

Михаил Фленов

# Библия

# C#

**2-е издание**

Программирование  
для .NET на C#

Базы данных

Графика и мультимедиа

Сетевое программирование

Повторное использование кода

Изучение языка  
на полезных примерах

Большое количество  
дополнительной  
информации на CD

**+CD**

**Михаил Фленов**

**Библия**

**С #**

**2-е издание**

Санкт-Петербург

«БХВ-Петербург»

2011

УДК 681.3.068+800.92С#

ББК 32.973.26-018.1

Ф69

**Фленов М. Е.**

Ф69 Библия С#. — 2-е изд., перераб. и доп. — СПб.: БХВ-Петербург, 2011. — 560 с.: ил. + CD-ROM

ISBN 978-5-9775-0655-7

Книга посвящена программированию на языке С# для платформы Microsoft .NET, начиная с основ языка и разработки программ для работы в режиме командной строки и заканчивая созданием современных приложений различной сложности (баз данных, графических программ и др.). Материал сопровождается большим количеством практических примеров. Подробно описывается логика выполнения каждого участка программы. Уделено внимание вопросам повторного использования кода. Во второе издание внесены исправления и добавлена глава по сетевому программированию. Компакт-диск содержит примеры программ, дополнительную справочную информацию, а также готовые компоненты, тестовые программы и изображения.

*Для программистов*

УДК 681.3.068+800.92С#

ББК 32.973.26-018.1

#### **Группа подготовки издания:**

Главный редактор	<i>Екатерина Кондукова</i>
Зам. главного редактора	<i>Игорь Шишигин</i>
Зав. редакцией	<i>Григорий Добин</i>
Редактор	<i>Нина Седых</i>
Компьютерная верстка	<i>Натальи Смирновой</i>
Корректор	<i>Наталья Першакова</i>
Дизайн серии	<i>Инны Тачиной</i>
Оформление обложки	<i>Елены Беляевой</i>
Зав. производством	<i>Николай Тверских</i>

Лицензия ИД № 02429 от 24.07.00. Подписано в печать 30.11.10.

Формат 70×100<sup>1/16</sup>. Печать офсетная. Усл. печ. л. 45,15.

Тираж 2000 экз. Заказ №

"БХВ-Петербург", 190005, Санкт-Петербург, Измайловский пр., 29.

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

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

# Оглавление

<b>ВВЕДЕНИЕ</b> .....	<b>1</b>
<b>БЛАГОДАРНОСТИ</b> .....	<b>5</b>
<b>БОНУС</b> .....	<b>7</b>
<b>СТРУКТУРА КНИГИ</b> .....	<b>9</b>
<b>ГЛАВА 1. ВВЕДЕНИЕ В .NET</b> .....	<b>11</b>
1.1. Платформа .NET .....	11
1.1.1. Кубики .NET.....	12
1.1.2. Сборки .....	14
1.1.3. Язык программирования.....	15
1.2. Обзор среды разработки Visual Studio .NET.....	16
1.2.1. Работа с проектами и решениями .....	17
1.2.2. <i>Server Explorer</i> .....	20
1.2.3. <i>Toolbox</i> .....	22
1.2.4. <i>Solution Explorer</i> .....	24
1.2.5. <i>Class View</i> .....	27
1.2.6. <i>Properties</i> .....	28
1.2.7. Работа с файлами .....	28
1.3. Простейший пример .NET-приложения.....	29
1.3.1. Проект на языке C#.....	30
1.3.2. Компиляция и запуск проекта на языке C#.....	30
1.4. Компиляция приложений.....	33
1.5. Поставка сборок.....	34
1.6. Формат исполняемого файла .NET .....	38
<b>ГЛАВА 2. ОСНОВЫ C#</b> .....	<b>41</b>
2.1. Комментарии.....	41
2.2. Переменная.....	42
2.3. Именованное.....	45

2.4. Работа с переменными .....	49
2.4.1. Строки и символы.....	52
2.4.2. Массивы.....	54
2.4.3. Перечисления .....	57
2.5. Простейшая математика .....	60
2.6. Логические операции .....	64
2.6.1. Условный оператор <i>if</i> .....	65
2.6.2. Условный оператор <i>switch</i> .....	67
2.6.3. Сокращенная проверка.....	68
2.7. Циклы .....	68
2.7.1. Цикл <i>for</i> .....	69
2.7.2. Цикл <i>while</i> .....	71
2.7.3. Цикл <i>do..while</i> .....	72
2.7.4. Цикл <i>foreach</i> .....	73
2.8. Управление циклом .....	75
2.8.1. Оператор <i>break</i> .....	75
2.8.2. Оператор <i>continue</i> .....	75
2.9. Константы.....	77
2.10. Нулевые значения.....	77

## **ГЛАВА 3. ОБЪЕКТНО-ОРИЕНТИРОВАННОЕ ПРОГРАММИРОВАНИЕ .....79**

3.1. Объекты на <i>C#</i> .....	79
3.2. Свойства .....	83
3.3. Методы .....	88
3.3.1. Описание методов.....	89
3.3.2. Параметры методов .....	91
3.3.3. Перегрузка методов .....	97
3.3.4. Конструктор .....	98
3.3.5. Статичность.....	101
3.3.6. Рекурсивный вызов методов.....	104
3.3.7. Деструктор.....	106
3.4. Метод <i>Main()</i> .....	108
3.5. Пространства имен .....	110
3.6. Начальные значения переменных .....	112
3.7. Объекты только для чтения .....	113
3.8. Объектно-ориентированное программирование .....	113
3.8.1. Наследование .....	114
3.8.2. Инкапсуляция.....	115
3.8.3. Полиморфизм.....	116
3.9. Наследование от класса <i>Object</i> .....	117
3.10. Переопределение методов .....	119
3.11. Обращение к предку из класса .....	121
3.12. Вложенные классы .....	122

3.13. Область видимости.....	124
3.14. Ссылочные и простые типы данных.....	126
3.15. Абстрактные классы.....	127
3.16. Проверка класса объекта .....	130
<b>Глава 4. КОНСОЛЬНЫЕ ПРИЛОЖЕНИЯ.....</b>	<b>131</b>
4.1. Украшение консоли.....	131
4.2. Работа с буфером консоли.....	134
4.3. Окно консоли .....	135
4.4. Запись в консоль .....	136
4.5. Чтение данных из консоли.....	138
<b>Глава 5. ВИЗУАЛЬНЫЙ ИНТЕРФЕЙС .....</b>	<b>141</b>
5.1. Приложение.....	141
5.1.1. Пространства имен .....	143
5.1.2. Потoki.....	145
5.1.3. Класс <i>Application</i> .....	146
5.2. Наследник <i>Form</i> для главной формы .....	147
5.2.1. Ресурсы программы.....	147
5.2.2. Файл для логики модуля .....	148
5.2.3. Именованiе формы .....	149
5.2.4. Код, сгенерированный дизайнером .....	150
5.2.5. Hello Visual World.....	153
5.3. Свойства формы .....	154
5.4. Методы формы.....	159
5.5. События на примере формы .....	160
5.6. Компоненты .NET.....	163
5.7. Общие компоненты .....	166
5.7.1. <i>Button</i> .....	167
5.7.2. <i>CheckBox</i> .....	170
5.7.3. <i>CheckedListBox</i> .....	170
5.7.4. <i>ComboBox</i> .....	173
5.7.5. <i>DateTimePicker</i> .....	175
5.7.6. <i>Label</i> и <i>LinkLabel</i> .....	176
5.7.7. <i>ListBox</i> .....	177
5.7.8. <i>ListView</i> .....	178
5.7.9. <i>PictureBox</i> .....	183
5.7.10. <i>ProgressBar</i> .....	184
5.7.11. <i>RadioButton</i> .....	185
5.7.12. <i>TextBox</i> .....	186
5.7.13. <i>TreeView</i> .....	186
5.8. Контейнеры .....	191
5.8.1. <i>GroupBox</i> .....	191

5.8.2. <i>Panel</i> .....	192
5.8.3. <i>TabControl</i> .....	192
5.8.4. <i>FlowLayoutPanel</i> и <i>TableLayoutPanel</i> .....	193
5.9. Меню и панели инструментов.....	195
5.9.1. <i>MenuStrip</i> .....	195
5.9.2. <i>ContextMenuStrip</i> .....	197
5.9.3. <i>ToolStrip</i> .....	199
5.9.4. <i>StatusStrip</i> .....	200
<b>ГЛАВА 6. ПРОДВИНУТОЕ ПРОГРАММИРОВАНИЕ .....</b>	<b>201</b>
6.1. Приведение и преобразование типов .....	201
6.2. Все в .NET это объекты .....	203
6.3. Работа с перечислениями <i>Enum</i> .....	204
6.4. Структуры.....	207
6.5. Дата и время.....	209
6.6. Класс строк.....	211
6.7. Перегрузка операторов.....	213
6.7.1. Математические операторы.....	214
6.7.2. Операторы сравнения.....	216
6.7.3. Операторы преобразования .....	217
6.8. Тип <i>var</i> .....	219
6.9. Шаблоны .....	220
<b>ГЛАВА 7. ИНТЕРФЕЙСЫ .....</b>	<b>225</b>
7.1. Объявление интерфейсов.....	225
7.2. Реализация интерфейсов.....	227
7.3. Использование реализации интерфейса .....	229
7.4. Интерфейсы в качестве параметров .....	232
7.5. Перегрузка интерфейсных методов .....	232
7.6. Наследование .....	235
7.7. Клонирование объектов.....	235
<b>ГЛАВА 8. МАССИВЫ.....</b>	<b>239</b>
8.1. Базовый класс для массивов.....	239
8.2. Невыровненные массивы.....	241
8.3. Динамические массивы.....	243
8.4. Индексаторы массива.....	245
8.5. Интерфейсы массивов .....	246
8.5.1. Интерфейс <i>IEnumerable</i> .....	247
8.5.2. Интерфейсы <i>IComparer</i> и <i>IComparable</i> .....	249
8.6. Оператор <i>yield</i> .....	253

8.7. Стандартные списки.....	253
8.7.1. Класс <i>Queue</i> .....	254
8.7.2. Класс <i>Stack</i> .....	255
8.7.3. Класс <i>Hashtable</i> .....	255
8.8. Типизированные массивы.....	257
<b>ГЛАВА 9. ОБРАБОТКА ИСКЛЮЧИТЕЛЬНЫХ СИТУАЦИЙ.....</b>	<b>261</b>
9.1. Исключительные ситуации.....	261
9.2. Исключения в C#.....	263
9.3. Оформление блоков <i>try</i> .....	267
9.4. Ошибки в визуальных приложениях.....	268
9.5. Генерирование исключительных ситуаций.....	270
9.6. Иерархия классов исключений.....	271
9.7. Собственный класс исключения.....	272
9.8. Блок <i>finally</i> .....	275
9.9. Переполнение.....	276
<b>ГЛАВА 10. СОБЫТИЯ В C#.....</b>	<b>281</b>
10.1. Делегаты.....	281
10.2. События и их вызов.....	282
10.3. Использование собственных делегатов.....	285
10.4. Делегаты изнутри.....	290
10.5. Анонимные методы.....	291
10.6. Динамическое создание компонентов.....	292
<b>ГЛАВА 11. ФОРМЫ.....</b>	<b>295</b>
11.1. Диалоговые окна.....	296
11.2. Редактирование объектов.....	300
11.3. Специфичный результат.....	304
11.4. Немодальные окна.....	307
11.5. Многодокументный интерфейс.....	307
<b>ГЛАВА 12. НЕБЕЗОПАСНОЕ ПРОГРАММИРОВАНИЕ.....</b>	<b>311</b>
12.1. Разрешение небезопасного кода.....	312
12.2. Указатели.....	313
12.3. Память.....	316
12.4. Системные функции.....	318
<b>ГЛАВА 13. ГРАФИКА.....</b>	<b>321</b>
13.1. Введение в <i>Graphics</i> .....	321
13.2. Рисование по событию <i>Paint</i> .....	323



13.3. Рисование без события <i>Paint</i> .....	325
13.4. Цвета .....	326
13.5. Перья .....	328
13.6. Кисти .....	330
13.7. Работа с картинками .....	331
13.8. Графический дизайнер .....	336
13.9. Рисование элементов списка <i>ListBox</i> .....	341
<b>ГЛАВА 14. ХРАНЕНИЕ ИНФОРМАЦИИ .....</b>	<b>345</b>
14.1. Реестр .....	345
14.2. Файловая система .....	350
14.3. Текстовые файлы .....	354
14.4. Бинарные файлы .....	357
14.5. XML-файлы .....	361
14.5.1. Создание XML-документов .....	362
14.5.2. Чтение XML-документов .....	365
14.6. Потoki <i>Stream</i> .....	369
14.7. Потoki <i>MemoryStream</i> .....	370
14.8. Сериализация .....	372
14.8.1. Отключение сериализации .....	375
14.8.2. Особенности сериализации .....	376
14.8.3. Управление сериализацией .....	378
<b>ГЛАВА 15. МНОГОПОТОЧНОСТЬ .....</b>	<b>383</b>
15.1. Класс <i>Thread</i> .....	384
15.2. Передача параметра в поток .....	387
15.3. Потoki с использованием делегатов .....	388
15.4. Конкурентный доступ .....	391
15.5. Доступ к компонентам .....	394
15.6. Пул потоков .....	396
15.7. Домены приложений .NET .....	398
<b>ГЛАВА 16. БАЗЫ ДАННЫХ .....</b>	<b>401</b>
16.1. ADO.NET .....	401
16.2. Строка подключения .....	404
16.3. Подключение к базе данных .....	408
16.4. Пул соединений .....	410
16.5. Выполнение команд .....	412
16.6. Транзакции .....	413
16.7. Наборы данных .....	415
16.8. Чтение результата запроса .....	420
16.9. Работа с процедурами .....	422

16.10. Методы <i>OleDbCommand</i> .....	427
16.11. Отсоединенные данные.....	429
16.12. Адаптер <i>DataAdapter</i> .....	433
16.12.1. Конструктор .....	434
16.12.2. Получение результата запроса.....	434
16.12.3. Сохранение изменений в базе данных.....	435
16.12.4. Связанные таблицы .....	437
16.12.5. Добавление данных .....	438
16.12.6. Удаление данных .....	440
16.13. Набор данных <i>DataSet</i> .....	441
16.13.1. Хранение данных в <i>DataSet</i> .....	441
16.13.2. Класс <i>DataRow</i> .....	444
16.13.3. Класс <i>DataColumn</i> .....	446
16.13.4. Класс <i>DataTable</i> .....	447
16.14. Таблицы в памяти.....	448
16.15. Выражения .....	450
16.16. Ограничения.....	452
16.17. Манипулирование данными .....	453
16.17.1. Добавление строк.....	454
16.17.2. Редактирование данных .....	455
16.17.3. Поиск данных.....	457
16.17.4. Удаление строк .....	457
16.18. Связанные данные .....	458
16.19. Ограничение внешнего ключа.....	462
16.20. Фильтрация данных.....	469
16.21. Представление данных <i>DataView</i> .....	471
16.22. Схема данных.....	475
<b>ГЛАВА 17. ПОВТОРНОЕ ИСПОЛЬЗОВАНИЕ КОДА .....</b>	<b>477</b>
17.1. Библиотеки.....	477
17.2. Создание библиотеки .....	478
17.3. Приватные сборки .....	482
17.4. Общие сборки .....	484
17.5. Создание пользовательских компонентов .....	486
17.6. Установка компонентов .....	493
<b>ГЛАВА 18. УДАЛЕННОЕ ВЗАИМОДЕЙСТВИЕ .....</b>	<b>495</b>
18.1. Удаленное взаимодействие в .NET.....	495
18.2. Структура распределенного приложения.....	497
18.3. Общая сборка .....	498
18.4. Сервер .....	499
18.5. Клиент.....	502

---

<b>ГЛАВА 19. СЕТЕВОЕ ПРОГРАММИРОВАНИЕ .....</b>	<b>505</b>
19.1. HTTP-клиент .....	505
19.2. Прокси-сервер .....	508
19.3. Класс <i>Uri</i> .....	509
19.4. Сокеты .....	511
19.5. Парсинг документа .....	522
19.6. Клиент-сервер .....	526
<b>ЗАКЛЮЧЕНИЕ.....</b>	<b>535</b>
<b>ОПИСАНИЕ КОМПАКТ-ДИСКА .....</b>	<b>537</b>
<b>СПИСОК ЛИТЕРАТУРЫ .....</b>	<b>539</b>
<b>ПРЕДМЕТНЫЙ УКАЗАТЕЛЬ .....</b>	<b>541</b>

# Введение

.NET — это платформа от компании Microsoft, которая состоит из полного набора инструментов для разработчиков (.NET Framework) и для пользователей. Сюда входят клиентская и серверная операционные системы (ОС), инструменты разработки, сервисы. В данной книге мы будем рассматривать .NET Framework, который нужен программистам для написания программ для платформы .NET, а также язык программирования C#, на котором мы и будем писать для .NET Framework. .NET Framework состоит из множества библиотек и классов, которые можно использовать для создания собственных приложений.

Меня очень часто спрашивают, как я отношусь к .NET. К первой версии .NET Framework я относился не очень хорошо, потому что не понял ее и не увидел преимуществ. Когда появилась вторая версия, я кардинально изменил свое мнение по поводу C#. Хотя нет, если быть точным, то мое отношение к .NET первой версии так и осталось скорее негативным, чем положительным или нейтральным. А вот ко второй версии и ко всем последующим я отношусь не то, чтобы положительно, а даже с восхищением.

Большинство языков программирования с богатой историей обладают одним большим недостатком. За время существования накапливается много устаревшего и небезопасного, но все эти недостатки накапливаются для сохранения совместимости с уже написанным кодом. Разработка абсолютно нового языка позволила компании Microsoft избавиться от всего старого и создать что-то новое, чистое и светлое. Наверно, это слишком громкие слова, но просто захотелось сказать эту глупость.

Чтобы понять, почему мне нравятся .NET и C#, давайте рассмотрим реальные преимущества, которые я бы выдел. Они уже есть в .NET и никуда не денутся. К основным преимуществам платформы .NET я бы отнес:

1. Универсальный API. На каком бы языке вы не программировали, вам предоставляются одни и те же имена объектов. Все языки для платформы .NET отличаются только синтаксисом, а классы используются из .NET Framework. Таким образом, все языки схожи, и вы выбираете только тот, который вам ближе именно по синтаксису. Я начинал изучать программирование с Basic, затем Pascal, C, C++, Assembler, Delphi, Java и сейчас .NET. При переходе с языка на язык приходится очень много времени тратить на изучение нового API. На платформе .NET больше не будет такой проблемы.

И тут преимущество не только в том, что все языки одинаковы, а в том, что улучшается возможность взаимодействия программ, написанных на разных языках. Раньше для того, чтобы программа на C++ без проблем взаимодействовала с кодом на Visual Basic или Delphi, приходилось применять различные трюки и уловки. В основном это было связано тем, что каждый язык по-своему обрабатывал и хранил строки. Сейчас такой проблемы нет, и все типы данных в C# абсолютно совместимы с Visual Basic .NET или другим языком платформы .NET.

Таким образом, программисты, использующие различные языки, могут работать над одним и тем же проектом и без швов сращивать модули на разных языках.

2. Защищенный код. Платформу Win32 очень часто ругали за ее незащищенность. В ней действительно есть очень слабое звено — незащищенность кода и возможность перезаписывать любые участки памяти. Самым страшным в Win32 являлась работа с массивами, памятью и со строками (последние являются разновидностью массива). С одной стороны, это предоставляет мощные возможности системному программисту, который умеет правильно распоряжаться памятью. С другой стороны, в руках неопытного программиста данная возможность превращается в уязвимость. Сколько раз нам приходилось слышать об ошибках переполнения буфера из-за неправильного выделения памяти? Уже и сосчитать сложно. На платформе .NET вероятность такой ошибки стремится к нулю, если вы используете управляемый код, и если Microsoft не допустит ошибок при реализации самой платформы.
3. Платформа .NET не является абсолютно безопасной, потому что существуют не только ошибки переполнения буфера, есть еще и ошибки логики работы программы.
4. Полная ориентированность на объекты. Объектно-ориентированное программирование (ООП) — это не просто дань моде, это мощь, удобство и скорость разработки. Платформа Win32 все еще основана на процедурах и наследует все недостатки этого подхода. Любые объектные надстройки, такие как Object Windows Library (OWL), Microsoft Foundation Classes (MFC) и др., решают далеко не все задачи.
5. Сборка мусора. Начинающие программисты очень часто путаются, когда нужно уничтожать объекты, а когда это делать не обязательно. Приходится долго объяснять, что такое локальные и глобальные переменные, распределение памяти, стек и т. д. Даже опытные программисты нередко путаются и допускают ошибки при освобождении памяти. А ведь если память освободить раньше, чем это можно сделать, то обращение к несуществующим объектам приведет к краху программы.
6. На платформе .NET за уничтожение объектов отвечает сама платформа, хотя вы и можете повлиять на этот процесс, но только косвенно. Таким образом, у вас не будет утечек памяти, а вместо того, чтобы думать об освобождении ре-

- сурсов, вы можете заниматься более интересными вещами. А это приводит и к повышению производительности труда.
7. Визуальное программирование. Благодаря объектно-ориентированному подходу стало проще создавать визуальные языки программирования. Если вы программировали на Visual C++, то наверно уже знаете, что этот язык далек от идеала, а визуальные возможности сильно ограничены по сравнению с такими языками, как Delphi и Visual Basic. Новый язык C# действительно визуален и по своим возможностям практически не уступает самой мощной (по крайней мере, до появления .NET) визуальной среде разработки Delphi. Визуальность упрощает создание графического интерфейса и ускоряет разработку, а значит, ваша программа сможет раньше появиться на рынке и захватить его. Как показывает практика, очень часто первый игрок снимает основные сливки.
  8. Компонентное представление. Так как платформа имеет полностью объектную основу, появилась возможность компонентно-ориентированного программирования, как это сделано в Delphi. В платформе Win32 были попытки создания компонентной модели с помощью ActiveX, но развертывание подобных приложений являлось слишком сложной задачей. Большинство разработчиков старались не связываться с этой технологией (в том числе и я), а если жизнь заставляла, то для развертывания приходилось создавать инсталляционные пакеты. Самостоятельно создавать пакет — сложно и неинтересно, поэтому среди разработчиков быстро получил распространение пакет InstallShield, позволяющий упростить задачу создания инсталляционных пакетов и упростить развертывание ActiveX. На платформе .NET установка новых пакетов сводится к простому копированию без необходимости регистрации в реестре.
  9. Распределенные вычисления. Платформа .NET ускоряет разработку приложений с распределенными вычислениями, что достаточно важно для корпоративного программного обеспечения. В качестве транспорта при взаимодействии используются технологии HTTP (Hypertext Transfer Protocol, протокол передачи гипертекстовых файлов), XML (Extensible Markup Language, расширяемый язык разметки) и SOAP (Simple Object Access Protocol, простой протокол доступа к объектам).
  10. Открытость стандартов. При рассмотрении предыдущего преимущества мы затронули открытые стандарты HTTP, XML и SOAP. Открытость — это неоспоримое преимущество, потому что предоставляет разработчику большую свободу.
  11. Поддержка устройств. На мой взгляд, это основная причина появления платформы. Дело в том, что язык программирования Java оказался наиболее пригодным для различных устройств, и на данный момент этот язык поддерживает большинство мобильных телефонов и карманных устройств. Платформа .NET способна отобразить этот лакомый кусок, и мне кажется, что именно для этого она делалась независимой от аппаратной части. .NET называют межплатформенной, потому что она создавалась с возможностью работы на разных платформах, но сама компания Microsoft не спешит поддерживать конкурентные

платформы, такие как Linux или Mac. Остается только наблюдать, что выберут производители устройств и программного обеспечения.

Список можно продолжать и дальше, но уже видно, что будущее у платформы есть. Какое это будет будущее — вот это большой и сложный вопрос. Но, глядя на деньги, которые были вложены в разработку и рекламную кампанию, можно говорить, что Microsoft не упустит своего и сделает все возможное для обеспечения долгой и счастливой жизни .NET Framework.

Данная книга призвана показать вам преимущества новой платформы и научить вас создавать собственные законченные приложения. Мы будем не просто рассматривать процесс расстановки компонентов на форме, а окунемся в глубину платформы и увидим жизнь ваших программ с разных сторон. Это позволит вам создавать более эффективные приложения.

Напоследок хочу выразить собственное мнение о платформе, которое не опирается на какие-либо факты. Как мы выбираем в нашей жизни какие-либо вещи? В основном по личному опыту. Да, качество товара тоже играет свою роль, но если вам не нравится, как выглядит майка, вы никогда ее не наденете.

Я знаком с .NET с момента появления первой версии и пытался изучать ее, несмотря на то, что синтаксис и возможности не вызывали восторга. Да, язык C# выбрал в себя все лучшее от C++ и Delphi, но все равно удовольствия было мало. Тем не менее, я продолжал его изучать в свободное время, т. к. люблю находиться на гребне волны, а не стоять в очереди за догоняющими.

Возможно, мое первоначальное отрицательно отношение было связано со скудными возможностями самой среды разработки Visual Studio, которая является стандартом, потому что после выхода финальной версии Visual Studio .NET и версии .NET Framework 2.0 мое отношение к языку начало меняться в противоположную сторону. Язык и среда разработки стали намного удобнее, а код читабельным, что очень важно при разработке больших проектов.

На данный момент мое личное отношение к языку и среде разработки Visual Studio .NET более чем положительное. Для меня эта платформа стала номером один в моих личных проектах (на работе я использую то, что требует работодатель). Я удивляюсь, как компания смогла разработать за такой короткий срок целую платформу, которая позволяет быстро решать широкий круг задач.

# Благодарности

Я хочу в очередной раз поблагодарить свою семью, которая отпустила меня на три месяца ради работы над этой книгой. На Новый год и по завершении работы над книгой я постараюсь отплатить им за терпение.

Я благодарю всех тех, кто помогал мне в работе над книгой в издательстве — редакторов и корректоров, дизайнеров и верстальщиков — всех, кто старался сделать эту книгу интереснее для читателя.

Спасибо всем моим друзьям, кто меня поддерживал, и всем тем, кто уговорил меня все же написать эту книгу, а это в большинстве своем посетители моего сетевого журнала [www.flenov.info](http://www.flenov.info) и сайта для программистов [www.hackishcode.com](http://www.hackishcode.com).



# Бонус

Я долго не хотел браться за этот проект, потому что люди в наше время не покупают книги, а качают их из Интернета, да и тратить свободное время, отрываясь от семьи, тяжело. Очень жаль, что люди не покупают книг. Издательство "БХВ-Петербург" устанавливает не такие высокие цены, и можно было бы отблагодарить всех тех людей, которые старались для читателя, небольшой суммой. Если вы скачали книгу, и она вам понравилась, то я прошу вас купить полноценную версию.

Я же со своей стороны постарался сделать книгу максимально интересной, а на прилагаемый компакт-диск выложил дополнительную информацию в виде статей и исходных кодов для дополнительного улучшения и совершенствования своих навыков. Обязательно убедитесь при покупке книги, что она содержит компакт-диск, потому что на нем очень много полезного и интересного. Это как бы дополнительный бонус для тех, кто покупает полноценную книгу.

# Структура книги

Раньше в своих предыдущих книгах я старался как можно быстрее начать показывать визуальное программирование в ущерб начальным теоретическим знаниям. В этой книге я решил потратить немного времени на теоретические знания, чтобы потом рассмотрение визуального программирования шло как по маслу. Чтобы теория вводной части не была слишком скучной, я старался писать как можно интереснее и по возможности придумывал интересные примеры.

Первые четыре главы мы будем писать программы, но они будут без графического интерфейса. Информация будет выводиться в консоль. В мире, где властвует графический интерфейс, окна, меню, кнопки и панели, консольная программа может выглядеть немного дико. Но командная строка еще жива и, наоборот, набирает популярность, и ярким примером тут является PowerShell (это новая командная строка, которая поставляется в Windows Server 2008 и может быть установлена в Windows Vista или Windows 7).

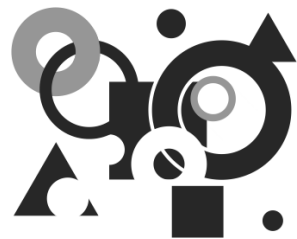
Визуальное программирование мы начнем изучать, как только познакомимся с основами .NET и объектно-ориентированным программированием. Не всегда примеры будут интересными, но я постарался придумать что-то познавательное, полезное и максимально приближенное к реальным задачам, которые вам придется решать в будущем, если вы свяжете свою работу с программированием или просто будете создавать что-то для себя.

Рассмотрев работу с компонентами, мы перейдем к графике и к наиболее важному вопросу для тех, кто работает профессионально в компаниях, — к базам данных. Да, профессиональные программисты чаще всего работают именно с базами данных, потому что в компаниях компьютеры нужны в основном для управления данными и информацией, которая должна быть структурированной и легкодоступной.

Я не пытался переводить файл справки, потому что Microsoft уже сделала это для нас. Моя задача — научить вас понимать программирование и уметь строить код. А за подробным описанием классов, методов и т. д. всегда можно обратиться к MSDN (Microsoft Development Network, это обширная библиотека технической информации, которая располагается на сайте [www.msdn.com](http://www.msdn.com)), который теперь доступен и на русском языке. На сайте есть перевод всех классов, методов и свойств с подробным описанием и простыми примерами.

Мир программ и программирования очень интересен и познавателен, особенно для тех, кто любит изучать что-то новое. Никакая книга не способна охватить сразу абсолютно все области программирования даже на каком-то одном языке, поэтому вам может быть придется покупать что-то для дальнейшего изучения. Но я надеюсь, что материала данной книги и дополнительной информации с компакт-диска вам хватит как минимум на год или даже на написание собственного большого проекта. А после этого, быть может, появится следующее издание Библии С#.

Пора переходить к более интересным вещам и начать погружение в мир .NET.



## ГЛАВА 1

# Введение в .NET

Платформа Microsoft .NET Framework состоит из набора базовых классов и CLR (Common Language Runtime, общезыковая среда выполнения). Базовые классы, которые входят в состав .NET Framework, поражают своей мощностью, универсальностью, удобством использования и разнообразием. В данной главе мы познакомимся с платформой .NET более подробно. Если вы сможете понять эти основы, то проще будет изучать последующий материал и понимать, почему и зачем разработчики что-то сделали именно так, а не иначе.

После вводной информации в саму платформу мы не будем тянуть время, а сразу начнем изучать язык программирования C# и знакомиться с объектно-ориентированным программированием. Я люблю делать упор на практику и считаю ее наиболее важной в нашей жизни. А по ходу практических занятий будем ближе знакомиться с теорией.

Для чтения этой и последующих глав рекомендуется иметь установленную среду разработки. Для написания большинства примеров этой книги использовалась Visual Studio 2008, некоторые примеры написаны уже в 2010. Но вне зависимости от того, в какой среде разработки писал их я, они могут быть скомпилированы в обеих средах. Желательно повторять все, что описывается в книге, чтобы ощутить это собственными руками. Для большинства примеров достаточно даже бесплатной версии среды разработки Visual C# Express Edition, которую можно скачать с сайта компании Microsoft ([www.microsoft.com/express/](http://www.microsoft.com/express/)).

Мы рассмотрим только самые основы .NET Framework, чтобы те, кто не читал об этой платформе, смогли получить необходимые для начала знания. Мы изучаем программирование, поэтому на него и сделаем упор.

## 1.1. Платформа .NET

Во введении мы уже затронули основные преимущества .NET, а сейчас рассмотрим платформу более подробно. На рис. 1.1 показана схема изменения платформы за последние годы. В основе всего, конечно, же стоит операционная система (ОС). Самое интересное, что Microsoft упоминает только Windows-системы (Vista, 2000, XP, 7 и т. д.). В этом списке нет других платформ, хотя существует

реализация .NET для Linux, которая называется Mono. Просто эта реализация выпускается независимым от Microsoft разработчиком, хотя и при существенной поддержке.

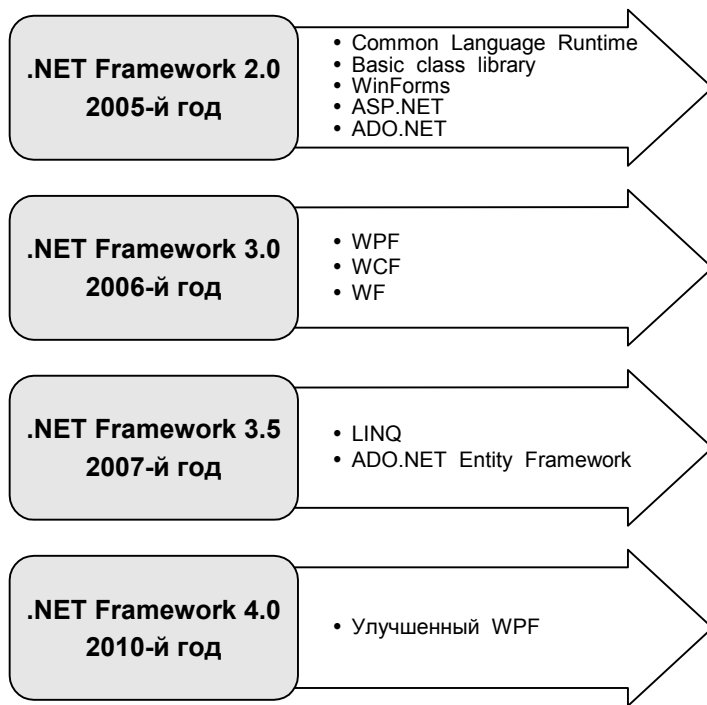


Рис. 1.1. Основа платформы .NET

Платформа .NET состоит из множества различных технологий, серверов и постоянно расширяется. Стоит только посмотреть на составляющие .NET Framework, которые постоянно появляются, и можно понять, что платформа не стоит на месте и будет расширяться в будущем, надеюсь, в таком же темпе. Я специально указал на схеме годы выпуска, чтобы была видна скорость развития.

С выходом .NET Framework 4 и среды разработки Visual Studio 2010 компания Microsoft дала четко понять, в каком направлении будет развиваться пользовательская платформа в дальнейшем. Это будет WPF (Windows Presentation Foundation, новейшая библиотека построения визуальных интерфейсов на основе языка XML). Визуальный интерфейс среды разработки переписан с использованием этой технологии и показывает программистам ее возможности и мощь.

### 1.1.1. Кубики .NET

Если отбросить всю рекламу, которую нам предлагают, и взглянуть на проблему глазами разработчика, то .NET описывается следующим образом: с помощью

среды разработки Visual Studio .NET вы можете разрабатывать приложения любой сложности с использованием .NET Framework, которые очень просто интегрируются с серверами и сервисами от Microsoft.

Основа всего, центральное звено платформы — это .NET Framework. Давайте посмотрим на основные составляющие платформы:

1. Операционная система. Понятное дело, что все должно работать поверх ОС, но теоретически может работать на специализированном железе, которое умеет выполнять код .NET. На данный момент .NET Framework — это как бы виртуальная машина, которая является промежуточным звеном между ОС и приложением. Но мне кажется, что недалек тот день, когда все перевернется, и .NET станет основой, а Win32-приложения начнут выполняться в виртуальной машине, как сейчас работают MS-DOS-приложения. Но это только плод моего большого воображения, который не связан с реальными планами Microsoft.
2. CLR (Common Language Runtime). Поверх ОС работает среда выполнения CLR. Это и есть виртуальная машина, которая обрабатывает IL-код (Intermediate Language, промежуточный язык) программы. Код IL — это аналог бинарного кода для платформы Win32 или байт-кода для виртуальной машины Java. Во время запуска приложения IL-код налету компилируется в машинный код под то железо, на котором запущена программа. Да, сейчас все работает на процессорах Intel, но все может быть, никто не запрещает реализовать платформу на процессорах других производителей.
3. Базовые классы .NET Framework. Как и библиотеки на других платформах, здесь нам предлагается обширный набор классов, которые упрощают создание приложения. С помощью этих компонентов вы можете строить свои приложения как бы из блоков. Когда я услышал это выражение в отношении MFC, то долго смеялся, потому что построение приложений с помощью классов MFC больше похоже на копание в нефтяной скважине лопатой, но никак не на строительство из блоков. Компоненты .NET реально упрощают программирование, и разработка приложений с помощью расстановки компонентов действительно стала похожа на строительство домика из готовых блоков.
4. Расширенные классы .NET Framework. В предыдущем пункте говорилось о базовых классах, которые реализуют базовые возможности. Также выделяют более сложные компоненты доступа к базам данных, XML и др.
5. Web-сервисы, Web-формы, Windows-формы. Это основа любого приложения. Если вы программировали на Delphi, то аналогом может служить класс `TForm`, который реализует все необходимые окну программы возможности.

.NET не является переработкой Java, у них общее только то, что они являются виртуальными машинами и выполняют не машинный код, а байт-код. Да, они оба произошли из C++, но "каждый пошел своей дорогой, а поезд пошел своей" (из песни Макаревича).

## 1.1.2. Сборки

Термин сборок в .NET переводят и преподносят по-разному. Я встречал много различных описаний и переводов, например компоновочные блоки или бинарные единицы. На самом деле, если не углубляться в терминологию, а посмотреть на сборки глазами простого программиста, то окажется, что это просто файлы, являющиеся результатом компиляции. Именно конечные файлы, потому что среда разработки может сохранить на диске после компиляции множество промежуточных файлов.

Наиболее распространены два типа сборки — библиотеки, которые сохраняются в файлах с расширением `dll`, и исполняемые файлы, которые сохраняются в файлах с расширением `exe`. Несмотря на то, что расширения файлов такие же, как и у Win32-библиотек и приложений, это все же совершенно разные файлы по своим внутренностям. Программы .NET содержат не инструкции процессора, как в классических Win32-приложениях, а IL-код (промежуточный язык, но можно также встретить термины CIL (Common Intermediate Language) или MSIL (Microsoft Intermediate Language)). Этот код создается компилятором и сохраняется в файле. Когда пользователь запускает программу, то она на лету компилируется в машинный код и выполняется на процессоре.

Так что я не буду в книге использовать мудрые фразы типа "компоновочный блок" или "бинарная единица", а просто буду называть вещи своими именами — исполняемый файл, библиотека и т. д. в зависимости от того, что мы компилируем. А если нужно сказать в общем, не привязываясь к типу, я просто буду говорить "сборка".

Благодаря тому, что IL-код не является машинным, а интерпретируется JIT-компилятором (Just-In-time Compiler, компилятор периода выполнения), то говорят, что код управляем этим JIT-компилятором. Машинный код выполняется напрямую процессором, и ОС не может управлять этим кодом. А вот IL-код выполняется на .NET платформе, и она уже решает, как выполнять, какие процессорные инструкции использовать, а также берет на себя множество рутинных вопросов безопасности и надежности выполнения.

Тут нужно сделать еще одно замечание, чтобы понять, почему программы .NET внешне не отличаются от классических приложений и имеют те же расширения. Это сделано для того, чтобы скрыть сложности реализации от конечного пользователя. Зачем ему знать, как выполняется программа и на чем она написана? Это совершенно не имеет значения. Как я люблю говорить, главное — это качество программы, а как она реализована, на каком языке и на какой платформе, нужно знать только программисту и тем, кто в этом заинтересован. Конечного пользователя это не интересует.

Помимо кода в сборке хранится информация об используемых типах данных. Метаданные сборки очень важны с точки зрения описания объектов и их использования. В файле есть метаданные не только данных, но и самого исполняемого фай-

ла, где хранятся версия сборки и ссылки на используемые внешние сборки. В последнем утверждении кроется одна интересная мысль — сборки могут ссылаться на другие сборки, что позволяет нам создавать многомодульные сборки.

Чаще всего код делят по сборкам по принципу логической завершенности. Допустим, что наша программа реализует работу автомобиля. Все, что касается работы двигателя, можно поместить в отдельный модуль, а все, что касается трансмиссии, — в другой. Помимо этого, каждый модуль будет разбивать составляющие двигателя и трансмиссии на более мелкие составляющие с помощью классов. Код, разбитый на модули, проще сопровождать, обновлять, тестировать и загружать пользователю. Пользователю нет необходимости загружать все приложение, можно предоставить возможность обновления через Интернет, и программа будет сама скачивать только те модули, которые были обновлены.

Теперь еще на секунду хочу вернуться к JIT-компиляции и сказать об одном сильном преимуществе этого метода. Когда пользователь запускает программу, то она компилируется так, чтобы максимально эффективно выполняться на железе и ОС компьютера, где сборка была запущена на выполнение. Для персональных компьютеров существует множество различных процессоров, и, компилируя программу в машинный код, чтобы он выполнялся на всех компьютерах, программисты очень часто не используют современные инструкции процессоров, дабы не сужать рынок продаж. JIT-компилятор может использовать эти инструкции и оптимально скомпилировать программу под текущий процессор. Кроме того, разные ОС обладают разными функциями, и JIT-компилятор также может использовать возможности ОС максимально эффективно.

Из-за компиляции кода во время запуска первый запуск на компьютере может оказаться достаточно долгим. Но когда платформа сохранит результат компиляции в кэше, последующий запуск будет выполняться намного быстрее.

За выполнение сборки отвечает среда выполнения CLR. Основа CLR реализована в библиотеке `mscorlib.dll`, которая называется Microsoft .NET Runtime Execution Engine (библиотека исполнения кода) и отвечает за выполнение сборок. Основные типы данных .NET реализованы в библиотеке `mscorlib.dll`.

### 1.1.3. Язык программирования

Несмотря на то, что платформа .NET не зависит от языка, и для нее можно писать на многих различных языках, самым популярным является C#, потому что он разрабатывался именно для этой платформы и создавался с нуля. У языка нет какого-то старого наследия, которое нужно сохранить, поэтому он вобрал в себя все лучшее из других языков программирования.

На данный момент этих знаний нам будет достаточно. Перейдем к рассмотрению среды разработки Visual Studio .NET и нового языка C#, и параллельно с прак-



тическими знаниями будем углубляться в платформу. Таким образом, закрепим теоретические знания на практике.

## 1.2. Обзор среды разработки Visual Studio .NET

Познакомившись с основами платформы .NET, переходим непосредственно к практике. Для этого мы бросим беглый взгляд на новую среду разработки и посмотрим, что она предоставляет нам, как программистам.

Итак, запускаем Visual Studio .NET. На момент написания этой книги я использовал последнюю вышедшую версию — 2010. Перед нами открывается главное окно, которое состоит из нескольких панелей (рис. 1.2).

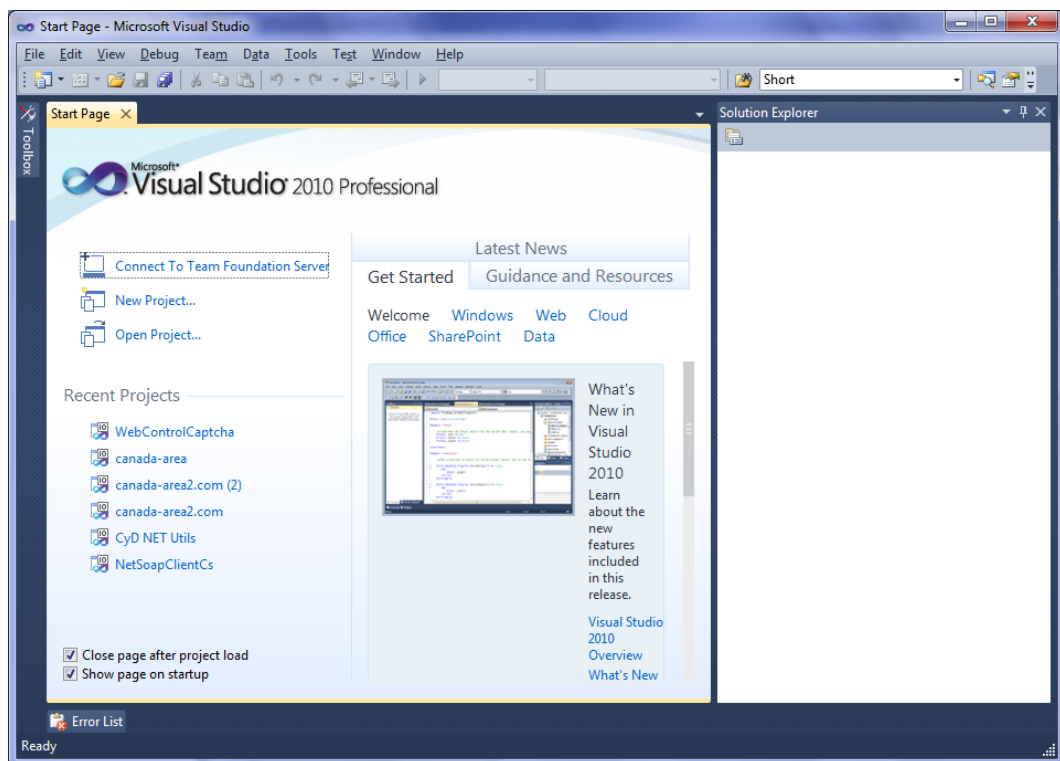


Рис. 1.2. Главное окно Visual Studio .NET

В центре окна расположена рабочая область, в которой в виде вкладок будут открываться файлы. На данный момент в рабочей области открыта HTML-страничка, в которой можно создать новый или открыть существующий проект, а также почитать последние новости из мира разработки.

## 1.2.1. Работа с проектами и решениями

В Visual Studio любая программа заключается в проект. Проект — это как папка для файлов. Он обладает определенными свойствами (например, платформа и язык, для которого создан проект) и может содержать файлы с исходным кодом программы, который необходимо скомпилировать в исполняемый файл. Проекты могут объединяться в решения (Solution). Более подробно о решениях и проектах можно почитать в файле Documents/Visual Studio 2008.doc на компакт-диске.

Для создания нового проекта выберите в меню **File | New | Project** (Файл | Новый | Проект). Перед вами откроется окно **New Project** (Новый проект), как на рис. 1.3. Слева расположено дерево проектов. Здесь можно выбрать тип проекта. Все типы проектов объединены по папкам.

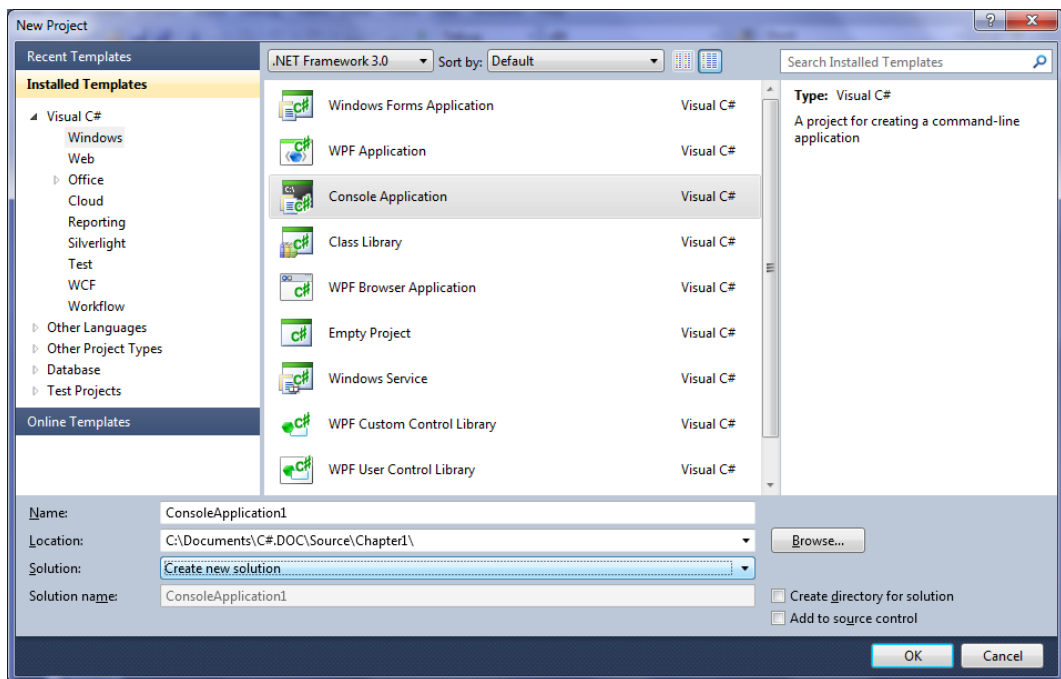


Рис. 1.3. Окно создания нового проекта

В последней версии Visual Studio во время первого запуска среда разработки спрашивает, какой язык программирования вы предпочитаете. Я выбрал C#, поэтому раздел с этим языком в окне создания проектов у меня находится на первом месте. Остальные языки попадают в раздел **Other Languages**. Проекты для создания пакетов инсталляций и развертывания могут находиться в **Other Project Types**. Почему я говорю "могут"? Дерево разделов изменяется от версии к версии и от типа среды разработки, хотя после перехода с 2008 на 2010 изменений оказалось немного.

Среда разработки Visual Studio может работать и компилировать проекты на нескольких языках: Visual C++, Visual C#, Visual Basic и т. д. Для каждого языка есть своя папка в дереве **Other Project Types**, но количество папок зависит от типа установки. Бесплатные версии среды разработки Express Edition рассчитаны для работы с определенным языком программирования.

В списке **Templates** (Шаблоны) посередине появляются типы проектов выбранного раздела. Например, если в дереве проектов слева выбрать тип проекта **Other Languages | Visual C++**, то в списке **Templates** появятся ярлыки для создания проектов этого типа.

Внизу окна создания нового проекта есть два поля ввода:

- ◆ **Name** — здесь вы указываете имя будущего проекта;
- ◆ **Location** — расположение папки проекта.

При задании имени и расположения будьте внимательны. Допустим, что в качестве пути вы выбрали **C:\Projects**, а имя **MyProject**. Созданный проект будет находиться в папке **C:\Projects\MyProject**. То есть среда разработки создаст папку с именем проекта, указанным в поле **Name**, в папке, указанной в поле **Location**.

В Visual Studio 2008 Express Edition можно создавать проекты без указания их расположения. Проект будет создан в памяти (а точнее во временном хранилище), и вы можете начинать писать код и даже компилировать и выполнять его. Если же выбрать сохранение проекта, который вы создали в памяти, то тогда вам будет предложено выбрать папку и имя файла проекта, ведь во время создания вы не указывали этой информации, и среда разработки не знает, где нужно сохранить файл.

Файл проекта, в котором находятся все настройки и описания входящих в проект файлов, имеет расширение `csproj`. На самом деле этот файл имеет формат XML и его легко просмотреть в любом текстовом редакторе или специализированной программе. Вы даже можете редактировать его в текстовом редакторе, но в большинстве случаев проще использовать специализированные диалоговые окна, которые предоставляет среда разработки.

Настройки, которые находятся в файле проекта `csproj`, легко изменять с помощью окна **Properties** (Свойства). Это окно можно вызвать выбрав меню **Project | Xxxx properties**, где Xxxx — имя вашего проекта. Если вы назвали проект HelloWorld, то имя файла проекта будет HelloWorld.csproj, а меню для вызова свойств будет выглядеть как **Project | HelloWorld properties**.

Внизу окна создания нового проекта могут располагаться два переключателя или выпадающий список с выбором (если у вас уже открыт какой-то проект в среде разработки):

- ◆ **Add to Solution** — добавить в текущее решение;
- ◆ **Create new solution** — закрыть решение. Текущее решение будет закрыто и создано новое.

Давайте создадим новый пустой C#-проект, чтобы увидеть, из чего он состоит. Выделите в дереве проектов слева раздел **Visual C#** и затем в списке **Templates** выберите **Windows Forms Application**. Укажите имя и расположение проекта и нажмите кнопку **OK**.

Вот теперь панели, расположенные справа, заполнились информацией, и имеет смысл рассмотреть их более подробно. Но сначала посмотрим, что появилось слева в окне. Это панель **Toolbox** (Инструментальные средства). У вас она может иметь вид тонкой полоски. Наведите на нее указатель мыши, и выдвинется панель, как на рис. 1.4. Чтобы закрепить **Toolbox** на экране, щелкните по кнопке с изображением иголки в заголовке панели (слева от кнопки закрытия панели).

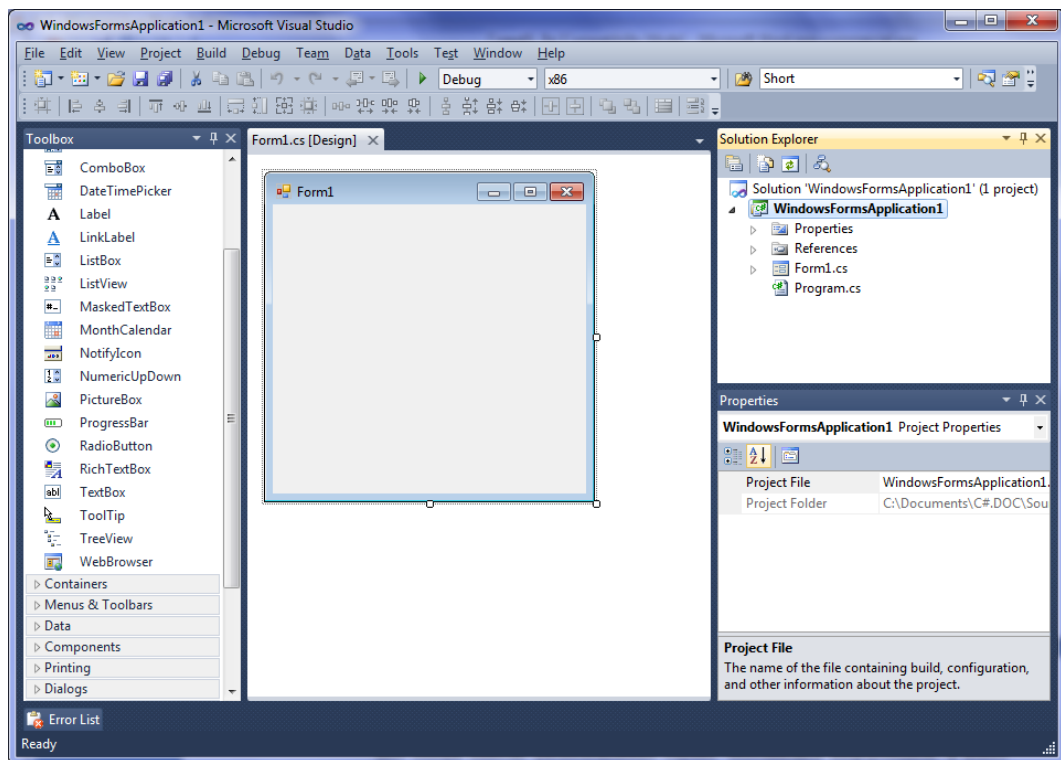


Рис. 1.4. Окно Visual Studio с открытым проектом

Слева внизу окна находятся три панели:

- ◆ **Server Explorer** — панель позволяет просматривать соединения с базами данных и редактировать данные в таблицах прямо из среды разработки;
- ◆ **Toolbox** — на этой панели располагаются компоненты, которые вы можете устанавливать на форме;
- ◆ **Data Sources** — источники данных.

Любая из этих панелей может отсутствовать, а могут присутствовать и другие панели, потому что среда разработки Visual Studio настраиваема, и панели можно закрывать, а можно и отображать на экране, выбирая их имена в меню **View** (Вид). Давайте познакомимся с этими и другими панелями, которые стали нам доступны.

## 1.2.2. Server Explorer

Панель **Server Explorer** (рис. 1.5) представляет собой дерево, состоящее из двух основных разделов:

- ◆ **Data Connections** — в этом разделе можно создавать и просматривать соединения с базами данных;
- ◆ **Servers** — зарегистрированные серверы. По умолчанию зарегистрирован только ваш компьютер.

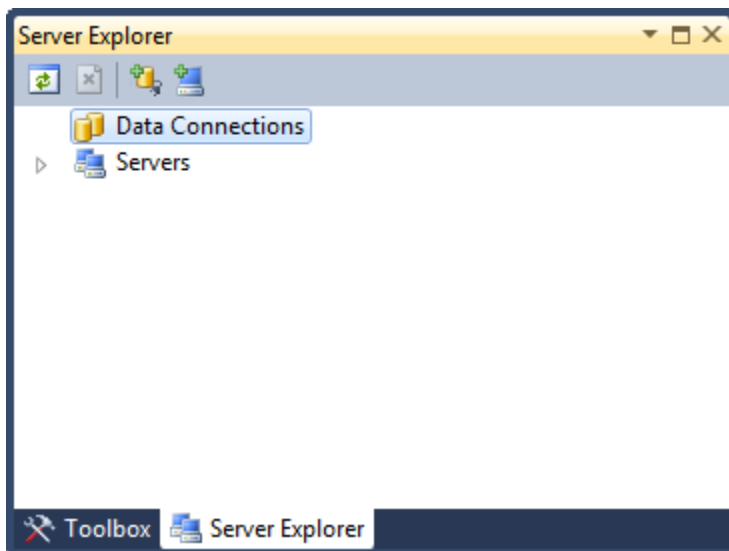


Рис. 1.5. Панель **Server Explorer**

Щелкните правой кнопкой мыши по разделу **Data Connections**. В контекстном меню можно увидеть два очень интересных пункта: **Add Connection** (Добавить соединение) и **Create New SQL Server Database** (Создать новую базу данных). Мы этого делать не будем, а попробуем создать соединение с уже существующей таблицей или базой данных.

Выберите пункт контекстного меню **Add Connection**, и перед вами появится стандартное окно соединения с базой данных (рис. 1.6).

Раскройте ветку созданного соединения, и перед вами появится содержимое этой базы данных. Для Access это будут разделы:

- ◆ **Tables** — таблицы, которые есть в базе данных;
- ◆ **Views** — представления. Я их часто называю хранимыми запросами выборки, потому что это запросы `SELECT`, которые хранятся в самой базе данных;
- ◆ **Stored Procedures** — хранимые процедуры.

Вы можете просматривать содержимое таблиц и хранимых запросов прямо из среды разработки. Для этого дважды щелкните по имени таблицы, и ее содержимое появится в отдельном окне рабочей области.

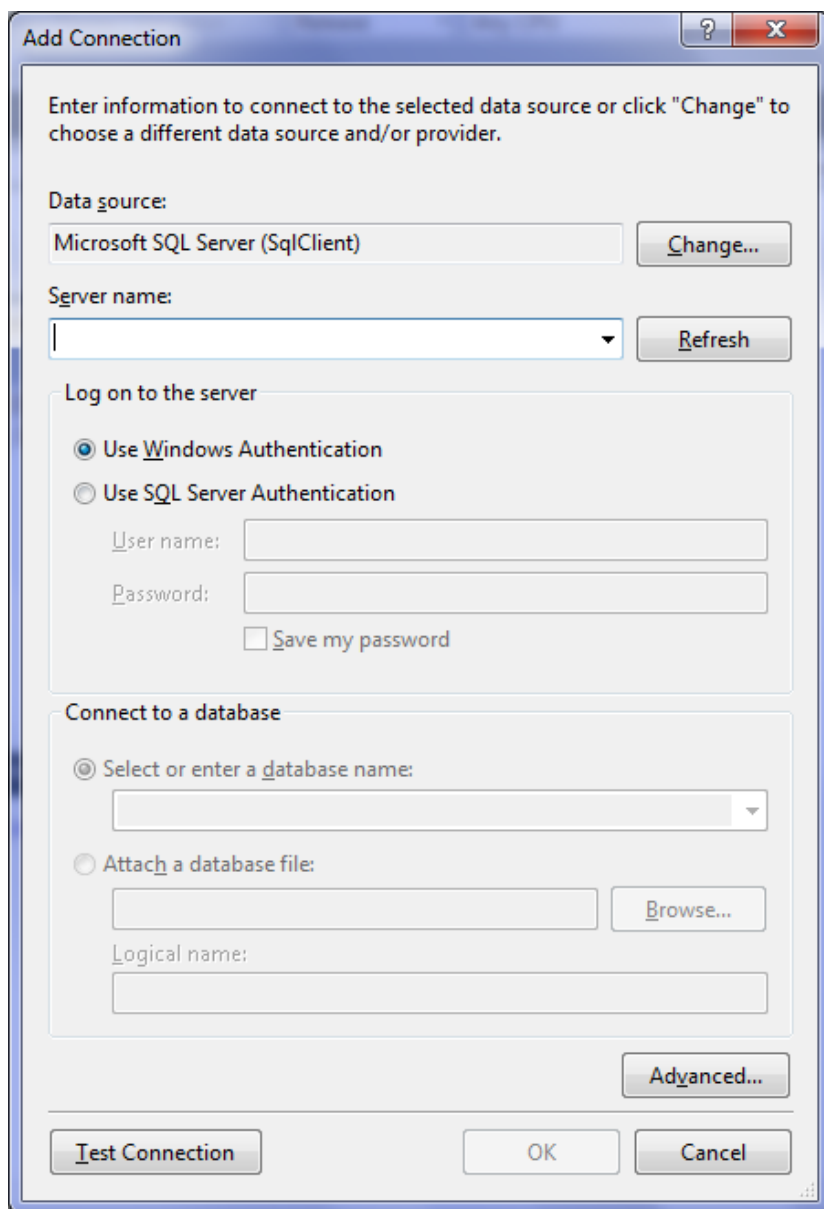


Рис. 1.6. Окно настройки подключения с базой данных

Теперь переходим к рассмотрению раздела **Servers**. Щелкните правой кнопкой мыши по имени раздела. В контекстном меню вы можете выбрать **Add server** для добавления нового сервера. В моей домашней версии Standard Edition этого пункта в контекстном меню нет, но на работе в Visual Studio 2005 Professional он есть.

Раскрыв ветку своего компьютера, вы можете увидеть список его событий, установленных сервисов, серверов и т. д. Если у вас установлен Microsoft SQL Server,