



Вячеслав Понамарев



Программирование на C++ / C# в Visual Studio .NET 2003

- Платформы .NET и .NET Framework
Основные понятия
- Среда Visual Studio .NET 2003
Рекомендации по настройке и расширению
среды разработки
- Языки C++ и C#
Примеры создания программ
- Web-приложения
Создание полноценного Web-магазина
компьютерной техники



МАСТЕР ПРОГРАММ

УДК 681.3.068+800.92С++/С#
ББК 32.973.26-018.1
П56

Понамарев В. А.

П56 Программирование на С++/С# в Visual Studio .NET 2003. —
СПб.: БХВ-Петербург, 2004. — 352 с.: ил.

ISBN 5-94157-402-9

В книге рассматриваются особенности разработки приложений в среде Visual Studio .NET 2003 с применением языков программирования С++ и С#. Проводится сравнительный анализ алгоритмических языков. Показано удобство создания приложений на языке С#, а также аналогичность этого языка языку Visual Basic .NET. Раскрываются основы работы с данными, изучаются технологии ADO и XML. Для повышения наглядности материала используются схемы, рисунки, таблицы, приводятся также поясняющие программы-примеры. Уделяется внимание самостоятельному созданию больших проектов с последующим ознакомлением с механизмом отладки приложений.

Для программистов

УДК 681.3.068+800.92С++/С#
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>Елена Яковлева</i>
Компьютерная верстка	<i>Ольга Сергиенко</i>
Корректор	<i>Зинаида Дмитриева</i>
Оформление серии	<i>Via Design</i>
Дизайн обложки	<i>Игоря Цырульникова</i>
Зав. производством	<i>Николай Тверских</i>

Лицензия ИД № 02429 от 24.07.00. Подписано в печать 27.01.04.

Формат 70×100^{1/16}. Печать офсетная. Усл. печ. л. 28,38.

Тираж 3000 экз. Заказ №

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

Гигиеническое заключение на продукцию, товар № 77.99.02.953.Д.001537.03.02
от 13.03.2002 г. выдано Департаментом ГСЭН Минздрава России.

Отпечатано с готовых диапозитивов
в Академической типографии "Наука" РАН
199034, Санкт-Петербург, 9 линия, 12.

ISBN 5-94157-402-9

© Понамарев В. А., 2004
© Оформление, издательство "БХВ-Петербург", 2004

Содержание

Введение	9
На кого рассчитана книга	9
Структура и особенности книги	9
Соглашения, используемые в книге	10
ЧАСТЬ I. ПЛАТФОРМА .NET	11
Глава 1. Платформы .NET и .NET Framework	13
Что такое Microsoft .NET	13
Архитектура платформы .NET	14
Цели создания .NET Framework	15
.NET Framework	15
Глава 2. Среда CLR	18
Что такое Common Language Runtime	18
Исполняемые файлы CLR	18
Пример на C++	19
Пример на Visual Basic .NET	21
Пример на C#	22
Переносимые исполняемые файлы .NET	23
Метаданные	24
Просмотр метаданных	25
Сборки и манифесты	27
Глава 3. Программирование в среде .NET	30
Общая модель программирования	30
Глава 4. Работа с распределенными .NET-компонентами	37
Распределенные компоненты	37

ЧАСТЬ II. СРЕДА РАЗРАБОТКИ VISUAL STUDIO .NET 2003	41
Глава 5. Среда разработки и ее настройка	43
Описание и назначение основных окон Visual Studio .NET 2003.....	43
Окно редактора.....	46
Окно просмотра решения.....	51
Окно свойств.....	52
Окно вывода и окно команд.....	53
Настройки среды.....	54
Работа с макросами Visual Studio	60
Глава 6. Расширение среды разработки	63
Создание дополнений.....	63
Глава 7. Проекты и решения.....	72
Оперирование проектами и решениями.....	72
Создание решений и проектов	73
Добавление файлов и элементов в решения и проекты.....	74
Глава 8. Работа с редактором кода	76
Редактор кода Visual Studio .NET.....	76
Использование механизма IntelliSense.....	78
Глава 9. Работа с файлами ресурсов	83
Файлы ресурсов.....	83
Создание файлов ресурсов.....	84
Редакторы ресурсов.....	87
Редактор Accelerator Editor.....	87
Редактор Binary Editor.....	89
Редактор Dialog Editor.....	90
Редактор HTML Editor.....	91
Редактор Image Editor.....	92
Редактор Menu Editor.....	92
Редактор String Editor.....	94
Редактор Toolbar Editor.....	96
Редактор Version Information Editor.....	97
ЧАСТЬ III. ЯЗЫКИ C++ И C#	99
Глава 10. Язык C# как развитие языка C++	101
Особенности C#	101
Web-интеграция.....	102
Исключение ошибок.....	102
Простота использования.....	103
Типовая защищенность	105
Удобство и современность C#	105
Дополнительные возможности	108

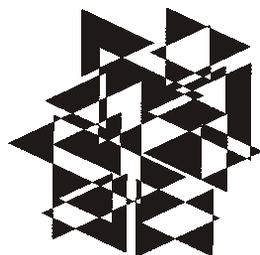
Глава 11. Синтаксис языков C# и C++	110
Основы языка C#	110
Простая программа на C#	110
Комментарии	111
Типы данных	114
Простые типы данных	114
Структурные типы данных	117
Ссылочные типы данных	118
Строковый тип данных	119
Типы перечисления	119
Определенность значений	120
Преобразование типов данных	121
Работа с массивами	122
Одномерные массивы	123
N-мерные массивы	123
Свободные массивы	124
Операции с массивами	124
Ввод и вывод в консольных приложениях	126
Операции и операторы	128
Выражения и операции	128
Унарные операции	129
Бинарные операции	130
Прочие операции	136
Порядок выполнения операций	142
Операторы	142
Оператор перехода	143
Оператор условия	144
Оператор выбора	146
Операторы цикла	148
Глава 12. Объектное и компонентное программирование на C++ и C#	155
Объекты и компоненты	155
Интерфейсы	158
Свойства	159
Использование делегатов и событий	162
Объявление делегатов	162
Типы делегатов	165
Внутренняя структура делегатов	167
Одиночные делегаты	167
Комбинированные делегаты	169
Структура комбинированного делегата	171
Обратные вызовы	172
События	173
Использование событий	181
Особенности использования событий	184
Обработка исключений	188
Блоки <i>try...catch</i>	188
Блок <i>finally</i>	190

Способы обработки исключений.....	191
Обработка нескольких исключений.....	191
Обработка и передача исключений.....	193
Восстановление нормальной работы после исключений.....	194
ЧАСТЬ IV. ДОСТУП К ДАННЫМ И ИСПОЛЬЗОВАНИЕ XML	199
Глава 13. Технология ADO.NET	201
Архитектура ADO.NET	201
Преимущества ADO.NET	202
Производительность.....	203
Продуктивность.....	203
Межплатформенное взаимодействие	204
Масштабируемость.....	204
Работа с ADO.NET.....	205
Объект <i>DataSet</i>	205
Пример создания набора данных	207
Объект <i>DataTable</i>	217
Отношения в наборе данных	218
Глава 14. Работа с XML-файлами.....	222
Краткие сведения об XML	222
XML-анализаторы.....	223
Анализаторы на основе деревьев	224
Анализаторы на основе потоков.....	231
Преобразование XML-формата	236
ЧАСТЬ V. СОЗДАНИЕ WEB-ПРИЛОЖЕНИЙ В VISUAL STUDIO .NET 2003.....	241
Глава 15. Принципы создания Web-приложений	243
Понятие Web-службы	243
Архитектура Web-служб.....	244
Форматы связи Web-служб.....	244
Описание Web-службы на языке WSDL.....	245
Открытие Web-службы.....	251
Поставщики Web-служб	253
Клиенты Web-служб.....	259
Web-клиент на основе HTTP GET	259
Web-клиент на основе HTTP POST и SOAP.....	259
Безопасность Web-служб.....	263
Безопасность на уровне системы.....	263
Безопасность на уровне приложений.....	264
Глава 16. Создание Web-форм	266
ASP и ASP.NET.....	266
Технология ASP.....	266
Технология ASP.NET.....	267

Основные пространства имен и классы ASP.NET	268
Пространство имен <i>System.Web.UI</i>	268
Класс <i>Control</i>	268
Класс <i>Page</i>	270
Класс <i>UserControl</i>	271
Пространство имен <i>System.Web.UI.HtmlControls</i>	271
Пространство имен <i>System.Web.UI.WebControls</i>	275
Синтаксис Web-форм	277
Директивы	277
Блоки объявления кода.....	278
Синтаксис элементов управления HTML.....	279
Синтаксис нестандартных элементов управления	280
Выражения привязки данных	280
Теги серверных объектов	281
Разработка приложения ASP.NET.....	282
Компоненты Web-формы	282
События Web-формы	285
Серверные элементы управления	285
Создание Web-служб с помощью ASP.NET.....	286
Привязка данных и применение шаблонов.....	289
Управление состоянием сеансов	290
Управление состояниями сеансов в ASP.NET.....	291
Глава 17. Начало проекта	294
Обзор ранних технологий.....	294
Постановка задачи	295
Создание и подготовка базы данных	296
Глава 18. Создание проекта	300
Структура интернет-магазина	300
Главная страница — выбор типа заказа.....	300
Выбор типовой конфигурации компьютера.....	301
Выбор дополнительного оборудования.....	303
Подбор комплектующих	304
Отправка заказа по e-mail (файл Done.asp)	318
Глава 19. Принципы отладки приложений.....	321
Простейшая отладка.....	321
Трассировка времени исполнения	324
Создание проверок.....	326
ЧАСТЬ VI. ПРИЛОЖЕНИЯ	329
Приложение 1. Список сайтов.....	331
Приложение 2. Редакции Visual Studio .NET 2003.....	332

Приложение 3. Системные требования для установки Visual Studio .NET 2003	334
Приложение 4. Языки .NET	335
Приложение 5. Общие типы данных языков .NET	337
Приложение 6. Сокращения и термины	339
Предметный указатель	345

ГЛАВА 1



Платформы .NET и .NET Framework

В этой главе приводятся сведения о платформах .NET и .NET Framework. Описываются цели создания и преимущества платформ .NET и .NET Framework.

Что такое Microsoft .NET

Первое заявление корпорации Microsoft о переходе на платформу .NET было летом 2000-го года на конференции PDC 2000 в городе Орlando штата Флорида.

Платформа .NET создавалась как среда разработки приложений под Windows с новым интерфейсом программирования. В .NET интегрированы службы компонентов COM+, работа с XML, среда Web-разработки ASP, а также ориентация на создание интернет-приложений и поддержка новейших Web-сервисов (SOAP, UDDI и WSDL).

Вся платформа .NET состоит из различных продуктов, которые можно условно разделить на четыре группы:

- *средства разработки* — т. е. языки программирования (Visual C, C#, Visual Basic .NET, Visual Java), среда Common Language Runtime (CLR, общеязыковая среда выполнения), библиотека классов для создания разнообразных приложений, а также инструментальная среда разработки Visual Studio .NET;
- *Web-сервисы* — т. е. возможность применения коммерческих Web-сервисов (таких, как .NET MyServices), которые необходимы для создания Web-приложений, требующих идентификации пользователей;
- *специализированные серверы* — набор серверов SQL Server, Exchange Server, BizTalk и др., объединенных в одно семейство серверов .NET Enterprise Servers. Эти серверы обеспечивают работу с базами данных, с электронной почтой, и многое другое;

- *поддержка устройств* — встроенная поддержка новых устройств ("умных" устройств), которые могут работать с технологиями .NET (например, мобильные телефоны).

Архитектура платформы .NET

В упрощенном общем виде платформа .NET состоит из пяти основных компонентов (рис. 1.1).

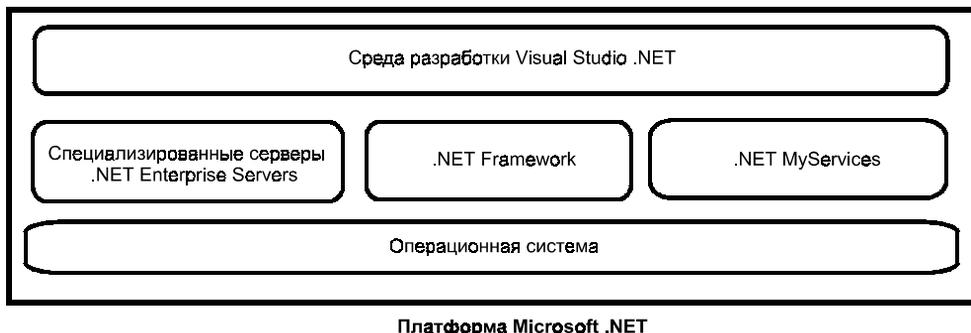


Рис. 1.1. Архитектура платформы .NET

На нижнем уровне платформы находится операционная система (Windows XP, Windows 2000, Windows ME или Windows CE). Кроме перечисленных операционных систем может использоваться и другое специализированное программное обеспечение для так называемых "умных" устройств.

На уровне, расположенном выше уровня операционных систем (рис. 1.1), находятся сразу три компонента:

- *специализированные серверы .NET Enterprise Servers* — набор серверных продуктов, таких как Application Center, BizTalk Server, Commerce Server, Exchange Server, Host Integration Server, Internet Security Acceleration Server и SQL Server;
- *набор Web-сервисов .NET MyServices* — представляющих собой готовые блоки кода, которые разработчик может включать (за дополнительную плату) в свои проекты;
- *.NET Framework* — новая инфраструктура разработки и исполнения Windows-приложений, которая включает в себя общезыковую среду выполнения CLR, а также общую структуру классов, которые можно использовать в любом языке программирования, относящемся к семейству .NET.

На верхнем уровне архитектуры .NET располагается среда разработки приложений Visual Studio .NET. *Visual Studio .NET* — это интегрированная среда

разработки (Integrated Development Environment, IDE), которая содержит разнообразные средства для создания приложений на языках .NET.

Цели создания .NET Framework

Одной из главных целей создания .NET Framework является упрощение построения Windows-приложений.

Для реализации таких приложений было создано большое количество сред разработки, каждая из которых отличалась от других библиотеками классов, синтаксисом, программным интерфейсом приложений (API, Application Programming Interface) и др. Никакой стандартизации между этими интерфейсами и библиотеками классов просто не существовало.

Использование .NET Framework позволяет программисту использовать набор базовых классов, который подходит для любого языка программирования, относящегося к .NET (C++, C#, Visual Basic .NET и др.). Для успешной работы с любым из этих языков программирования не нужно всякий раз изучать новый API.

Перечислим основные цели создания .NET Framework:

- обеспечение одинаковых возможностей объектно-ориентированного программирования, независимо от используемого языка программирования;
- обеспечение неконфликтности версий создаваемого программного обеспечения и упрощение развертывания приложений, за счет регистраций общих DLL (Dynamic Link Library, динамически подключаемая библиотека) в глобальном кэше сборок (Global Assembly Cache, GAC);
- обеспечение безопасности выполнения кода приложения. .NET Framework предоставляет множество функций безопасности, обеспечивающих надежную защиту от несанкционированного проникновения в приложение или систему. В частности, вы можете защитить даже определенную часть исполняемого кода (например, метод).

Рассмотрим архитектуру .NET Framework.

.NET Framework

В общем виде, .NET Framework является простым системным приложением, работающим в операционной системе, такой как Windows (хотя, могут использоваться и другие операционные системы, например, относящиеся к семейству UNIX).

.NET Framework состоит из двух основных частей:

- *CLR* (Common Language Runtime, общезыковая среда выполнения);
- *библиотеки классов .NET Framework*.

На рис. 1.2 представлена упрощенная схема архитектуры .NET Framework.

Главным компонентом .NET Framework является CLR. О среде CLR речь пойдет в гл. 2. Но пока отметим, что CLR занимается активизацией объектов, отладкой, обработкой исключений, проверкой типов, распределением памяти и ресурсов для объектов, а также осуществляет "сборку мусора", т. е. при прекращении использования того или иного объекта происходит автоматическое уничтожение объекта. Таким образом, CLR осуществляет те же самые действия, которые выполняет виртуальная Java-машина (Java Virtual Machine, JVM) в языке Java.

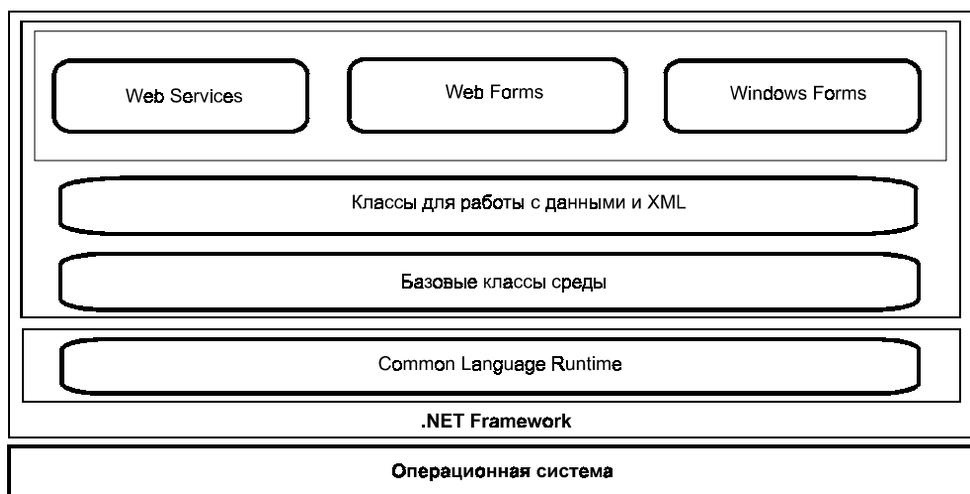


Рис. 1.2. Архитектура .NET Framework

CLR и JVM во многом похожи, но отличаются уже тем, что JVM поддерживает только язык программирования Java, а CLR поддерживает любые языки программирования, которые можно представить на общем для языков .NET промежуточном языке Microsoft Intermediate Language (MSIL, промежуточный язык Microsoft).

Другое отличие JVM и CLR заключается в том, что код Java с помощью JVM может работать на различных платформах (на которых установлена сама JVM), а код .NET работает пока только на платформе Windows.

Библиотеку классов .NET Framework можно условно поделить на несколько частей, согласно их предназначению.

Первая часть — набор базовых классов среды. Это стандартные классы, обеспечивающие работу со строками, ввод и вывод данных, многопоточность, работу с сетью, безопасность и многое другое. Базовые классы похожи по своему составу и функциональности на такие классы, как MFC, ATL и др.

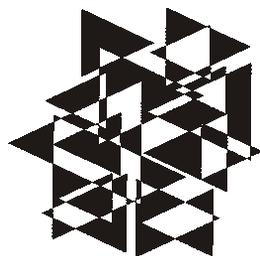
Вторая часть — это набор классов для работы с данными и XML (eXtensible Markup Language). Классы для работы с данными включают в себя класс работы с SQL (Structured Query Language, язык структурированных запросов), а также набор классов ADO.NET для манипулирования постоянными данными. Классы для работы с XML позволяют манипулировать XML-данными, а также выполнять XML-преобразования и XML-поиск.

Третья часть библиотеки расширяет возможности классов для работы с данными и XML и представляет три различных набора классов для поддержки трех технологий:

- *классы Web Services* — поддерживают разработку облегченных распределенных компонентов;
- *классы Web Forms* — позволяют быстро и эффективно создавать Web-приложения, в которых используется графический интерфейс пользователя (Graphical User Interface, GUI). Методы создания интерфейсов пользователя практически ничем не отличаются от создания интерфейсов пользователя на том же Visual Basic. Вы просто помещаете нужные элементы управления на Web-форму, задаете значения нужных свойств и создаете обработчики событий;
- *классы Windows Forms* — позволяют создавать обычные Windows-приложения, в которых используются стандартные элементы управления Windows. Вы можете рассматривать этот набор классов как улучшенную версию классов MFC.

Итак, мы кратко рассмотрели, из каких компонентов состоит .NET и .NET Framework. В гл. 2 речь пойдет о такой важной части .NET Framework, как среда Common Language Runtime. Мы уточним, как в этой среде поддерживаются и исполняются .NET-компоненты, которые носят название *сборок* (assemblies).

ГЛАВА 2



Среда CLR

В этой главе речь пойдет о среде Common Language Runtime. Мы рассмотрим, что такое CLR и исполняемые файлы CLR. Кроме того, раскрывается понятие метаданных, а также сборок и манифестов.

Что такое Common Language Runtime

Общезыковая среда выполнения (CLR), как уже упоминалось в *гл. 1*, входит в состав .NET Framework. CLR лежит в основе технологии .NET. Она создана, что называется, "с нуля". В основе CLR не лежит ни одна из ранее используемых в различных языках программирования библиотек, таких как MFC, ATL и др.

Среда CLR управляет выполнением кода, написанного на любом из языков семейства .NET. Данная среда занимается созданием различных объектов, выделяет под них системные ресурсы, выполняет проверку безопасности и исполняет код объектов. Кроме того, CLR отвечает за так называемую "сборку мусора", т. е. уничтожение неиспользуемых объектов и освобождению от них системных ресурсов.

Среда CLR оперирует так называемыми *сборками*.

Сборки (assemblies) — представляют собой .NET-компоненты, которые являются переносимыми исполняемыми файлами (Portable Executable, PE). Эти файлы являются файлами с расширениями exe или dll и состоят из метаданных и кода. О метаданных речь пойдет далее в этой главе, а пока рассмотрим исполняемые файлы CLR.

Исполняемые файлы CLR

Рассмотрим на небольшом примере (консольной программе, выводящей текстовое сообщение) особенности языков программирования C++, C# и Visual Basic .NET.

Пример на C++

Язык C++ в среде .NET несколько отличается от раннего C++ тем, что он расширился за счет нескольких специфичных ключевых слов. Это позволяет программам, написанным на C++ .NET, использовать новые возможности (такие, как "сборка мусора").

В листинге 2.1 приведен код простого консольного приложения, выводящего на экран строку текста. Заметьте, что данный код создается средой Visual Studio .NET автоматически, при создании нового консольного проекта при помощи диалогового окна **New Project** (рис. 2.1).

Листинг 2.1. Простое консольное приложение на C++

```
// это главный файл проекта для приложения VC++,
// сгенерированный мастером приложений (Application Wizard)
#include "stdafx.h"
#using <mscorlib.dll>
using namespace System;
int _tmain()
{
    // TODO: замените код примера, расположенный ниже, на свой собственный
    Console::WriteLine(S"Hello World");
    return 0;
}
```

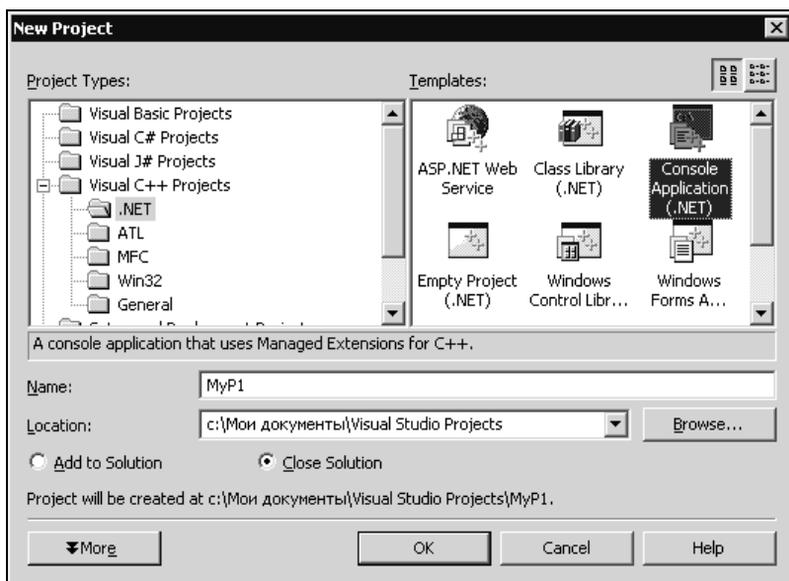


Рис. 2.1. Создание нового консольного приложения на C++

Попробуем разобраться с этим кодом. Как вы можете видеть, это вполне обычная программа на C++. Отличие заключается лишь в использовании директивы компилятора `#using` во второй строке кода. Данная директива делает доступными все типы из указанной библиотеки DLL. Директива `#using` может показаться аналогом директивы `#include`, но в отличие от нее импортирует типы любой сборки .NET, независимо от языка, на котором была написана эта сборка (естественно, это должен быть язык, относящийся к семейству .NET).

Далее все достаточно просто. В методе `_tmain` происходит вызов статического метода `WriteLine()` класса `Console`. Класс `Console` — это тип, который находится в библиотеке `microsoft.dll`.

Кроме того, в следующей строке кода, с помощью оператора `using namespace`, компилятору сообщается, что будут использованы типы из пространства имен `System`. В противном случае код строки

```
Console::WriteLine(S"Hello World");
```

пришлось бы заменить на такое выражение:

```
System::Console::WriteLine(S"Hello World");
```

Примечание

Краткие описания пространств имен вы можете прочитать в гл. 3.

Если запустить это приложение (клавиша `<F5>` или пункт меню **Debug | Start**), то вы увидите результат (рис. 2.2).

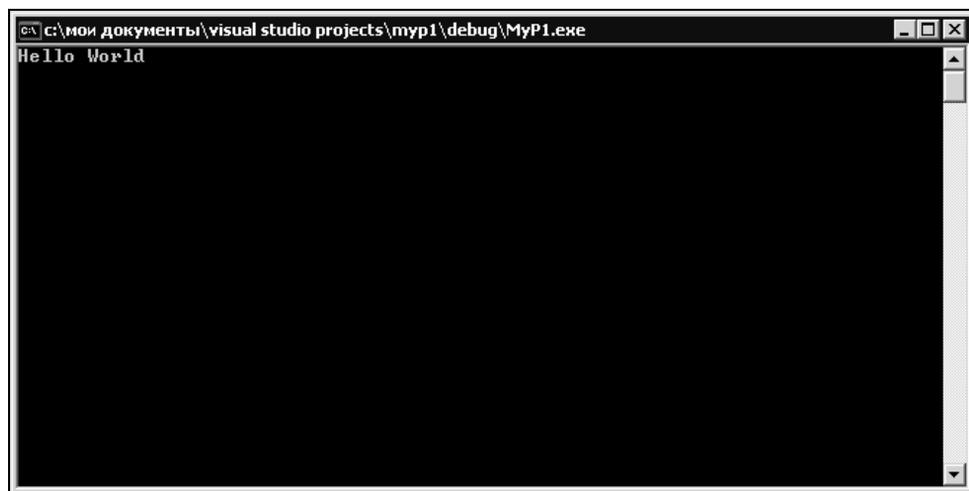


Рис. 2.2. Результат выполнения консольного приложения

Пример на Visual Basic .NET

В листинге 2.2 приведен пример консольного приложения, выводящего одну строку текста на экран, написанного на языке Visual Basic .NET.

Листинг 2.2. Простое консольное приложение на Visual Basic .NET

```
Module Module1
Sub Main()
Console.WriteLine("Hello World")
End Sub
End Module
```

Как вы можете видеть, вызывается тот же самый метод `WriteLine()` класса `Console` (см. листинг 2.1). Если внимательно посмотреть на информацию о данной сборке — файл `AssemblyInfo.vb`, то будет виден следующий текст (листинг 2.3).

Листинг 2.3. Содержимое файла `Assembly.vb`

```
Imports System
Imports System.Reflection
Imports System.Runtime.InteropServices

' общая информация о сборке устанавливается с помощью следующего набора
' атрибутов. Вы можете изменить значения этих атрибутов для задания
' информации о сборке

' просмотр значений атрибутов сборки

<Assembly: AssemblyTitle("")>
<Assembly: AssemblyDescription("")>
<Assembly: AssemblyCompany("")>
<Assembly: AssemblyProduct("")>
<Assembly: AssemblyCopyright("")>
<Assembly: AssemblyTrademark("")>
<Assembly: CLSCompliant(True)>

' следующий GUID является идентификатором ID библиотеки типов, если
' данный проект запланирован как COM-проект
<Assembly: Guid("B8980C3E-3F83-4174-A9CA-C4114ACD51F3")>

' информация о версии данной сборки состоит из следующих четырех значений
',
',
', Major Version
', Minor Version
```

```
' Build Number
' Revision
'
' вы можете указать все значения самостоятельно, или использовать
' значения по умолчанию, с помощью знака '*', как показано ниже

<Assembly: AssemblyVersion("1.0.*")>
```

Как видно из листинга 2.3, в самом начале программы происходит импорт типов из пространств имен `System`, `System.Reflection` и `System.Runtime.InteropServices`. Таким образом, используется то же самое пространство имен `System`, как и в примере на C++ (см. листинг 2.1).

Пример на C#

Рассмотрим реализацию консольной программы вывода текстового сообщения на языке C#, который был разработан фирмой Microsoft специально для среды .NET. Язык C# заимствовал синтаксис у языков C++ и Java. Это очень простой для понимания язык. Кстати, большинство утилит и классов .NET написаны именно на этом языке. Итак, пример простого консольного приложения на C# приведен в листинге 2.4.

Листинг 2.4. Простое консольное приложение на C#

```
using System;

namespace ConsoleApplication3
{
    /// <summary>
    /// общее описание класса Class1
    /// </summary>
    class Class1
    {
        /// <summary>
        /// главная точка входа в приложение
        /// </summary>
        [STAThread]
        static void Main(string[] args)
        {
            //
            // TODO: стартовый код приложения добавьте здесь
            //
            Console.WriteLine("Hello World");
        }
    }
}
```

Здесь также используется пространство имен `System`. Отличие программы на C# от C++ заключается фактически лишь в том, что метод `Main()` в программе на C# является открытой статической функцией класса (в нашем примере — класса `Class1`), а не глобальной функцией, как в C++.

Если внимательно посмотреть на тексты листингов 2.2, 2.3 и 2.4, то можно заметить аналогию между языками Visual Basic .NET и C#. В Visual Basic .NET используются ключевые слова `Import` и `Module`, а в C# — `using` и `class`. Ну и конечно, в C# метод обозначается ключевым словом `void`, а в Visual Basic — `Sub`.

Переносимые исполняемые файлы .NET

Любой исполняемый файл операционной системы Windows (с расширениями `exe` или `dll`) должен соответствовать специальному формату *PE* (Portable Executable, переносимый исполняемый файл).

Формат PE является производным от формата *COFF* (Common Object File Format, общий объектный формат файлов).

Операционная система Windows "понимает" формат PE-файлов, поэтому может загружать и исполнять файлы с расширениями `exe` и `dll`. Таким образом, любой компилятор генерирует исполняемые файлы Windows в соответствии с форматами PE или COFF.

Обычные PE-файлы Windows можно разделить на две секции:

- *заголовки PE/COFF* (включая ссылки на содержимое PE-файла);
- *двоичные образы*. Включают в себя несколько секций:
 - `.Data`;
 - `.Rdata`;
 - `.Rsrc`;
 - `.Text`;
 - другие секции, созданные с помощью компилятора языка C++, используя директиву компилятора `pragma`. В этих секциях можно хранить, например, зашифрованные данные.

Рассмотрим внутреннее строение обычного PE-файла Windows (рис. 2.3).

Для поддержки новых возможностей среды CLR Microsoft немного изменила формат PE-файла, генерируемого средой .NET. А именно, были добавлены несколько новых секций (рис. 2.4).

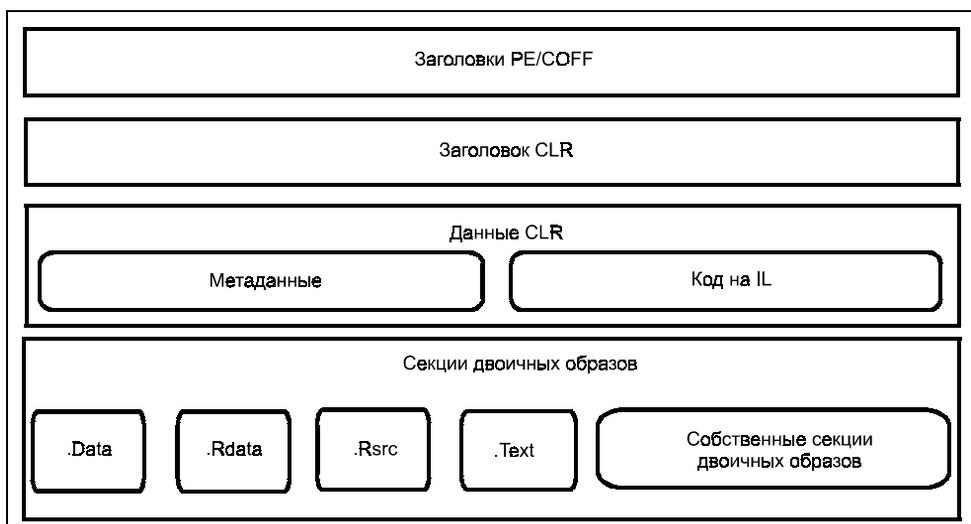
Добавлены следующие секции:

- *заголовок CLR* — содержит информацию, указывающую, что PE-файл является исполняемым файлом .NET;



Обычный PE-файл Windows

Рис. 2.3. Формат обычного PE-файла Windows



Обычный PE-файл Windows

Рис. 2.4. Формат PE-файла, генерируемый средой .NET

- данные CLR* — определяют, как будет выполняться программа. Состоят из:
- *метаданных*;
 - *IL-кода*.

Рассмотрим теперь непосредственно метаданные и IL-код.

Метаданные

Метаданные (metadata) — это данные о данных. Другими словами, метаданные — это данные о ресурсах (например, к метаданным относятся определения типов, сведения о версиях, ссылки на внешние сборки и многое другое).