Лицей информационных технологий

Мартин Дрейер

C# для школьников

УЧЕБНОЕ ПОСОБИЕ

Книга выпускается при поддержке компании Microsoft

Microsoft®

www.microsoft.ru
www.dreamspark.ru

Интернет-Университет
Информационных Технологий
www.intuit.ru

БИНОМ.
Лаборатория знаний
www.lbz.ru

Москва
2009

УДК
ББК
Д

Дрейер М.

Д С# для школьников: Учебное пособие / М. Дрейер. Перевод с англ. под ред. В. Биллига— М.: Интернет-Университет Информационных Технологий; БИНОМ. Лаборатория знаний, 2009. — 128 с.: ил., табл. — (Лицей информационных технологий).

ISBN (БИНОМ.ЛЗ)

Эта книга знакомит юного читателя (предполагаемый возраст — 12–16 лет) с объектно-ориентированным программированием, и автор предлагает начать обучение с реальных программ в среде Windows.

Новый язык программирования Microsoft C# (произносится «си-шарп») — очень мощный и в то же время простой в использовании. Он высоко ценится современными ИТ-специалистами и является хорошим выбором для тех, кто только ступает на путь программирования.

УДК
ББК

По вопросам приобретения обращаться:
«БИНОМ. Лаборатория знаний»
Телефон (499) 157-1902, (499) 157-5272,
e-mail: Binom@lbz.ru, http://www.Lbz.ru

ISBN (БИНОМ.ЛЗ)

Предисловие редактора перевода

Читать, писать и считать мы учим детей, едва им исполнится 7 лет. Многие овладевают этими навыками значительно раньше – чтение в 4 года совсем не редкость. Относится ли и программирование к так называемым базисным умениям? Можно ли учиться этой премудрости с 7 лет? Полагаю, что в ближайшие десятилетия так и будет: программирование, как вторая грамотность, станет повсеместным явлением.

Знакомый мне победитель международной школьной олимпиады по информатике, теперь уже студент, свои первые программы написал еще во втором классе. Впрочем, не он один. По этой дороге идут многие школьники, и число их растет.

Когда маленькие девочки, играя в куклы, дают им имена — это Маша, а вот это Даша, — то, по сути, они занимаются именованием объектов, совсем как в объектно-ориентированном программировании. И говоря кукле Маше: «Если будешь себя хорошо вести, то я одену тебя в новое платье», они используют базисную конструкцию языков программирования — оператор If – then, а напоминая кукле Даше о том, что «уже 12 часов дня — пора спать», создают события и выполняют их обработку. В мире маленьких мальчиков работа с объектами идет столь же интенсивно.

Сегодня компьютер есть в каждом доме, и вполне обычное дело, когда ребенок, еще не умеющий толком читать, нажимает на кнопки, легко справляясь с компьютерными играми. Игры — хорошее дело, я и сам иногда увлекаюсь ими. Но если компьютер используется только для игр – это беда, болезнь.

Компьютер — инструмент для созидания. Чем раньше мы будем учиться созиданию, тем лучше. Быть потребителем готовых продуктов несложно, но чрезмерное потребление просто вредно.

Книг, предназначенных для юных программистов, совсем немного. И Мартин Дрейер написал одну из них. В ней есть забавные картинки, многочисленные примеры, читать ее интересно и полезно, даже тем, кто вовсе не собирается стать программистом.

Привлекая аналогии из реального мира, Мартин Дрейер рассказывает об основных понятиях объектно-ориентированного программирования — классах, полях, методах и событиях класса. Особенностью книги является то, что повествование о фундаментальных понятиях сочетается с применением самых современных технологий промышленного программирования. Язык программирования C#, среда разработки Visual Studio .Net, база данных SQL Server — все это новые технологии, созданные фирмой Microsoft.

Фундаментальность и новые технологии — это правильный подход к обучению программированию.

Интересны и предлагаемые примеры. Все начинается, как обычно, с простых иллюстраций. Однако уже к середине книги появляются «настоящие» приложения, а к концу, где речь идет о работе с базами данных, для их понимания требуются профессиональные знания. Такие примеры демонстрируют возможности того, что можно делать программно.

Трудно, однако, ожидать, что школьник, прочитав эту книгу, сможет написать собственный пример работы с базой данных или с XML-документом, но если у него возникнет желание погрузиться в глубины сказочно интересного мира программирования, цель, которую ставил автор, будет достигнута.

Редактор перевода:
профессор кафедры информатики Тверского государственного университета
Владимир Биллиг

Введение

Эта книга знакомит юного читателя (предполагаемый возраст — 12–16 лет) с объектно-ориентированным программированием, и автор предлагает начать обучение с реальных программ в среде Windows.

Новый язык программирования Microsoft C# (произносится «си-шарп») — очень мощный и в то же время простой в использовании. Он высоко ценится современными ИТ-специалистами и является хорошим выбором для тех, кто только ступает на путь программирования.

Об авторе

Мартин Дрейер (Martin Dreyer) — в прошлом школьный учитель, сейчас работает в ЮАР, возглавляет группу разработчиков программного обеспечения. Имеет диплом о высшем образовании по направлению «Естественные науки» и степень бакалавра по направлению «Вычислительная техника и информационные системы».

Часть 1. Первое знакомство

Начнем скорее

Мне кажется, я знаю, о чем подумали, пролистав первые страницы этой книги:«Хочу написать программу! И как можно скорее! Самое интересное – программировать. Не стоит тратить время на утомительное чтение – хватит слов, пора перейти к делу!»

Честно говоря, мне давно хочется познакомиться с начинающим программистом, который смог бы сопротивляться такому желанию. Ведь как обычно бывает? Представьте: вам купили новый велосипед, и что же вы сделаете? Приметесь читать инструкцию? Конечно, нет. Вскочите на своего коня, понесетесь, расшибете лицо и только потом обнаружите: тормоза-то у этой модели устроены совсем не так, как у других велосипедов!

Но ведь мы хотим добиться успеха? Поэтому давайте договоримся: будем набираться знаний постепенно, с самой первой главы, а когда установите программу на компьютер, попробуйте выполнить некоторые примеры и даже изменить их, если хватит смелости.

Хочу предупредить: первые программы покажутся вам исключительно трудными, как езда вслепую. Ведь даже из-за небольшой ошибки работа зайдет в тупик, а для решения проблемы нужны знания. Вот тогда-то станет понятно: пришло время заняться теорией, как это делают профессионалы. Не унывайте и не сдавайтесь, если что-то не будет получаться. И найдите, пожалуйста, время, чтобы дочитать книгу до конца.

Очень важная часть

В программировании, как и во всех других профессиях, наибольшего успеха добиваются люди, любящие свою работу. Если вы получаете удовольствие от того, чем занимаетесь, то непременно захотите изучать новые возможности и двигаться вперед. В таком случае, разрабатывая новые проекты, вы будете набираться знаний и опыта.

Итак, самое главное: **наслаждайтесь** освоением мира программирования C#!

Для разработки программ на C# вам понадобятся:

1. Пакет Microsoft .NET Framework 3.5, который можно бесплатно загрузить на странице http://msdn.microsoft.com/ru-ru/netframework/aa569263.aspx.
2. Среда разработки .NET Framework. Мы рекомендуем версию Microsoft Visual C# Express Edition, ее бесплатная загрузка предлагается на сайте Microsoft: http://www.microsoft.com/rus/express/vcsharp. Пакет установки Visual C# 2008 Express включает .NET Framework, поэтому устанавливать приложение, указанное в пункте 1, не придется.
3. И наконец, нужно скопировать файлы примеров из книги на диск своего компьютера. Распакуйте файл examples.zip, сохраните его в любой папке и запомните, где она расположена – находящиеся в ней файлы вам понадобятся позже.

Дополнительная информация о Microsoft Visual C# Express Edition

Программы на C# можно писать в простом текстовом редакторе (например, в Блокноте) и компилировать при помощи так называемого «Компилятора C#», входящего в состав .NET Framework. Однако мы советуем использовать Visual C# Express – упрощенную среду для разработки приложений для Windows и консольных приложений на языке C#. Она поможет сделать первые шаги в постижении программирования, а после прочтения книги вы сможете двигаться дальше, постепенно усложняя задачи.

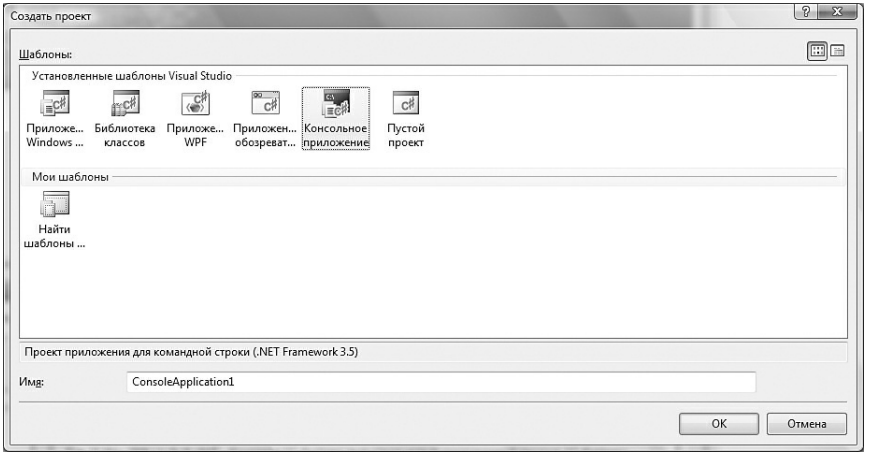
Важно понять, что эта книга **не предназначена** для обучения работы в Visual C# Express Edition.

В Visual C# Express есть множество замечательных функций, таких, например, как автоматическое написание кода C# при перетаскивании на форму какой-либо кнопки или другого элемента управления. Это замечательная возможность и ею можно пользоваться, но **цель нашей книги — обучить языку C#,** причем с самых его основ. Сначала мы хотим научить вас писать код C# вручную, только при таком условии можно разобраться в его устройстве.

Дополнительная информация о среде разработки Visual C# Express опубликована на сайте компании Microsoft: http://www.microsoft.com/rus/express/vcsharp.

Создание новой программы (проекта) в Visual C# Express

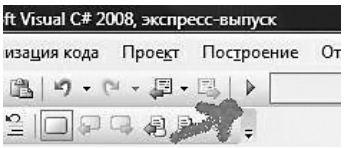
- Чтобы запустить Visual C# Express, в меню «Пуск» укажите «Все программы» и затем выберите «Microsoft Visual C# Express Edition». Если на рабочем столе уже есть нужный вам значок (ярлык), просто щелкните по нему.
- Для создания нового проекта откройте меню File («Файл»), щелкните New project («Новый проект») и выберите тип проекта. Мы начнем с нескольких консольных приложений, а позднее перейдем к приложениям для Windows.



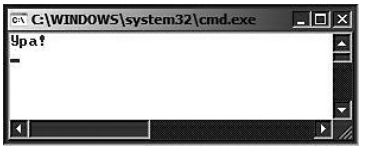
- Попробуем создать новое консольное приложение и нажмем «OK». Откроется студия разработки с множеством окон. Центральное окно предназначено для программирования, здесь будет показан автоматически созданный код класса **Program.cs**. Удалите его полностью и наберите текст программы, приведенной ниже. Помните наши договоренности: не скопируйте, а именно наберите – вручную. *Разумный* человек сумеет справиться с *ленью*!

```
using System;
class PleaseSayUra
{
    static void Main()
    {
        Console.WriteLine("Ура!");
        Console.ReadLine();
    }
}
```

Убедитесь, что набранный вами текст В ТОЧНОСТИ соответствует представленному в книге. Затем нажмите кнопку «Выполнить» (или клавишу F5).

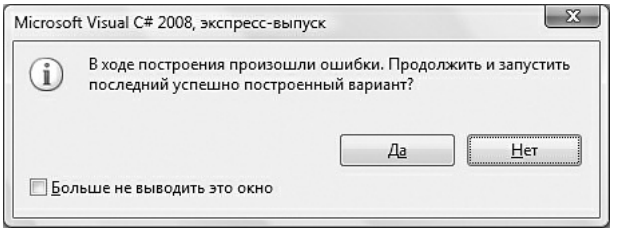


- Если все сделано без ошибок, откроется окно консоли с текстом «Ура!».



Для завершения программы надо нажать на кнопку «X» в верхнем правом углу окна или на любую клавишу.

- Если в коде программы обнаружится ошибка, вы получите предупреждение. Обратите внимание на то, что в языке программирования C# в конце каждого предложения или выражения должна стоять точка с запятой. Попробуем удалить последнюю точку с запятой («;») и затем выполнить программу – на экране появится следующее сообщение об ошибке:



В большинстве случаев нужно нажать на кнопку «Нет», и тогда в окне «Список ошибок» вы увидите некоторые сведения, включая указание номера строки, в которой ошибка была найдена. Щелкните дважды на это описание, чтобы перейти к нужной строке и исправить ошибку.

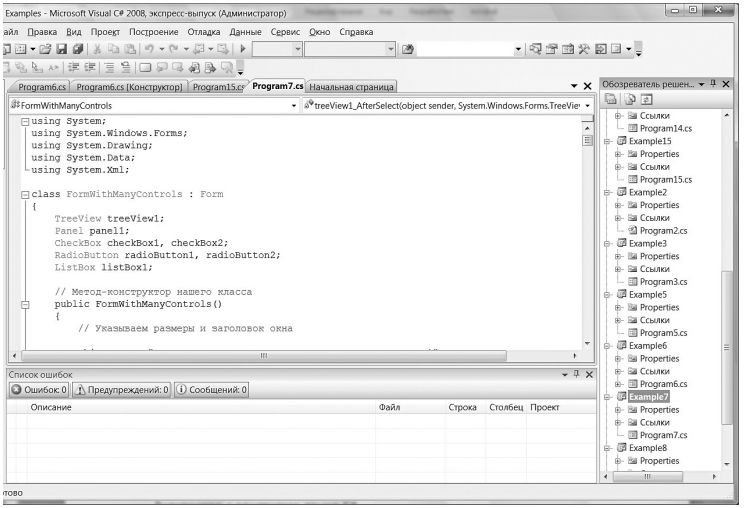
Выполнение примеров программ, прилагаемых к книге

Эта книга построена на изучении примеров программ – от самых простых (наподобие программы «Ура», код которой был только что приведен) до сложных, работающих с базами данных. Каждый пример реализован как отдельный проект. Для удобства все проекты объединены в одно решение (Solution) и находятся в общей папке с именем Examples, которая хранится в архивированном виде Examples.zip на диске, прилагаемом к книге. Скопируйте эту папку на жесткий диск вашего компьютера и разархивируйте ее, чтобы получить папку Examples, где и лежат папки и несколько файлов. Папка с именем databases хранит базы данных, необходимые для некоторых проектов. Папки с именами Example1, ..., Example15 содержат отдельные проекты на C# – именно эти примеры обсуждаются в книге. Каждая папка имеет сложную внутреннюю структуру, состоящую из других папок и файлов.

В папке Examples находится центральный для нас файл Examples.sln, где хранится решение с проектами. Запуск любого проекта будем выполнять с его помощью: если щелкнуть по этому файлу, то решение со всеми проектами будет загружено в студию разработки. Теперь можно запустить любой пример. Для этого достаточно:

- Выбрать нужный вам проект из тех, что показаны в окне студии: если, скажем, нужен пример с номером 10, выберите проект с именем Example10.
- Установить курсор на имени проекта, нажать правую кнопку мыши и в появившемся контекстном меню выбрать пункт «Назначить запускаемым проектом» (Set as StartUp Project).
- Запустить проект на выполнение. Это можно сделать разными способами: если нажмете на клавишу F5, проект начнет работать в отладочном режиме, а комбинация клавиш Ctrl+F5 запустит проект без отладки. Другие варианты: выбрать в меню Debug соответствующие пункты или щелкнуть мышью по специальной кнопке «Выполнить».

На рисунке показан внешний вид студии разработки, который формируется в процессе работы с представленными в книге примерами:



Знакомство с понятиями языка C#

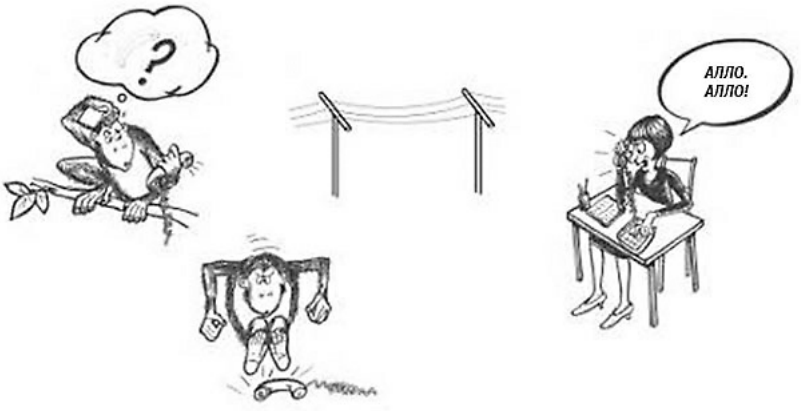
Во II части книги дается общее описание ключевых понятий. Такое краткое изложение называется «строительным блоком». Рассмотрим пример строительного блока.

Строительный блок: Классы	

Все программы на языке C# создаются внутри классов. Основная структура класса имеет следующий вид:	`class Animal`
	`{`
	`}`

Возможно, вы захотите ознакомиться с возможностями Visual C# Express, просмотреть файлы, содержащиеся в разделе «Справка», и попробуете понять, что же собой представляет эта программа.

Совсем скоро вы убедитесь, что для диалога с компьютером одних лишь средств программирования недостаточно — нужно нечто большее.



Поэтому настало время перейти к следующей части книги, чтобы *учиться общению* с компьютером на языке программирования C#.

Часть 2. Учимся общаться с компьютером

Люди и компьютеры

Написание программ осложняется тем, что компьютер совсем не похож на людей и «воспитание» у нас разное. Если бы это было не так, то и компьютер, и мы одинаково бы воспринимали окружающий мир и умели бы говорить на одном языке. Тогда нам было бы легче понимать друг друга: достаточно попросить компьютер нарисовать кота — и все.

Вы скажете: «В чем же дело? Давайте научим компьютер понимать нас!». Но эти машины устроены совсем не так, как человеческий мозг, и «видят» мир иначе, поэтому обучить их мышлению, подобному нашему, — непростая задача.

Впрочем, выход есть: надо найти то, что поможет прийти к согласию:

- общий «взгляд на мир» и
- общий язык, который способны понимать как люди, так и компьютеры.

Эта часть книги полностью посвящена изучению способов общения с компьютером. Надо только «развернуть» свое сознание, чтобы увидеть вещи так, как будто вы находитесь в роли компьютера. При этом придется пойти на взаимные уступки: компьютер сможет научиться многому из того, что делаем мы (например, запомнит некоторые слова), но и вы должны будете уяснить некоторые особенности его мира. Попытайтесь забраться в «голову» компьютера и представить, как можно было бы изложить мысли.

Компьютеры не могут догадываться о том, что мы думаем, чего от них хотим и каким образом они могут выполнить наши желания. Остается только мечтать о том дне, когда они будут откликаться на один лишь взгляд и движение бровей в их сторону, но сейчас приходится четко излагать свои идеи и даже давать компьютерам указания, которым они обязаны следовать. Короче говоря, взаимопонимания между людьми и машинами до сих пор нет, и если мы хотим добиться от них помощи, то придется преодолеть эту пропасть.

Итак, **давайте ненадолго забудем о компьютерах** и представим, что вам крайне необходимо организовать общение с неким пришельцем. А история такова...

- К счастью или нет, но что-то произошло во Вселенной, и вы оказались на маленьком пустынном острове посреди океана один на один со странным существом. После отчаянного «Привет, меня зовут Света, а тебя?», вы, в конце концов, решаете дать своему спутнику имя – Алджи.

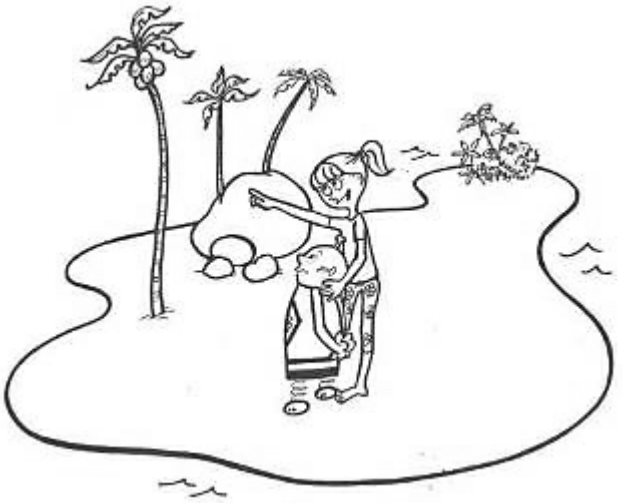
- Подступивший голод дает о себе знать. По счастью на высокой пальме висят несколько кокосов, и Алджи мог бы достать их. Вот только как объяснить ему, что надо сделать? На песке вы рисуете изображение кокосовой пальмы и, указывая на него, произносите слово «ко-кос». Вас поражает, что Алджи моментально реагирует на этот рисунок, и от-

лично воспринимает ваши дальнейшие объяснения. Обнаруживается, что у этого создания талант к языкам. До наступления сумерек Алджи овладевает русским.

- Волнуясь от осознания собственного успеха, вы решаетесь попросить: «Алджи, пожалуйста, подними меня на плечи, чтобы я могла достать кокосы!»
  Он смотрит на вас озадаченно. Затем с таким же видом переводит взгляд на изображение кокосовой пальмы. Потом опять на вас. И снова на рисунок. Бесполезно — он не понимает! Осознав свою ошибку, вы хватаетесь за голову и растерянно показываете на *настоящую* пальму. Но Алджи по-прежнему не может понять. Оказывается, он понимает только рисунки. Ага! Теперь-то становится ясно, что, хотя Алджи и может использовать ваш язык, он фактически *НЕ понимает* реальных вещей, о которых вы говорите.

- Каким-то образом надо прийти к общей точке зрения. На первый взгляд, все просто: раз удалось научить языку, то можно обучить и правильному пониманию объектов реального мира. Но и через трое суток кокосы по-прежнему висят на пальме. Уроки оказываются бесполезными, и вы теряете контроль над ситуацией по мере того, как перестаете понимать, в чем же дело.

  Вы не знаете, сон это, галлюцинации или видение, но в ярком, туманном свете появляется фигура в белом. Низкий голос подсказывает: «Вот отличный способ восприятия окружающего мира! Только так вы и Знифф12 (вот оно – настоящее имя Алджи!) сможете понять друг друга». Неизвестно сколько времени проходит, пока образы магически проникают в ваш разум, обучая новому способу описания мира. Наконец, успокоившись, вы выбираетесь из этого состояния.
- Встрепенувшись, хватаете Алджи за плечи: «Я все поняла, приятель, слушай внимательно то, что я тебе скажу». Алджи с нахмуренным видом вслушивается в вашу речь.



- «В окружающем нас мире существуют различные классы объектов — растения, животные, голодные люди, инопланетяне и так далее».
- «Сейчас я хотела бы рассказать тебе об объектах, существующих на этом острове. Прежде всего, если ты еще не заметил, меня очень интересует эта кокосовая пальма (между прочим, она входит в **класс**, называемый «растениями»). Мне интересен и вон тот слон. Ух, ты! Странно, что я не заметила его раньше». (Вы указываете на каждый объект, о котором говорите.)
- «Слон входит в **класс** под названием «животные» и имеет некоторые важные для нас свойства. Во-первых, все животные могут *двигаться*. Во-вторых, слон очень *тяжелый*, наверное, тонны две – это нам пригодится. В-третьих, у него скверный характер, что тоже может оказаться полезным».

- «Ну, а теперь, Алджи, слушай меня... В окружающем нас мире непрерывно происходят события. Я проголодалась — это событие. Кокосы созревают — и это событие. Иногда мы сами можем вызывать события, чтобы выполнить некоторое действие. У меня есть отличная мысль: давай вызовем событие «*беспокойство слона*», которое заставит его помочь нам выполнить действие «*добывание кокосов*».
- «Разделим обязанности. Я вызову **событие** (привлеку к тебе внимание слона), а ты заставишь его достать для нас кокосы, договорились?» *При этом Алджи робко смотрит на вас и признается:* «Света, я слышу твои слова, но не понимаю, *как* это сделать». Похлопывая товарища по спине, вы говорите: «Не беспокойся, я никогда не стану просить тебя о чем-либо, сначала не объяснив тебе метода выполнения этого действия. Вот что нужно будет сделать после того, как я скажу: «Достань кокосы»:
  - Выбеги из-за этого камня и встань рядом с пальмой. Слон немедленно понесется к тебе.
  - В тот момент, когда он достигнет нарисованной мной на песке линии, беги изо всех сил (поворачивай в разные стороны, бегать взад и вперед не надо).
  - Он ударится о пальму и на некоторое время оцепенеет, но силой удара кокосы будут сбиты, и мы позже сможем собрать их.
  - После этого возвращайся назад и сообщи результат».

- Алджи надолго замирает, переваривая задание, потом неожиданно достает суперсовершенное устройство связи, в течение некоторого времени издает звуки, напоминающие верещание модема, и затем снова кладет устройство в карман. Вы спрашиваете: «Что это было, Алджи?». Он отвечает: «Позволь мне перевести, Света...
- Существует **класс** объектов, называемых Икс-зниффферами.

- Я один из **объектов** такого класса, но есть и другие.
- Все объекты этого класса обладают следующими **свойствами**:
  – name (имя, например, мое имя — Знифф12);
  – earShape («форма_уха», значение этого свойства — «трубка» — всегда постоянно);
  – likeToWatchTV («подобно_просмотру_ТВ», в некоторых случаях имеет значение истины, а в других — лжи).
- Должен признаться, мое появление здесь не случайно. Ты попала в популярное ТВ-шоу «Остаться в живых», а минуту назад я позвонил своему продюсеру и сказал, что меня не удовлетворяет обещанная плата за такое опасное задание. Он собирается немедленно выполнить **метод** «ОтправитьСпасательнуюГруппу».

На этом история заканчивается. Смысл в том, что вам с Алджи удалось разработать общий способ описания мира: вы используете одни термины — класс, объект, метод, свойство — и одинаково понимаете их значение. Это, наряду с использованием единого языка, дает вам возможность общения.

Теперь представим, что вам нужно объясниться с компьютером. Решение будет аналогичным рассказанной истории, и проблему можно решить путем согласования общего способа восприятия мира, который сможет обеспечивать взаимопонимание человека и машины.

Классы и объекты в языке C#

Подход с использованием «классов, объектов, свойств, событий и методов» (о нем и было рассказано в истории) не является единственно возможным. Это только один из способов, который поможет объяснить пришельцу, как устроен окружающий нас мир. Крайне важно договориться о том, что вы будете описывать его при помощи одинаковых образов. Такой взгляд на мир (кстати, он называется «объектно-ориентированным» подходом) стал очень популярным способом описания вещей, и фактически язык C# требует придерживаться именно его.

Итак, изучение языка C# — это изучение способов описания вещей при помощи классов, объектов, событий, методов и т. д.

Если вы хотите, чтобы компьютер выполнил определенные действия, то вам придется сделать следующее:

- выучить язык C# и научиться описывать вещи с ориентацией на объекты;
- установить на компьютере приложение C# и другие сопутствующие программы. Тогда **компьютер** сможет понимать язык C# и работать с объектами, описанными в соответствии с объектно-ориентированным подходом.

Вот и все: общее представление о мире и общий язык. Вскоре будете общаться с компьтером, как старые приятели.

Теперь необходимо понять, что собой представляет ориентация на объекты с точки зрения программиста.

- Окружающий нас мир можно разделить на различные классы вещей: например, «коты», «дома» или «деревья». Классы можно считать именованными категориями, позволяющими группировать сходные объекты. Надо найти такой способ описания классов, который позволял бы компьютеру оперировать с классами и выполнять действия с их объектами.

- Каждый **класс** вещей включает нужные нам объекты. Например, к классу «кот» относится мой кот Пушок – он является объектом класса котов. Дом на углу моей улицы не что иное, как объект класса «дом», а дуб, видимый из окна спальни, – объект класса «дерево». Если есть определенный объект, с которым должен работать компьютер, то необходимо создать некоторое описание, представляющее такой объект.
- Для объектов характерны различные свойства, которые помогают четко описать объекты. Их можно сравнить с прилагательными в русском языке. Например, моего большого упитанного Пушка можно описать такими свойствами, как *рост*, *вес* и т. д.
- События — это некоторые особые состояния, в которые может попадать объект. В реальном мире события происходят вокруг нас непрерывно. Например, когда я глажу Пушка, для него наступает приятное событие, которое можно назвать «Нирвана». Если налетает сильный ветер, ломающий ветви уже упоминавшегося здесь дуба, то с деревом происходит событие под названием «Ураган». При возникновении события, как правило, выполняются определенные действия, влияющие на состояние объекта. Действия по обработке события выполняются не самим объектом, а другими объектами, которые могут получать сообщение о возникающих событиях. При работе с компьютерным приложением, где есть видимые объекты, например, компьютерные игры, событиями могут быть «нажатие на кнопку», «нажатие на определенную клавишу» или «щелчок кнопки мыши».
- Объект может выполнять действия. Например, кот Пушок лижет свою шерсть — это действие. В компьютерном мире определенные действия выполняются, когда происходят интересные события, например, при нажатии на кнопку выполняется действие «рисуем кота».
- Хотите верьте, хотите нет, но компьютер может не иметь представления о том, как надо рисовать кота. В этом случае необходимо дать описание, которое будет ему понятно, — метод рисования кота, это набор пошаговых инструкций, определяющих порядок выполнения конкретных действий.

В последующих разделах мы рассмотрим каждый аспект объектно-ориентированного программирования и приведем более детальное описание. Начинаем учиться программировать на языке C#.

В окружающем нас мире можно выделить различные группы вещей. Назовем их классами.

Вспомним, как создаются классы в школе. Детей определенного возраста или с определенными способностями объединяют в группу и называют ее классом. Такой подход позволяет уравнять их в правах: все будут изучать одни и те же предметы и сдавать одни и те же экзамены.

Но это не означает, что все ученики класса одинаковы, у них всего лишь имеются некоторые общие особенности.

Рассмотрим другой пример: в группе живых организмов можно выделить два класса — «растения» и «животные».

Теперь попробуем научиться выражать свои мысли так, чтобы *компьютер* смог понять то, что мы пытаемся сказать.

В C# описание класса для компьютера может иметь следующий вид:

```
class Animal
{
}
```

Из этого небольшого отрывка кода компьютер понимает следующее:

- мы собираемся изучать **класс** объектов;
- мы назвали этот класс **Animal**;
- описание этого класса будет расположено между фигурными скобками **{ ... }**.

Класс «Animal» имеет смысл определить в программе, выводящей на экран изображения животных или сохраняющей интересные факты о них. При решении других задач надо будет описывать другие классы, например, классы «MyPhotoProgram» или «MyGradeCalculator». Нужно понимать, что классы объединяют не только те объекты, которые существуют *в реальном* мире. Они полезны и при работе с абстрактными объектами.

Строительный блок: Классы	

Все программы на языке C# представляют собой описание множества **классов**.
Компьютеру известно слово «class», которое должно быть написано строчными буквами, но имя класса может быть любым и содержать как прописные, так и строчные буквы, причем использование пробелов не допускается.
	`class Animal`
	`{`
	`}`

Немного позже мы расскажем о том, что происходит *внутри* описания класса (между фигурными скобками). Но сначала рассмотрим объекты.

Каждый класс описывает, как устроены объекты класса. В процессе работы с классом объектов «Animal» могут появиться и другие:

- кот Барсик, принадлежащий вашей тете;
- безымянная корова, увиденная вами всего один раз;
- горилла по прозвищу Джереми;
- другие объекты.

Создание объектов

Вы уже знаете, как вести разговор с компьютером о классе животных. Возможно, в какой-то момент вам захочется поговорить и об объектах. Например, о том, что «есть одно существо в классе животных, которого зовут Барсиком».

Это можно выразить следующим образом:

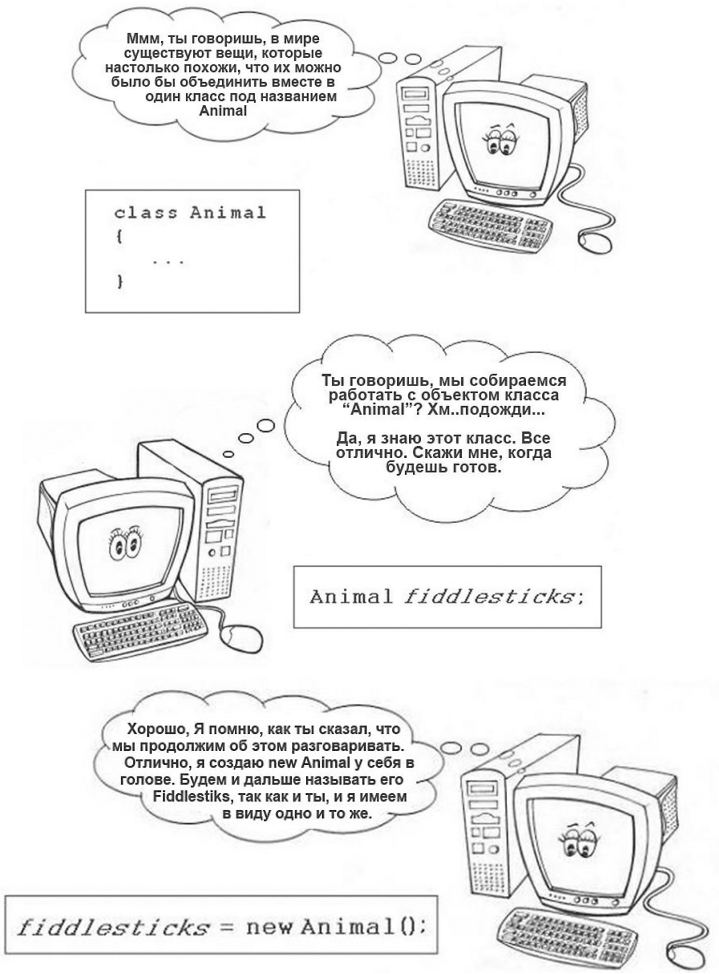
```
Animal Barsik;
```

Мы написали небольшой отрывок кода, и здесь нужно остановиться и подумать, иначе можно упустить главное. Компьютер понимает следующее: «Этот человек хочет поговорить об объекте в классе «Animal» и назвать его «Barsik». Я не знаю, что такое Barsik, и мне все равно – пусть называет, как хочет».

Теперь дополним приведенный выше код еще одним выражением:

```
Barsik = new Animal();
```

Может показаться странным, но нужно сделать именно так, потому что зрения компьютера в этом есть смысл. Подумайте «как машина» и представьте себе следующее:



- Первое выражение (Animal Barsik) предупреждает компьютер, что вскоре мы попросим его работать с объектом из класса «Animal». Теперь у компьютера появляется возможность проверить свои знания об этом классе. Он просматривает всю программу и ищет нужное описание. Не найдя такого класса, он сообщит вам, что сделать ничего не сможет. Если же отыскать нужное описание *удастся*, компьютер продолжит выполнение программы, ожидая вашей дальнейшей просьбы совершить некоторое действие с объектом класса «Animal» по имени Barsik.
- Второе выражение (Barsik = new Animal();) напоминает компьютеру: «Помнишь объект Barsik? Теперь мне нужно выполнить с ним действие. Выдели немножко памяти в своей голове, где ты хранил бы данные об этом животном». Компьютер запоминает новый объект класса «Animal» и понимает, что это объект Barsik, упомянутый ранее. Изучив класс «Animal», компьютер кое-что знает о Барсике и держит эту информацию наготове, понимая, что может получить от вас дополнительные сведения об этом объекте.

Теперь рассмотрим все три момента и представим себе, как компьютер «думает» о них.

Заметьте, в первом блоке мы описываем класс объектов в общем, а в блоках 2 и 3 выделяем определенный объект в классе Animal.

//Примечание редактора. По-английски fiddlestick означает «смычок», а fiddlesticks — «вздор, чепуха». Но, как уже говорилось, имя может быть любым, в том числе и «чепушинка».

Строительный блок: Объекты

Объекты — это отдельные, четко обозначенные элементы некоторого класса.
Класс дает общее описание объектов, указывает «на что они похожи».
	`class Animal`
	`{`
	`}`

Рассмотрим процесс объявления и создания объектов класса.	
Если планируется работать с определенным объектом, необходимо заранее сообщить компьютеру, к какому классу он принадлежит, для того чтобы компьютер проверил свои знания о нужном нам классе. Поэтому сначала указываем имя класса, а затем даем имя объекту.	`Animal percyThePorcupine;` `//ПерсиДикообраз`
Этот процесс называется «объявлением объекта». Затем необходимо попросить компьютер «создать экземпляр» класса.	
На основе своих знаний о классе компьютер создаст реальный объект, с которым он сможет выполнять действия. В нашем примере «percyThePorcupine» становится экземпляром «Animal», или можно сказать, что	`percyThePorcupine =` `new Animal();`

percyThePorcupine — это «объект» типа «Animal». Но в любом случае нам уже точно известно, что мы говорим об определенном объекте класса «Animal». Это называется «созданием объекта».	

Теперь мы знаем, как рассказать об определенном объекте в классе, но поскольку класс, который мы рассматриваем, достаточно прост (можно сказать, он совсем пустой), то работать с ним неинтересно. Итак, в следующем уроке мы добавим к нему некоторые элементы.

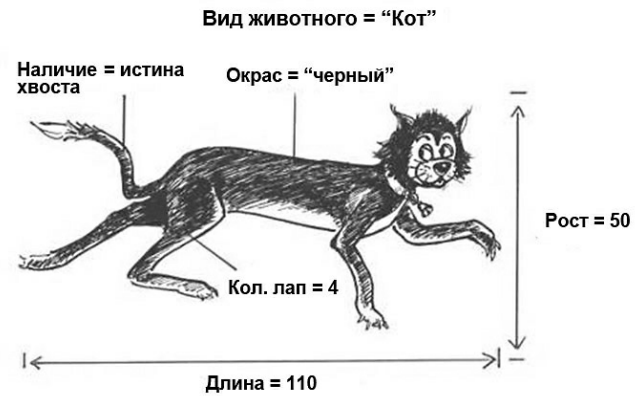
Свойства объектов

Если бы я попросил вас перечислить несколько свойств животных, то список мог быть следующим:

- Вид животного (Kind of animal).
- Рост (Height).
- Длина (Length).
- Количество лап (Number of legs).
- Окрас (Color).
- Наличие хвоста (Has a tail).
- Является ли млекопитающим (Is a mammal).

При рассмотрении свойств этих животных вы можете присвоить **значения** каждому из них:

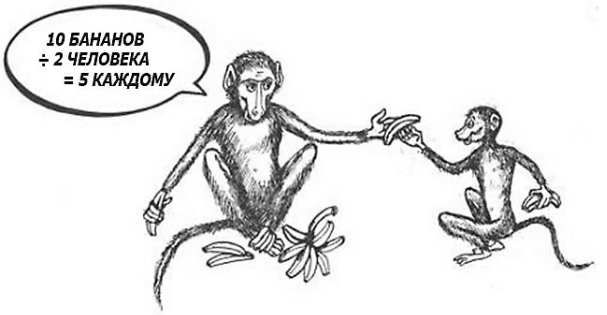
- Kind of animal = «Cat» («Кот»).
- Height = 50 cm (большой котяра!).
- Length = 110 cm (это рысь скорее, а не домашняя кошка!).
- Number of legs = 4.
- Color = «Black» (черный).
- Has tail = true (истина).
- Is mammal = true (истина).



В C# эти свойства называются полями. (Слово «свойства» используется для несколько иной цели, но здесь мы не станем вдаваться в подробности.)

Теперь рассмотрим нечто, о чем в реальной жизни беспокоиться не приходится, но в мире компьютеров оказывается очень важным. Обратите внимание: все указанные выше свойства кажутся однородными, однако фактически существуют разные типы полей. Приведем несколько примеров.

- Значения полей *height*, *length* и *number of legs* являются **числами**. Как правило, компьютер использует эти значения для выполнения математических операций.



Эти поля можно было бы назвать так: «numberOfBananas» («количествоБананов»), «numberOfPeople» («количествоЛюдей») и «numberEachOneGets» («количествоПолучаемоеКаждым»). Они должны иметь числовой тип, поскольку с их помощью мы выполняем некие математические вычисления.

Примечание. В языке C# имена полей не могут содержать пробелы. Используйте в именах только буквы и числа, причем первой всегда ставится буква.

- *Поля kindOfAnimal* и *color* относятся к строковому типу. Значением таких полей может быть любой текст. В нашем примере значения задаются словами «cat» и «black», заключенными в кавычки.

Надо отметить, что в строковом поле (в строке) текст может содержать числа. Например, в тексте «Сегодня это уже 17-й крокодил, которого я вижу!» присутствуют буквы, пробелы, числа и знаки препинания, но с числами, находящимися в строке, нельзя выполнять математические операции.

- Поле типа *hasTail* всегда будет принимать значение ИСТИНА или ЛОЖЬ (**true/false**), точно так же, как любые поля типа ДА или НЕТ («yes/no»). Такие поля относятся к логическому типу.

Каждый «тип» обрабатывается компьютером особым образом, поэтому при определении полей мы должны указывать их тип. В одной программе мы определим поле *numberOfLegs* как строку, поскольку мы не планируем выполнять математические операции над этим полем, а в другом случае, когда надо будет узнать, сколько конечностей у 500 одинаковых объектов, может потребоваться использование такого поля в качестве числового.

Тип «String» (строка)

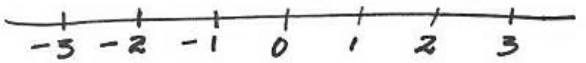
Для того чтобы попросить компьютер рассматривать некое свойство как строку, состоящую из букв, достаточно просто объявить объект следующим образом:

```
string kindOfAnimal;
```

Пока мы не задаем значение в этом поле, а всего лишь сообщаем, с каким типом данных будем работать, и присваиваем полю имя. Поставьте себя на место компьютера и представьте, как бы вы отреагировали, если бы человек написал для вас следующее:

Числовые типы

Извещая компьютер о том, что значение поля нужно рассматривать как число, необходимо выбрать один из нескольких числовых типов, известных в языке C#. Наиболее распространенный — *integer*. Из курса математики вы должны знать, что это не что иное, как целое число, которое может быть отрицательным, положительным или нулем.



То, что значение поля должно быть целым числом, проще всего сообщается так:

```
int numberOfLegs;
```

А вот что подумает в этом случае наш цифровой друг:

Тип «Boolean» (логическое значение)

Если вам известно, что некоторое поле будет всегда принимать значения «true» («истина») или «false» («ложь») и ничего другого, то использование в C# поля типа «true/false» может оказаться полезным. Это самый простой из всех типов, хотя такие поля носят довольно странное название — булевые (Boolean), по имени математика Джорджа Буля.

Извещая компьютер о том, что поле «hasTail» должно принимать значения «true/false», используйте следующий код:

```
bool hasTail;
```

Машина будет думать следующим образом:

Итак, поля представляют собой элементы информации, позволяющие лучше описать объекты в классе. Они всегда имеют определенный тип.

Добавление полей в класс

Ранее, когда мы знакомили компьютер с классом животных, то просто дали классу название – «Animal», добавили в код фигурные скобки и пообещали рассказать о нем больше, а точнее — сообщить о том, что находится внутри него. Запись выглядела так:

```
class Animal
{
  ...
}
```

А теперь мы готовы добавить несколько полей в определение класса.

```
class Animal
{
    string kindOfAnimal;
    string name;
    int numberOfLegs;
    int height;
    int length;
    string color;
    bool hasTail;
    bool isMammal;
    bool spellingCorrect;
}
```

Код должен быть аккуратным и легко читаемым, для чего и создаются о**тступы** полей, показанные выше. Это делается при помощи клавиши TAB на клавиатуре – для одной строки или, выделив сразу несколько строк, для всех сразу.

Поля и объекты

Итак, мы объяснили компьютеру, что ВСЕ объекты класса «Animal» имеют перечисленные выше поля. Теперь давайте попросим его создать экземпляр, представляющий определенный объект класса «Animal» (как мы делали это в предыдущем разделе), и сообщим компьютеру некоторую информацию об этом объекте, присвоив значения каждому полю.

```
Animal Barsik;
Barsik = new Animal();
Barsik.kindOfAnimal = “Cat”;
Barsik.name = “Кот Барсик”;
Barsik.numberOfLegs = 4;
Barsik.height = 50;
Barsik.length = 110;
Barsik.color = “Black”;
Barsik.hasTail = true;
Barsik.isMammal = true;
```

Обратите внимание, что в *кавычки* (“”) заключается значение только *строковых* полей. В конце каждого выражения ставится точка с запятой (;), после чего полученная конструкция становится оператором присваивания.

На данном этапе стоит упомянуть, что выбранное нами имя объекта «Barsik» не имеет никакого значения для компьютера, который будет работать с объектом, даже если мы заменим имя «Barsik» на «X» или любое другое:

```
Animal X;
X = new Animal();

X.kindOfAnimal = “Cat”;
X.height = 50;
X.length = 110;
X.color = “Black”;
X.hasTail = true;
X.isMammal = true;
```

Примечание редактора. То, что компьютер способен обрабатывать любые имена (проблемы не составит даже кот, которого зовут ГЩ12я), не означает, что объекты можно именовать как попало. Имя многое говорит человеку, поэтому крайне важно давать объектам содержательные имена, позволяющие лучше понимать смысл программы.

Еще одно важное замечание. Каждый объект можно рассматривать как коробочку, внутри которой находятся ячщички (поля). У двух объектов одного класса коробочки одинаковые, но значения, хранящиеся в ящичках (полях), — разные. Например, два объекта Миша и Гриша имеют поле «Рост», но Миша высокий, а Гриша маленького роста.

Строительный блок: Поля	

Обычно в классе присутствует одно поле или несколько полей, которые позволяют описывать определенные черты объектов этого класса.	`class SchoolKid`
	`{`
	`string firstName;`
	`string lastName;`
	`int age;`
	`bool isWideAvake;`
	`}`

Поля всегда имеют определенный тип. Допустимых типов очень много, вот описание наиболее распространенных из них:

Тип	**Описание**	**Примеры значений.**
string	Данный тип предполагает, что значением поля может быть слово или предложение, состоящее из символов алфавита.	«Petr» («Петя») «Eight-legged Octopus» («Восьминогий Осьминог») «Girls are too clever to fall out of their prams» («Девочки слишком умные, чтобы выпасть из коляски»)
int	Сокращение от слова «integer». Значение поля может быть целым числом.	328 28000 -520
bool	Сокращение от слова «boolean». Значение поля может быть либо true, либо false.	True — истина False — ложь

При определении поля в программе необходимо указать его *тип* и *имя*:

```
string lastname;
```

Обозначение *типа* должно быть известно компьютеру. Указанные выше основные типы пишутся строчными буквами, однако некоторые могут содержать и прописные буквы.

Давая *имя* полю, можно использовать любое слово, например, «VsegdaNacheku» – здесь сочетаются заглавные и строчные буквы. Обратите внимание: пробелы ставить нельзя.

При работе с определенным объектом полям присваиваются значения, которые помогают описывать объект. Если, например, у нас есть объект с именем *schoolKid (Школьник)*, то его полям можно было бы присвоить следующие значения:

```
schoolKid.firstName = "Petr";
schoolKid.age = 13;
```

Имя объекта и имя поля разделяются точкой (.).

Итак, `schoolKid.firstName = "Petr"` означает следующее: «*Petr* становится значением поля *firstName (Имя)* объекта *schoolKid*».

Закрытые, защищенные и открытые поля

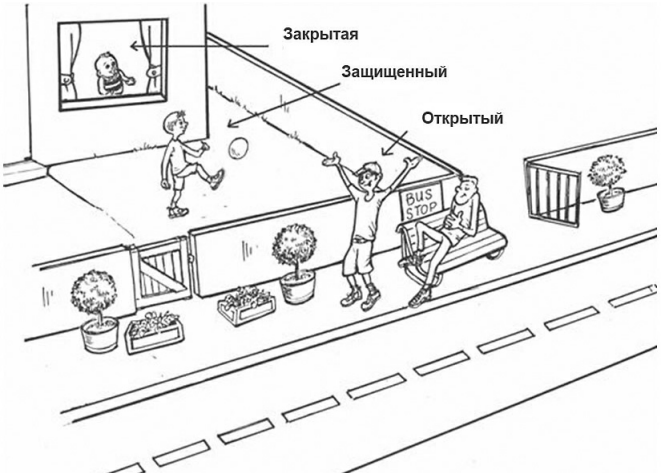
На первом этапе изучения языка C# этот раздел не столь важен, но в дальнейшем вам могут встречаться слова «private», «protected» и «public». Давайте обсудим их смысл.

```
class Animal
{
    string kindOfAnimal;
    string name;
    int numberOfLegs;
    int height;
    int length;
    string color;
    bool hasTail;
    bool isMammal;
    bool spellingCorrect;
}
class MyTestClass
{
    Animal myAnimal;
}
```

Объекты одного класса могут быть полями в другом классе. Например, в классе *MyTestClass* появится поле myAnimal, принадлежащее классу *Animal*.

При объявлении полей так, как показано выше, **мы можем решить, должны ли другие классы иметь к ним доступ**. Иногда очень важно делать поля закрытыми (private) — для того, например,

чтобы другой класс не помещал неверные значения в поля и не нарушал работу программы. Использование полей «private», «protected» и «public» определяет степень защиты объектов класса.



На рисунке изображены дети, живущие в одной семье. Обратите внимание на то, как обеспечивается безопасность каждоо из них. Малютка Света *закрыта (private) в комнате* — ее не выпускают из дома, и она имеет наиболее высокий уровень защиты. С ней могут общаться люди, обладающие наибольшим доверием. Борису разрешается находиться как на переднем, так и на заднем дворе, но он по-прежнему в некоторой степени *защищен (protected)*. Толя, старший ребенок, является *открытым (public) для других,* и любой желающий может общаться с ним.

Итак, в начало описания поля мы можем добавить еще одно слово:

```
class Animal
{
    public string kindOfAnimal;
    public string name;
    public int numberOfLegs;
    public int height;
    public int length;
    public string color;
    bool hasTail;
    protected bool isMammal;
    private bool spellingCorrect;
}
```

Вы можете поинтересоваться: а что произойдет, если мы не станем включать эти слова? В таком случае по умолчанию задается значение «private» и компьютер воспринимает код, как если бы в начале объявления было написано слово «private». Поэтому в последнем примере поле *hasTail* автоматически становится закрытым (private).

Давайте уточним значения этих слов:

- **Private** – «объекты только этого класса могут обращаться к данному полю».
- **Public** – «объекты любого класса могут обращаться к этому полю».
- **Protected** – «только объекты классов-наследников могут обращаться к полю». Если построен класс *Animal*, то другой класс, например, класс *Mammal* (Млекопитающее), может объявить себя наследником класса *Animal*.

Следующие примеры двух классов демонстрируют описанный принцип полей **public** и **private**. Поля типа protected мы оставим без обсуждения, а сведения о них можно найти в справке Visual C# Express.

Как и ранее, сначала мы определяем несколько закрытых и открытых полей в классе «*Animal*» и затем пытаемся обратиться к ним из класса «Zoo».

```
class Animal
{
    public string kindOfAnimal;
    public string name;
    public int numberOfLegs;
    public int height;
    public int length;
    public string color;
    bool hasTail;
    protected bool isMammal;
    private bool spellingCorrect;
}

class Zoo
{
    Animal a = new Animal ();
        // Следующая строка будет выполнена успешно, поскольку классу “Zoo”
        // разрешено обращаться к открытым полям в классе “Animal”
    a.kindOfAnimal = “Kangaroo”;
        // Обе следующие строки НЕ будут выполнены, поскольку классу “Zoo”
        // не разрешено обращаться к закрытым или защищенным полям
    a.isMammal = false;        // Попытка обращения к защищенному методу
    a.spellingCorrect = true;  // Попытка обращения к закрытому методу
 }
```

Методы класса

Если бы классы позволяли только описывать свойства объектов, то пользы от них было бы мало. Безусловно, мы хотим, чтобы с их помощью выполнялись некоторые действия:

- выводились слова на экран;
- решались задачи;

- копировались данные с веб-узла;
- регулировалась яркость фотографии;
- ...и выполнялись еще тысячи операций.

А теперь вспомните уроки, на которых вас учили сложению чисел. Вероятней всего, сначала учитель объяснял, **КАК** это делается, описывая, возможно, каждый шаг. Другими словами, в первую очередь рассматривался **МЕТОД** решения задачи определенного типа, то есть набор пошаговых инструкций, поясняющих порядок выполнения действий.

А следуя уже известному вам методу решения задач, вы справитесь с любыми похожими задачами,.

Подобным образом, для того чтобы компьютер выполнял нужные нам действия, надо написать код, разъясняющий порядок их выполнения, то есть описать метод, который будет использоваться для выполнения необходимых действий, а затем применять этот метод при решении задачи подобного типа.

В языке C# метод представлен отрывком кода с описанием порядка выполнения определенных действий. Когда впоследствии мы вызываем код, компьютер читает его и выполняет именно то, что предписано.

Простой метод для компьютера может иметь следующий вид:

```
void SayHello()
{
}
```

Но метод не имеет смысла, если ничего не делается, поэтому давайте зададим код между фигурными скобками.

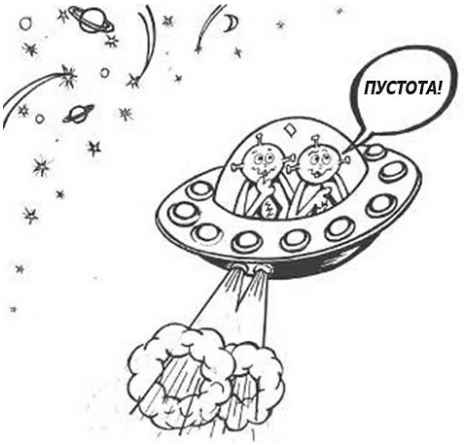
```
void SayHello()
{
    Console.WriteLine("Hello");
}
```

(Придадим коду аккуратный вид и при помощи клавиши TAB создадим в методе отступы.)

Если вызвать этот метод, то на экран будет выведено слово «Hello». Задача не слишком сложная, поэтому и метод довольно прост.

Что означает VOID?

Странное слово «void» в приведенном выше примере может несколько озадачить, и вы можете поинтересоваться, зачем вообще его использовали. Вкратце остановимся на этом. Вспомните фильм о космических путешествиях: представьте безграничное пустое пространство во вселенной. Void означает пустоту, «ничего».



Использование слова «void» перед именем метода означает, что, когда завершается выполнение метода, возвращается пустое значение, то есть по завершении определенных действий, которые выполняет метод, он никаких значений не возвращает. В приведенном выше примере метода «SayHello» нас это вполне устраивает, поскольку после написания слова «Hello» возвращать ничего не нужно. Все необходимое было сделано во время работы метода.

Вы сможете лучше понять смысл этого слова, когда мы будем рассматривать случаи, где метод должен будет возвращать некоторый результат. Подобный пример появится немного позже.

Вызов метода

Написав приведенный выше код, мы объяснили компьютеру, КАК выводить приветствие на экран, но не сказали, КОГДА ему нужно воспользоваться приобретенным умением.

Это выглядит так, как если бы на уроке вам рассказали, как складывать числа, после чего вы записали метод, но решить задачку или пример вас не попросили.

Чтобы заставить компьютер выполнить действие, нужно вызвать метод. Для этого надо просто написать имя метода и поставить рядом круглые скобки:

```
SayHello();
```

Когда компьютер встречает эту строку, он понимает, что нужно выполнить команду, но встроенного метода с таким именем у компьютера нет, поэтому он лихорадочно начинает искать написанный вами метод с именем «SayHello». Если найти такой метод удается, он быстро выполняет все предписания, записанные в коде метода.

Затем возвращается к самому началу и, довольный успехом, получает вашу благодарность.

Строительный блок: Объявление и вызов метода

Чаще всего в классе присутствует один или несколько методов. Они выполняют определенные действия. Методами они называются потому, что именно в них описывается метод выполнения действий – пошаговые инструкции, задающие порядок выполнения операций.
Строка, начинающаяся с двух символов «слеш» (//поля), называется комментарием. Комментарии не влияют на выполнение компьютером программного кода. Они поясняют код.
Операция «+» определена над строками. Она называется сцеплением строк, или конкатенацией. Результатом операции является приписывание второй строки в конец первой.

Как уже говорилось, объекты класса Person могут объявляться и создаваться в методах другого класса, выступающего в роли клиента класса Person.

```
class Person
{
 // Поля
 public string firstName;
 public string lastName;
 // Метод
 public void ShowFullName()
 {
  Console.WriteLine("Name is
  " + firstName + " " +
  lastName);
 }
}

 Person Petr;
 Petr = new Person();
 Petr.firstName = "Petr";
```

Когда встречается вызов метода ShowFullName, компьютер находит в классе Person метод с указанным именем и — шаг за шагом — выполняет написанный в нем код.

```
Petr.lastName = "Ivanov";
Petr.ShowFullName();
```

Как выполняется метод? Параметры метода

В предыдущем разделе мы познакомились с методами вкратце, но о них можно рассказать гораздо больше. Теперь рассмотрим более подробно, как передавать методу значения и получать значения, созданные при работе метода.

Предположим, вам потребовалось, чтобы компьютер вывел на экран следующий текст:

```
Hello Jo
Hello Sam
Hello You
```

Один из возможных способов — написать отдельный метод для каждого случая:

```
void WriteHelloJo()
{
    Console.WriteLine("Hello Jo");
}

void WriteHelloSam()
{
    Console.WriteLine("Hello Sam");
}

void WriteHelloYou()
{
    Console.WriteLine("Hello You");
}
```

Затем необходимо вызвать их следующим образом:

```
 WriteHelloJo();
 WriteHelloSam();
 WriteHelloYou();
```

Но ведь все три метода очень похожи. А что если написать один метод WriteHello, дополнив его соответствующими параметрами, и при каждом вызове просто передавать значение параметра, отличающее один вызов от другого?

Вот как это можно сделать:

```
void WriteHello(string someName)
{
    Console.WriteLine("Hello " + someName);
}
```

и затем вызвать метод следующим образом:

```
WriteHello("Jo");
WriteHello("Sam");
WriteHello("You");
```

Как видим, код позволяет сэкономить и занимаемое пространство, и затраченные усилия. Всегда старайтесь делать код как можно более кратким – чем короче программа, тем умнее программист.

//Примечание редактора. Умный программист пишет не только короткий, но и понятный код. Хороший код всегда содержит комментарии умного программиста.

Напишем метод подобным «умным» образом:

```
void WriteHello(string someName)
{
    Console.WriteLine("Hello " + someName);
}
```

Фактически мы говорим: «Каждый раз при вызове этого метода я буду подставлять строку символов с каким-либо именем. Любая *подставляемая* строка должна выводиться после слова «Hello».

Код в скобках (`string someName`) называется параметром. Параметр позволяет подставлять значение в метод при его вызове.

Когда вас обучали сложению, учитель не рассказывал о сложении всех существующих пар чисел, он просто научил методу и затем задавал разные задачи: «Сложите 2 и 5, а теперь 7 и 3». Это похоже на то, как если бы вам излагали метод сложения чисел, используя параметры: неважно, какие значения у параметров, — зная метод, всегда можно найти ответ для заданных значений.

Компьютеру все равно, какое имя вы присвоите параметру, важно, чтобы оно было единым при использовании во всем методе. Например, следующий код будет выполнен правильно:

```
void WriteHello(string x)
{
    Console.WriteLine("Hello " + x);
}
```

А этот с ошибкой:

```
void WriteHello(string someName)
{
    Console.WriteLine("Hello " + someBodysName);
}
```

Вы заметили: параметры «someName» и «someBodysName» отличаются — наш «электронный друг» не разберется в этой путанице и «разгневается».

Кроме того, в методе можно использовать не один параметр, а несколько, но нужно обязательно разделить их запятыми:

```
void WriteHello(string firstName, string lastName)
{
    Console.WriteLine("Hello " + firstName + " " + lastName);
}
```

А при вызове метода необходимо подставить правильное количество значений:

```
WriteHello("Petr", "Ivanov");
```

В данном случае на экран будет выведен текст «Hello Petr Ivanov».

Ошибки в задании типов параметров

Представьте, что когда вы впервые изучали сложение чисел, учитель неожиданно задал вам такую задачу: «Сложите число *5* и *цветок*».

Как бы вы поступили? Наверное, ответили бы, что цветок — это не число, и выполнить сложение невозможно. Правильно.

Подобным же образом компьютеру не понравится, если вы укажете значение *неверного типа*. Такая ошибка часто встречается у программистов. Поэтому если что-то не получается, вернитесь к началу и убедитесь, что значения, указанные вами при вызове метода, имеют тип, соответствующий тому, который определен в самом методе.

Строительный блок: Параметры

Чтобы в методе выбиралось нужное значение, необходимо указать соответствующие параметры.

Каждый раз при вызове метода мы должны убедиться, что подставляем правильный тип значений в параметры. В приведенном примере мы подставляем два целых числа, так как параметры метода «LuckyNumber» были определены как целые числа.

```
class Person
{
 // Поля
 string firstName;
 string lastName;
 // Метод
 public void LuckyNumber(
  int numberOfTeeth, int age)
 {
  Console.WriteLine(
   «Счастливое число» +
   numberOfTeeth * age);
 }
}

Person Petr;
Petr= new Person();
Petr.LuckyNumber(24, 14);
```

//Примечание редактора. Внимательный читатель спросит: «Складывая строку с числом, не делаем ли мы ту же ошибку, как в случае сложения цветка с числом при вызове метода Console.WriteLine?». Здесь мы полагаемся на то, что компьютер (точнее, компилятор языка C#) умеет справляться с этим — сначала он автоматически преобразовывает число в строку и только потом выполняет операцию сложения — сцепление строк.

Приводимые до сих пор методы были void-методами. Они выводили некий текст на экран и затем возвращались назад к месту их вызова, как бы говоря: «Свое дело я сделал и вернулся в исходную точку». Однако иногда необходимо вернуть некоторое значение в точку вызова метода. В этих случаях следует написать метод, который будет возвращать значение, отличное от void.

Приведем пример. Напишем метод, который будет выполнять поиск количества конечностей указанного животного и затем отправлять полученное число туда, откуда этот метод был вызван.

Следует помнить, что с помощью метода мы показываем компьютеру, КАК выполнять определенное действие. Сначала я напишу то, чего хочу добиться от него, на русском языке, а затем на C#:

- если животное, о котором мы говорим, — слон, то number of legs = 4;
- иначе, если животное, о котором мы говорим, — индейка, то number of legs = 2;

- иначе, если животное, о котором мы говорим, — устрица, то number of legs = 1;
- иначе, если мы говорим о каких-либо других животных, то number of legs = 0.

```
int NumberOfLegs(string animalName)
{
    if (animalName == "слон") //Если название животного – слон
    {
        // Возвращаемое значение 4
        return 4;
    }
    else if (animalName == "индейка") //Иначе, если животное – индейка
    {
        // Возвращаемое значение 2
        return 2;
    }
    else if (animalName == "устрица")//Иначе, если животное – устрица
    {
        // Возвращаемое значение 1
    return 1;
    }
    else //Иначе (при всех других условиях)
    {
        // Возвращаемое значение 0
        return 0;
    }
}
```

Теперь мы можем вызвать метод. Давайте сделаем это дважды:

```
int i;
    //Переменная "i" будет хранить значение числа конечностей.
i = NumberOfLegs("индейка");
    //Теперь i = 2, получив значение, возвращенное методом NumberOfLegs
Console.WriteLine("У индейки конечностей – " + i);
i = NumberOfLegs("обезьяна");
    //Теперь i = 0. Догадайтесь, почему!
Console.WriteLine("У обезьяны конечностей – " + i);
```

На экран будет выведен текст: «У индейки конечностей — 2», «У обезьяны конечностей — 0». Итак, метод возвращает значение, которое можно принять в точке его вызова.

Мы определили метод именно так:

```
int NumberOfLegs(string animalName)
{
 ...
}
```

А не так:

```
void NumberOfLegs(string animalName)
{
...
}
```

И не так:

```
string NumberOfLegs(string animalName)
{
...
}
```

Дело в том, что нам необходимо, чтобы в данном случае метод возвращал **целое число** — не «пустое» значение (void), не строку букв, а именно целое число. А для работы с целыми числами используется тип данных «Integer», или «int» в сокращенном варианте.

При написании метода мы всегда указываем тип данных, возвращаемых этим методом. Если возвращать значение не надо, используется «void» – для возврата пустого значения.

```
void JustWriteSomething(string someThing)
{
   Console.WriteLine(someThing);
}
```

И наконец: возможно, вы догадались, что слово «**return**» возвращает значение. Когда компьютер встречает это слово, происходит выход из метода и возврат запрашиваемого значения.

Строительный блок: Возвращаемые значения

Иногда необходимо получить значение из метода. В таком случае вместо указания типа «void», сообщающего, что «никакого значения возвращено не будет», мы указываем определенный тип данных, возвращаемых методом.

```
class Person
{
 // Поля
 string firstName;
 string lastName;
 // Метод
 int LuckyNumber(int numberOfTeeth,
  int age)
 {
  return (numberOfTeeth * age);
 }
}
```

Возвращаемые значения автоматически становятся доступными в любом месте, где мы вызываем метод.

```
Person Anna;
Anna = new Person();
int num = Anna.LuckyNumber(24, 14);
```

Например, используя возвращенное значение, мы могли бы сначала сохранить ответ в переменной части выражения и затем использовать значение переменной в отдельном выражении.	`Console.WriteLine(«Счастливое число Анны:» + num);`
Или мы могли бы вызвать метод непосредственно в выражении WriteLine.	`Console.WriteLine(«Счастливое число Анны:» + Anna.LuckyNumber(24, 14));`

Доступ к методам, аналогично доступу к полям класса, регулируется с помощью ключевых слов. По умолчанию все методы будут рассматриваться как private (закрытые), то есть они применяются только внутри своего класса. Чтобы разрешить их использование для других классов, можно добавить слово «public» в начало объявления метода.

```
public void JustWriteSomething(string someThing)
{
    Console.WriteLine(someThing);
}
```

В реальном мире людям запрещается входить в некоторые помещения без специального разрешения. Например, в ресторанах только повара и официанты могут проходить на кухню – это закрытая зона. В то же время обеденный зал предназначен для свободного доступа, и в нем могут находиться любые лица. Подобным же образом некоторый код закрыт для других классов.

Мы уже рассматривали пример с закрытыми и открытыми полями. Дополним его: введем закрытые (Private) и открытые (Public) *методы* в класс «Animal» и затем пытаемся обратиться к ним из класса «Zoo».

```
class Animal
{
    //Поля
```

```
    public string kindOfAnimal;
    public string name;
    public int numberOfLegs;
    public int height;
    public int length;
    public string color;
    bool hasTail;
    protected bool isMammal;
    private bool spellingCorrect;
    //Методы
    // Открытый метод, получающий информацию о том, чем питается животное
    public string GetFoodInfo()
    {
    // Представим, что здесь расположен код, выполняющий поиск по базе данных
    …
    }

    // Закрытый метод для проверки правильности написания вида животного
    private void SpellingCorrect()
    {
     // Представим, что здесь расположен код для проверки правописания
    …
    }

    // Защищенный метод, определяет существование данного вида животного
    protected bool IsValidAnimalType()
    {
        //код для проверки существующих видов животных
     …
    }
}

 class Zoo
 {
    Animal a = new Animal ();
    a.name = “Kangaroo”;
    string food;
    bool animalExists;

    // Следующий код будет выполнен успешно, поскольку классу “Zoo” разрешено
    // обращаться к открытым методам в классе “Animal”
    food = a.GetFoodInfo(); // Вызов открытого метода

    // Обе следующие строки НЕ будут выполнены, поскольку классу “Zoo”
```

```
    // не разрешено обращаться к закрытым или защищенным методам
    a.spellingCorrect();       // Попытка вызова закрытого метода
    animalExists = a.IsValidAnimalType();     // Попытка вызова защищенного метода
}
```

Очень часто встречаются классы с особым типом метода, называемым «**конструктором**». С точки зрения синтаксиса его особенность состоит в том, что имя *метода-конструктора совпадает с именем класса* и в объявление конструктора не включается *тип возвращаемого значения*. Содержательная специфика связана с предназначением конструктора — он нужен для создания (конструирования) объекта. Использование этого метода в классах помогает приобрести хороший практический опыт.

//Примечание редактора. Классов без конструктора не бывает, поскольку объект класса можно создать только путем вызова конструктора класса. Даже если программист не добавит в класс конструктор, это будет сделано по умолчанию, но параметров такой конструктор не имеет. Полезно иметь в классе конструктор с параметрами, роль которых уже пояснялась. Подобных конструкторов может быть несколько.

```
class Person
{
    // Поля
    string firstName;
    string lastName;
    // Метод-конструктор для класса Person
    public Person()
    {
        firstName = “Johnny”;
        lastName = “Rocket”;
    }
}
```

Метод-конструктор вызывается по-особому: при каждом создании экземпляра класса с помощью конструкции new.

Напоминание:
под «экземпляром класса» мы понимаем *определенный* объект класса. Например, в одном из предыдущих разделов мы выделили «Гориллу Джереми» как определенный объект, или экземпляр класса «Animal».

Итак, если мы выполним следующий код:

```
Person p = new Person();
Console.WriteLine(p.lastName);
```

то в результате на экране появится слово «Rocket». Написав конструкцию *new Person(), мы тем самым дали указание компьютеру вызвать конструктор класса Person* для создания нового объек-

та этого класса. Он будет связан с переменной p, у которой задано значение «Rocket» для поля lastName.

Приведем аналогичный пример из реальной жизни. В некоторых странах новорожденного регистрируют, согласно закону, еще в родильном доме, чтобы он как можно скорее стал членом общества и получил все гражданские права.

Это действие можно сравнить с методом-конструктором, выполняемым для класса. Прежде чем новый экземпляр класса сможет что-либо сделать, выполняется метод-конструктор. В него можно включить любые планируемые к выполнению действия, прежде чем объект будет считаться «готовым к жизни».

Конструкторы с параметрами

В конструктор можно включить параметры. Приведем пример класса с двумя различными конструкторами:

```
class Person
{
   // Поля
   string firstName;
   string lastName;
   // Первый метод-конструктор
   public Person()
   {
       firstName = “Johnny”;
       lastName = “Rocket”;
```

```
    }
    // Второй метод-конструктор
    public Person(string f, string l)
    {
        this.firstName = f;
        this.lastName = l;
    }
}
```

Таким образом, мы получили *два разных способа конструирования объекта*.

Например, этот:

```
Person p = new Person();
```

В таком случае в поле p.lastName будет автоматически подставлено значение «Rocket».

Или этот:

```
Person p = new Person("Petr", "Ivanov");
```

Тогда в поле p.lastName будет подставлено значение «Ivanov».

Слово «**this**» относится к **объекту, который мы создаем**, то есть фактически указывается: «подставлять в поле имени и фамилии *этого объекта* любые значения, передаваемые методу-конструктору».

//Примечание редактора. В данном случае слово this можно добавить и в первый конструктор – для уточнения имени полей. Ничего не изменится, если во втором конструкторе имена полей будут указываться без this. Но иногда такое уточнение необходимо. Например, если во втором конструкторе имя параметра задавать не одной буквой f, а именем firstName, отражающим суть параметра, то без уточнения this в имени поля не обойтись, иначе компьютер запутается, не понимая, где имя поля, а где имя параметра метода.

События

В реальном мире события происходят непрерывно, причем некоторые *от нас* совсем *не зависят*. Например, восход и закат солнца (хотел бы я посмотреть, как вы пытаетесь заставить солнце вставать и садиться). *Другие* события мы вызываем сами: скажем, заставляем громкоговоритель издавать звуки.

Рассмотрим несколько событий, которые часто встречаются в мире компьютеров:

- нажатие на кнопку, изображенную на экране монитора;
- истечение времени таймера;
- перемещение мыши;
- нажатие клавиши на клавиатуре.

Очевидно, что, нажав на кнопку, мы хотим заставить компьютер выполнить определенное действие. (Если нет – зачем вообще ее трогать?) Но компьютер ждет не только подтверждения,

что нажатие этой кнопки имеет для вас какое-то значение, но и указания на действие, которое нужно выполнить.

Рассмотрим подробнее этот пример, поскольку нажатие на кнопку, возможно, является наиболее распространенным событием, и детально разберем порядок работы. Допустим, в вашей программе есть объект — кнопка с именем *mrButton, на которой написано* «Нажми меня».

По ходу обсуждения **попробуйте все делать сами**. Для начала:

- Запустите Visual C# Express.
- Создайте новый проект приложения Windows: в меню File («Файл») выберите Create project («Создать проект») и затем тип проекта Windows Application («Приложение Windows Forms»).
- В Visual C# Express откроется несколько файлов, где содержится «скелет» кода программы.
- В окне обозревателя решений справа (в списке всех файлов) удалите файл с именем Form1.cs.
- Дважды щелкните имя файла Program.cs и удалите весь автоматически вставленный «скелет» кода.
- Чтобы создать программу с экземпляром кнопки, наберите следующий код в окне Program.cs так, как показано ниже (написание слов *курсивом или жирным шрифтом можно не учитывать*.

```
using System;
using System.Windows.Forms;

class MyButtonClass: Form
{
    private Button mrButton;
    // Метод-конструктор
    public MyButtonClass()
    {
        mrButton = new Button();
        mrButton.Text = “Нажми меня”;
        this.Controls.Add(mrButton);
    }
```

```
    // Основной метод
    static void Main()
    {
        Application.Run(new MyButtonClass());
    }
}
```

Выполните программу при помощи клавиши F5 (или щелкните по зеленой кнопке «Выполнить»). Если возникнут сообщения об ошибках, тщательно проверьте, нет ли опечаток в коде. Если программа будет выполнена успешно, вы увидите форму с кнопкой «Нажми меня». Пока при нажатии на кнопку никаких действий происходить не будет. Конечно, вы ожидали другого результата, но все еще впереди.

Событие нажатия кнопки. Указание действия в случае события

Теперь мы должны задать **метод**, выполняющий действие при нажатии на кнопку. Такие методы называются обработчиками событий, поскольку они именно «обрабатывают» событие. В приведенном ниже примере выделенный маркером код обработчика события просто изменяет надпись на кнопке, поэтому он совсем короткий:

```
using System;
using System.Windows.Forms;

class MyButtonClass: Form
{
    private Button mrButton;
    // Метод-конструктор
    public MyButtonClass()
    {
        mrButton = new Button();
        mrButton.Text = “Нажми меня!”;
        this.Controls.Add(mrButton);
    }
    // Основной метод
    static void Main()
    {
        Application.Run(new MyButtonClass());
    }
    // Метод-обработчик событий
    void MyButtonClickEventHandler(object sender, EventArgs e)
    {
        mrButton.Text = “Вы нажали меня!”;
    }
}
```

Ваша программа еще выполняется?

- Остановите ее (нажмите на кнопку X в верхнем правом углу окна, в котором открыта форма).
- Добавьте в программу выделенный код и нажмите на клавишу F5 для выполнения измененной программы.
- Попробуйте теперь щелкнуть по кнопке «Нажми меня». И теперь ничего не происходит?!

Поскольку вы уже прочитали те страницы, где мы говорили про методы, то должны узнать основную структуру приведенного выше метода. Слово «void» означает, что по его завершении ничего не возвращается. Мы назвали этот метод «MyButtonClickEventHandler».

Возможно, то, что вы видите, кажется немного странным. Вы понимаете, что в скобках присутствуют два параметра, но почему у них такие необычные типы (object *sender*, EventArgs *e*)? К сожалению, с методами обработчиков событий *нельзя использовать собственные типы параметров*. Когда у кнопки возникает событие «Click», она посылает сообщение о нем операционной системе, а та находит и вызывает соответствующий обработчик события. При вызове такого метода система сама определяет типы параметров и передает обработчику их значения. *Это факт, с которым* мы ничего поделать не можем.

Поэтому придется просто смириться и всегда использовать ожидаемые типы параметров с обработчиком событий. Очень часто подставляемые параметры имеют тип «object» и «EventArgs». В приведенном выше примере мы выбрали имена параметров «sender» и «e», но могли бы выбрать любые другие — для компьютера важны имена *типов* этих параметров. Например, следующий код будет работать точно так же, как и код, рассмотренный в предыдущем примере. Можете проверить это сами, изменив имена параметров в вашей программе на «x» и «y».

//Примечание редактора. Ничего удивительного: эти параметры вообще не используются в тексте обработчика события. Но в ряде случаев они полезны. Например, при работе с мышкой обработчику события могут быть переданы координаты объекта, на котором нажата кнопка мыши.

```
void MyButtonClickEventHandler(object x, EventArgs y)
{
   mrButton.Text = "Вы нажали меня!";
}
```

В первом параметре обычно содержится некоторая информация об объекте, который инициировал событие. Второй параметр относится к данным о самом событии.

Очень важно знать следующее: система всегда подставляет некоторые значения в эти два параметра, но зачастую необходимости в их использовании нет — они отправляются в метод обработчика событий «на всякий случай».

Подключение метода обработчика событий к событию

Вы удивляетесь, почему ничего не происходит при нажатии на кнопку? Дело в том, что указанный метод вызывается только тогда, когда мы *свяжем с ним событие нажатия на кнопку*, ука-

зав в программе: при нажатии на кнопку необходимо перейти к определенному обработчику событий.

Когда в программе используются разные кнопки и несколько обработчиков событий, без такого уточнения не обойтись,. так как компьютер должен знать, какой именно метод следует выполнять при нажатии на определенную кнопку.

Код для связывания события объекта с методом обработчика события выглядит тоже несколько странно. Мы опять выделим его маркером.

```
using System;
using System.Windows.Forms;

class MyButtonClass : Form
{
    private Button mrButton;
    // Метод-конструктор

    public MyButtonClass()
    {
        mrButton = new Button();
        mrButton.Text = “Нажми меня”;
        mrButton.Click += new System.EventHandler(MyButtonClickEventHandler);
        this.Controls.Add(mrButton);
    }
    // Основной метод
    static void Main()
    {
        Application.Run(new MyButtonClass());
    }
    // Метод-обработчик событий
    void MyButtonClickEventHandler(object sender, EventArgs e)
    {
        mrButton.Text = “Вы нажали меня!”;
    }
}
```

С компьютерного языка это можно перевести следующим образом:

«Путем нажатия на кнопку mrButton надо связать событие Click с методом обработчика событий, который называется *MyButtonClickEventHandler*».

При нажатии на кнопку приведенная выше строка кода позволяет системе вызвать метод обработчика событий, после его выполнения надпись на кнопке меняется на «Вы нажали меня!».

Чтобы использовать этот код, остановите свою программу, добавьте в нее выделенный код и нажмите клавишу F5 для выполнения программы. Нажмите на кнопку, и надпись на ней изменится. Рабочую программу — пример события нажатия на кнопку — можно найти в папке примеров, прилагаемых к книге (Проект Example5).

//Примечание редактора. Программа работает, но можно немного улучшить ее внешний вид. Сейчас положение кнопки на форме и ее размеры установлены по умолчанию. Для изменения этих параметров добавьте в конструктор класса строчки, выделенные маркером, и снова запустите программу.

```
public MyButtonClass()
    {
        mrButton = new Button();
        mrButton.Text = "Нажми меня";
        mrButton.Top = 100;
        mrButton.Left = 100;
        mrButton.Height = 50;
        mrButton.Width = 70;

        mrButton.Click += new System.EventHandler(MyButtonClickEventHandler);
        this.Controls.Add(mrButton);
    }
```

Теперь мы попробуем описать основную структуру метода обработчика событий мыши. Вероятно, в этом случае вы *захотите* использовать информацию, подставляемую в параметр *MouseEventArgs*, хотя бы для того, чтобы выяснить, какая из кнопок мыши нажимается.

```
public void TheMouseIsDown(object sender, MouseEventArgs e)
{
    if (e.Button = MouseButtons.Left)
        this.Text = "Нажата левая кнопка мыши";
}
```

Далее показано, как связать событие с методом. В переводе с компьютерного в тексте написано следующее: «Если при выполнении этой программы нажимается кнопка мыши, надо перейти к методу *TheMouseIsDown*», которому известно, как следует обрабатывать события мыши:

```
this.MouseDown += new MouseEventHandler(TheMouseIsDown);
```

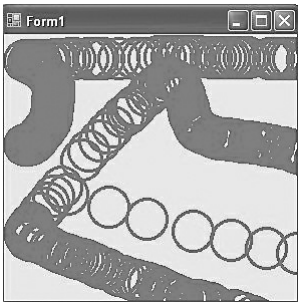
Можно внести некоторые улучшения, чтобы при запуске следующего обработчика код, содержащийся в нем, делал окно более широким или узким в зависимости от нажимаемой кнопки.

```
public void TheMouseWasClicked(object sender, MouseEventArgs e)
{
    // При нажатии левой кнопки
    if (e.Button == MouseButtons.Left)
        // Расширение текущего окна
        this.Width = this.Width + 100;
    else if (e.Button == MouseButtons.Right)
        // Сужение текущего окна
        this.Width = this.Width - 100;
}
```

Другой обработчик событий позволяет обнаружить перемещение мыши и рисовать окружность в том месте, где находится курсор:

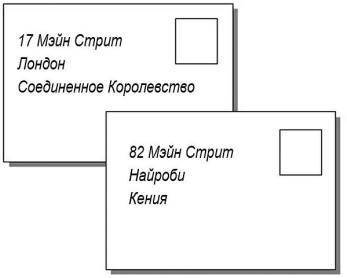
```
public void TheMouseMoved(object sender, MouseEventArgs e)
{
    // Подготовка области рисования
    System.Drawing.Graphics g = this.CreateGraphics();

    // Использование красной ручки
    System.Drawing.Pen redPen = new System.Drawing.Pen(Color.Red, 3);

    // Рисуем окружность как эллипс с равными осями.
    // Окружность рисуется в охватывающем ее квадрате.
    // Координаты X и Y левого верхнего угла квадрата
    // определяются координатами текущего положения мыши.
    g.DrawEllipse(redPen, e.X, e.Y, 40, 40);

    // Очистка
    g.Dispose();
}
```

На снимке экрана показано, как это выглядит при перемещении мыши:



Пространства имен и почтовая служба

В мире существуют сотни, а может быть, и тысячи, улиц, названных именем А.С. Пушкина. Но если на конверте указана одна из них, то как письмо находит своего получателя? Естественно, адрес состоит не только из улицы. Мы, по меньшей мере, добавляем название страны и города.



Очевидно, если мы укажем адрес на конверте:

- улица Пушкина, 17;
- улица Пушкина, 82,

то почта не распознает, где эта улица и где этот дом.

Можно написать так:

- улица Пушкина, 17, г. Москва;
- улица Пушкина, 82, г. Алма-Ата

Так-то лучше, а если существуют два города с одинаковым названием? Например, город Москва есть в США! Поэтому для надежности добавим еще и название страны:

- улица Пушкина, 17, г. Москва, Российская Федерация;
- улица Пушкина, 82, г. Алма-Ата, Республика Казахстан.

В России адрес принято писать так:

- Российская Федерация, г. Москва, улица Пушкина, 17;
- Республика Казахстан, г. Алма-Ата, улица Пушкина, 82.

Теперь письмо обязательно дойдет до адресата. Например, второе письмо доставят самолетом в Республику Казахстан, из аэропорта его перевезут на почтамт города Алма-Аты, а потом почтальон пойдет на улицу Пушкина и найдет дом под номером 82.

Формат (Республика Казахстан, г. Алма-Ата, ул. Пушкина) можно считать «пространством имен» для отправки писем.

Пространства имен и программный код

Какое же отношение все это имеет к программированию?

Предположим, разработчиками корпорации Microsoft написан класс «Point», используемый для рисования фигуры в определенной *точке*, но и вы создали класс с именем «Point», например, для загрузки фотографии человека, *указывающего на что-либо*.

//Примечание редактора. Слово Point в английском языке многозначное и означает как «точку», так и глагол «указывать».

Очевидно, что оба класса выполняют абсолютно разные действия, но и тот и другой можно назвать «Point». С каким же из них будет работать программа?

Безусловно, имеет смысл использовать разные имена. В среде .NET для этого можно задействовать различные «пространства имен». Например, так:

- Microsoft.Drawing.*Point*
- Anna.PictureStuff.*Point*

Имена классов сохранены («Point»), но пространства имен перед именами класса позволяют четко видеть, в каком из них находится класс. И теперь, если мне нужно использовать класс Point из пространства Anna, я могу создать объект Point при помощи следующего метода-конструктора:

```
Anna.PictureStuff.Point annaPoint = new Anna.PictureStuff.Point();
```

Для работы с классом Point от Microsoft подойдет следующий код:

```
Microsoft.Drawing.Point microsoftPoint = new Microsoft.Drawing.Point();
```

Несмотря на то что оба моих объекта принадлежат классу «Point», они относятся к разным классам и выполняют разные действия.

Как создать пространство имен и поместить в него собственный класс

Разместить собственный класс в пространстве имен очень просто. Раньше мы давали описание класса следующим образом:

```
class Animal
{
...
}
```

Теперь создайте пространство имен и внутри него опишите класс:

```
namespace Anna
{
    class Animal
    {
        ...
    }
}
```

Или, если хотите, сделайте так:

```
namespace Anna.FunStuff
{
    class Animal
    {
        ...
    }
}
```

Пространства имен могут быть вложенными друг в друга. Тогда их имена разделяются точками. В последнем примере класс *Animal* входит в пространство имен *Anna.FunStuff*.

Все библиотеки классов, которые предлагают разработчики Microsoft, а также других организаций, помещаются в пространства имен. Поэтому, прежде чем использовать код из таких библиотек, необходимо понимать, как работать с пространствами имен.

Строительный блок: Пространство имен	

Для логического объединения классов одной тематики рекомендуется помещать их в одно пространство имен. Имя пространства **должно иметь содержательный смысл.**	`namespace Charles.SchoolUtilities`
	`{`
	`  class Animal`
	`  {`
	
	`  }`
	`}`

Экземпляр или объект приведенного выше класса можно создавать, вызывая конструктор класса.	`Charles.SchoolUtilities.Animal  cat;` `cat = new Charles.SchoolUtilities.Animal();`

Если нужно создать много объектов одного пространства имен, то в начале программы в предложении «using» можно указать пространство имен. Тогда компьютер сам будет находить классы в этом пространстве.	`// говорим компьютеру, что будем использовать`
	`//классы из этого пространства имен`
	`using Charles.SchoolUtilities;`
	`// создание объекта Animal из пространства`
	`//имен Charles.SchoolUtilities`
	`Animal cat = new Animal();`

Наследование

Так как эта книга адресована тем, кто только начинает знакомиться с искусством программирования, мы не станем рассматривать слишком сложные вещи, поэтому расскажем о наследовании немного.

Иногда вам будет встречаться объявление классов следующего вида:

```
class MyFancyClass : Form
{
...
}
```

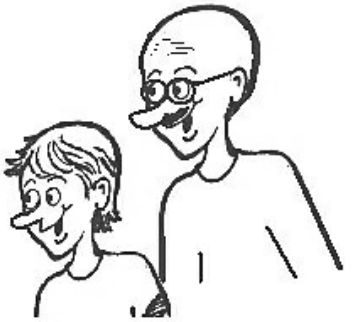
Сравните его с обычным объявлением класса:

```
class MyFancyClass
{
...
}
```

Поясним, что же происходит, когда после имени класса через двоеточие указывается имя другого класса и объявлению придается совсем другой смысл.

Наследование среди людей

Человек, как правило, наследует определенные качества от своих родителей. У вас может быть цвет волос как у мамы, а нос — папин.



Это не означает, что вы полностью похожи на своих родителей. Несомненно, вы обладаете различными уникальными качествами и способностями, но определенные свойства характера и физические особенности «заложены» в вас при рождении.

Наследование кода

При написании программного кода было бы полезно иметь возможность наследования всего набора способностей существующего класса, будь то собственный класс или чей-то еще.

Приведем пример. Определим два класса — «Animal» и «Bird». Класс «Bird» объявим наследником класса «Animal».

```
class Animal
{
    //Поля класса
    public string kindOfAnimal;
    public string name;
    public int numberOfLegs;
    public int height;
    public int length;
    public string color;
    bool hasTail;
    protected bool isMammal;
    private bool spellingCorrect;
    …
}
    // Класс “Bird” – наследник класса “Animal”
class Bird : Animal
{
    public string featherColor;
     …
}
```

В реальном мире *птица — это вид животного*, но у птиц есть свои отличительные признаки, характерные только для них. Тогда имеет смысл классу *Bird* наследовать все признаки класса *Animal* и стать обладателем ряда дополнительных признаков. В данном случае мы определим одно специальное поле, характерное для птиц, — featherColor — цвет перьев.

Итак, пишем:

```
class Bird : Animal
{
...
}
```

Таким образом мы сообщаем следующее: «Я определяю новый класс «Bird», но он должен автоматически наследовать все свойства класса «Animal». Иными словами, класс «Bird» **является производным** от класса «Animal».

При создании экземпляра «Bird» мы можем обращаться как к полям, определенным в классе Bird, так и к наследуемым полям класса «Animal», конечно, если эти поля не являются закрытыми:

```
Bird b = new Bird();
b.kindOfAnimal = “Орел”;
```

```
b.isMammal = false;
b.featherColor = "темно-желтый";
```

//Примечание редактора. Это фрагмент кода, о корректности которого нельзя судить без знания контекста. Доступ к свойствам зависит от того, где появляется приведенный выше текст: если из метода класса Bird, то доступ к защищенному (protected) свойству isMammal разрешен, поскольку класс Bird — потомок класса Animal; если же из метода класса клиента, то это свойство будет недоступно.

*Более важное замечание: о наследовании можно говорить, только когда объекты двух классов связаны отношением «**является**». Каждая птица (объект класса Bird) является животным (объектом класса Animal). Поэтому класс Bird может быть объявлен наследником класса Animal, но класс Car, автомобили, нельзя объявлять наследником класса Animal, поскольку автомобили **не являются** животными.*

Для простоты мы не включили в программу объявление *методов* в классах «Animal» и «Bird», но для них действуют те же правила доступа, что и для полей. Производный класс может вызывать любые методы в родительском классе, если они не объявлены закрытыми.

Наследование возможностей работы с окнами

Предположим, вам нужно написать программу, выполняемую в обычном окне, и понадобятся возможности изменения размера окна, его разворачивания, сворачивания, перетаскивания и некоторые другие. Имеет смысл, чтобы ваш класс «наследовал» возможности класса, который уже работает с подобным типом интерфейса. Обычным выбором родительского класса становится класс System.Windows.Forms.Form.

В таком случае пишем:

```
class MyFancyClass : Form
{
...
}
```

Фактически мы говорим: «Я создаю собственный класс, но он должен автоматически наследовать все возможности класса «Form».

Когда следует использовать наследование

Для использования класса совсем не обязательно наследовать от него поля! Наследование лучше всего подходит для тех случаев, когда то, чего вы пытаетесь добиться, можно получить от существующего класса, дополнительно расширив его или настроив.

Строительный блок: Наследование

Класс может наследовать свойства и действия другого класса.

```
public class Musician
{
 public string name;
 public int age;
 public bool canSing;
}
```

Класс «Guitarist» наследует все поля из класса «», а кроме того, имеет два собственных поля.
Имя класса Musician, написанное после двоеточия, сообщает компьютеру, что новый класс «Guitarist» наследует поля и методы родительского класса «Musician».

```
public class Guitarist : Musician
{
 public string guitarType;
 public string guitarBrand;
}
```

Создав объект класса Guitarist, можно установить его свойства, если они являются открытыми для доступа.

```
Guitarist g = new Guitarist();
g.name = «Jimbo van Helsdingen»;
g.ageInYears = 28;
g.canSing = false;
g.guitarType = «Acoustic»;
g.guitarBrand = «Gibson»;
```

Часть 3. Программирование в .NET Framework

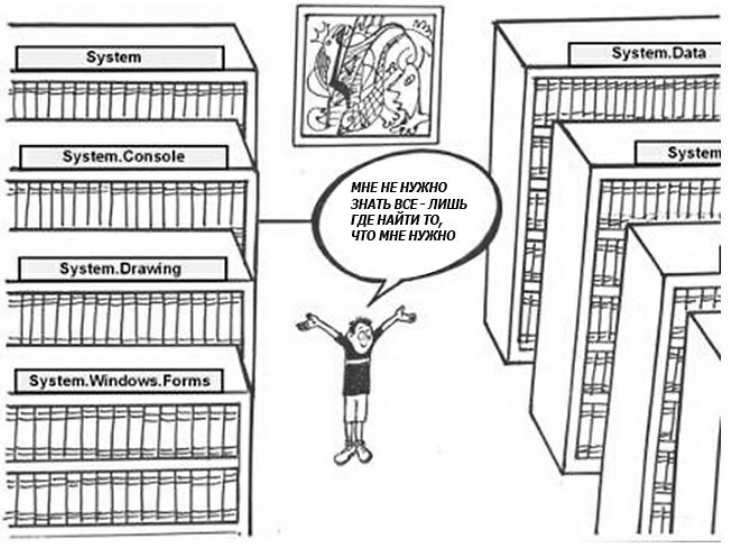
Что такое .NET Framework?

Программировать на языке C# без .NET Framework — это все равно что писать ручкой без чернил. Сколько ни води ею по бумаге, написать ничего не удастся.

Так что же такое .NET Framework? Это платформа программирования, разработанная корпорацией Microsoft, и язык C# создавался специально для .нее. Платформа .NET Framework состоит из двух частей.

1. Во-первых, она содержит огромную библиотеку классов, которые можно вызывать из программ, созданных на C#. Такая возможность избавляет от необходимости писать все заново.
2. Во-вторых, в ее состав входит среда выполнения, управляющая запуском и работой готовых программ (это происходит абсолютно незаметно для вас и не вызывает никаких затруднений).

Итак, при написании программы на языке C# (или на любом другом языке .NET) наряду с созданием собственного кода вызываются классы, хранящиеся в библиотеке.



Библиотека .NET framework содержит огромное количество классов, и некоторые из них настолько сложны, что мы даже не будем пытаться рассмотреть все сразу. Расскажем лишь о тех, которые, на наш взгляд, наиболее полезны для начинающих программистов. Со временем вы станете запоминать классы, которыми пользуетесь чаще, чем другими, и постепенно хорошо их

освоите. Ведь когда часто берешь в библиотеке любимую книгу, то легко запоминаешь, на какой полке ее искать.

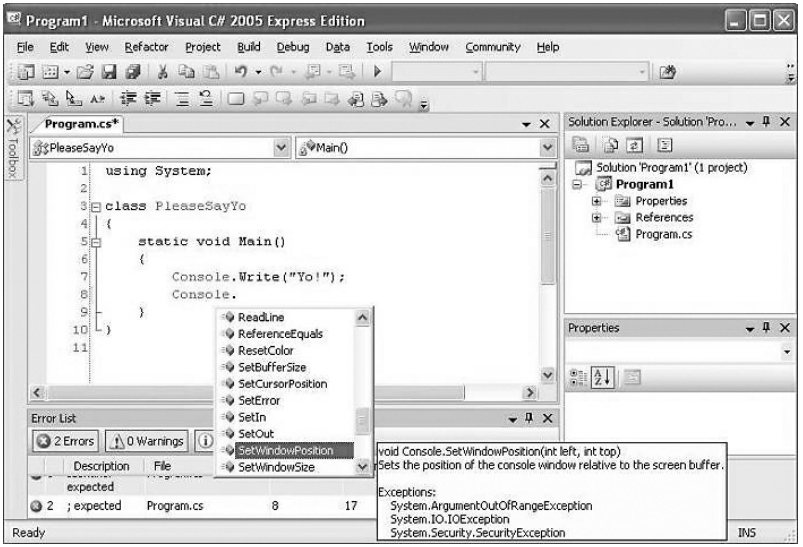
Обращаю ваше внимание: в этой части книги приведено много примеров программ, которые вы должны сначала **испытать в работе**, а затем попытаться изменить их, чтобы добиться **новых результатов**. Ранее мы уже рассказывали, как запускать примеры, представленные на вложенном в книгу диске.

Как изменять образцы программ и расширять их возможности

Я слышу вашевозмущение: «Легко сказать: измени программу! Но откуда мне знать, в каком направлении двигаться? В приведенных примерах используются лишь некоторые методы и классы .NET Framework, но ведь их очень много! Как о них узнать?»

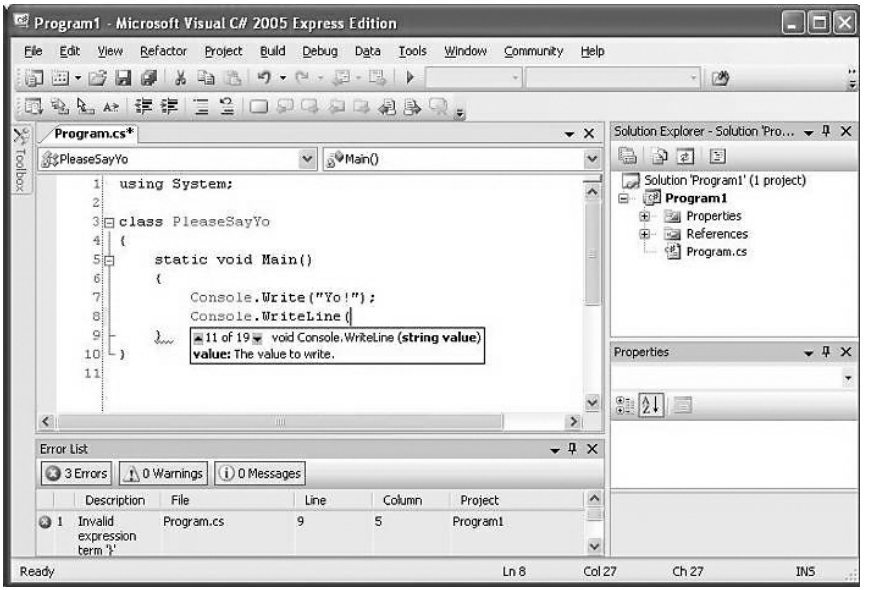
Для этого есть несколько способов:

- Просмотреть библиотеку классов .NET Framework SDK, которая включена в справочную систему Visual C# Express.
- Воспользоваться интеллектуальной подсказкой Visual C# Express. Как только вы введете в ее окошко имя пространства имен или класса и поставите точку, автоматически раскроется список всех доступных классов, методов, полей. Достаточно выбрать нужный элемент списка. Например, после ввода «Console.» вы увидите список методов и других членов класса Console.



Подсказка работает и для методов. Если поставите открывающую скобку после имени метода, то увидите типы параметров, которые он может принимать. Зачастую в C# используются одноименные методы с различными наборами параметров — в таком случае список надо «прокру-

чивать» (перемещаться по нему) вверх и вниз при помощи клавиш с изображенными на них стрелками. В следующем примере показано, что произойдет, если набрать «Console.WriteLine(». Visual C# Express подскажет о существовании 19 различных способов вызова метода WriteLine. Мы прокрутили список вниз до 11-й позиции (см. рисунок ниже).



Консольные приложения

Понятие «консоль» пришло к нам из тех времен, когда были популярны большие компьютеры, их называли мейнфреймами. Компания размещала в каком-нибудь помещении один гигантский

компьютер, а на рабочих местах служащих устанавливалась только клавиатура и простенький монитор, называвшийся консолью. Клавиатура и монитор подключались к тому самому «монстру», спрятанному от посторонних глаз в отдельной комнате. Такие мониторы не умели отображать графику — только текст. Информация передавалась в мейнфрейм при помощи клавиатуры – основного устройства ввода, а консоль – основное устройство вывода – позволяла компьютеру представлять информацию пользователю.

Сегодня мониторы большинства компьютеров имеют гораздо более совершенные возможности и способны отображать не только текст, но и, например, фотографии.

Однако при выполнении многих задач никаких графических изысков не требуется. Например, программа, получающая какие-либо данные с сервера в Интернете и сохраняющая их в файле на вашем компьютере, должна уметь выводить только два сообщения: «идет получение данных» и «готово». Зачем тратить время на разработку затейливого пользовательского интерфейса, который занимает много памяти? Именно по этой причине в библиотеку .NET включен класс для быстрого написания консольных приложений.

Не стоит относиться снисходительно к консольным приложениям и считать их слишком примитивными. Опытные программисты предпочитают не тратить усилия на интерфейсные ухищрения и работают в основном с консольными приложениями.

Конечно, если вашей программой будет пользоваться кто-то еще, то вы, вероятно, окажете ему большую услугу, сделав интерфейс чуть более дружественным, чем в стандартном консольном приложении.

Некоторые полезные методы:

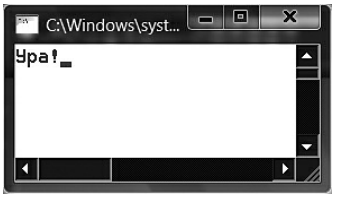
1. **Console.ReadLine** — считывает строку символов, введенную с клавиатуры (или иного устройства ввода).
2. **Console.WriteLine** — выводит текст на экран (или иное устройство вывода) начиная с новой строки.
3. **Console.Write** — выводит на экран ряд символов без перехода на новую строку.

Пример программы 1

Следующая программа выводит на экран слово «Ура!» и после этого ожидает нажатия клавиши ВВОД.

Код программы 1

```
using System;
 class PleaseSayUra
 {
  static void Main()
  {
    // Выводим слово на экран
    Console.Write("Ура!");
    // Ожидаем нажатия клавиши ВВОД
    Console.ReadLine();
  }
}
```



Пример программы 2

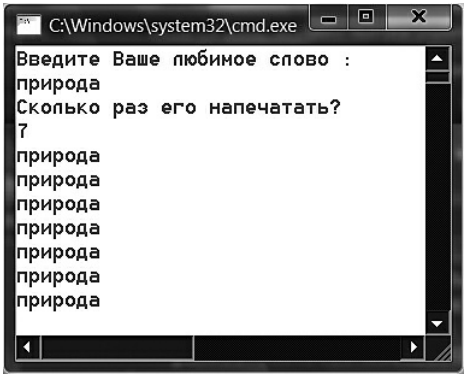
Следующая программа:

- просит пользователя ввести слово при помощи клавиатуры;
- спрашивает, сколько раз это слово должно быть выведено на экран;
- выводит слово на экран указанное количество раз, причем в каждом случае с новой строки.

Код программы 2

```
using System;

 class WriteAFewLines
 {
  static void Main()
  {
   Console.WriteLine("Введите Ваше любимое слово : ");
   // Сохраняем в строковой переменной введенное пользователем слово
   string favouriteWord = Console.ReadLine();
```

```
    Console.WriteLine("Сколько раз его напечатать? ");
    // Сохраняем в целочисленной переменной введенное число
    // (При неправильном вводе числа произойдет ошибка)
    int numberOfTimes = Convert.ToInt32(Console.ReadLine());

    // Выводим на экран слово указанное количество раз
    for (int i = 0; i < numberOfTimes; i++)
    {
     Console.WriteLine(favouriteWord);
    }

    // Ожидаем нажатия клавиши ВВОД
    Console.ReadLine();
  }
}
```



Приложения Windows Forms

Если вы захотите писать программы, похожие на привычные приложения Windows, то наверняка воспользуетесь классами из пространства имен System.Windows.Forms. Они позволяют задействовать кнопки, списки, текстовые поля, меню, окна сообщений и множество других «элементов управления». Элементы управления — это то, что вы помещаете в форму. Они нужны для вывода информации, например, текстовой (элемент управления Label) или графической (элемент управления PictureBox), либо для выполнения определенных действий, например, выбора значения или перехода к другой форме после нажатия кнопки. Возможно, создавая программы на C#, вы будете использовать классы из System.Windows.Forms.

Очевидно, что понятие «форма», принятое в программировании, родственно понятию «форма анкеты» или «форма документа», которое применяется в обычной жизни. Форма — это то, где можно в определенном порядке расположить различные элементы (текст, картинки, поля для за-

полнения и т. д.). Когда нам дают готовую форму документа и просят ее *заполнить*, мы обычно *читаем* содержащуюся в ней типовую информацию, а затем *вписываем недостающие данные в* определенные строки.

В программировании понятие формы во многом похоже: форма позволяет размещать текст, изображения, поля ввода, кнопки и т. п., добиваясь их точного расположения на экране. В консольном приложении на экран выводятся только строки текста.

Компания Microsoft предоставила в составе библиотеки классов .NET Framework огромное количество «элементов управления», которые можно помещать на формы. Освоив этот инструмент, вы сможете быстро создавать эффектные приложения.

Некоторые полезные классы из пространства имен System.Windows.Forms

Ниже приведены примеры классов элементов управления, которые можно размещать на формах:

- Label (Метка).
- Button (Кнопка).
- ListBox (Список).
- CheckBox (Флажок).
- RadioButton (Переключатель).
- MessageBox (Окно сообщений).
- Menu (Меню).
- TabControl (Управление вкладками).
- Toolbar (Панель инструментов).
- TreeView (Дерево).
- DataGrid (Сетка данных).
- PictureBox (Изображение).
- RichTextBox (Текстовое поле с поддержкой формата RTF).

Работа с примерами программ Windows Forms в Visual C# Express

Возможно, вы предпочтете не использовать уже заготовленные примеры проектов, а *разрабатывать их «с нуля»*. В таком случае нужно учесть, что для каждого проекта C# Express сразу же создает два файла (с именами **Form1.cs** и **Program.cs**) и наполняет их исходным кодом на языке C#, то есть вы изначально получаете простейшую, но полноценную программу. Предлагаемый нами способ работы с уже полученным проектом состоит в выполнении следующих действий:

- Удалите файл Form1.cs.
- Замените код в файле Program.cs на код примера, с которым вы работаете.

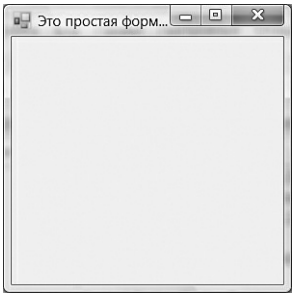
Оба этих действия не понадобятся, если вы открываете программы с помощью команды «Открыть проект» в меню «Файл» и находите нужный проект в той папке, куда его поместили после разархивации.

Пример программы 3

Рассмотрим пример простейшего приложения Windows Forms. Оно всего лишь создает новую форму и выводит определенный текст в заголовок окна формы.

Код программы 3

```
using System.Windows.Forms;

 class SimpleWindowsForm : Form
 {
  // Метод-конструктор нашего класса
  public SimpleWindowsForm()
  {
  // Указываем заголовок окна
  this.Text = “Это простая форма с заголовком”;
  }

  static void Main()
  {
   // Создаем новый экземпляр класса
   //и запускаем его на выполнение
   // В результате на экране дисплея откроется форма
   Application.Run(new SimpleWindowsForm());
  }
 }
```



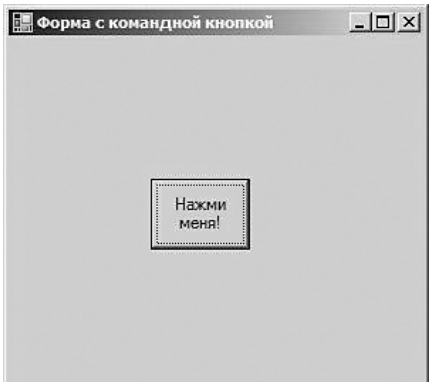
Пример программы 4

Следующий пример тоже достаточно прост, но мы делаем шаг вперед — размещаем на форме кнопку.

Код программы 4

```
using System.Windows.Forms;

 class SimpleWindowsFormWithButton : Form
  {
   Button button1;

   // Метод-конструктор нашего класса
   public SimpleWindowsFormWithButton()
   {
    // Указываем заголовок окна
    this.Text = “Форма с командной кнопкой”;

    // Добавляем кнопку в коллекцию элементов управления формы
    // Хотя на кнопке написано: “Нажми меня!”,
    // пока при нажатии ничего не происходит!
    button1 = new Button();
    button1.Text = “Нажми меня!”;
    button1.Top = 100;
    button1.Left = 100;
    button1.Height = 50;
    button1.Width = 70;
    this.Controls.Add(button1);
   }

  static void Main()
```

```
  {
    // Создаем и запускаем форму
    Application.Run(new SimpleWindowsFormWithButton());
  }
}
```



Пример программы 5

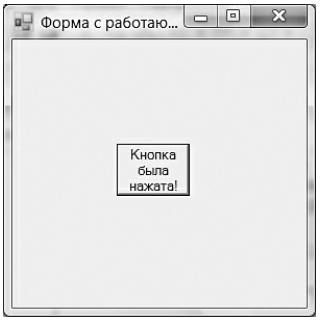
Кнопку на форму мы поместили, но при нажатии на нее ничего не происходит. Это скучно.

Нам нужно описать метод, который будет выполнять какое-либо действие при нажатии на кнопку. Пусть при этом текст в заголовке окна будет меняться. Поскольку такой метод отслеживает наступление некоторого события (в нашем случае – нажатие на кнопку) и затем каким-то образом обрабатывает его, он, напомним, называется «обработчик события». Кроме того, надо привязать обработчик события к соответствующему событию, то есть к нажатию на кнопку.

Код программы 5

```
using System;
using System.Windows.Forms;
using System.Drawing;

class FormWithWorkingButton : Form
{
    Button mrButton;
    // Метод-конструктор нашего класса
    public FormWithWorkingButton()
    {
        // Указываем заголовок окна
        this.Text = "Форма с работающей кнопкой!";
```

```
        // Добавляем кнопку и привязываем ее к обработчику события
        mrButton = new Button();
        mrButton.Text = “Нажми меня”;
        mrButton.Top = 100;
        mrButton.Left = 100;
        mrButton.Height = 50;
        mrButton.Width = 70;
        mrButton.Click += new System.EventHandler(mrButton_Click);
        this.Controls.Add(mrButton);
    }

    static void Main()
    {
        // Создаем и запускаем форму
        Application.Run(new FormWithWorkingButton());
    }

    // Обработчик события, срабатывающий при нажатии кнопки
    void mrButton_Click(object sender, EventArgs e)
    {
        // Изменяем заголовок окна
        mrButton.Text = “Кнопка была нажата!”;
    }
}
```



Пример программы 6

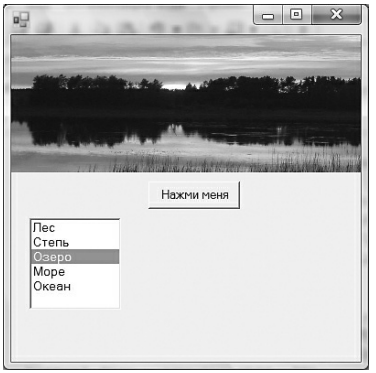
Мы добились успеха: наша программа умеет выполнять основные действия. Теперь добавим на форму несколько новых элементов управления, аккуратно разместим их и немного поработаем с ними. Возьмем элементы управления 4-х типов: Button, ListBox, MessageBox и PictureBox.

Обратите внимание: кроме System.Windows.Forms в этом примере упоминается пространство имен System.Drawing. Дело в том, что мы используем элемент управления PictureBox, а для работы с изображениями требуются классы Drawing.

Код программы 6

```
using System.Windows.Forms;
using System.Drawing;

class MyForm : Form
{
 // Объявим элемент ListBox как поле класса:
 // нам придется обращаться к нему из разных методов
 ListBox listBox1;

 // Метод-конструктор нашего класса
 public MyForm()
 {
     //Размеры формы
     this.Size = new Size(400, 400);
     // Создадим элемент PictureBox, поместим в него изображение,
      // добавим его на форму
      PictureBox pictureBox1 = new PictureBox();
      pictureBox1.SizeMode = PictureBoxSizeMode.StretchImage;
      Bitmap image1 = new Bitmap ("..//..//images//Zakat.jpg");
      pictureBox1.ClientSize = new Size(this.Width, 150);
      pictureBox1.Image = (Image)image1;
      this.Controls.Add(pictureBox1);
      // Создаем объект Button, определяем некоторые из его свойств
      Button button1 = new System.Windows.Forms.Button();
      button1.Location = new Point(150, 160);
      button1.Size = new Size(100, 30);
      button1.Text = "Нажми меня";
      button1.Click += new System.EventHandler(button1_Click);
      this.Controls.Add(button1);

      // Создаем ListBox, определяем свойства и добавляем на форму
      listBox1 = new System.Windows.Forms.ListBox();
      listBox1.Location = new System.Drawing.Point(20, 200);
      listBox1.Size = new Size(100, 100);
      listBox1.Items.Add("Лес");
      listBox1.Items.Add("Степь ");
      listBox1.Items.Add("Озеро");
      listBox1.Items.Add("Море");
```

```
        listBox1.Items.Add("Океан");
        listBox1.SelectedIndex = 2;
        this.Controls.Add(listBox1);
    }

    // Обработчик события, срабатывающий при нажатии кнопки
    void button1_Click(object sender, System.EventArgs e)
    {
     // Выводим сообщение с указанием выбранного в списке пункта
     MessageBox.Show(this, "Вы выбрали  " + listBox1.SelectedItem,
     "Уведомление", MessageBoxButtons.OK);
    }

    static void Main()
    {
     // Создаем и запускаем форму
     Application.Run(new MyForm());
    }

    private void InitializeComponent()
    {
        this.SuspendLayout();
        //
        // MyForm
        //
        this.BackColor = System.Drawing.SystemColors.Control;
        this.ClientSize = new System.Drawing.Size(292, 273);
        this.Name = "MyForm";
        this.ResumeLayout(false);
    }
}
```



Пример программы 7

Итак, настала пора испытать свои силы. Попробуем написать одну действительно большую программу, используя целый ряд полезных элементов управления. Объем кода может показаться **пугающе большим**, но программа вам пригодится, когда нужно будет вспомнить, для чего нужен тот или иной элемент управления.

Программа весьма интересна и стоит того, чтобы вы тщательно с ней поработали. После ее запуска открывается форма, в которой меню построено при помощи элемента управления TreeView. Пункты меню представлены в виде листьев на ветке дерева. Выбор того или иного листочка приводит к тому, что обработчик события заселяет форму новыми элементами управления. Поскольку эти элементы задаются определенным пунктом меню, их тип зависит от того, какой именно пункт выбран. С добавляемыми элементами тоже можно работать. Благодаря этому проекту появляется возможность познакомиться с самыми разными элементами управления, а главное — понять, как они создаются и как их использовать в собственных проектах.

Досконально разбирать текст всей программы необязательно, но когда у вас возникнут вопросы по использованию, например, элемента CheckBox, вы можете просмотреть ту ее часть, которая касается этого элемента.

Напоминаем: для того чтобы использовать элементы управления PictureBox и DataGridView, требуются пространства имен System.Drawing, System.Data и System.Xml.

Код программы 7

```
using System;
using System.Windows.Forms;
using System.Drawing;
using System.Data;
using System.Xml;

class FormWithManyControls : Form
{
    TreeView treeView1;
    Panel panel1;
    CheckBox checkBox1, checkBox2;
    RadioButton radioButton1, radioButton2;
    ListBox listBox1;

    // Метод-конструктор нашего класса
    public FormWithManyControls()
    {
        // Указываем размеры и заголовок окна

        this.Text = "Форма, включающая различные элементы управления!";
        this.Height = 800; this.Width = 900;
```

```
        // Добавляем элемент TreeView в качестве своеобразного меню

        treeView1 = new TreeView();
        treeView1.BackColor = Color.BurlyWood;
        treeView1.Dock = DockStyle.Left;
        treeView1.AfterSelect +=
        new System.Windows.Forms.TreeViewEventHandler(treeView1_AfterSelect);

        TreeNode tn = new TreeNode("Элементы");
        tn.Expand();

        tn.Nodes.Add(new TreeNode("[Очистить]"));
        tn.Nodes.Add(new TreeNode("Label"));
        tn.Nodes.Add(new TreeNode("Button"));
        tn.Nodes.Add(new TreeNode("CheckBox"));
        tn.Nodes.Add(new TreeNode("RadioButton"));
        tn.Nodes.Add(new TreeNode("ListBox"));
        tn.Nodes.Add(new TreeNode("TextBox"));
        tn.Nodes.Add(new TreeNode("TabControl"));
        tn.Nodes.Add(new TreeNode("DataGridView"));
        tn.Nodes.Add(new TreeNode("MainMenu"));
        tn.Nodes.Add(new TreeNode("ToolBar"));
        tn.Nodes.Add(new TreeNode("PictureBox"));
        tn.Nodes.Add(new TreeNode("RichTextBox"));

        treeView1.Nodes.Add(tn);

        this.Controls.Add(treeView1);

        // Добавляем панель для размещения остальных элементов управления

        panel1 = new Panel();
        panel1.Dock = DockStyle.Right;
        panel1.BorderStyle = BorderStyle.Fixed3D;
        panel1.Width = this.Width - treeView1.Width;

        this.Controls.Add(panel1);
    }

    // Обработчик событий, срабатывающий при выборе одного из узлов дерева
    // TreeView
    private void treeView1_AfterSelect
    (object sender, System.Windows.Forms.TreeViewEventArgs e)
    {
```

```
// Выполнение соответствующего действия при выборе любого из узлов

if (e.Node.Text == "[Очистить]")
{
    // Удаляем с панели все элементы управления
    panel1.Controls.Clear();
}
else if (e.Node.Text == "Button")
{
    // Добавляем на панель кнопку

    Button button1 = new Button();
    button1.Text = "Нажми меня!";
    button1.Location = new Point(300, 20);
    button1.Width = 120;
    button1.Height = 40;
    button1.Click += new EventHandler(button1_Click);

    panel1.Controls.Add(button1);
}
else if (e.Node.Text == "Label")
{
    // Добавляем на панель метку

    Label label1 = new Label();
    label1.Text =
    "Это метка. Используется для вывода текста на экран!";

    label1.Location = new Point(180, 70);
    label1.Width = 400;
    label1.Click += new EventHandler(label1_Click);

    panel1.Controls.Add(label1);
}
else if (e.Node.Text == "CheckBox")
{
    // Добавляем на панель несколько флажков

    checkBox1 = new CheckBox();
    checkBox1.Text = "Я способный!";
    checkBox1.Location = new Point(20, 40);
    checkBox1.Width = 150;
    checkBox1.CheckedChanged +=
    new EventHandler(CheckBox_CheckedChanged);
```

```
        panel1.Controls.Add(checkBox1);

        checkBox2 = new CheckBox();
        checkBox2.Text = "Я скромный!";
        checkBox2.Location = new Point(20, 80);
        checkBox2.Width = 150;
        checkBox2.CheckedChanged +=
        new EventHandler(CheckBox_CheckedChanged);
        panel1.Controls.Add(checkBox2);
    }
    else if (e.Node.Text == "RadioButton")
    {
        // Добавляем на панель несколько переключателей

        radioButton1 = new RadioButton();
        radioButton1.Text = "Я добрый!";
        radioButton1.Location = new Point(20, 120);
        radioButton1.Width = 150;
        radioButton1.Height = 30;
        //radioButton1.Size = new Size(20, 100);

        radioButton1.CheckedChanged +=
        new EventHandler(RadioButton_CheckedChanged);

        panel1.Controls.Add(radioButton1);

        radioButton2 = new RadioButton();
        radioButton2.Text = "Я трудолюбивый!";
        radioButton2.Location = new Point(20, 160);
        radioButton2.Width = 150;
        radioButton2.Height = 30;
        //radioButton2.Size = new Size(20, 100);

        radioButton2.CheckedChanged +=
        new EventHandler(RadioButton_CheckedChanged);

        panel1.Controls.Add(radioButton2);
    }
    else if (e.Node.Text == "ListBox")
    {
        // Добавляем на панель список

        listBox1 = new ListBox();
        listBox1.Items.Add("Зеленый");
```

```
        listBox1.Items.Add("Желтый");
        listBox1.Items.Add("Голубой");
        listBox1.Items.Add("Серый");

        listBox1.Location = new Point(20, 250);
        listBox1.Width = 100; listBox1.Height = 100;
        listBox1.SelectedIndexChanged +=
        new EventHandler(listBox1_SelectedIndexChanged);

        panel1.Controls.Add(listBox1);
    }
    else if (e.Node.Text == "TextBox")
    {
        // Добавляем на панель текстовое поле

        TextBox textBox1 = new TextBox();
        textBox1.Multiline = true;
        textBox1.Text = "Это текстовое окно. Сюда можно вводить текст!" +
            "\r\n" + " Сотрите этот текст и введите свой!";
        textBox1.Location = new Point(180, 100);
        textBox1.Width = 400; textBox1.Height = 40;

        panel1.Controls.Add(textBox1);
    }
    else if (e.Node.Text == "DataGridView")
    {
        // Добавляем на панель таблицу, заполненную данными из файла xml

        DataSet dataSet1 = new DataSet("Пример DataSet");
        dataSet1.ReadXml("..//..//images//marks.xml");

        DataGridView dataGridView1 = new DataGridView();
        dataGridView1.Width = 250;
        dataGridView1.Height = 150;
        dataGridView1.Location = new Point(20, 500);
        dataGridView1.AutoGenerateColumns = true;
        dataGridView1.DataSource = dataSet1;
        dataGridView1.DataMember = "предметы";
        //dataGridView1.DataMember = "оценка";

        //dataGridView1.ColumnCount = 2;
        panel1.Controls.Add(dataGridView1);

    }
```

```
else if (e.Node.Text == "TabControl")
{
    // Добавляем на панель элемент управления вкладками
    // и наполняем каждую вкладку содержимым

    TabControl tabControl1 = new TabControl();
    tabControl1.Location = new Point(190, 150);
    tabControl1.Size = new Size(300, 300);


    TabPage tabPage1 = new TabPage("Вадик");
    PictureBox pictureBox1 = new PictureBox();
    pictureBox1.SizeMode = PictureBoxSizeMode.StretchImage;
    pictureBox1.Image = new Bitmap("..//..//images//Vadik.jpg");

    pictureBox1.Size = new Size(300, 200);
    tabPage1.Controls.Add(pictureBox1);
    Label labelV = new Label();
    labelV.Top = 200;
    labelV.Size = new Size(300, 50);
    labelV.Text = "Это Вадик! Он любит купаться и работать на компьютере!";
    tabPage1.Controls.Add(labelV);
    tabControl1.TabPages.Add(tabPage1);

    TabPage tabPage2 = new TabPage("Его компьютер");
    PictureBox pictureBox2 = new PictureBox();
    pictureBox2.SizeMode = PictureBoxSizeMode.StretchImage;
    pictureBox2.Image = new Bitmap("..//..//images//comp.jpg");
    pictureBox2.Size = new Size(300, 200);
    tabPage2.Controls.Add(pictureBox2);
    Label labelC = new Label();
    labelC.Top = 200;
    labelC.Size = new Size(300, 50);
    labelC.Text = "Это компьютер Вадика! Пока Вадик купается, " +
        "он разрешает работать на компьютере ящерице!";
    tabPage2.Controls.Add(labelC);
    tabControl1.TabPages.Add(tabPage2);

    TabPage tabPage3 = new TabPage("Компьютерия");
    PictureBox pictureBox3 = new PictureBox();
    pictureBox3.SizeMode = PictureBoxSizeMode.StretchImage;
    pictureBox3.Image = new Bitmap("..//..//images//terra.jpg");
    pictureBox3.Size = new Size(300, 200);
    tabPage3.Controls.Add(pictureBox3);
```

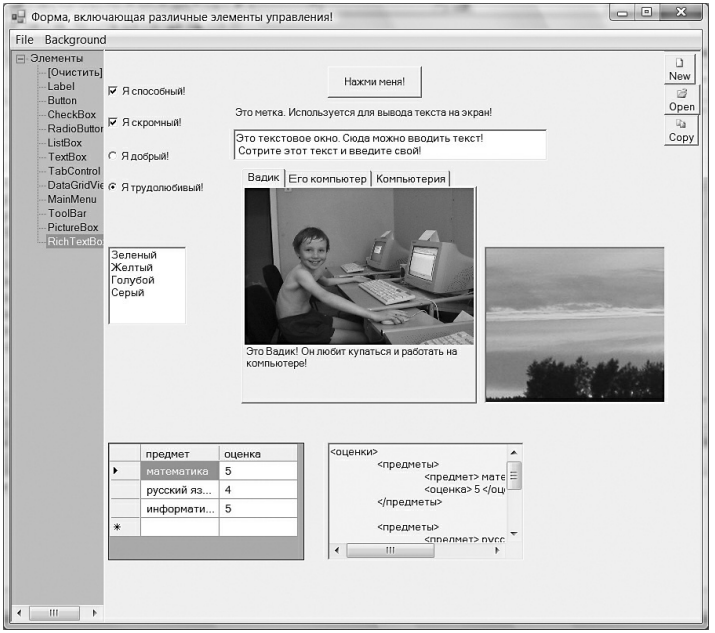
```
        Label labelT = new Label();
        labelT.Top = 200;
        labelT.Size = new Size(300, 50);
        labelT.Text = "Это страна Компьютерия!" +
            " Она расположена на берегу реки Тверца в Тверской области!";
        tabPage3.Controls.Add(labelT);
        tabControl1.TabPages.Add(tabPage3);

        panel1.Controls.Add(tabControl1);
    }
    else if (e.Node.Text == "PictureBox")
    {
        // Добавляем на панель изображение

        PictureBox pictureBox1 = new PictureBox();
        pictureBox1.Image = new Bitmap("..//..//images//Zakat.jpg");
        pictureBox1.BorderStyle = BorderStyle.Fixed3D;
        pictureBox1.Location = new Point(500, 250);
        pictureBox1.Size = new Size(250, 200);

        panel1.Controls.Add(pictureBox1);
    }
    else if (e.Node.Text == "RichTextBox")
    {
        // Добавляем поле для ввода текста с форматированием
        // Загружаем в него данные из файла XML

        RichTextBox richTextBox1 = new RichTextBox();
        richTextBox1.LoadFile("..//..//images//marks.xml",
        RichTextBoxStreamType.UnicodePlainText);
        richTextBox1.WordWrap = false;
        richTextBox1.BorderStyle = BorderStyle.Fixed3D;
        richTextBox1.BackColor = Color.Beige;
        richTextBox1.Size = new Size(250, 150);
        richTextBox1.Location = new Point(300, 500);
       // panel1.Height - richTextBox1.Height - 5);

        panel1.Controls.Add(richTextBox1);
    }
    else if (e.Node.Text == "MainMenu")
    {
        // Добавляем классическое "меню" (появляется в верхней части окна)
        MainMenu mainMenu1 = new MainMenu();
```

```
        MenuItem menuItem1 = new MenuItem("File");
        menuItem1.MenuItems.Add("Exit",
        new EventHandler(mainMenu1_Exit_Select));
        mainMenu1.MenuItems.Add(menuItem1);

        MenuItem menuItem2 = new MenuItem("Background");
        menuItem2.MenuItems.Add("Choose",
        new EventHandler(mainMenu1_ColorOwn_Select));
        menuItem2.MenuItems.Add("White",
        new EventHandler(mainMenu1_ColorWhite_Select));
        mainMenu1.MenuItems.Add(menuItem2);

        this.Menu = mainMenu1;

        MessageBox.Show("Главное меню добавлено в окно " +
        "Испытайте его после нажатия OK.");

    }
    else if (e.Node.Text == "ToolBar")
    {
        // Добавляем на панель элемент "панель управления" с кнопками
        // быстрого вызова

        ToolBar toolBar1 = new ToolBar();
        toolBar1.Size = new Size(100, 100);
        toolBar1.Dock = DockStyle.Right;
        ImageList imageList1 = new ImageList();
        imageList1.Images.Add(new Bitmap("../../images/new.gif"));
        imageList1.Images.Add(new Bitmap("../../images/open.gif"));
        imageList1.Images.Add(new Bitmap("../../images/copy.gif"));
        toolBar1.ImageList = imageList1;

        ToolBarButton toolBarbutton1 = new ToolBarButton("New");
        toolBarbutton1.ImageIndex = 0;
        toolBar1.Buttons.Add(toolBarbutton1);

        ToolBarButton toolBarbutton2 = new ToolBarButton("Open");
        toolBarbutton2.ImageIndex = 1;
        toolBar1.Buttons.Add(toolBarbutton2);

        ToolBarButton toolBarButton3 = new ToolBarButton("Copy");
        toolBarButton3.ImageIndex = 2;
        toolBar1.Buttons.Add(toolBarButton3);
```

```
            toolBar1.ButtonClick +=
            new ToolBarButtonClickEventHandler(toolBar1_Click);

            panel1.Controls.Add(toolBar1);
        }

    }

    /* Обработчики событий для добавленных выше элементов управления */

    // Обработчик события, срабатывающий при щелчке мышью на метке
    void label1_Click(object sender, System.EventArgs e)
    {
        MessageBox.Show
        ("Да, у меток тоже есть событие Click. Но для них включение событий -
        редкость.");
    }

    // Обработчик события, срабатывающий при нажатии кнопки
    void button1_Click(object sender, System.EventArgs e)
    {
        MessageBox.Show("Пора, наконец-то вы нажали меня!");
    }

    // Обработчик события, срабатывающий при установке или снятии флажка
    void CheckBox_CheckedChanged(object sender, System.EventArgs e)
    {
        if (checkBox1.Checked && checkBox2.Checked)
        {
            MessageBox.Show("У Вас все получится!");
        }
        else if (checkBox1.Checked)
        {
            MessageBox.Show("Не здорово быть умным и не скромным!");
        }
        else if (checkBox2.Checked)
        {
            MessageBox.Show("Скромность украшает. Хорошо бы еще быть умным!");
        }
        else
        {
            MessageBox.Show("Ни скромности, ни таланта?");
        }
    }
```

```
// Обработчик события, срабатывающий при нажатии переключателя
void RadioButton_CheckedChanged(object sender, System.EventArgs e)
{
    if (radioButton1.Checked)
    {
        MessageBox.Show("Доброту все любят!");
    }
    else if (radioButton2.Checked)
    {
        MessageBox.Show("Это замечательно!");
    }
}

// Обработчик события, срабатывающий при выборе одного из пунктов списка
void listBox1_SelectedIndexChanged(object sender, System.EventArgs e)
{
    switch (listBox1.SelectedItem.ToString())
    {
        case ("Зеленый"): treeView1.BackColor = Color.Green; break;
        case ("Желтый"): treeView1.BackColor = Color.Yellow; break;
        case ("Голубой"): treeView1.BackColor = Color.Blue; break;
        case ("Серый"): treeView1.BackColor = Color.Gray; break;

    }
}

// Обработчик события, срабатывающий при выборе в меню пункта "White"
void mainMenu1_ColorWhite_Select(object sender, System.EventArgs e)
{
    treeView1.BackColor = Color.White;
}

// Обработчик события, срабатывающий при выборе в меню цвета
void mainMenu1_ColorOwn_Select(object sender, System.EventArgs e)
{
    ColorDialog colorDialog1 = new ColorDialog();
    colorDialog1.Color = treeView1.BackColor;
    colorDialog1.ShowDialog();
    treeView1.BackColor = colorDialog1.Color;
}

// Обработчик события, срабатывающий при выборе в меню пункта "exit"
void mainMenu1_Exit_Select(object sender, System.EventArgs e)
{
```

```
        if (
        MessageBox.Show("Вы уверены, что хотите закончить работу?",
        "Exit confirmation", MessageBoxButtons.YesNo)
        == DialogResult.Yes
       )
        {
            this.Dispose();
        }
    }

    // Обработчик события, срабатывающий при нажатии кнопки на панели инструмен-
тов
    void toolBar1_Click
    (object sender, System.Windows.Forms.ToolBarButtonClickEventArgs e)
    {
        if (e.Button.Text == "Open")
        {
            MessageBox.Show("Здесь мог бы открываться файл!");
        }
        else if (e.Button.Text == "New")
        {
```



```
            MessageBox.Show("Здесь мог бы создаваться файл!");
        }
        else if (e.Button.Text == "Copy")
        {
            MessageBox.Show("Здесь мог бы копироваться файл!");
        }
    }

    static void Main()
    {
        // // Создаем и запускаем форму
        Application.Run(new FormWithManyControls());
    }
}
```

Рисование

Классы, объединенные в пространство имен Drawing, позволяют работать с различными изображениями. Существует два основных типа компьютерных изображений.

Растровые представляют собой набор точек. Примером могут служить фотографии и значки.

Векторная графика — это изображения, составленные из геометрических фигур: линий, окружностей, прямоугольников и т. д. Например, план дома удобно представлять в виде векторного изображения.

Для начала продемонстрируем работу с растровой графикой. На компьютере часто приходится выполнять обработку изображений, например, фотографий. Для этого в библиотеке классов .NET Framework имеется немало полезных средств.

Пример программы 8

Для создания программы, отображающей на форме рисунок, хранящийся в файле, нам понадобится специальный элемент управления. Для этой цели прекрасно подходит PictureBox.

Код программы 8

```
using System;
using System.Windows.Forms;
using System.Drawing;

class PictureDisplayer : Form
{
    Bitmap image1;
    PictureBox pictureBox1;
```

```
// Метод-конструктор нашего класса
public PictureDisplayer()
{
    // Указываем размеры и заголовок окна

    this.Text = "Искусство аборигенов";
    this.Size = new Size(302, 240);

    // Подготавливаем поле для размещения изображения

    pictureBox1 = new PictureBox();
    pictureBox1.SizeMode = PictureBoxSizeMode.StretchImage;
    pictureBox1.BorderStyle = BorderStyle.Fixed3D;
    pictureBox1.ClientSize = new Size(300, 196);

    // Добавляем изображение в элемент PictureBox

    image1 = new Bitmap(@"../../images/Iskusstvo.jpg");
    pictureBox1.Image = (Image)image1;

    // Добавляем PictureBox (с изображением) на форму

    this.Controls.Add(pictureBox1);

}

static void Main()
{
    // Создаем и запускаем форму
    Application.Run(new PictureDisplayer());
}
```

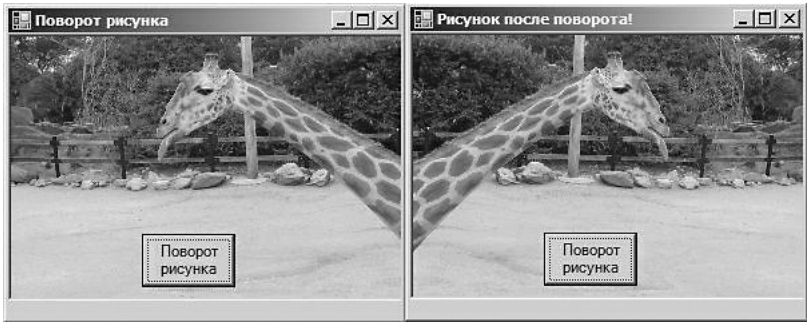
Пример программы 9

Следующая программа загружает фотографию с диска и после нажатия кнопки «flip» (Перевернуть) позволяет получить ее зеркальное отражение, расположенное по горизонтали:

Код программы 9

```
using System;
using System.Windows.Forms;
using System.Drawing;

class PictureFlipper : Form
{
    Button button1;
    Bitmap image1;
    PictureBox pictureBox1;

    // Метод-конструктор нашего класса
    public PictureFlipper()
    {
        // Указываем размеры и заголовок окна

        this.Text = “Поворот рисунка”;
        this.Size = new Size(302, 240);

        // Добавляем на форму кнопку

        button1 = new Button();
        button1.Text = “Поворот рисунка”;
        button1.Location = new Point(100, 150);
        button1.Size = new Size(70, 40);
        button1.Click += new System.EventHandler(button1_Click);
        this.Controls.Add(button1);

        // Добавляем элемент PictureBox на форму

        pictureBox1 = new PictureBox();
        pictureBox1.SizeMode = PictureBoxSizeMode.StretchImage;
        pictureBox1.BorderStyle = BorderStyle.Fixed3D;
        pictureBox1.ClientSize = new Size(300, 196);

        // Добавляем изображение в элемент PictureBox

        image1 = new Bitmap(@”../../images/Giraf.jpg”);
```

```
            pictureBox1.Image = (Image)image1;

            // Добавляем на форму элемент PictureBox

            this.Controls.Add(pictureBox1);

        }

        static void Main()
        {
            // Создаем и запускаем форму
            Application.Run(new PictureFlipper());
        }

        // Обработчик события, срабатывающий при нажатии кнопки
        void button1_Click(object sender, EventArgs e)
        {
            // Flip the image along the X axis (horizontally)
            image1.RotateFlip(RotateFlipType.RotateNoneFlipX);

            // Повторно вставляем изображение в элемент PictureBox
            pictureBox1.Image = (Image)image1;

            // Обновляем заголовок окна
            this.Text = "Рисунок после поворота!";
        }
}
```



Теперь перейдем к примерам работы с векторной графикой — изображениям, составленным из геометрических фигур.

Во всех примерах будут создаваться кнопка и обработчик событий, отвечающий за то, чтобы работа с графикой начиналась только после нажатия кнопки.

Необходимо усвоить несколько важных принципов. Они вполне логичны, но все-таки следует уяснить их, чтобы избежать возможных затруднений.

1. В обычном мире, прежде чем нарисовать линию, окружность, прямоугольник или иную фигуру, необходимо выбрать карандаш нужного цвета с грифелем определенной толщины. Для отрисовки на компьютере простейших фигур надо сначала создать **объект Pen (Перо)**. Например, с помощью данного фрагмента кода создается объект Pen, который рисует зеленую линию толщиной 3 пикселя:

```
Pen myGreenPen = new Pen(Color.Green, 3);
```

2. Для рисования **фигур с заливкой** потребуется нечто вроде кисти с красками. Предварительно следует создать **объект Brush (Кисть),** а затем выбрать цвет заливки и один из многочисленных типов кисти. В следующем фрагменте кода создается объект SolidBrush (Сплошная кисть) голубого цвета:

```
SolidBrush myBlueBrush = new SolidBrush(Color.Blue);
```

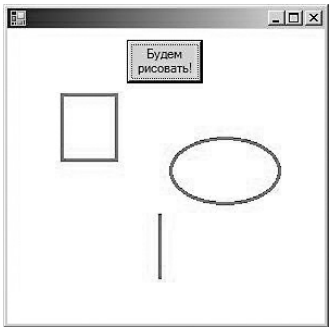
Пример программы 10

В этой программе в методе, названном DrawSomeShapes, рисуется линия, прямоугольник и эллипс.

Код программы 10

```
using System;
using System.Windows.Forms;
using System.Drawing;

class SimpleShapeMaker : Form
{
    // Метод-конструктор нашего класса
    public SimpleShapeMaker()
    {
        // Меняем цвет фона формы на белый

        this.BackColor = Color.White;

        // Добавляем на форму кнопку и привязываем ее к обработчику событий

        Button button1 = new Button();
        button1.Text = "Будем рисовать!";
        button1.Location = new Point(110, 10);
        button1.Size = new Size(70, 40);
        button1.BackColor = Color.LightGray;
        button1.Click += new System.EventHandler(button1_Click);
```

```
        this.Controls.Add(button1);
    }

    // Обработчик события, срабатывающий при нажатии кнопки
    void button1_Click(object o, System.EventArgs e)
    {
        // Вызов метода
        DrawSomeShapes();
    }

    // Метод для отрисовки на поверхности формы нескольких фигур
    void DrawSomeShapes()
    {
        // Подготовка области рисования на форме
        Graphics g = this.CreateGraphics();

        // Подготавливаем перо, рисующее красную линию толщиной 3 пикселя
        Pen redPen = new Pen(Color.Red, 3);

        // С помощью пера рисуем прямую линию, прямоугольник и эллипс
        g.DrawLine(redPen, 140, 170, 140, 230);
        g.DrawRectangle(redPen, 50, 60, 50, 60);
        g.DrawEllipse(redPen, 150, 100, 100, 60);

        // Очистка
        g.Dispose();
    }

    static void Main()
    {
        // Создаем и запускаем форму
        Application.Run(new SimpleShapeMaker());
    }
}
```



Пример программы 11

Теперь попробуем поиграть с мышкой – работать с графикой удобнее при помощи мыши, а не клавиатуры. Мы будем обрабатывать как растровые, так и с векторные изображения, используя некоторые события мыши.

Постараемся освоить некоторые новые действия, а именно — действия с точечными рисунками. Тратить время на подробное описание мы не будем, но *небольшое вступление* необходимо, чтобы рассказать о принципах работы приведенного ниже кода.

- Компьютерные программы формируют изображение на экране монитора, управляя цветом и яркостью маленьких точек, которые называются *пикселями*.
- Цвет пикселя определяется тремя цветовыми компонентами: красной (red), зеленой (green) и синей (blue) – в языках программирования часто используется сокращение RGB. Цвет и яркость пикселя можно изменять, регулируя интенсивность компонентов RGB, в пределах от 0 до 255 единиц. Например:
  - если red=255, green=0, blue=0 — цвет пикселя будет ярко-красным;
  - если red=255, green=255, blue=0 — цвет пикселя желтый.
- Компьютер может отслеживать положение курсора мыши, определяемое координатами X и Y (горизонтальная и вертикальная координаты). Так, верхний левый угол экрана имеет координаты X=0 и Y=0.

Код программы 11

```
using System;
using System.Windows.Forms;
using System.Drawing;

class FunWithTheMouse : Form
{
    // Объявляем объекты, доступные для разных методов

    PictureBox pictureBox1;
    Label label1;
    Point spotClicked;

    // Метод-конструктор нашего класса
    public FunWithTheMouse()
    {
        // Задаем размеры окна

        this.Size = new Size(640, 480);

        // Загружаем рисунок в элемент PictureBox и вставляем в форму

        pictureBox1 = new PictureBox();
```

```
        pictureBox1.Image = (Image)new Bitmap(@"../../images/Dog.bmp");
        pictureBox1.SizeMode = PictureBoxSizeMode.Normal;
        pictureBox1.Dock = DockStyle.Fill;
        this.Controls.Add(pictureBox1);

        // Добавляем метку с инструкциями в нижнюю часть экрана

        label1 = new Label();
        label1.BackColor = Color.Wheat;
        label1.Dock = DockStyle.Bottom;
        label1.Text =
        "При нажатой левой кнопке мыши можно рисовать прямоугольники. " +
        "Нажатая правая кнопка изменяет яркость прямоугольника " +
        "Нажав SHIFT и перемещая мышь, рисуем желтые кружки.";
        label1.TextAlign = ContentAlignment.MiddleCenter;
        this.Controls.Add(label1);

        // Привязываем PictureBox к обработчикам событий мыши

        this.pictureBox1.MouseDown += new MouseEventHandler(MouseButtonIsDown);
        this.pictureBox1.MouseUp += new MouseEventHandler(MouseButtonIsUp);
        this.pictureBox1.MouseMove += new MouseEventHandler(TheMouseMoved);
    }

    // Обработчик событий, срабатывающий при ПЕРЕМЕЩЕНИИ мыши
    public void TheMouseMoved(object sender, MouseEventArgs e)
    {
        // Если на клавиатуре нажата клавиша SHIFT
        if ((Control.ModifierKeys & Keys.Shift) == Keys.Shift)
        {
            // Подготовка области рисования на изображении
            System.Drawing.Graphics g =
                        this.pictureBox1.CreateGraphics();

            // Используем желтое перо
            System.Drawing.Pen yellowPen = new
                           System.Drawing.Pen(Color.Yellow, 3);

            // Рисуем окружность (эллипс, вписанный в квадрат)
            // Верхний левый угол квадрата имеет координаты X и Y
            // текущего положения мыши.
            g.DrawEllipse(yellowPen, e.X, e.Y, 40, 40);

            // Очистка
```

```
            g.Dispose();
        }
    }

    // Обработчик событий, срабатывающий при НАЖАТИИ кнопки мыши
    public void MouseButtonIsDown(object sender, MouseEventArgs e)
    {
        // Запоминаем точку, в которой произошло нажатие кнопки мыши.
        // Когда кнопка будет отпущена, нам понадобятся ее координаты

        spotClicked.X = e.X;// горизонтальная координата
        spotClicked.Y = e.Y;// вертикальная координата
    }

    // Обработчик событий, срабатывающий при ОТЖАТИИ кнопки мыши
    public void MouseButtonIsUp(object sender, MouseEventArgs e)
    {
        /* Пользователь отпустил кнопку мыши! */

        // Создаем прямоугольник (пока он еще не виден), ограничивающий
        // область изображения, с которой пользователь будет работать

        Rectangle r = new Rectangle();

        // Левый верхний угол прямоугольника соответствует точке,
        // в которой была нажата кнопка мыши.
        // Мы сохранили ее координаты.

        r.X = spotClicked.X;
        r.Y = spotClicked.Y;

        // Ширина и высота прямоугольника вычисляется
        // путем вычитания координат мыши в точке нажатия
        // из текущих координат (в точке отжатия кнопки).

        r.Width = e.X - spotClicked.X;
        r.Height = e.Y - spotClicked.Y;

        if (e.Button == MouseButtons.Left)
        {
            /* Если была нажата и отпущена левая кнопка мыши,
            рисуем видимый контур прямоугольника */
```

```
            // Подготовка области рисования на изображении
            Graphics g = this.pictureBox1.CreateGraphics();

            // Рисуем красный контур прямоугольника

            Pen redPen = new Pen(Color.Red, 2);
            g.DrawRectangle(redPen, r);
        }
        else
        {
            // Если была нажата другая кнопка, вызываем
            // метод, подсвечивающий область изображения

            ChangeLightness(r);
        }
    }

    // Метод, увеличивающий яркость выбранного участка изображения
    // путем увеличения яркости каждого пикселя этого участка
    public void ChangeLightness(Rectangle rect)
    {
        int newRed, newGreen, newBlue;
        Color pixel;

        // Копируем изображение, загруженное в PictureBox

        System.Drawing.Bitmap picture = new
                        Bitmap(this.pictureBox1.Image);

        // Операция увеличения яркости может занять много времени,
        // пользователя предупреждают, если выбран большой участок.

        if ((rect.Width > 150) || (rect.Height > 150))
        {

            DialogResult result = MessageBox.Show(
            "Выделенная область велика! " +
            "Изменение яркости может требовать значительного времени!",
            "Warning", MessageBoxButtons.OKCancel);

            // При нажатии кнопки Cancel (Отмена) выходим из метода
            // и возвращаемся к месту его вызова
```

```
            if (result == DialogResult.Cancel) return;
        }

        /* Перебираем последовательно все пиксели данного участка
        и удваиваем значение яркости компонент RGB пикселей */

        // Перебор по горизонтали слева направо...

        for (int x = rect.X; x < rect.X + rect.Width; x++)
        {
            // и по вертикали сверху вниз...

            for (int y = rect.Y; y < (rect.Y + rect.Height); y++)
            {
                // Считываем текущий пиксель

                pixel = picture.GetPixel(x, y);

                // Увеличиваем яркость цветовых компонент пикселя

                newRed = (int)Math.Round(pixel.R * 2.0, 0);
                if (newRed > 255) newRed = 255;
                newGreen = (int)Math.Round(pixel.G * 2.0, 0);
                if (newGreen > 255) newGreen = 255;
                newBlue = (int)Math.Round(pixel.B * 2.0, 0);
                if (newBlue > 255) newBlue = 255;

                // Присваиваем пикселю новые цветовые значения

                picture.SetPixel
                (x, y, Color.FromArgb(
                (byte)newRed, (byte)newGreen, (byte)newBlue));
            }
        }

        // Помещаем измененную копию изображения в PictureBox,
        // чтобы изменения отобразились на экране

        this.pictureBox1.Image = picture;

    }
```

```
    static void Main()
    {
        // Создаем экземпляр класса формы

        Application.Run(new FunWithTheMouse());
    }
}
```

Часть 4. Базы данных и XML

Введение в работу с базами данных

Большинство приложений должны работать с **базами данных**. Любой программист из крупной компании подтвердит, что роль баз данных в мире вычислительных технологий очень велика. Обладая умением оперировать с базами данных, можно создавать самые разнообразные и действительно **полезные** приложения.

Возможно, у вас на компьютере уже установлена система управления базами данных, например, Microsoft Access. В качестве альтернативы подойдет Microsoft SQL Server Express Edition, с помощью которого C можно научиться работать с базами данных SQL Server, используемыми во многих крупнейших компаниях по всему миру. SQL Server Express включен в пакет установки Visual C# Express, так что, возможно, он имеется и у вас.

Для работы с базами данных в библиотеке .NET Framework предусмотрены классы пространства имен System.Data. База данных в корне отличается от таких объектов, как изображения или документы текстовых редакторов, которые часто называют *неструктурированными*. Информация в базе данных **структурирована** и обычно хранится в **таблицах**, а каждая таблица состоит из **строк** и **столбцов**.

В программировании строки называют **записями**, а столбцы задают **поля записи**. Столбец таблицы определяет тип данных, хранимых в поле записи.

Ниже представлена таблица базы данных, содержащая информацию о планетах. Она имеет следующие столбцы: PlanetName (название планеты), DistanceFromSun (расстояние до Солнца) и Inhabitants (обитатели).

Таблица Planet

PlanetName	DistanceFromSun (единица измерения – тысяча колометров)	Inhabitants
Меркурий	57909	Меркуриане
Венера	108200	Венериане
Земля	149600	Земляне
Марс	227940	Марсиане
Юпитер	778400	Юпитериане
Знок	7208100	Знокиане
Сатурн	1423600	Сатурниане
Уран	2867000	Ураниане
Нептун	4488400	Нептуниане
Плутон	5909600	Плутониане

Из таблицы видно, что, например, Венера находится на расстоянии 108 200 тысяч километров от Солнца, а существа, ее населяющие, называются венериане.

Вот еще одна таблица, в которой содержатся сведения о ежегодной численности обитателей разных планет.

Эта совершенно секретная информация, нигде ранее не обнародованная. была получена с инопланетного звездолета, потерпевшего крушение в самой глубине пустыни Гоби. Иметь возможность ознакомиться с ней — большая честь. Очевидно, представители инопланетной цивилизации тоже используют базы данных SQL Server, что и позволило нам привести эти сведения в качестве примера.

Таблица Population

PlanetName	Year	Population
Меркурий	2008	40000
Венера	2008	25
Земля	2008	6000000000
Марс	2008	450000
Юпитер	2008	8326300200
Знок	2008	325000
Сатурн	2008	1000000
Уран	2008	7849345700
Нептун	2008	<NULL>
Плутон	2008	<NULL>
Меркурий	2009	35000
Венера	2009	3
Земля	2009	6500000000
Марс	2009	326800
Юпитер	2009	8451780500
Знок	2009	8700
Сатурн	2009	750000
Уран	2009	8237456000
Нептун	2009	<NULL>
Плутон	2009	<NULL>

Если продолжить разговор о Венере, то, обратите внимание, что в 2008 г. на Венере обитало 25 венериан, а через год, вероятно, в результате извержений вулканов, их осталось всего трое.

Не следует путать базы данных с электронными таблицами. Хотя способ представления данных в электронных таблицах внешне похож на тот, что использовался в приведенных выше примерах, обработка данных происходит в них иначе.

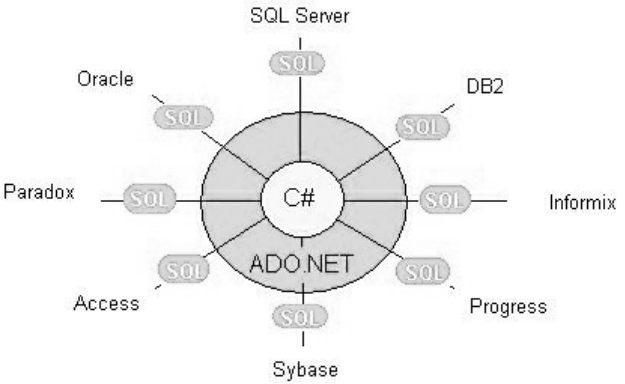
Язык SQL

Существует множество разнообразных систем управления базами данных: Microsoft Access, Oracle, DB2, Microsoft SQL Server, Informix, mySQL, и это далеко не полный список. А нам необходимо обратиться к базе данных из программы на языке C# и «объяснить» ей, что мы ищем.

Проще всего включить в код C# фрагмент на «языке базы данных», с помощью которого выполняется запрос к базе данных на получение нужных столбцов и строк.

(По правде говоря, существует еще и промежуточный уровень – ADO.NET, но сейчас мы не станем заострять на нем внимание.)

Много лет назад специалисты по базам данных договорились об использовании «единого языка баз данных», который понимали бы большинство существующих баз данных. Они назвали его SQL (от англ. Structured Query Language — язык структурированных запросов). Не следует путать *язык* SQL с *системой управления базами данных* SQL Server, разработанной корпорацией Microsoft. *Язык* SQL поддерживают системы управления базами данных (СУБД), выпускаемые самыми разными производителями.



Приступая к разговору о том, как использовать C# при работе с базами данных, сначала познакомимся с основами языка SQL. Ниже приведены примеры операторов на языке SQL и описывается результат их выполнения.

Основными командами SQL являются SELECT (для выбора некоторых данных), INSERT (для добавления новых данных) и UPDATE (для изменения информации, уже существующей в базе данных). Приведем примеры каждой команды.

Как правило, оператор SELECT записывается следующим образом:

```
SELECT <имена извлекаемых столбцов>
FROM <соответствующие таблицы базы данных>
WHERE <условие выбора, которое должно выполняться>
```

Рассмотрим пример выбора данных оператором Select:

```
SELECT *   FROM PLANET
```

Символ звездочка * означает выбор всех столбцов таблицы. Поэтому данный оператор вернет из базы данных всю таблицу Planet со всеми строками и столбцами.

Рассмотрим оператор

```
SELECT PlanetName, Inhabitants        FROM PLANET
```

Этот запрос возвращает столбцы «PlanetName» и «Inhabitants» со всеми строками из таблицы PLANET.

PlanetName	Inhabitants
Меркурий	Меркуриане
Венера	Венериане
Земля	Земляне
Марс	Марсиане
Юпитер	Юпитериане
Знок	Знокиане
Сатурн	Сатурниане
Уран	Ураниане
Нептун	Нептуниане
Плутон	Плутониане

Запрос

```
 SELECT PlanetName, Inhabitants       FROM PLANET
WHERE PlanetName='Venus'
```

возвращает столбцы «PlanetName» и «Inhabitants» из таблицы PLANET. В результат его выполнения будут включены только те строки, которые содержат значение «Venus» в столбце PlanetName.

PlanetName	Inhabitants
Венера	Венериане

Запрос:

```
 SELECT PlanetName, Population        FROM POPULATION
WHERE Population<100000
```

возвращает строки столбцов PlanetName и Population из таблицы POPULATION, для которых значение столбца Population меньше 100000.

PlanetName	Population
Меркурий	40000
Венера	25
Нептун	<NULL>
Плутон	<NULL>
Меркурий	35000
Венера	3
Знок	8700
Нептун	<NULL>
Плутон	<NULL>

Как правило, оператор INSERT записывается следующим образом:

```
INSERT INTO <таблица базы данных, к которой добавляются строки>
(<столбцы, в которые будут добавляться значения>)
 INSERT INTO PLANET
 (PlanetName, DistanceFromSun, Inhabitants)
 VALUES
 (Флафф, 23500000, 'Флаффиане')
```

Оператор INSERT добавляет в таблицу PLANET новую строку. Он не возвращает какого-либо результата в программу, но мы покажем, какой вид примет таблица после его выполнения.

PlanetName	DistanceFromSun	Inhabitants
Меркурий	57909	Меркуриане
Венера	108200	Венериане
Земля	149600	Земляне
Марс	227940	Марсиане
Юпитер	778400	Юпитериане
Знок	7208100	Знокиане
Сатурн	1423600	Сатурниане
Уран	2867000	Ураниане
Нептун	4488400	Нептуниане
Плутон	5909600	Плутониане
Флафф	23500000	Флаффиане

Как правило, оператор UPDATE записывается следующим образом:

```
UPDATE < таблица базы данных, в которую вносятся изменения>
SET <столбцы, в которые необходимо внести изменения> = <новые значения>
 UPDATE PLANET
 SET PlanetName= 'Стафф', Inhabitants='Стаффиане'
 WHERE PlanetName='Флафф'
```

Оператор изменяет некоторые значения в той строке, где столбец PlanetName имеет значение «Флафф», и не возвращает результат в программу. Далее показано, какой вид примет таблица после его выполнения.

PlanetName	DistanceFromSun	Inhabitants
Меркурий	57909	Меркуриане
Венера	108200	Венериане
Земля	149600	Земляне
Марс	227940	Марсиане
Юпитер	778400	Юпитериане
Знок	7208100	Знокиане

Сатурн	1423600	Сатурниане
Уран	2867000	Ураниане
Нептун	4488400	Нептуниане
Плутон	5909600	Плутониане
Стафф	23500000	Стаффиане

Связи и объединение таблиц базы данных

Если рассмотреть таблицы внимательно, то можно заметить, что между таблицами PLANET и POPULATION существует связь: и в той и в другой есть столбец с именем «PlanetName», который **связывает** обе таблицы. Благодаря этой связи можно собрать всю информацию об определенной планете.

Например, можно выбрать из обеих таблиц все строки, касающиеся Венеры:

PLANET		
PlanetName	**DistanceFromSun**	**Inhabitants**
Венера	108200	Венериане

POPULATION		
PlanetName	**Year**	**Population**
Венера	2008	25
Венера	2009	3

А затем объединить полученные данные в одну большую таблицу:

```
SELECT *
FROM PLANET INNER JOIN POPULATION ON PLANET.PlanetName=POPULATION.planetName
WHERE PlanetName='Венера'
```

PLANETS_AND_POPULATION				
PlanetName	**DistanceFromSun**	**Inhabitants**	**Year**	**Population**
Венера	108200	Венериане	2008	25
Венера	108200	Венериане	2009	3

Используемые СУБД

При работе с определенной СУБД необходимо учитывать ее особенности. С этой целью вводится понятие «провайдер данных» – у каждой СУБД он свой. С точки зрения программиста провайдер данных — это набор классов, обеспечивающих связь с базой данных и осуществляющих доступ к данным. Так, например, за связь с базой данных Microsoft SQL Server отвечает класс SqlConnection, а за связь с базой данных Access — OleDbConnection. Первый является частью SQLDataProvider, второй — OLEDbProvider.

Программный код меняется в зависимости от того, с какой базой предстоит работать. Для демонстрации возможностей использования баз данных, работающих под управлением различных СУБД, мы создали базу данных Planets в Microsoft SQL Server и в Access. Поскольку формат базы данных в Access 2007 отличается от формата, принятого в предыдущих версиях, реализация

выполнена для версий Access 2007 и Access 2003. Естественно, что все особенности построения баз данных находят отражение в программном коде.

Три варианта базы данных Planets помещены в каталог с именем databases. Файлы Planets.mdf и Planets.ldf, задают базу данных SQL Server, файл Planets.accdb создан для базы данных Access 2007, а Planets.mdb — для базы данных Access 2003.

В программном коде указывается путь к файлам базы данных, поэтому папка databases должна находиться в том же каталоге, где находится программа, задающая решение, — в нашем случае это файл Examples.sln.

При работе с базой данных SQL Server программный код зависит от того, с какой версией СУБД приходится работать — с Microsoft SQL Server или Microsoft SQL Server Express Edition.

В приведенном ниже примере показаны четыре возможных варианта работы:

1. работа с базой данных при использовании Microsoft SQL Server Express Edition;
2. работа с базой данных при использовании Microsoft SQL Server;
3. работа с базой данных Access 2003;
4. работа с базой данных Access 2007.

Три варианта закомментированы, а один готов к использованию, в данном тексте это вариант работы с Microsoft SQL Server. В зависимости от того, что установлено на компьютере читателя, следует нужный вариант раскомментировать, а ненужный — закомментировать.

У большинства пользователей на компьютерах установлен комплект Microsoft Office, возможно, в нем имеется и приложение Access, так что для начала можно поработать с базами данных, подготовленными в этом приложении.

Когда вам понадобятся более широкие возможности программирования, мы рекомендуем вам установить SQL Server Express. К тому же умение работать с SQL Server гораздо выше ценится в деловой сфере, и чем скорее вы освоите этот инструмент, тем лучше. Загрузить его можно бесплатно по адресу: http://www.microsoft.com/rus/express/sql/download.

Обращение к базам данных из программы, написанной на языке C#

Далее в трех примерах программ на C# мы будем использовать классы *Connection*, *Command*, *DataReader*, *DataAdapter*. Префиксы *Sql* и *OleDb* указывают на то, с каким вариантом базы данных Planets мы работаем.

Пример программы 12

Следующая программа подключается к базе данных и посылает ей SQL-запрос. Затем выполняется несколько циклов получения результата запроса, после чего каждое значение из столбца PlanetName передается в элемент управления ListBox.

Код программы 12

```
using System.Windows.Forms;
using System.Data;
```

```
// Пространство имен для работы с базами данных SQL Server
using System.Data.SqlClient;
using System.Drawing;
// Пространство имен для работы с базами данных Access
using System.Data.OleDb;
class SimpleDataAccess : Form
{
    public SimpleDataAccess()
    {
        // Указываем заголовок окна
        this.Text = "Работа с базой данных. Чтение данных.";
        // Добавляем элементы управления – метку и список
        Label labelCaption = new Label();
        labelCaption.Text = "Планеты солнечной системы!";
        labelCaption.Location = new Point(30, 10);
        labelCaption.Width = 200;
        labelCaption.Parent = this;

        ListBox listPlanets = new ListBox();
        listPlanets.Location = new Point(30, 50);
        listPlanets.Width = 100;
        listPlanets.Parent = this;

        // Формируем запрос к базе данных –
        //запрашиваем информацию о планетах
        string sql = "SELECT * FROM PLANET";
        string connectionString;

        /*
        //Вариант 1
        // Подключаемся к базе данных SQL Server Express Edition

        // Указываем физический путь к базе данных PLANETS
        string dbLocation =
        ("../../../databases/planets.mdf");

        connectionString = @"data source=.\SQLEXPRESS;" +
               "User Instance=true;Integrated Security=SSPI;" +
          "AttachDBFilename=" + dbLocation;
        SqlConnection connection1 = new SqlConnection(connectionString);
       */

        //Вариант 2
        // Подключаемся к базе данных SQL Server 2005
```

```
connectionString =
     "data source = localhost; Initial Catalog = Planets;" +
      "Integrated Security = SSPI";
SqlConnection connection1 = new SqlConnection(connectionString);

//Открываем соединение
connection1.Open();

SqlCommand command1 = new SqlCommand(sql, connection1);
SqlDataReader dataReader1 = command1.ExecuteReader();
// Организуем циклический перебор полученных записей
//и выводим название каждой планеты в список
while (dataReader1.Read())
{
     listPlanets.Items.Add(dataReader1["PlanetName"]);
}

// Очистка
dataReader1.Close();
connection1.Close();
/*

//Вариант 3. Связывание с базой данных Access 2003 - *.mdb
connectionString = "Provider=Microsoft.Jet.OLEDB.4.0;" +
      @"Data Source= ../../../databases/planets.mdb";


//Вариант 4. Связывание с базой данных Access 2007 - *.accdb
connectionString = "Provider=Microsoft.Ace.OLEDB.12.0;" +
     @"Data Source= ../../../databases/planets.accdb";


OleDbConnection connection = new OleDbConnection(connectionString);
connection.Open();
OleDbCommand command = new OleDbCommand(sql, connection);
OleDbDataReader dataReader = command.ExecuteReader();

// Организуем циклический перебор полученных записей
//и выводим название каждой планеты в список
while (dataReader.Read())
{
     listPlanets.Items.Add(dataReader["PlanetName"]);
}
```

```
            // Очистка
            dataReader.Close();
            connection.Close();
             * */
        }

        static void Main()
        {
            // Создаем и запускаем форму
            Application.Run(new SimpleDataAccess());
        }
}
```

Пример программы 13

В этой программе на экран выводится *несколько столбцов* данных. Для их представления мы воспользуемся элементом DataGridView.

Программа выполняет тот же запрос, что и в предыдущем примере, но помещает результат в объект DataSet, который подключается к элементу управления DataGridView, а тот автоматически отображает все данные.

Подключение источника данных к визуальному элементу управления называется **привязкой,** или **связыванием данных**.

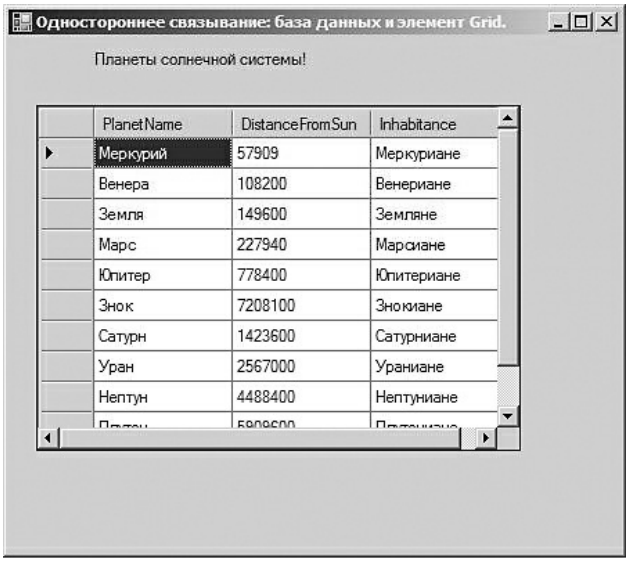
Код программы 13

```
using System.Windows.Forms;
using System.Data;
// Пространство имен для работы с базами данных SQL Server
```

```
using System.Data.SqlClient;
using System.Drawing;
// Пространство имен для работы с базами данных Access
using System.Data.OleDb;
class DataInGrid : Form
{
    public DataInGrid()
    {
        //Изменяем размеры формы
        this.Width = 450;
        this.Height = 400;

        // Указываем заголовок окна
        this.Text = "Одностороннее связывание:" +
            " база данных и элемент Grid.";
        // Добавляем элементы управления – метку и таблицу
        Label labelCaption = new Label();
        labelCaption.Text = "Планеты солнечной системы!";
        labelCaption.Location = new Point(60, 10);
        labelCaption.Width = 200;
        labelCaption.Parent = this;

        // Добавляем элемент DataGridView на форму

        DataGridView dataGridView1 = new DataGridView();
        dataGridView1.Width = 350;
        dataGridView1.Height = 250;
        dataGridView1.Location = new Point(20, 50);
        dataGridView1.DataMember = "Table";
        dataGridView1.AutoResizeColumns();
        this.Controls.Add(dataGridView1);


        // Формируем запрос к базе данных –
        //запрашиваем информацию о планетах
        string sql = "SELECT * FROM PLANET";
        string connectionString;
        // DataSet сохраняет данные в памяти
        //данные хранятся в виде таблиц данных DataTable
        DataSet dataSet1 = new DataSet();

        /*
        //Вариант 1
        // Подключаемся к базе данных SQL Server Express Edition
```

```
// Указываем физический путь к базе данных PLANETS
string dbLocation =
("../../../databases/planets.mdf");

connectionString = @"data source=.\SQLEXPRESS;" +
       "User Instance=true;Integrated Security=SSPI;" +
  "AttachDBFilename=" + dbLocation;
SqlConnection connection1 = new SqlConnection(connectionString);
*/
/*
//Вариант 2
// Подключаемся к базе данных SQL Server 2005
connectionString =
     "data source = localhost; Initial Catalog = Planets;" +
      "Integrated Security = SSPI";
SqlConnection connection1 = new SqlConnection(connectionString);

//Открываем соединение
connection1.Open();

// DataAdapter - посредник между базой данных и DataSet
SqlDataAdapter sqlDataAdapter1 = new SqlDataAdapter();

// Создаем объект DataAdapter,
//передаем ему данные запроса
sqlDataAdapter1.SelectCommand =
new SqlCommand(sql, connection1);

// Данные из адаптера поступают в DataSet
sqlDataAdapter1.Fill(dataSet1);

// Связываем данные с элементом DataGridView
DataGridView1.DataSource = dataSet1;

// Очистка
connection1.Close();
 * */

/*
//Вариант 3. Связывание с базой данных Access 2003 - *.mdb
connectionString = "Provider=Microsoft.Jet.OLEDB.4.0;" +
     @"Data Source= ../../../databases/planets.mdb";

*/
```

```
        //Вариант 4. Связывание с базой данных Access 2007 - *.accdb
        connectionString = "Provider=Microsoft.Ace.OLEDB.12.0;" +
            @"Data Source= ../../../databases/planets.accdb";


        OleDbConnection connection = new OleDbConnection(connectionString);
        connection.Open();

        OleDbDataAdapter dataAdapter = new OleDbDataAdapter();
        dataAdapter.SelectCommand = new OleDbCommand(sql, connection);

        dataAdapter.Fill(dataSet1);
        dataGridView1.DataSource = dataSet1;

        // Очистка
        connection.Close();
    }

    static void Main()
    {
        // Создаем и запускаем форму
        Application.Run(new DataInGrid());
    }
}
```



Пример программы 14

Научившись отображать данные с помощью элемента управления DataGridView, вы, наверное, обратили внимание, что внесенные в базу изменения не сохраняются. Поэтому применим другой подход и осуществим «**двухстороннюю привязку данных**», которая позволит не только просматривать открывшуюся таблицу, но и вводить изменения в элемент DataGridView, добавляя новые строки, изменяя содержимое имеющихся строк и удаляя ненужные. Объекты класса DataAdapter способны выполнять как операцию Select, получая данные запроса из базы данных, так и команды Insert, Update, Delete, изменяя содержимое таблицы базы данных.

Здесь мы позволим себе маленькую хитрость (это называется «повысить производительность труда») и не станем самостоятельно прописывать операторы UPDATE и INSERT. Пространство имен System.Data содержит небольшой, но очень полезный класс CommandBuilder, который умеет создавать команды SQL и автоматически их выполнять.

Код программы 14

```
using System;
using System.Windows.Forms;
using System.Data;
// Пространство имен для работы с базами данных SQL Server
using System.Data.SqlClient;
using System.Drawing;
using System.Diagnostics;
class DataInOutGrid: Form
{
    SqlDataAdapter dataAdapter;
    DataGridView dataGridView;

    public DataInOutGrid()
    {
        //Изменяем размеры формы
        this.Width = 450;
        this.Height = 400;

        // Указываем заголовок окна
        this.Text = "Двустороннее связывание:" +
            " база данных и элемент Grid.";
        // Добавляем элементы управления -
        //метку, таблицу и командную кнопку
        Label labelCaption = new Label();
        labelCaption.Text = "Планеты!";
        labelCaption.Location = new Point(60, 10);
        labelCaption.Width = 200;
        labelCaption.Parent = this;
```

```
// Добавляем элемент DataGridView на форму

dataGridView = new DataGridView();
dataGridView.Width = 350;
dataGridView.Height = 250;
dataGridView.Location = new Point(20, 50);
dataGridView.AutoResizeColumns();
this.Controls.Add(dataGridView);

// Добавляем командную кнопку
Button buttonSave = new Button();
buttonSave.Location = new Point(100, 320);
buttonSave.Width = 220;
buttonSave.Text = "Сохранить изменения в базе данных!";
buttonSave.Click +=
    new System.EventHandler(ButtonSave_Click);
buttonSave.Parent = this;

// Формируем запрос к базе данных -
//запрашиваем информацию о планетах
string sql = "SELECT * FROM PLANET";
string connectionString;
// DataTable сохраняет данные в памяти как таблицу
DataTable dataTable = new DataTable();
/*
//Вариант 1
// Подключаемся к базе данных SQL Server Express Edition

// Указываем физический путь к базе данных PLANETS
string dbLocation =
("../../../databases/planets.mdf");

connectionString = @"data source=.\SQLEXPRESS;" +
      "User Instance=true;Integrated Security=SSPI;" +
  "AttachDBFilename=" + dbLocation;
SqlConnection connection1 = new SqlConnection(connectionString);
*/

//Вариант 2
// Подключаемся к базе данных SQL Server 2005
connectionString =
    "data source = localhost; Initial Catalog = Planets;" +
     "Integrated Security = SSPI";
SqlConnection connection = new SqlConnection(connectionString);
```

```
        //Открываем соединение
        connection.Open();

        //Создаем команду
        SqlCommand sqlCommand = new SqlCommand(sql, connection);
        //Создаем адаптер
        // DataAdapter - посредник между базой данных и DataSet
        dataAdapter = new SqlDataAdapter(sqlCommand);

        //Создаем построитель команд
        //Для адаптера становится доступной команда Update
        SqlCommandBuilder commandBuilder =
            new SqlCommandBuilder(dataAdapter);

        // Данные из адаптера поступают в DataTable
        dataAdapter.Fill(dataTable);
        // Связываем данные с элементом DataGridView
        dataGridView.DataSource = dataTable;
        // Очистка
        connection.Close();
    }


    static void Main()
    {
        // Создаем и запускаем форму
        Application.Run(new DataInOutGrid());
    }

    void ButtonSave_Click(object sender, System.EventArgs args)
    {
        try
        {
            dataAdapter.Update((DataTable)dataGridView.DataSource);
            MessageBox.Show("Изменения в базе данных выполнены!",
                "Уведомление о результатах", MessageBoxButtons.OK);
        }
        catch(Exception)
        {
            MessageBox.Show("Изменения в базе данных выполнить не удалось!",
                    "Уведомление о результатах", MessageBoxButtons.OK);
        }
    }
}
```

На следующем рисунке показано содержимое базы данных, в которую добавлены две строки с названиями недавно открытых фантастических планет Солярис и Обитаемый остров (или коротко – Остров).

Поэкспериментируйте, изменяя существующие значения и вводя новые. Нажмите кнопку «Сохранить изменения» и закройте форму. Перезапустив программу, убедитесь, что все значения были сохранены в базе данных. Если вы работаете с Access, то измените код в соответствии с образцом, приведенным в программах Example 12 и Example 13.

Работа с XML-данными

Классы пространства имен System.Xml позволяют работать с XML-данными разными способами. Наиболее часто выполняются следующие задачи:

- открытие XML-документа;
- чтение фрагмента XML-данных для извлечения некоторых значений;
- сохранение XML-файла на диск.

Краткое введение в XML

Язык XML широко распространен, и, скорее всего, вы уже слышали о нем. Этот язык программирования удобен для восприятия как человека, так и машины. Большинство данных, с которыми работают различные вычислительные системы, обычным людям кажутся китайской грамотой, но XML-документы записываются в виде обычного текста.

Например, можно составить XML-документ для хранения на диске географических данных:

`<?xml version="1.0" encoding="utf-8" ?>`	**Каждый XML-документ начинается с такой строки, встречая которую, программа будет знать, что имеет дело именно с XML**
`-<Планета>` `Земля`	**Внешний блок Планета**

`- <Континент>`
`Южная Америка`
`<Страна столица="Рио-де-Жанейро">`
`Бразилия`
`</Страна>`
`<Страна столица="Буэнос-Айрес">`
`Аргентина`
`</Страна>`
`</Континент>`
	Вложенный блок Континент, содержащий информацию о материке Южная Америка
	Блоки более глубокого уровня вложения, содержат информацию о странах данного континента

`- <Континент>`
`Азия`
`<Страна столица="Дели">`
`Индия`
`</Страна>`
`<Страна столица="Бангкок">`
`Тайланд`
`</Страна>`
`</Континент>`
`</Планета>`
	Еще один вложенный блок Континент

XML весьма похож на HTML, но в HTML набор тегов фиксирован, каждый тег имеет строгий синтаксис и семантику. В XML можно определять *собственные теги*, соблюдая лишь общие правила синтаксиса записи текста.

Далее мы обсудим значение двух терминов, без знания которых невозможно дальнейшее изучение языка XML.

Элементы

XML-документ представляет собой запись в виде скобок, состоящую из **элементов**. Каждый элемент начинается с открывающей скобки — открывающего тега — и заканчивается закрывающей скобкой — закрывающим тегом. Открывающий и закрывающий теги заключаются в угловые скобки (<>). Признаком закрывающего тега является слеш, следующий за угловой скобкой (</>).

У элемента есть имя, оно задается в открывающем теге и повторяется в закрывающем теге, а между открывающим и закрывающим тегами вводят значение элемента и его содержание. В со-

держание элемента могут быть вложены **элементы**, являющиеся частью этого содержания. Элементы с одним именем нередко повторяются, даже если они не вложены один в другой.

В приведенном выше XML-документе один из элементов с именем <Континент> имеет значение «Южная Америка». В него вложены два элемента с именем <Страна>, содержательно задающие названия стран этого континента.

Имя элемента в открывающем теге	Значение	Закрывающий тег
<Страна>	Бразилия	</Страна>
<Школьник>	Ваня Курочкин	</Школьник>

Атрибуты

В открывающем теге можно задавать дополнительные свойства элемента. Для этого используются **атрибуты**. Каждый атрибут задается парой «имя = «значение»». Например, если для элемента <Страна> надо указать столицу, то можно создать *атрибут* элемента с именем «столица». А для элемента <Школьник> задать атрибуты «возраст» и «класс». В приведенном ниже примере у элемента «Country» со значением «Argentina» атрибут «capital» имеет значение «Buenos Aires».

Открывающий тег					Значение элемента	Закрывающий тег
Имя элемента	Имя атрибута		Значение атрибута			
<Страна	столица	=	«Буэнос-Айрес»	>	Аргентина	</Страна>

Читать подобные документы не составляет труда, а поскольку они имеют четкую структуру, то и компьютер можно научить этому без особых проблем. Например, составить такой набор инструкций: «Начать просмотр документа; при нахождении символа «<» — читать *имя элемента*. При нахождении символа «>» — читать *значение элемента*...» и так далее.

Некоторые элементы могут иметь только атрибуты и не иметь значения и содержания. Тогда открывающий и закрывающий теги объединяются. Например, элемент <Страна> можно задать следующим образом:

```
<Страна название = “Аргентина” столица = “Буэнос-Айрес” />
```

Для описания элемента использовались два атрибута, но не значение элемента.

Пример программы 15

Следующая программа считывает данные из XML-файла и отображает их на форме.

В ней используются три класса из пространства имен System.Xml:

- XmlDocument. Объекты этого класса задают Xml-документ. При создании объекта содержимое документа может быть прочитано из файла, получено из других источников или создано в ходе работы программы.

- XmlNodeList. Объекты этого класса могут содержать некоторый список элементов Xml-документа, найденный, например, в результате поиска.
- XmlNode. Объект этого класса задает один Xml-элемент.

Для поиска нужных фрагментов Xml-документа в программе используются так называемые **выражения xPath**. Они позволяют указать, какие именно элементы необходимо извлечь из XML-документа. Выражение xPath вида «/континент/страна» означает: «найти все элементы с именем «страна», вложенные в элемент с именем «континент»».

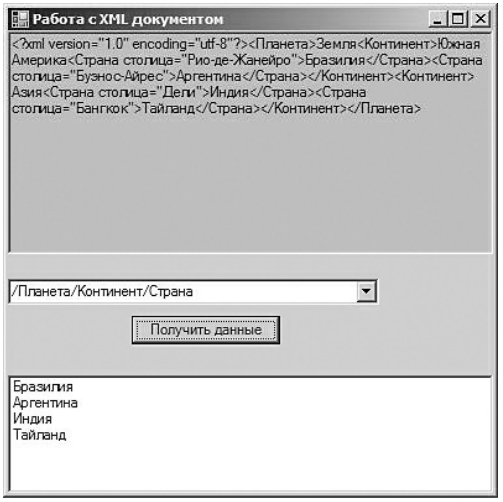
Код программы 15

```
using System;
using System.Windows.Forms;
using System.Drawing;
// Пространство имен для работы с XML-данными
using System.Xml;
//Пространство имен для работы с выражениями xPath
using System.Xml.XPath;
class XmlRetriever: Form
{
    ComboBox comboBox1;
    Button button1;
    ListBox listBox1;
    RichTextBox richTextBox1;
    XmlDocument xmlDoc;

    // Метод-конструктор нашего класса
    public XmlRetriever()
    {
        // Задаем заголовок и размеры окна
        this.Text = "Работа с XML-документом";
        this.Size = new Size(400, 400);

        // Создаем объект XmlDocument, используя xml-файл
        xmlDoc = new XmlDocument();
        xmlDoc.Load("../../docs/Planets.xml");

        // Создаем объект TextBox для вывода данных
        richTextBox1 = new RichTextBox();
        richTextBox1.Dock = DockStyle.Top;
        richTextBox1.AcceptsTab = true;
        richTextBox1.Height = 180;
        richTextBox1.ReadOnly = true;
        richTextBox1.BackColor = Color.Silver;
```

```
        // Помещаем XML-данные в элемент TextBox
        richTextBox1.Text = xmlDoc.OuterXml;
        this.Controls.Add(richTextBox1);

        // Создаем объект ComboBox
        // В элементы списка этого объекта записываем
        //различные выражения XPath,
        //позволяющие искать нужные элементы XML-документа
        comboBox1 = new ComboBox();
        comboBox1.Location = new Point(0, 200);
        comboBox1.Width = 300;
        comboBox1.Items.Add("/Планета");
        comboBox1.Items.Add("/Планета/Континент");
        comboBox1.Items.Add("/Планета/Континент/Страна");
        comboBox1.Items.Add(
            "/Планета/Континент/Страна[@столица='Рио-де-Жанейро']");
        comboBox1.SelectedIndex = 0;
        this.Controls.Add(comboBox1);

        // Создаем командную кнопку для поиска и отображения
        // соответствующих элементов Xml-документа
        button1 = new Button();
        button1.Text = "Получить данные";
        button1.Location = new Point(100, 230);
        button1.Width = 120;
        button1.Click += new EventHandler(Button1_Click);
        this.Controls.Add(button1);

        // Создаем элемент ListBox для отображения элементов
        listBox1 = new ListBox();
        listBox1.Dock = DockStyle.Bottom;
        listBox1.Location = new Point(10, 10);
        this.Controls.Add(listBox1);
    }

    static void Main()
    {
        // Создаем и запускаем новый экземпляр класса
        Application.Run(new XmlRetriever());
    }

    // Обработчик события, срабатывающий при нажатии кнопки
    void Button1_Click(object sender, EventArgs e)
    {
```

```
        XmlNodeList xmlNodes;
        XmlNode xmlElement;
        string elementValue;

        // Используем охраняемый блок try-catch,
        // что позволит в случае ошибок в выражениях XPath
        // перехватывать обработку исключений, выдавать сообщение
        // об ошибке и нормально продолжить работу приложения
        try
        {
            // Выбираем из XML-документа элементы, которые соответствуют
            // выражению XPath, заданному выбранным элементом ComboBox
            xmlNodes = xmlDoc.SelectNodes(comboBox1.Text);

            // Производим циклический перебор найденных элементов,
            // добавляя каждый элемент в ListBox
            listBox1.Items.Clear();
            for (int i = 0; i < xmlNodes.Count; i++)
            {
                xmlElement = xmlNodes[i];
                if (xmlElement.HasChildNodes)
                {
                    elementValue = xmlElement.FirstChild.Value.Trim();
                    listBox1.Items.Add(elementValue);
                }
            }
        }
        catch (XPathException ex)
        {
            const string errorMessage =
               "Ошибка в задании выражения XPath!" +
               "\r\n" + "Соответствующие данные в документе не найдены!" +
               "\r\n" + "Попробуйте задать другое выражение!";
            MessageBox.Show(errorMessage +"\r\n" + ex.Message);
        }
    }
}
```

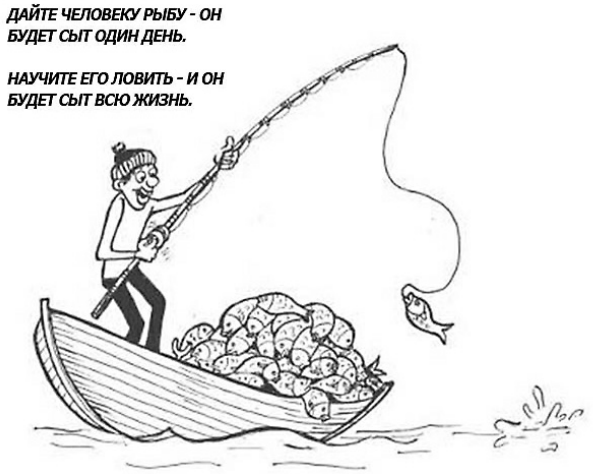
Программа отображает исходный XML-файл в элементе управления RichTextBox. Далее из списка элементов ComboBox можно выбрать одно из выражений xPath и нажать на командную кнопку. В результате в элемент управления ListBox будут выведены элементы Xml-документа, заданные выражением XPath. Вы можете задать собственное выражение XPath в элементе ComboBox. После запуска программы поэкспериментируйте с вводом и выбором выражения xPath, после чего нажмите кнопку «Получить данные».


Работа с XML документом
<?xml version="1.0" encoding="utf-8"?><Планета>Земля<Континент>Южная Америка<Страна столица="Рио-де-Жанейро">Бразилия</Страна><Страна столица="Буэнос-Айрес">Аргентина</Страна></Континент><Континент>Азия<Страна столица="Дели">Индия</Страна><Страна столица="Бангкок">Тайланд</Страна></Континент></Планета>
/Планета/Континент/Страна
Получить данные
Бразилия
Аргентина
Индия
Тайланд

Часть 5. За рамками этой книги

К сожалению, люди еще не изобрели способа загружать необходимую информацию прямо в голову. Но стоит ли об этом сожалеть? Ведь процесс обучения доставляет немало удовольствия. Вспомните свой восторг, когда заработала первая написанная вами программа. А ведь она была лишь ничтожной частью тех побед, которые еще педстоит одержать.

Автор этой книги вовсе не ставил перед собой задачу рассказать все о программировании на языке C#. Самое главное – сформировать некую базу из основных принципов и предоставить достаточное количество примеров с реальными задачами, опираясь на которые юные программисты приступят к самостоятельному освоению этого искусства.

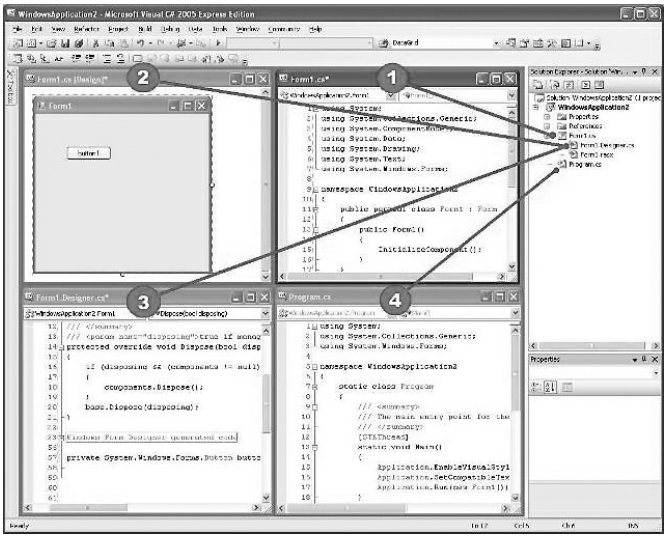


Проекты и среда разработки

Во всех примерах, приведенных в книге, построение проекта несколько отличается от способа построения реальных проектов. В наших примерах вначале строится проект типа «Windows Application», а затем выполняются не совсем обычные действия. Из проекта удаляется построенная автоматически форма, а текст построенного автоматически класса Program заменяется кодом прилагаемого примера. Благодаря этим манипуляциям весь код примера содержится в одном файле, что облегчает его восприятие. Второе существенное отличие состоит в том, что все элементы управления, появляющиеся на форме, создаются программным путем. В реальной практике заполнение формы элементами управления производится вручную путем перетаскивания элементов с инструментальной панели. Гораздо проще поместить элемент управления в нужное место формы и придать ему необходимые размеры, видя перед собой и форму, и элементы управления. Так работать гораздо эффективнее, чем программно задавать для элемента управления свойства Location, Height и Width. Оба способа работы эквивалентны по результатам, поскольку в итоге создается один и тот же код. В студии разработки визуальная работа программиста по проектированию интерфейса формы поддерживается тем, что там есть специальный инструментарий, который называется «Дизайнер»; он следит за выполняемыми «вручную» действиями, и создает код, эквивалентный этим действиям. В конечном итоге в проекте появляется файл, созданный «Дизайнером», и программист не должен его изменять .

Итак, в реальном проекте есть файлы, создаваемые автоматически и не подлежащие изменению, но есть и такие, куда программисты добавляют свой код. В наших проектах имеется только один файл, который мы полностью создаем сами.

На приведенном ниже снимке экрана показана студия разработки с открытым Windows-проектом. Для того чтобы более наглядно показать все создаваемые элементы, четыре окна, отображающие файлы проекта, помещены в один рисунок.



Давайте рассмотрим все четыре окна.

1. Представление кода с файлом **Form1.cs** — здесь содержится файл, *в который записывается код* формы. Сюда же добавляются обработчики событий элементов управления.
2. Представление конструктора с файлом **Form1.Design.cs** — визуальное представление формы, куда можно перетаскивать элементы управления (например, кнопки) с панели инструментов. Это избавляет от необходимости программировать элементы вручную.
3. Представление кода с файлом **Form1.Designer.cs** — когда в область конструктора перетаскивается, к примеру, кнопка, «Дизайнер» из Visual C# Express добавляет соответствующий фрагмент кода с описанием данного экземпляра кнопки. Код сохраняется в файле, а файл используется только самой системой и служит для представления в коде тех элементов, которые добавляются в область конструктора. Обычно код в этом файле программист не изменяет и ничего в него не добавляет.
4. Представление кода с файлом **Program.cs**. Этот файл содержит стандартный метод Main() и код для автоматического создания экземпляра класса Form. Метод Main — это точка «большого взрыва». Он вызывается системой извне, и именно с него начинается выполнение проекта. В случае приложений Windows код метода Main создается автоматически. Этот файл тоже не изменяется.

Подобное разделение кода на несколько файлов стало возможным благодаря существованию так называемых **разделяемых классов** — это означает, что части класса могут находиться в разных файлах. **Разделять класс на несколько файлов** не обязательно, однако такой подход имеет большое преимущество: код со специальным предназначением выделяется в отдельный файл. Благодаря применению подобной схемы каждому предоставляется свое рабочее пространство. На приведенной выше иллюстрации ваша территория — это два верхних окна: представление конструктора, где программист самостоятельно размещает элементы управления, и файл Forms1.cs, где он создает обработчики события.

Если удобства работы вас не соблазняют, вы можете в любой момент удалить файлы Program.cs и Form1.cs и начать работу с собственными файлами.

Дополнительные советы

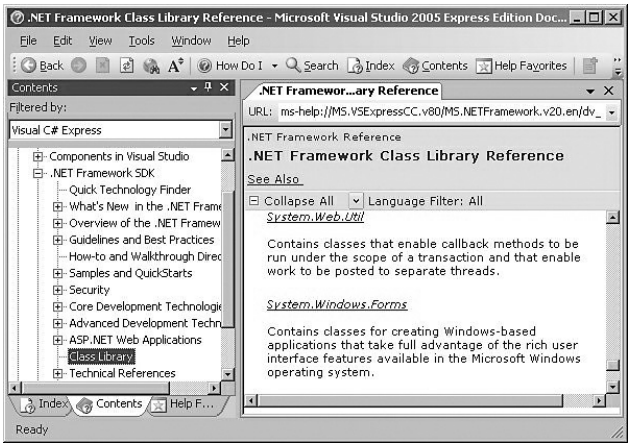
Как и при изучении любого нового предмета, во время чтения книги у вас, несомненно, будут возникать многочисленные вопросы. Попытаемся ответить на те, которые задаются наиболее часто:

1. Что делать, если Visual C# Express выдает сообщение об ошибке?
   Успокоиться и попытаться понять, что хочет сказать компьютер. Сделать это непросто, но попробуйте поставить себя на его место — обычно помогает.
   Почитайте разделы, касающиеся отладки программ в Visual C# Express. Крайне важно освоить пошаговое выполнение операторов программного кода с одновременной проверкой значений переменных. Ошибки, появляющиеся в ходе этого процесса, автоматически не устраняются, но вы получаете гораздо больше полезной информации, благодаря которой можно понять причину ошибки и исправить ее. Хороший детектив всегда вначале собирает улики. Поисковую работу придется выполнять вам.

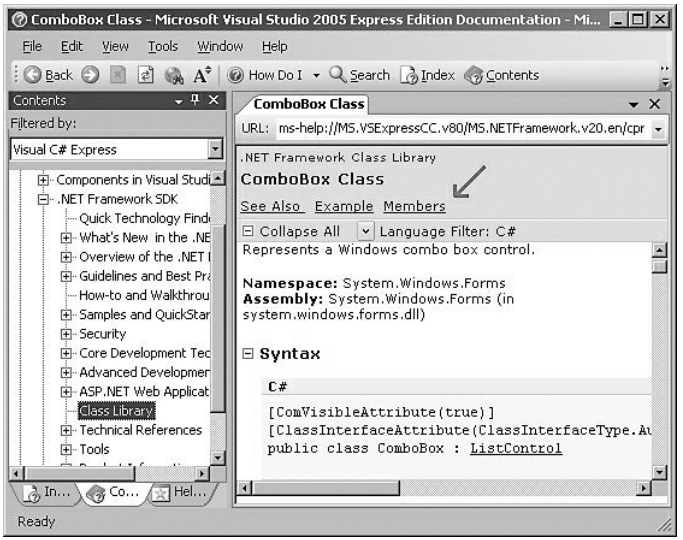
Если отладка не принесла желаемого результата, попробуйте найти подсказку в справке или в Интернете. Если повезет, можете наткнуться на идеи, которые позволят взглянуть на проблему с другой стороны.

2. Как узнать, какие классы доступны в библиотеке классов .NET Framework Class Library?
3. Какие методы есть у этих классов?
4. И какие параметры необходимы этим методам?

Visual C# Express содержит подробный справочный материал по всем классам в библиотеке .NET Framework Class Library. Чтобы просмотреть справку, нажмите «Справка -> Оглавление» и выберите тему «.NET Framework SDK». В приведенном ниже примере требуется найти сведения о том, какие классы доступны в пространстве имен System.Windows.Forms. Для этого мы выбираем Class library («Библиотека классов») и, нажимая на кнопку со стрелкой, прокручиваем страницу вниз до заголовка «System.Windows.Forms».



При переходе по гиперссылке *Systems.Windows.Forms* появляется список классов (в нашем примере это «Кнопка», «Метка», «Поле со списком» и т. п.). При выборе определенного класса (например, «Поле со списком») на экране появляется нужная картинка (см. рисунок ниже). Если теперь щелкнуть по гиперссылке «Элементы», откроется страница с перечислением всех методов, свойств и событий этого класса и подробной информацией о них.



Одним из главных достоинств этого справочного материала является огромное количество содержащихся в нем примеров. Иногда информация покажется вам слишком общей, но стоит нажать кнопку «Пример», и все сразу становится на свои места. Если вы не нашли подходящего примера в справочном материале, выполните поиск по всей справке — в большинстве случаев полезные примеры обнаружатся в других ее частях.

5. Где найти дополнительные сведения?

В какой-то момент вы поймете, что без других источников информации уже не обойтись. Это означает, что пришло время обратиться к своим друзьям-программистам и поинтересоваться, что именно они программируют в Visual Basic и как решают возникающие проблемы.

Рекомендуем посетить следующие веб-узлы:
http://www.microsoft.com/rus/express/vcsharp.
http://msdn.microsoft.com/rucoding4fun
http://msdn.microsoft.com/ru-ru/vcsharp/aa336717.aspx или
http://forums.microsoft.com/MSDN/ShowForum.aspx?ForumID=160&SiteID=1
www.gotdotnet.ru
http://www.codeproject.com/?cat=3

Не стесняйтесь задавать вопросы в форумах. Однако сначала ознакомьтесь с уже существующими сообщениями и проверьте, не задавал ли кто-нибудь до вас точно такой же вопрос.

Приятного путешествия в мир программирования!

Оглавление

Учебное издание

Дрейер Мартин

C# ДЛЯ ШКОЛЬНИКОВ

Учебное пособие

Литературный редактор *С. Перепелкина*
Корректор *Ю. Голомазова*
Компьютерная верстка *Н. Овчинникова*
Дизайн обложки *М. Автономова, М. Купцова*

Подписано в печать 25.08.2009. Бумага офсетная. Формат 70x90 $^{1}/_{16}$.
Гарнитура Таймс. Усл. печ. л. 8. Печать офсетная.
Тираж 2000 экз. Заказ №

ООО «ИНТУИТ.ру»
Интернет-Университет Информационных Технологий, www.intuit.ru
Москва, Электрический пер., 8, стр.3.
E-mail: admin@intuit.ru, http://www.intuit.ru

ООО «БИНОМ. Лаборатория знаний»
Москва, проезд Аэропорта, д. 3
Телефон: (499) 157-1902, (499) 157-5272
E-mail: Binom@lbz.ru, http://www.Lbz.ru