного кода следует большое внимание уделять как можно более точным названиям объектов, переменных, процедур. Тогда смысл написанных операторов будет более "прозрачен". Автор в своих примерах старался присваивать соответствующие названия на русском языке, что обеспечило еще большую структурированность написанных программ. Вообще, следует помнить, что антонимом слова "структура" будет — "хаос".

Таким образом, бóльшая структуризация программы — это путь для повышения производительности разработки, уменьшения количества ошибок, и это есть "хороший стиль программирования".

ГЛАВА 1

Простейшие программы с экранной формой и элементами управления

Пример 1. Форма, кнопка, метка и диалоговое окно

После инсталляции системы программирования Visual Studio 2012, или кратко VS 12, включающей в себя Visual C# 12, загрузочный модуль системы devenv.exe будет, скорее всего, расположен в папке: C:\Program Files\Microsoft Visual Studio 12.0\Common7\IDE.

После запуска системы мы увидим начальный пользовательский интерфейс, показанный на рис. 1.1.

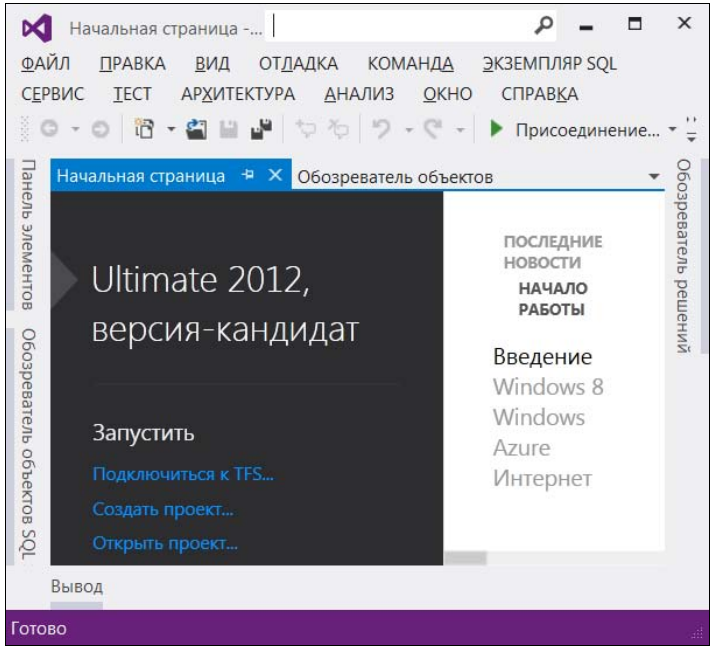

Рис. 1.1. Фрагмент стартовой страницы системы Visual Studio 12

Чтобы запрограммировать какую-либо задачу, необходимо в пункте меню **File** выполнить команду **New Project**. В левой колонке открывшегося окна **New Project** находится список установленных 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. Выберем язык Visual C#, а затем в области шаблонов (в средней колонке) щелкнем на шаблоне (**Templates**) **Windows Forms Application**. Теперь введем имя проекта (**Name**): `First` и щелкнем на кнопке **OK**. В результате увидим окно, представленное на рис. 1.2.

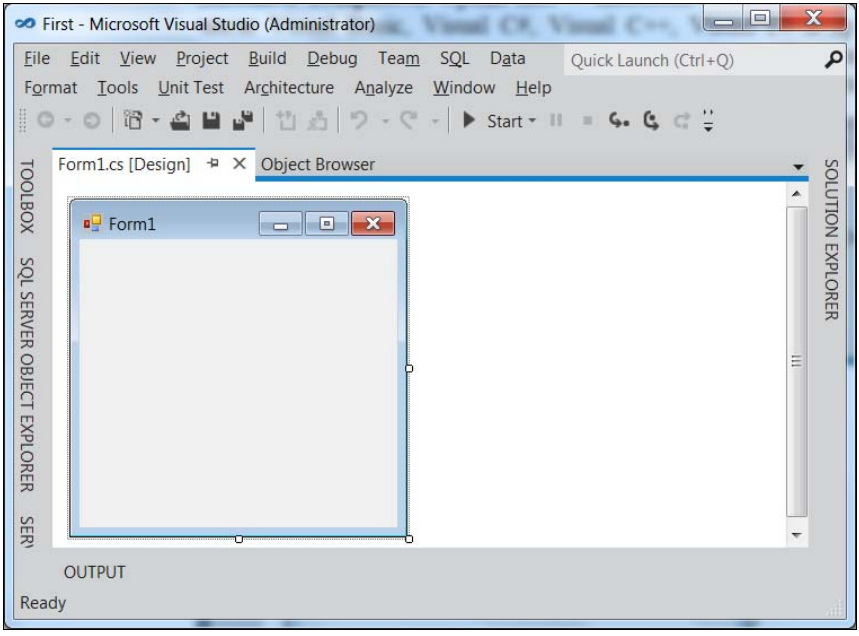

Рис. 1.2. Окно для проектирования пользовательского интерфейса

В этом окне изображена *экранная форма* — **Form1**, в которой программисты располагают различные компоненты графического интерфейса пользователя или, как их иначе называют, *элементы управления*: поля для ввода текста **TextBox**, командные кнопки **Button**, строчки текста в форме (метки **Label**, которые не могут быть отредактированы пользователем) и проч.

О визуальном программировании

Обратите внимание — здесь используется самое современное так называемое *визуальное программирование*, предполагающее простое перетаскивание мышью в форму тех или иных элементов управления из панели **Toolbox**, где все они и расположены. Таким образом, сводится к минимуму непосредственное написание программного кода.

В вашей первой программе мы создадим экранную форму с надписью, например, "Microsoft Visual C#". Кроме нее в форме будет расположена командная кнопка с надписью "Нажми меня". При нажатии кнопки откроется диалоговое окно с сообщением "Всем привет!"

Пояснение

Написать такую программку — вопрос 2–3 минут. Но сначала я хотел бы буквально парой слов пояснить основной принцип современного объектно-ориентированного подхода к программированию. А заключается он в том, что в программе все, что может быть определено как имя существительное, называют *объектом*. Так в нашей программе имеется четыре объекта: форма **Form**, надпись на форме **Label**, кнопка **Button** и диалоговое окно **MessageBox** с текстом "Всем привет!" — окно с приветом ☺.

Теперь давайте добавим в форму названные элементы управления. Для этого нам понадобится панель элементов управления **Toolbox**. Если в данный момент вы панель элементов не видите, то ее можно добавить, например, с помощью комбинации клавиш <Ctrl>+<Alt>+<x> или выполнив команду меню **View | Toolbox**. Итак, добавьте метку **Label** и кнопку **Button** в форму, щелкнув двойным щелчком на этих элементах в панели **Toolbox**. А затем расположите их примерно так, как показано на рис. 1.3.

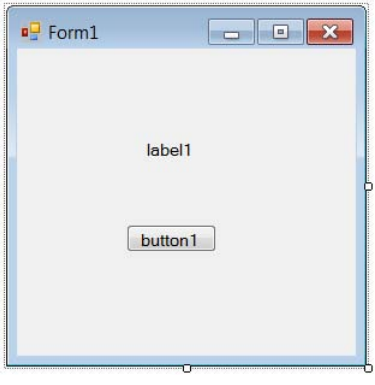

Рис. 1.3. Форма первого проекта

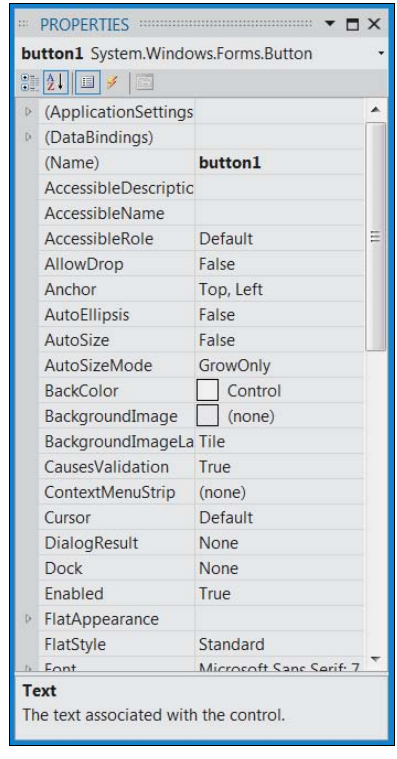

Рис. 1.4. Свойства кнопки `button1`

Любой такой объект можно создать самостоятельно, а можно воспользоваться готовым. В данной задаче мы пользуемся готовыми визуальными объектами, которые можно также просто перетаскивать мышью из панели элементов управления **Toolbox** в окно формы.

При этом нам нужно знать, что каждый объект имеет *свойства* (**Properties**). Например, свойствами кнопки являются (рис. 1.4): имя кнопки (**Name**) — в нашем случае **Button1**, надпись на кнопке (**Text**), расположение кнопки (**Location**) в системе координат формы **X**, **Y**, размер кнопки (**Size**) и т. д. Свойств много, все их можно увидеть в панели

свойств **Properties** (см. рис. 1.4), которая откроется, если щелкнуть правой кнопкой мыши в пределах формы и выбрать в контекстном меню команду **Properties**.

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

Вернемся к нашей задаче. Для объекта **Label1** выберем свойство **Text** и напишем у этого поля: `Microsoft Visual C# 12` (вместо текста `label1`). Для объекта `button1` также в свойстве `Text` напишем `Нажми меня`.

Итак, что объекты имеют свойства, мы уже поняли. Следует также 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. Управляют тем или иным событием посредством написания процедуры обработки события в программном коде. Для этого вначале нужно получить *"пустой" обработчик события* (т. е. заготовку записи программного кода). В нашей задаче единственным событием, которым мы управляем, является щелчок на командной кнопке. Для получения пустого обработчика этого события следует в свойствах кнопки **Button1** (см. рис. 1.4) щелкнуть на значке молнии **Events** (События) и в списке всех возможных событий кнопки **Button1** выбрать двойным щелчком событие **Click**. После этого перед нами откроется вкладка программного кода **Form1.cs** (рис. 1.5).

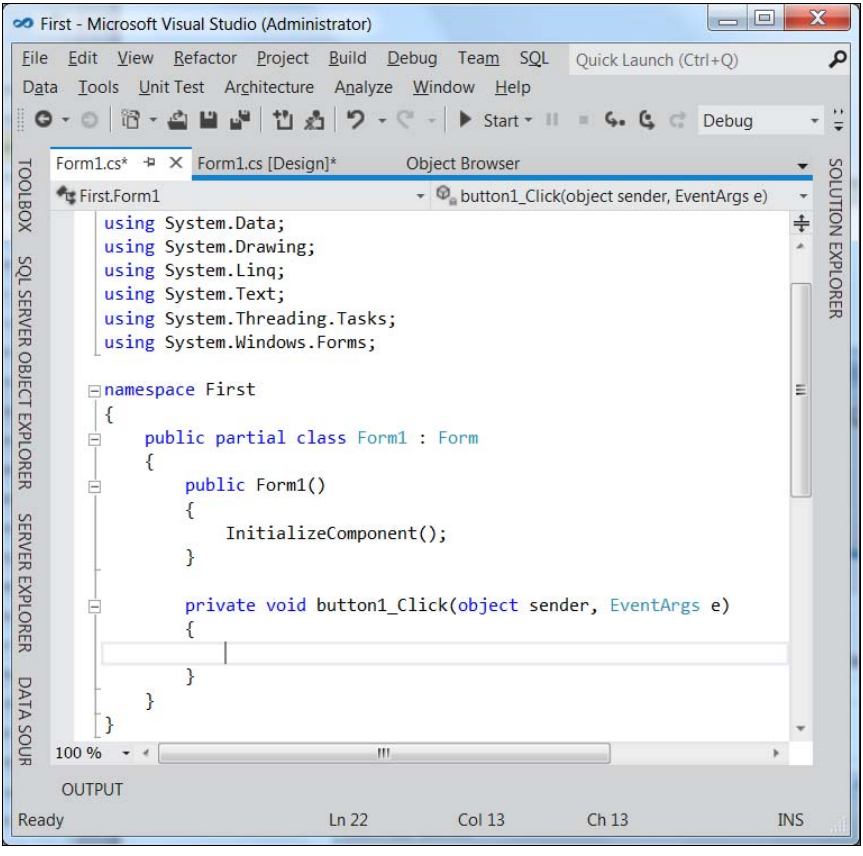

Рис. 1.5. Вкладка программного кода

На вкладке **Form1.cs** мы увидим, что управляющая среда Visual C# 12 сгенерировала четыре строки программного кода. Читателю не стоит ужасаться, глядя на этот непонятный код — постепенно многое прояснится.

Итак, на рис. 1.5 показан программный код, описывающий упомянутый нами "пустой" обработчик события `Button1_Click`. Здесь между фигурными скобками, следующими за оператором `private void`, мы можем написать команды, подлежащие выполнению после щелчка пользователем на командной кнопке **button1**.

Как можно видеть, у нас теперь две вкладки: вкладка программного кода **Form1.cs** и вкладка визуального проекта программы **Form1.cs [Design]** (другое название этой вкладки — *дизайнер формы*). Переключаться между ними можно мышью или нажатием комбинации клавиш <Ctrl>+<Tab>, как это принято для переключения между вкладками в Windows, а также функциональной клавишей <F7>.

Напомню, что по условию задачи после щелчка на кнопке **Button1** должно открыться диалоговое окно с надписью: "Всем привет!" Для этого в фигурных скобках обработчика события напишем:

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

Этой записью вызывается *метод* (программа) `Show` объекта `MessageBox` с текстом "Всем привет!" Оператор точка (`.`) указывает системе найти метод `Show` среди методов объекта `MessageBox`. Ну вот, я здесь "нечаянно" проговорился, что объекты, кроме свойств, имеют также и *методы*, т. е. программы, которые обрабатывают объекты.

> ***Совет***
>
> Заметьте, что в языках с С-подобным синтаксисом в конце каждого оператора ставят точку с запятой (`;`). И чтобы не задумываться, в каком режиме в данный момент пребывает ваша клавиатура: в режиме ввода русских букв или английских — *очень удобно*, не переключаясь в другой режим, ввести точку с запятой набором на боковой клавиатуре цифр `59`, удерживая при этом нажатой клавишу <Alt>.

Таким образом, мы написали *процедуру обработки события* щелчка **Click** на кнопке **Button1**. Теперь нажмем клавишу <F5> и проверим работоспособность программы (рис. 1.6).

Поздравляю, вы написали свою первую программу на MS Visual C# 12!

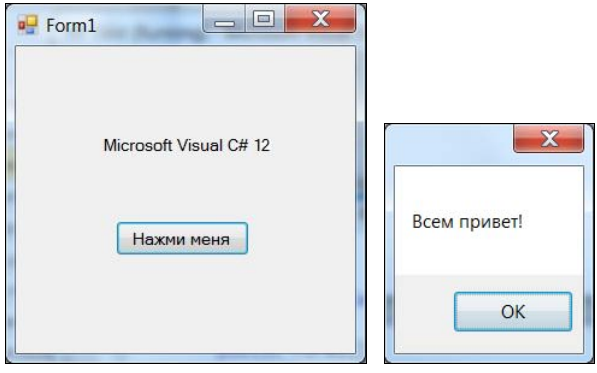

Рис. 1.6. Пример работы программы

Убедиться в работоспособности программы можно, открыв решение First.sln в папке First сопровождающего книгу электронного архива.

> ***Напоминание***
>
> Электронный архив с примерами, рассмотренными в книге *(см. приложение)*, можно скачать с FTP-сервера издательства по ссылке **ftp://ftp.bhv.ru/9785977508889.zip**, а также со страницы книги на сайте **www.bhv.ru**.

Пример 2. Событие *MouseHover*

Немного усложним задачу предыдущего примера. Добавим для объекта **Label1** обработку еще одного события — **MouseHover**. Событие **MouseHover** наступает, когда пользователь указателем мыши "зависает" над каким-либо объектом, причем именно "зависает" (от англ. *hover* — реять, парить), а не просто проводит мышью над объектом. Можно сказать также, что событие **MouseHover** происходит, когда указатель мыши *наведен* на элемент.

> ***Примечание***
>
> Существует еще событие **MouseEnter** (Войти), когда указатель мыши *входит в пределы* области элемента управления (в данном случае метки **Label1**), но сейчас мы воспользуемся именно событием **MouseHover**.

Таким образом, программа в нашем примере должна содержать на экранной форме текстовую метку **Label** и кнопку **Button**. Метка будет отображать текст "Microsoft Visual C# 12", а при щелчке по командной кнопке, на которой по-прежнему написано "Нажми меня", откроется диалоговое окно с сообщением "Всем привет!" Кроме того, когда указатель мыши окажется наведен на текстовую метку (то самое событие **MouseHover**), должно открыться диалоговое окно с текстом "Событие Hover".

Для решения этой задачи запустим Visual Studio 12, щелкнем на пункте меню **New Project**. В открывшемся окне **New Project** в левой колонке найдем пункт **Visual C#**, а затем в области **Templates** (Шаблоны) — в средней колонке — выберем шаблон **Windows Forms Application**. В качестве имени проекта введем имя `Hover` и щелкнем на кнопке **OK**.

В дизайнере формы из панели **Toolbox** перетащим на форму метку **Label** и кнопку **Button**, а затем немного уменьшим размеры формы на свой вкус. Теперь добавим три обработчика событий в программный код. Для этого в панели **Properties** следует щелкнуть на значке молнии (**Events**) и двойными щелчками последовательно выбрать:

- событие загрузки формы **Load**;
- событие щелчок на кнопке button1 — **Click**;
- событие "зависания" указателя мыши на текстовой метке label1 — **MouseHover**.

При этом осуществится переход на вкладку программного кода **Form1.cs** и среда Visual Studio 12 сгенерирует три пустых обработчика события. Например, обработчик последнего события будет иметь вид:

```
private void label1_MouseHover(object sender, EventArgs e)
{
}
```

Между этими двумя фигурными скобками вставим 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!

Уточнение позиции

А сейчас я слегка войду в противоречие с самим себой. Ранее я говорил про визуальную технику программирования, направленную на минимизацию "ручного" написания программного кода. А сейчас хочу сказать про *наглядность*, *оперативность*, *технологичность* работы программиста. Посмотрите на свойства каждого объекта в панели **Properties**. Вы видите, как много строчек. Если вы меняете какое-либо свойство, 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. Как уже отмечалось во *введении*, написанная только что программа имеет свойство быстро забываться. Если вы посмотрите на строчки кода, которые писали неделю назад, то почувствуете, что многое забыли, по прошествии же месяца вы смотрите на написанную вами программу, как на чужую. Поэтому при написании программ на первое место выходят *понятность*, *ясность*, *очевидность* написанного программного кода. Для этого каждая система программирования имеет те или иные средства. Кроме того, сам программист должен придерживаться некоторых правил, помогающих ему работать *производительно* и *эффективно*.

Назначать свойства объектов в программном коде удобно или сразу после инициализации компонентов формы (после процедуры InitializeComponent), или при обработке события **Form1_Load**, т. е. события загрузки формы в оперативную память при старте программы.

Создадим простой обработчик этого события. Для этого, как и в предыдущих случаях, можно выбрать нужное событие в панели свойств объекта, а можно еще проще — щелкнуть двойным щелчком в пределах проектируемой формы на вкладке **Form1.cs [Design]**. В любом случае на вкладке программного кода будет сформирован "пустой" обработчик события.

Заметим, что для формы умалчиваемым событием, для которого можно двойным щелчком получить пустой обработчик, является событие загрузки формы **Form1_Load**, а для командной кнопки **Button** и метки **Label** таким событием является одиночный щелчок мышью на этих элементах управления. То есть, если щелкнуть двойным щелчком в дизайнере формы по кнопке, то получим в программном коде пустой обработчик одиночного щелчка **Button1_Click**, аналогично получаем программный код и для обработки одиночного щелчка по метке **Label**.

Итак, вернемся к событию загрузки формы — для него управляющая среда сгенерировала пустой обработчик:

```
private void Form1_Load(object sender, EventArgs e)
{
}
```

Здесь, между двумя этими фигурными скобками, обычно вставляют свойства различных объектов и даже часто пишут много строчек программного кода. Назначим свойству **Text** объекта **label1** значение `Microsoft Visual C# 12`:

```
label1.Text = "Microsoft Visual C# 12";
```

Аналогично для объекта **button1**:

```
button1.Text = "Нажми меня";
```

О средстве IntelliSense

Совершенно необязательно писать каждую букву приведенных команд. Например, для первой строчки достаточно написать `la` — появится раскрывающееся меню, откуда вы сможете выбрать нужные для данного контекста ключевые слова. Это работает IntelliSense — очень мощное и полезное современное средство редактирования программного кода (его иногда называют *суфлером*). И если вы от Visual Studio 12 перешли в другую систему программирования, где подобный сервис отсутствует, то станете ощущать сильный дискомфорт.

Пользуясь функцией IntelliSense, очень удобно после ввода оператора разрешения области действия или оператора-точки — в обоих случаях вводим символ точки (.) — получать список допустимых вариантов дальнейшего ввода. Можно выделить элемент и нажать клавишу <Tab> или <Enter> или щелкнуть двойным щелчком по элементу, чтобы вставить его в код. Так что 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 весьма длинные имена ключевых слов, имена переменных и проч.

Я призываю вас, уважаемые читатели, также использовать в своих программах для названий переменных, объектов наиболее ясные, полные имена, причем предпочтительно на вашем родном *русском языке*. Потому что на первое место выходят ясность, прозрачность программирования, а громоздкость названий с лихвой 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 или через ссылку `this`, или используя ключевое слово `base`:

```
base.Text = "Приветствие";
```

Написав последнюю строчку кода, мы должны увидеть на экране программный код, представленный в листинге 1.1.

Листинг 1.1. Фрагмент файла Form1.cs, содержащего программный код с тремя обработчиками событий

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
namespace Hover
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
```

```
        private void Form1_Load(object sender, EventArgs e)
        {
            // Процедура обработки события загрузки формы
            this.Text = "Приветствие";
            // или base.Text = "Приветствие";
            label1.Text = "Microsoft Visual C# 12";
            button1.Text = "Нажми меня";
        }
        private void button1_Click(object sender, EventArgs e)
        {
            // Процедура обработки события "щелчок на кнопке"
            MessageBox.Show("Всем привет!");
        }
        private void label1_MouseHover(object sender, EventArgs e)
        {
            // Обработка события, когда указатель мыши "завис" над меткой:
            MessageBox.Show("Событие Hover!");
        }
    }
}
```

Комментарии, поясняющие работу программы, в окне редактора кода выделены зеленым цветом, чтобы в тексте он выразительно отделялся от прочих элементов программы. В языках с С-подобным синтаксисом (C#, C++, PHP, Java, JavaScript) комментарий пишут после двух прямых слэшей (`//`) или внутри пар `/* */`.

> ***Совет***
>
> Уважаемые читатели, даже если вам кажется весьма очевидным то, что вы пишете в программном коде, *добавьте комментарий*. Как показывает опыт, даже весьма очевидный замысел программиста забывается удивительно быстро. Человеческая память отметает все, что по оценкам организма считается ненужным. В любом случае, даже если текст программы вполне ясен, в начале программы обязательно опишите ее назначение и способ использования, т. е. как бы "преамбулу" программы. Далее в примерах мы будем следовать этому правилу.

На рис. 1.7 приведен пример работы программы.

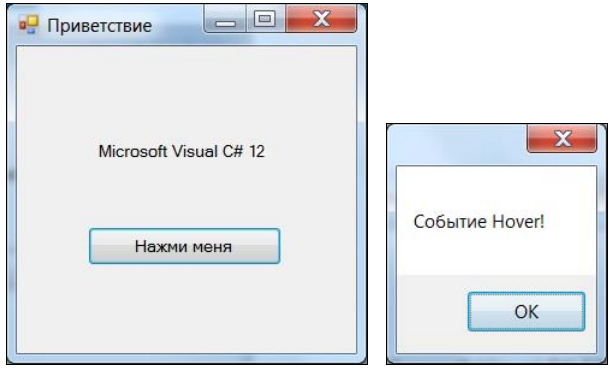

Рис. 1.7. Пример работы программы

Обычно в редакторах программного кода используется моноширинный шрифт, в котором все символы имеют одинаковую ширину — от точки до прописной русской буквы "Ш". По умолчанию в редакторе программного кода Visual C# 12 задан шрифт Consolas. Однако если пользователь привык к шрифту Courier New, то шрифт по умолчанию 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 сопровождающего книгу электронного архива.

Пример 3. Выбор нужной даты

Условие следующей нашей задачи состоит том, чтобы, например, при заказе железнодорожных билетов или при регистрации в качестве отдыхающего в санатории (ситуаций может быть много) при щелчке на соответствующем элементе управления появлялся бы календарь, из которого можно было бы выбрать необходимую дату. Это может быть дата отправления поезда, или дата окончания отдыха в санатории, или другие ситуации, связанные с выбором нужной даты.

Для решения этой задачи запускаем систему программирования Visual Studio 12 и щелкаем на пункте **New Project**. В открывшемся окне **New Project** выбираем пункт **Visual Basic**, а затем в области шаблонов (в средней колонке) — шаблон (**Templates**) **Windows Forms Application**. В качестве имени проекта введем имя `ВыборДаты` и щелкнем на кнопке **OK**.

В дизайнере формы из панели **Toolbox** перетащим на форму командную кнопку **Button**, метку **Label** и элемент **DateTimePicker**. Этот элемент непосредственно выполняет функцию выбора даты. Если щелкнуть на стрелке элемента **DateTimePicker**, то раскроется календарь для выбора даты (рис. 1.8). Также мы хотим, чтобы календарь элемента **DateTimePicker** раскрывался при нажатии кнопки **Button**. В итоге выбранная дата должна отображаться в текстовой метке **Label**.

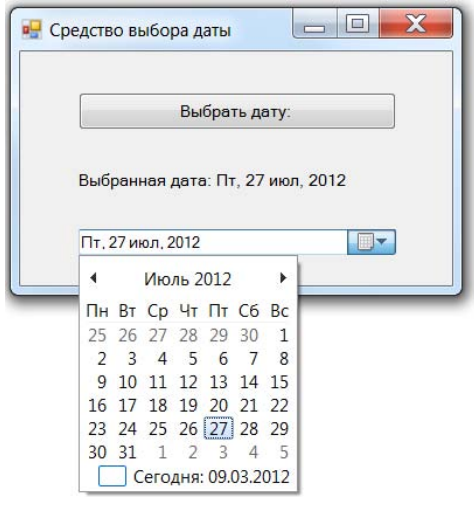

Рис. 1.8. Выбор нужной даты из раскрывающегося календаря

Немного изменим указателем мыши размеры экранной формы и расположим элементы управления, как показано на рис. 1.8. Двойной щелчок в пределах формы вызовет переход на вкладку программного кода **Form1.cs**. При этом среда Visual Studio сгенерирует пустой обработчик события загрузки формы **Form_Load** (см. листинг 1.2). Таким же образом, т. е. двойным щелчком на командной кнопке в дизайнере формы, мы получим пустой обработчик события "щелчок на кнопке". Нам еще понадобится обработчик события **ValueChanged**. Чтобы его получить, перейдем на вкладку конструктора формы **Form1.cs[Design]** и в панели свойств, щелкнув на значке молнии (**Events**), для элемента **DateTimePicker1** двойным щелчком выберем событие **ValueChanged** (можно также на вкладке конструктора формы просто щелкнув двойным щелчком на элементе **dateTimePicker1**). Это приведет к созданию пустого обработчика этого события на вкладке **Form1.vb**. Добавим на вкладке программного кода недостающие команды (листинг 1.2).

Листинг 1.2. Файл Form1.cs, содержащий программный код выбора необходимой даты

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
// Программа выбора нужной даты
namespace ВыборДаты
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
        private void Form1_Load(object sender, EventArgs e)
        {
            // Обработка события загрузки формы:
            this.Text = "Средство выбора даты";
            dateTimePicker1.Format = DateTimePickerFormat.Custom;
            dateTimePicker1.CustomFormat = "ddd, dd MMM, yyyy";
            button1.Text = "Выбрать дату:";
            label1.Text = String.Format(
                          "Сегодня: {0}", dateTimePicker1.Text);
        }
        private void dateTimePicker1_ValueChanged(
                                    object sender, EventArgs e)
        {
```

```
            // Обработка события изменения даты:
            label1.Text = String.Format(
                "Выбранная дата: {0}", dateTimePicker1.Text);
        }
        private void button1_Click(object sender, EventArgs e)
        {
            // Обработка события "щелчок на кнопке":
            // Передаем фокус на элемент управления dateTimePicker1:
            dateTimePicker1.Focus();
            // Имитируем нажатие клавиши <F4>:
            SendKeys.Send("{F4}");
        }
    }
}
```

Как видно из программного кода, при обработке события загрузки формы задается нужный формат отображения даты:

- вначале (первые три буквы "ddd") — вывод дня недели в краткой форме;
- затем ("dd MMM") — числа и названия месяца, также в краткой форме;
- и, наконец, вывод года ("yyyy").

При обработке события изменения даты ValueChanged текстовой метке label1 присваиваем выбранное значение даты. При этом пользуемся методом String.Format. Использованный формат "Выбранная дата: {0}" означает — взять нулевой выводимый элемент, т. е. свойство Text объекта DataTimePicker1, и записать его вместо фигурных скобок.

При обработке события "щелчок на кнопке" вначале передаем фокус на элемент управления DateTimePicker1. Теперь для раскрытия календаря можно использовать нажатие клавиши <F4>.

> ***Пояснение***
>
> Стандартно функциональная клавиша <F4> обеспечивает раскрытие списка в приложениях Windows. Например, если в браузере Internet Explorer передать фокус щелчком мыши на адресную строку, то после нажатия клавиши <F4> раскроется список веб-страниц, которые вы уже посещали, и вам останется лишь выбрать в этом списке нужный ресурс. В данном случае мы имитируем нажатие клавиши <F4> — метод Send посылает активному приложению сообщение о нажатии соответствующей клавиши.

Пример работы программы показан на рис. 1.8.

Заметим, что подобный пример использования раскрывающегося календаря приведен в документации Visual Studio на сайте Microsoft **www.msdn.com**. Автором внесены в него лишь некоторые изменения.

Убедиться в работоспособности программы можно, открыв решение ВыборДаты.sln в папке ВыборДаты сопровождающего книгу электронного архива.

Пример 4. Ввод данных через текстовое поле *TextBox* с проверкой типа методом *TryParse*

При работе с формой очень часто ввод данных организуют через элемент управления **TextBox** (Текстовое поле). 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, если введено не число, сообщает об этом пользователю.

Решая сформулированную задачу, запускаем Visual Studio 12 и выбираем пункт меню **File | New | Project**. В открывшемся окне **New Project** выбираем пункт **Visual C#**, а затем в области шаблонов (в средней колонке) — шаблон (**Templates**) **Windows Forms Application**. В качестве имени проекта введем имя `Корень` и щелкнем на кнопке **OK**. Далее из панели элементов управления **Toolbox** (если в данный момент вы не видите панель элементов, то ее можно добавить, например, с помощью комбинации клавиш <Ctrl>+<Alt>+<x> или меню **View | Toolbox**) в форму указателем мыши перетаскиваем текстовое поле **TextBox**, метку **Label** и командную кнопку **Button**. Поместить названные элементы на проектируемую экранную форму можно также двойным щелчком мыши на каждом элементе в панели **Toolbox**. Таким образом, в форме будут находиться три элемента управления. Расположим их так, как показано на рис. 1.9.

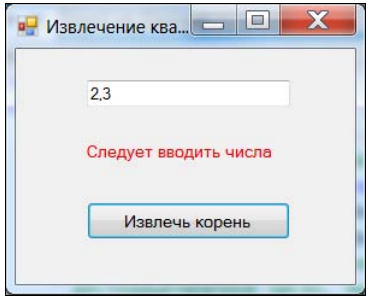

Рис. 1.9. Пример работы программы, если введено не число

Теперь нам следует изменить некоторые свойства элементов управления. Это удобно сделать при обработке загрузки формы. Чтобы получить пустой обработчик загрузки формы, двойным щелчком щелкнем на проектируемой экранной форме. Сразу после этого мы попадаем на вкладку программного кода **Form1.cs**. Здесь задаем свойствам формы (к форме обращаемся посредством ссылки `base`), командной кнопки **Button1**, текстового поля **TextBox1** и метке **Label1** следующие значения:

```
base.Text = "Извлечение квадратного корня";
button1.Text = "Извлечь корень";
textBox1.Clear(); // — очистка текстового поля
label1.Text = null; // или = String.Empty
```

Нажмем 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, двойным щелчком щелкнув

по этой кнопке на вкладке дизайнера формы. Между двумя появившимися строчками программируем диагностику правильности вводимых данных, конвертирование строковой переменной в переменную типа `Single` и непосредственное извлечение корня (листинг 1.3).

Листинг 1.3. Извлечение корня с проверкой типа методом `TryParse`

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
// Программа вводит через текстовое поле число, при щелчке на командной
// кнопке извлекает из него квадратный корень и выводит результат на метку
// label1. В случае ввода не числа сообщает пользователю об этом, выводя
// красным цветом предупреждение также на метку label1
namespace Корень
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
        private void Form1_Load(object sender, EventArgs e)
        {
            button1.Text = "Извлечь корень";
            label1.Text = String.Empty;
            // или label1.Text = null;
            base.Text = "Извлечение квадратного корня";
            textBox1.Clear(); // Очистка текстового поля
            textBox1.TabIndex = 0; // Установка фокуса в текстовом поле
        }
        private void button1_Click(object sender, EventArgs e)
        {
        Single X; // — из этого числа будем извлекать корень
        // Преобразование из строковой переменной в Single:
        bool Число_ли = Single.TryParse(
                    textBox1.Text,
                    System.Globalization.NumberStyles.Number,
                    System.Globalization.NumberFormatInfo.CurrentInfo,
                    out X);
        // Второй параметр — это разрешенный стиль числа (Integer,
        // шестнадцатеричное число, экспоненциальный вид числа и проч.).
        // Третий параметр форматирует значения на основе текущего языка
        // и региональных параметров из Панели управления — Язык и
        // региональные стандарты числа допустимого формата: метод
        // возвращает значение в переменную X
```

```
        if (Число_ли == false){
            // Если пользователь ввел не число:
            label1.Text = "Следует вводить числа";
            label1.ForeColor = Color.Red; // — цвет текста на метке
            return; // — выход из процедуры или Return
        }
        // Извлечение корня с преобразованием в тип Single:
        var Y = (Single)Math.Sqrt(X);
        // или var Y = Convert.ToSingle(Math.Sqrt(X));
        label1.ForeColor = Color.Black; // — черный цвет текста на метке
        label1.Text = String.Format("Корень из {0} равен {1:F5}", X, Y);
        }
    }
}
```

Здесь при обработке события "щелчок мышью" на кнопке **Извлечь корень** проводится проверка, число ли введено в текстовом поле. Проверка осуществляется с помощью функции `TryParse`. Первым параметром метода `TryParse` является анализируемое поле `TextBox1.Text`. Второй параметр — это разрешаемый для преобразования стиль числа, он может быть целого типа (`Integer`), шестнадцатеричным (`HexNumber`), представленным в экспоненциальном виде, и проч. Третий параметр указывает, на какой основе формируется допустимый формат, — в нашем случае мы использовали `CurrentInfo`, т. е. на основе текущего языка и региональных параметров.

> ***Примечание***
>
> По умолчанию при инсталляции русифицированных версий Windows XP и Windows 7 разделителем целой и дробной части числа принимается запятая. Однако эту установку можно изменить, если в Панели управления выбрать опцию **Язык и региональные стандарты**, затем на вкладке **Форматы** щелкнуть на кнопке **Дополнительные параметры** и в открывшейся вкладке указать в качестве разделителя целой и дробной частей вместо запятой — точку. В обоих случаях (и для запятой, и для точки) метод `TryParse` будет работать так, как указано на вкладке **Форматы**.

Четвертый параметр метода `TryParse` возвращает результат преобразования. Кроме того, функция `TryParse` возвращает булеву переменную `True` или `False`, которая сообщает, успешно ли выполнено преобразование. Как видно из текста программы, если пользователь ввел не число (например, какие-либо буквы), то на метку `Label1` выводится красным цветом текст "Следует вводить числа". Далее, поскольку ввод неправильный, организован выход из программы обработки события `Button1_Click` с помощью оператора `return`.

На рис. 1.9 показан пример работы программы как раз в этом случае. Из рисунка видно, что функция `TryParse` не восприняла введенные символы `2,3` как число, поскольку автор специально для демонстрации данного примера указал на вкладке **Форматы** точку в качестве разделителя целой и дробной частей.

Если пользователь ввел все-таки число, то будет выполняться следующий оператор извлечения квадратного корня `Math.Sqrt(X)`. Математические функции Visual Studio 12 являются методами класса `Math`. Их можно увидеть, набрав `Math` и введя так называемый оператор разрешения области действия (`.`). В раскрывающемся списке вы увидите множество математических функций: `Abs`, `Sin`, `Cos`, `Min` и т. д. и два свойства — две кон-