Государственный университет – Высшая школа экономики

Факультет Бизнес-Информатики

Кафедра Основ информатики
и прикладного программного обеспечения

C#

Объектно-ориентированный язык программирования

Пособие к практическим занятиям -

Введение в .NET

Проф. Забудский Е.И.

Москва 2005

Введение:

характеристика платформы .NET

Microsoft Visual Studio .NET – это полнофункциональная Интегрированная Среда Разработки приложений **.NET** на языках программирования C#, C++, J#, Visual Basic и др. Она состоит из средств разработки графического интерфейса, включает редакторы, мастеров, отладчик и поддержку исполнения приложения и др. В [**3**] детально описывается среда **MS VS .NET 2003**.

Будущее и настоящее программирования связано именно со средой – **Microsoft Visual Studio .NET**.

// **КОММЕНТАРИЙ**. На сайте http://www.firststeps.ru/ “Первые шаги” представлено много интересных обучающих материалов по различным интегрированным средам и языкам программирования, в том числе представлены C# и платформа .NET (step by step).

Содержание

1.	Концепции .NET …… ……	4
1.1.	Назначение технологии .NET……..	4
1.2.	Что такое технология .NET ……..	5
1.3.	Библиотека базовых классов (Framework BCL) ……	6
1.4.	Общеязыковая среда исполнения приложений ……	7
1.5.	Языки программирования ……	9
2.	Общие сведения об интегрированной среде разработки приложений C#Builder …	11
	Список литературы ……	12
	П р и л о ж е н и я	
	1. C# & **.NET** по шагам: http://www.firststeps.ru/dotnet/dotnet1.html	13
	2. Microsoft .NET Framework SDK . Документация . **Web-ресурс** ……...	14

1. Концепции .NET

Платформа .NET — это надстройка над операционными системами, позволяющая создавать прикладные программы, работающие без перекомпиляции на различных аппаратных средствах и с различными операционными системами: Windows, Linux и др. Уже появились первые версии Windows, поддерживающие .NET (Microsoft Windows Server 2003). И в недалеком будущем, по-видимому, разработка приложений для Интернета, распределенных приложений, а также и отдельных программ будет, прежде всего, ориентироваться на .NET. Компания Borland разработала новую среду программирования — C#Builder. C#Builder — средство программирования, обеспечивающее создание прикладных программ для Microsoft .NET. Язык C#, на котором основан C#Builder, создан специально для .NET, объединяет наиболее сильные стороны языков C++ и Java, и станет, вероятно, основным инструментом программирования в .NET.

Компания Borland стала первым независимым разработчиком программного обеспечения, получившим лицензию от корпорации Майкрософт на использование программной платформы **.NET Framework.** Поэтому разработанные с помощью интегрированной среды разработки **C#Builder** приложения будут полностью совместимыми с другими приложениями и библиотеками **.NET**.

Однако **полно**функциональной Интегрированной Средой Разработки приложений **.NET** на языках программирования C#, C++, J#, Visual Basic и др. является среда **Microsoft Visual Studio .NET**.

Информация данного раздела является не исчерпывающей, но очень важной. Понимание основ **.NET**-технологии позволит разобраться во многих вопросах, которые будут возникать в процессе разработки приложений.

1.1. Назначение технологии .NET

В недавнем прошлом языки программирования, операционные системы и программные платформы создавались в условиях, когда основной аппаратной платформой для приложений был настольный компьютер. Когда программы с настольных компьютеров переместились в Интернет, существующие средства разработки по необходимости были расширены дополнительными программными интерфейсами приложений (**API** – Application Programming Interface – программный интерфейс приложений) и другими средствами. Чаще всего новые возможности добавлялись со стороны языков программирования или средств, позволявших им работать в Интернете. Хотя обычные инструментальные средства хорошо работали, и с их помощью была создана сама сеть Интернет, в них имелись узкие места, которые предстояло преодолеть. Технология **.NET** была создана для обеспечения возможности создания компьютерных приложений для Интернета, в которых бы решались проблемы распространения приложений, обеспечения безопасности, использования компонентов различных версий. Центральной частью платформы **.NET** является общеязыковая среда исполнения приложений ***CLR* (Common Language Runtime)** — система исполнения виртуального кода, в которой и решаются проблемы распространения прикладных программ, проблемы безопасности, и отслеживаются версии используемых программных компонентов. При программировании в машинных кодах такие возможности отсутствуют.

1.2. Что такое технология .NET

.NET — это программная платформа создания распределенных приложений. Она включает в себя: ***1)*** библиотеку базовых классов — Framework BCL (*Base Class Library*), ***2)*** общеязыковую среду исполнения приложений — CLR (*Common Language Runtime*), ***3)*** языки программирования. С помощью этих средств можно создавать приложения различного назначения, включая *Windows Forms*, *ADO.NET*, ASP.NET и *Web services*.

Windows Forms -— это набор библиотек, используемых для создания графических интерфейсов пользователя в традиционных приложениях, выполняемых на клиентских компьютерах. Они инкапсулируют многое из Win32 API, обеспечивая объектно-ориентированный подход, облегчающий создание клиентских приложений *Windows*.

ADO.NET (ActiveX Data Object .NET – библиотека классов .NET доступа к данным) — это набор объектно-ориентированных классов, предназначенных для построения в многозвенных приложениях компонентов данных и компонентов доступа к данным. Они построены таким образом, что разработчики могут создавать свои собственные высокопроизводительные источники данных для индивидуальных баз данных. В состав C#Builder_входит источник данных Borland Date Provider, способный работать с разнотипными базами данных. .NET включает в себя также OleDb и ODBC — системы доступа к источникам данных по традиционным стандартам, которые не имеют пользовательских источников данных.

ASP.NET (Active Server Pages .NET) — это технология создания приложений .NET, содержащая модель программирования *Web Forms*, в которой приложения *Web* могут создаваться и исполняться через Интернет, а доступ к ним осуществляется с помощью браузера. Это усовершенствованная модель программирования *Web*, в которой код компилируется на сервере, а результат передается клиенту как традиционный файл HTML. Технология ASP.NET является объектно-ориентированной и поддерживает модель серверных компонентов, способствующую повторному использованию кода.

ASP.NET — это группа компонентов, используемых для упрощения создания приложений на базе браузера.

Браузер — это приложение, которое позволяет пользователю в удобной форме просматривать документы в формате HTML (Hyper Text Marcup Language). Браузеры широко используются для отображения информации в Internet.

Web services — это новый подход к формированию гетерогенных (неоднородных) систем в Интернете, стандартизованный и независимый от платформ. *Web services* .NET использует объектно-ориентированную инфраструктуру программной модели ASP.NET, но предоставляет дополнительно модель обмена сообщениями, основанную на открытых стандартах. Используя такие открытые стандарты, как XML – Extensible Markup Language, SOAP – Simple Object Access Protocol – протокол передачи сообщений, WSDL – Web Service Description Language и UDDI – Universal Description, Discovery, and Integration (перечислены языки, и протоколы, используемые в сетевых задачах), *Web services* взаимодействует с любыми другими стандартизованными сервисами *Web*, независимо от платформ и транспортных протоколов.

Web services — это важная часть новой технологии, которая обещает изменить пути использования Internet, а с ними — представления о том, как разрабатывать приложения и использовать их.

Перечисленное — это лишь несколько наиболее распространенных типов приложений, которые можно создавать, используя технологию **.NET**. Если вы познакомитесь с обширной библиотекой **.NET Framework BCL**, то обнаружите, что она может удовлетворить разнообразные потребности программистов.

1.3. Библиотека базовых классов (Framework BCL)

Библиотека базовых классов .NET Base Class Library (**BCL**) **содержит тысячи типов**, применение которых способствует значительному повышению производительности разработки приложений **.NET.** Чтобы познакомиться с ее возможностями, необходимо потратить определенное время. Прежде чем создавать пользовательский тип, проверьте эту библиотеку — скорее всего такой тип уже существует.

В таблице приведены основные пространства имен и описание типов библиотеки BCL

Таблица. Пространства имен .NET

Пространство имен	Описание
System	Наиболее часто используемые типы.
System.CodeDom	Позволяет создавать типы, которые автоматизируют работу с исходными кодами, т.е. компиляторы и генераторы кодов.
System. Collection	Коллекции типов, таких как **ArrayList, Hashtable** и **Stack**.
System. Configuration	Типы для работы с различными видами конфигурационных файлов **XML**.
System. Data	Большая часть типов, предназначенных для программирования баз данных **ADO.NET**. Другие типы относятся к пространствам имен, специфичным для той или иной базы данных и интерфейса данных.
System. Diagnostics	Типы **Process, EvantLog** и **Performance Counter**.
System. Director/Services	Управляемый интерфейс доступа к **Windows Active Directory Services**.
System. Drawing	Типы **CGI+.**
System. EnterprizeServices	Типы **Com+.**
System. Globallization	Типы поддержки специфических для различных стран календарей, особенностей форматирования и языков.
System. IO	Типы **Directory, File** и **Stream**.
System .Management	API для выполнения задач управления средой **Windows**.
System. Messaging	Типы для работы с очередями сообщений.
System. Net	Доступ к типам сетевых протоколов.
System. Reflection	**API** для метаданных программных сборок.
System .Resources	Типы для управления специфичными для разных, стран ресурсами.
System. Runtime	Поддержка возможностей сетевого взаимодействия моделей COM, взаимодействия в распределенных приложениях и сериализации
System. Security	Типы безопасного доступа к коду, типы, основанные на правилах безопасности, и криптографические типы.
System. ServiceProcess	Типы для создания **Windows Services**.
System. Text	Кодировка / декодировка текста, взаимная трансляция байтовых массивов и строк, класс **StringBuilder** и регулярные выражения.
System.Timers	Типы таймеров.
System. Threading	Типы потоков и объектов синхронизации.
System. Web	Типы **HTTP Communications, ASP.NET** и **Web Services**.
System.Windows	Типы Windows Forms.
System.XML	Вся поддержка типов **XML**, включая **XML Schema**, **XmlTextreaders** / **XmlTextwriters**, **Xpath, XML Serialization** и **XSLT**.

1.4. Общеязыковая среда исполнения приложений

Общеязыковая среда исполнения приложений — **CLR (*Common Language Runtime*)** — это система, обеспечивающая управляемое исполнение программного кода .NET. Центральным моментом в определении CLR является слово «управляемый код». CLR осуществляет исполнение, отслеживание версий программных компонентов и безопасность всего .NET -кода. По этой причине часто код .NET и C# называется «управляемым кодом». Весь программный код, предназначенный для исполнения в общеязыковой среде исполнения платформы .NET, является «управляемым кодом». Напротив, когда говорят в терминологии .NET о «неуправляемом коде», это относится к программному коду, который не обрабатывается системой CLR; к такому коду относятся программы, находящиеся в исполняемых файлах *.exe*, и в библиотечных файлах, которые транслируются непосредственно в машинные коды компьютера.

Управляемый код транслируется не в машинные команды, а в команды промежуточного языка **MSIL (Microsoft Intermediate Language)** [1], далее называемый просто промежуточным языком (**IL**). Этот язык является языком ассемблерного типа. *Система CLR загружает и оперативно (**Just-In-Time, JIT – как-раз-вовремя**) компилирует IL код в машинные команды во время исполнения программы.* **IL** специально создан для работы с разными языками программирования. Целью разработки **IL** является компилируемая спецификация и многообразие языков программирования, о чем разработчики приложений .NET (на языке. C# в частности) должны иметь четкое представление.

Программы .NET состоят из *программных* сборок (assemblies), которые являются минимальными логическими единицами распространения, идентификации и безопасности программного кода. Они отличаются от традиционных файлов *exe* тем, что программные сборки могут состоять из нескольких файлов. *Обычно программная сборка .NET упаковывается в виде одного исполняемого файла или в виде библиотечного файла*. Но она может также содержать модули, являющиеся неисполняемыми программными кодами, которые при этом могут повторно использоваться в других программных сборках.

Программные сборки — это самодокументируемые структуры. Поскольку они содержат свои собственные метаданные, нет необходимости в дополнительных хранилищах, таких как реестры или дополнительные каталоги. Программные сборки могут также содержать цифровую подпись и систему безопасности, защищающую их от воздействия других зловредных программ, а также обеспечивать безопасность типов. Под термином «тип» тут и далее понимается любая именованная абстракция, предопределенная в языке программирования или определенная пользователем. Важность безопасности типов состоит в том, что она делает программный код более устойчивым и помогает защитить его при неправильном использовании программы. Поскольку C# является языком программирования, обеспечивающим безопасность типов, он обнаружит большинство попыток неправильного использования типов в период компиляции и выдаст сообщения об ошибках. Например, в программе нельзя присвоить тип **double** типу **int**. Возможность обнаружения ошибок в период компиляции программы защищает от возникновения ошибок во время выполнения программы. Благодаря безопасности типов становится труднее злонамеренному коду вставить один тип в другой с целью неправильного управления памятью. Система ***CLR*** также проверяет безопасность типов программной сборки с целью выявления ошибок, которые не были обнаружены на стадии компиляции.

---

[1] Примечание: см. по этому вопросу также раздел 8 (стр. 27) в Пособии к практическим занятиям №1

Другой важной особенностью общеязыковой среды исполнения ***CLR***, является то, как она загружает исполняемый код. Понимание этого процесса позволит разобраться с безопасностью типов и защитой. Это также позволит изнутри взглянуть на проблемы быстродействия и управление памятью. На рисунке показано как ***CLR*** управляет жизненным циклом приложения.

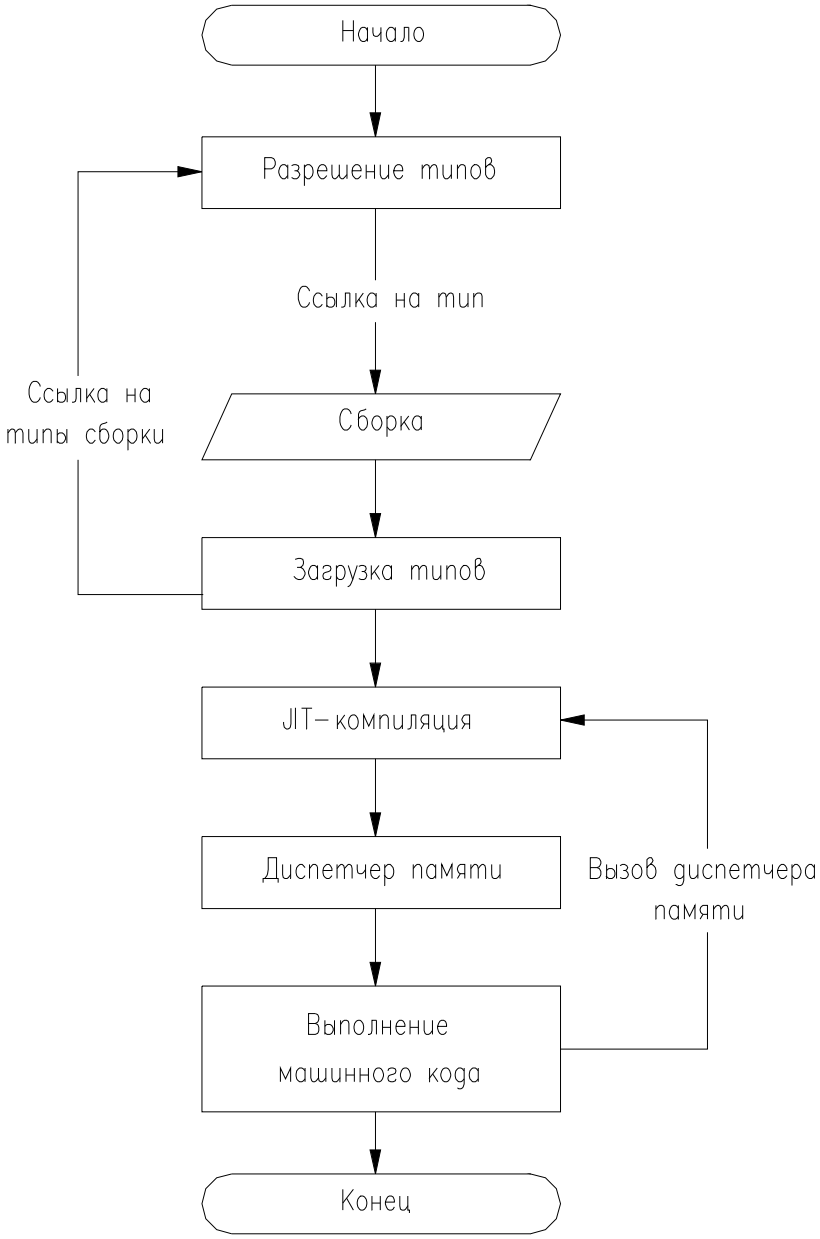

Рис. Алгоритм CLR управления исполнением приложения

Когда начинается исполнение программы **.NET**, **Windows** определяет, что это программная сборка **.NET**, и запускает ***CLR***. Затем ***CLR*** находит точку входа в программу и начинается процесс разрешения типов (**Resolve Types**), в котором определяются места расположения типов, используемых в программной сборке. Отождествленные программные сборки загружаются загрузчиком процесса (**Loader**).

Описанная схема объясняет, почему **компилятор .NET называется JIT (Just-In-Time, — оперативным) компилятором** — *типы загружаются по мере возникновения потребности в них, т.е. оперативно*. Процесс верификации обеспечивает безопасность типов и защиту программного кода от злонамеренного вмешательства. Когда **JIT**-компилятор встречает нужный ему тип, он обращается к процессу разрешения типов с целью поиска и загрузки необходимой программной сборки, для того чтобы продолжить свою работу.

Первоначальная **JIT**-компиляция при запуске приложения выполняется медленно. Дальнейшая производительность загрузки будет меняться в зависимости от выбираемого типа. Как уже отмечалось, это только начальные задержки, и производительность будет значительно увеличиваться после того, как тип присоединяется к работающему в памяти набору программных компонентов.

После компиляции типов в машинный код ими управляет диспетчер памяти. Типы-значения (структуры) размещаются в стеке, а типы-ссылки (классы) размещаются в динамически распределяемой области памяти. Периодически активируется процесс сборки мусора, заключающийся в удалении неиспользуемых объектов из памяти. Сборщик мусора удаляет сначала последние из созданных неиспользуемых объектов, в затем более старые. В целях оптимизации сборка мусора выполняется в среднем не чаще, чем возникают ошибки страницы памяти.

Программный код в памяти является машинным кодом, скомпилированным **JIT**-компилятором, который может непосредственно выполняться процессором компьютера. Работающий в памяти код остается там до тех пор, пока нехватка памяти не активизирует процесс сборки мусора. Оптимизация диспетчера памяти, связанная с учетом поколений программных компонентов, предотвращает возникновение ошибок из-за отсутствия нужной страницы памяти, что приводит к увеличению скорости выполнения приложения.

Когда процессору требуется исполнить тип, отсутствующий в оперативной памяти, генерируется запрос на его получение от **JIT**-компилятора. С целью оптимизации циклы исполнения, **JIT**-компиляции и разрешения типов минимизированы. Важно понимать, что именно эти циклы приводят к снижению скорости выполнения приложения. Для небольшой программы это не является проблемой, поскольку все ее компоненты легко умещаются в памяти. В случае больших программ необходимо уничтожать неиспользуемые объекты, как только они становятся ненужными. Это позволяет избежать нехватки памяти и хранить в памяти первоначально работающий набор программных компонентов.

CLR отвечает за все, что происходит в платформе .NET. От понимания того, как **CLR** работает, зависит качество разработки эффективно исполняющихся программ.

1.5. Языки программирования

Другим важным аспектом платформы **.NET** является *поддержка разных языков программирования*. Этой цели служит использование промежуточного языка (**IL**). В настоящее время существуют десятки языков программирования, которые работают с CLR посредством их компиляция в **IL**. **Кроме языка программирования C#** платформа .NET поставляется с такими языками**, как Visual Basic.NET, JScript.NET, J#.NET и Managed C++. Другие поставщики включают Borland Delphi.NET, Fujitsu COBOL.NET; Python.NET, Perl.NET** и некоторые другие языки программирования**.**

Система, которая позволяет одновременно использовать эти языки программирования, называется системой общих типов — **CTS** (**Common Type System**). Хотя в каждом из языков программирования типы представляются особым способом, базовое поведение и семантика каждого типа для **CLR** остается одинаковой. Система общих типов определяет, какие члены может иметь тип: поля, методы, события, свойства и индексаторы. Она также определяет их

области действия и видимости: **public** (общедоступный, открытый), **internal** (внутренний), *protected* (защищенный), **protected internal** (внутренний защищенный) и **private** (закрытый). Конечно, приведенное в предыдущей фразе описание дано с перспективой дальнейшего использования ключевых слов C#. В других языках программирования могут использоваться иные ключевые слова, но их семантика в **CLR** остается неизменной.

Что дает поддержка различных языков программирования? Чтобы понять это рассмотрим большое предприятие с несколькими программными проектами. Часто оказывается, что в этих проектах используются различные языки программирования. До тех пор пока программисты используют компонентную технологию, такую, как COM, они, вероятно, не смогут повторно использовать коды, перенося их из одного проекта в другой. Наличие нескольких языков программирования, поддерживающих одну платформу, дает больше возможностей для повторного использования кодов, что является выгодным во всех отношениях.

Возможность создания программ на различных языках также полезна при создании общего проекта разными предприятиями. Рынок продуктов независимых разработчиков программного обеспечения широк, и многие компании каждый день присоединяются к использованию технологии **.NET**. Они могут создавать свои приложения **.NET** на разных языках программирования, поддерживающих технологию **.NET**. Разрабатываемые ими программные компоненты могут повторно использоваться в других приложениях **.NET**, независимо от языка, на котором они написаны. Поддержка разных языков программирования в технологии .NET открывает новые рынки для производителей программных компонентов, что в прошлом оказывалось труднодостижимым.

Преимущества поддержки различных языков программирования

Поддержка разных языков программирования в технологии **.NET** помогает использовать преимущества каждого языка программирования. Например, группа разработчиков, программирующая несколько лет на языке **COBOL**, может быстро получить преимущества использования технологии **.NET**. Кроме того, они могут использовать библиотеки, написанные на других языках, без знания этих языков. Речь, конечно, не идет об использовании нескольких языков программирования при работе над одним проектом, поскольку это приведет к разрушению окружения, и сделает программу трудной в сопровождении (для внесения изменений). Имеется в виду следующее. Если разработчики проекта **A** создали повторно используемые компоненты (**DLL**) на C#, разработчики проекта **B** на языке COBOL могут использовать эти компоненты, не обращая внимание на то, на каком языке они были написаны. Поддержка разных языков программирования обеспечивает более легкий путь освоения технологии **.NET** программистами, ранее ее не использовавшими.

Конечно, совместимость между разными языками программирования не является абсолютной или автоматической. Разные языки имеют свои уникальные возможности, которые не поддерживаются в других языках программирования. Например, приложение **VB.NET** не будет работать с C# **DLL** с открытым методом, использующим типы указателей.

Для решения этих проблем была разработана спецификация общеязыковой среды исполнения **CLS** (**Common Language Specification**). По большей части в **CLS** определяются минимальные требования к языкам программирования, которые должны взаимодействовать между собой, в части того, что они могут экспонировать. Например, в C# нельзя экспонировать указатели и неподписанные типы, если хотеть, чтобы код был совместимым с **CLS**. Библиотеки, которые вы намерены сделать совместимыми с **CLS**, могут использовать несовместимые с **CLS** функции только для внутреннего употребления, но не в открытом интерфейсе.

2. Общие сведения об интегрированной среде разработки приложений C#Builder

C#Builder является инструментом создания приложений **.NET**. Знакомство с работой **CLR** и многочисленными повторно используемыми классами библиотеки **BCL** поможет создавать хорошо спроектированные и эффективные приложения на языке C#. В [**2**] подробно описывается, как создавать приложения **.NET** с помощью интегрированной среды разработки приложений **C#Builder**.

C#Builder – это Интегрированная Среда Разработки (**IDE** – англ. абб.) приложений **.NET** на языке программирования C#. Она состоит из средств разработки графического интерфейса, включает редакторы, мастеров, отладчик и поддержку исполнения приложения.

Microsoft Visual Studio .NET – это полнофункциональная Интегрированная Среда Разработки приложений **.NET** на языках программирования C#, C++, J#, Visual Basic и др. Она состоит из средств разработки графического интерфейса, включает редакторы, мастеров, отладчик и поддержку исполнения приложения и др. В [**3**] детально описывается среда **MS VS .NET 2003**. Будущее и настоящее программирования связано именно с этой средой.

Список литературы

1. Микелсен Клаус. Язык программирования C#. Лекции и упражнения. Учебник: пер. с англ./ Клаус Микелсен –СПб.: ООО «ДиаСофтЮП», 2002. – 656 с.

2. Джо Майо. C#Builder. Быстрый старт. Пер. с англ. – М.: ООО «Бином-Пресс», 2005 г. – 384 с.

3. Основы Microsoft Visual Studio .NET 2003 / Пер. с англ. - М.: Издательско-торговый дом «Русская Редакция», 2003. – 464 с. Брайан Джонсон, Крэйт Скибо, Марк Янг.

4. Герберт Шилдт. Полный справочник по C# . / Пер. с англ./ Издательство: Вильямс, 2004 г. 752 с.

5. Чарльз Петцольд Программирование в тональности C# / Пер. с англ. Издательство: Русская Редакция, 2004 г. - 512 стр.

6. Мэтт Вайсфельд. Объектно-ориентированный подход**:** Java**, .**Net**,** C++ . Второе издание / Пер. с англ. - М: КУДИЦ-ОБРАЗ, 2005. - 336 с.

Что значит освоить объектно-ориентированное программирование? Для этого недостаточно выучить синтаксис языка C#, Java или C++. Нужно разобраться в принципиальных положениях объектного подхода, понять, чем он отличается от других. И предлагаемая книга будет в этом отличным помощником. В ней на конкретных примерах разбираются все основные понятия объектно-ориентированного подхода. Советую прочесть эту книгу [**6**].

C# & .NET по шагам (Web-ресурс)

1 | 2 | 3 | 4

Шаг 1 - Разработка приложений в .NET (основы). (24.09.2001 - 2.3 Kb)
Шаг 2 - Как будет распространяться приложение (основы). (24.09.2001 - 3.8 Kb)
Шаг 3 - Нам нужен .Net Framework SDK. (24.09.2001 - 3.8 Kb)
Шаг 4 - Hello Word C#. (25.09.2001 - 2.4 Kb)
Шаг 5 - Hello Word VB. (25.09.2001 - 1.7 Kb)
Шаг 6 - Hello Word VC++. (25.09.2001 - 1.6 Kb)
Шаг 7 - Пространство имен. (26.09.2001 - 2.7 Kb)
Шаг 8 - Net ассемблер и дизассемблер. (26.09.2001 - 3.5 Kb)
Шаг 9 - Просмотр класса в EXE проекте ILDasm.exe. (26.09.2001 - 1.6 Kb)
Шаг 10 - Две основы Net. (27.09.2001 - 2 Kb)
Шаг 11 - Отладка. (27.09.2001 - 33 Kb)
Шаг 12 - ADO.NET (27.09.2001 - 10 Kb)
Шаг 13 - Попробуем OLEDB. (27.09.2001 - 6 Kb)
Шаг 14 - Типы данных - системные и языка программирования. (28.09.2001 - 3 Kb)
Шаг 15 - Windows Form. (28.09.2001 - 7 Kb)
Шаг 16 - Где взять редактор C#. (28.09.2001 - 21 Kb)
Шаг 17 - Избавляемся от консольного окна. (28.09.2001 - 9 Kb)
Шаг 18 - Создаем окно. (28.09.2001 - 6 Kb)
Шаг 19 - Добавляем меню. (28.09.2001 - 6 Kb)
Шаг 20 - Свойства (properties). (28.09.2001 - 3 Kb)
Шаг 21 - Обработка событий на форме. (30.09.2001 - 5 Kb)
Шаг 22 - Изменение размера формы. (30.09.2001 - 2 Kb)
Шаг 23 - Изменение положения формы. (30.09.2001 - 2 Kb)
Шаг 24 - Override. (30.09.2001 - 2 Kb)
Шаг 25 - Встраиваем элемент управления в окно. (30.09.2001 - 5 Kb)
Шаг 26 - Обработка сообщений элемента классом элемента. (30.09.2001 - 6 Kb)
Шаг 27 - Еще один редактор C#. (30.09.2001 - 30 Kb)
Шаг 28 - Создание меню подробнее. (01.10.2001 - 6 Kb)
Шаг 29 - Одномерные Массивы. (01.10.2001 - 3 Kb)
Шаг 30 - foreach. (01.10.2001 - 2 Kb)
Шаг 31 - Интерфейсы. (01.10.2001 - 3 Kb)
Шаг 32 - Коллекции. (01.10.2001 - 6 Kb)
Шаг 33 - Создаем обработчик событий меню. (01.10.2001 - 6 Kb)
Шаг 34 - Сохраняем данные в файл. (01.10.2001 - 7 Kb)
Шаг 35 - Добавляем строку состояния. (02.10.2001 - 5 Kb)
Шаг 36 - Панели на строке состояния. (02.10.2001 - 6 Kb)
Шаг 37 - Икона формы. (02.10.2001 - 9 Kb)
Шаг 38 - Диалог открытия файлов. (02.10.2001 - 14 Kb)
Шаг 39 - Отображаем картинку. (02.10.2001 - 12 Kb)
Шаг 40 - Создаем панель инструментов. (02.10.2001 - 6 Kb)
Шаг 41 - Net Classes первые вывод. (02.10.2001 - 6 Kb)
Шаг 42 - XML документация кода. (02.10.2001 - 6 Kb)
Шаг 43 - XML notepad. (02.10.2001 - 16 Kb)
Шаг 44 - Заголовок формы и пункт меню выход. (03.10.2001 - 4 Kb)
Шаг 45 - Создаем файл с ресурсами строк. (03.10.2001 - 5 Kb)

……..

1 | 2 | 3 | 4

Приложение 2

Microsoft .NET Framework SDK	Web-ресурс
Welcome to the Microsoft .NET Framework Software Development Kit. The .NET Framework SDK is the essential reference for developers who use the .NET Framework technologies.	
Getting Started For those who are new to the .NET Framework technologies, the Getting Started with the .NET Framework section of the .NET Framework SDK documentation is designed to point you in the right direction.	**QuickStarts, Tutorials and Samples** The .NET Framework SDK QuickStarts, tutorials, and samples are designed to quickly acquaint you with the programming model, architecture, and components that comprise the .NET Framework.
Documentation	**Tools and Debuggers**
The .NET Framework SDK documentation provides a wide range of overviews, programming tasks, and class library reference information that is designed to help you build efficient, powerful, and scalable applications based on the .NET Framework technologies.	The .NET Framework SDK tools and debuggers enable you to create, deploy, and manage applications and components that target the .NET Framework.
The Tool Developer's Guide provides useful information for developers wanting to build low-level development tools that operate within the .NET Framework, such as compilers, browsers, profilers, and debuggers.	We are working hard to make sure that the .NET Framework technologies and the SDK enable you to build great applications. Your feedback is extremely valuable to us. Please send us your comments and suggestions. You can also visit the .NET Framework SDK newsgroups.
For a list of the modifications to the class library for the .NET Framework version 1.1, see the Class Library Changes and Additions.	For known issues and late-breaking information see the Release Notes.
Note: To run a .NET Framework application, the .NET Framework must be present or installed by the application setup. To run applications that use Visual J#, the Visual J# .NET Redistributable Package 1.1 must be installed in addition to the .NET Framework. See the redistributable readme file for more information.	