Государственный университет – Высшая школа экономики

Факультет Бизнес-Информатики

Кафедра Основ информатики
и прикладного программного обеспечения

C#

Объектно-ориентированный язык программирования

Пособие к практическим занятиям - №1

Проф. Забудский Е.И.

Москва 2005

Тема 1. Объектно-ориентированный подход к разработке программного обеспечения

Два практических занятия
(4 часа)

Первые два занятия посвящены обучению работе с платформой .NET и изучению объектно-ориентированного языка C# на простых программах.

Сопоставляются две парадигмы компьютерной науки: процедурно-ориентированное программирование и объектно-ориентированное программирование

Подробно анализируются структурные элементы C#-программ: предложения языка, ключевые слова и идентификаторы, операторы, символы, комментарии, а также понятия, связанные с парадигмой объектно-ориентированного анализа и программирования: классы, пространства имен, объекты, методы, объявления и вызовы методов, типы и др.

Наряду с созданием примеров консольных приложений также рассматривается C#-программа, реализующая графический интерфейс на основе WinForms: демонстрируются результаты выполнения.

// **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).

Данное пособие распространяется свободно. Изложенный материал, предназначенный для публикации в “твердом” варианте, дополнен и откорректирован.

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.

© Забудский Е.И., 2005

Содержание

1. Две парадигмы компьютерной науки 4

1.1. Процедурно-ориентированное программирование и присущие ему проблемы (рис.1.1) 4

1.2. Объектно-ориентированное программирование и его преимущества (рис.1.2) 4

2. Терминология ООП. Первые шаги при разработке программы в ООП (рис.1.3)...... 7

3. Несколько слов о компонентно-ориентированном программировании 8

4. Настало время сказать о тех языках программирования, которые являются объектно-ориентированными (рис.1.4) 10

5. О программных продуктах фирмы Microsoft, предназначенных для реализации парадигмы ООП 10

5.1. Платформа .NET. Интегрированная среда C#Builder. 12

5.2. Среда Visual Studio .NET. C#-программа, реализующая графический интерфейс: *демонстрация результатов выполнения* 12

6. Настало время написать программу на C# 12

6.1. Семь шагов создания программы на C#. bat-файл запуска из командной строки ... 12

6.2. Каркас простейшей C#-программы. Пояснение семантики 13

6.3. C#-программа №1 – выводит на экран строки У. Шекспира, которые соответствуют тому, что чувствуют многие люди, когда начинают изучать ЯП 14

7. C#-программа №2 – интерактивная программа дружественная пользователю 14

7.1. Псевдокод C#-программы №2. Пространство имен.. 15

7.2. Основные элементы C#-программы №2 18

7.2.1. Комментарии 18

7.2.2. Определение класса 18

7.2.3. Идентификаторы (имена) 19

7.2.4. Фигурные скобки и блоки исходного кода (рис.1.5) 19

7.2.5. Метод Main() и его определение (рис.1.6) 20

7.2.6. Переменные: представление данных, хранимых в памяти компьютера (рис.1.7) ... 22

7.2.7. Запуск методов .NET-платформы 23

7.2.8. Общий механизм вызова метода 24

7.2.9. Присваивание значения переменной 25

7.2.10. Ветвление посредством оператора if 26

7.2.11. Завершение метода Main() и класса Hello 26

8. Компиляция в .NET исходного кода C# (рис.1.8) 27

Контрольные вопросы 29

Упражнения по программированию 30

Список литературы 31

П р и л о ж е н и я

1. C# & .NET по шагам: http://www.firststeps.ru/dotnet/dotnet1.html 32

2. Зарезервированные слова языка C# 33

1. Две ***парадигмы*** компьютерной науки:

1) процедурно-ориентированное программирование (модульное);

2) объектно-ориентированное программирование;
компонентно-ориентированное программирование.

// Paradeigma (греч.) – пример, образец.
Парадигма – *исходная концептуальная схема*, модель постановки проблемы и ее решения.

1.1. Процедурно-ориентированное программирование и присущие ему проблемы (рис. 1.1)

Одним из традиционных подходов к проектированию компьютерных программ был старомодный, процедурно-ориентированный стиль. ***В нем наивысший приоритет принадлежит действиям,*** которые выполняет элемент программы*, **в то время как** данные **,** с которыми программа работает, остаются как бы* на втором плане.

В процедурно-ориентированных программах **доступ к данным был возможен из любой части программы** (как это показано на рис. 1.1), а операции над ними — посредством любой инструкции.

Этот подход хорош для малых проектов, но при создании больших и сложных программ разработчику приходится отслеживать все возможные ветвления в коде программы, поскольку все части ее более или менее взаимосвязаны. Более того, ***различные фрагменты кода программы могут получить*** *доступ к одному и тому же значению данных* ***— причем они могут*** не только читать, но и изменять его. При разработке одной части программы можно и не знать о том, что данные, с которыми она работает, могут быть изменены другими частями программы. Ситуация очень быстро приближается к **полному хаосу**, когда над проектом совместно работают несколько программистов (**рис.1.1 выглядит вполне хаотическим**).

1.2. Объектно-ориентированное программирование и его преимущества (рис. 1.2)

Как же преодолеть эту проблему? Типичный человеческий подход к преодолению сложной проблемы — разбить ее на более простые части. Попробуем разбить предыдущий пример на четыре меньших, самостоятельных фрагмента. Результат показан на рис. 1.2. Здесь весь набор инструкций разделен на четыре **отдельных модуля**. В объектно-ориентированном мире такие **модули называются объектами**. Данные также разделены на четыре части, *так что каждый объект содержит лишь те данные, с которыми он работает*. Теперь при выполнении программы эти четыре объекта взаимодействуют, пересылая друг другу сообщения, активизируя инструкции и обмениваясь данными. Это не только значительно снизило сложность программы, но и позволило создать четыре самодостаточных модуля, каждый из которых может быть "извлечен" из программы и "вставлен" в нее снова. Теперь не так уж сложно добиться, чтобы модули создавались и поддерживались разными программистами.

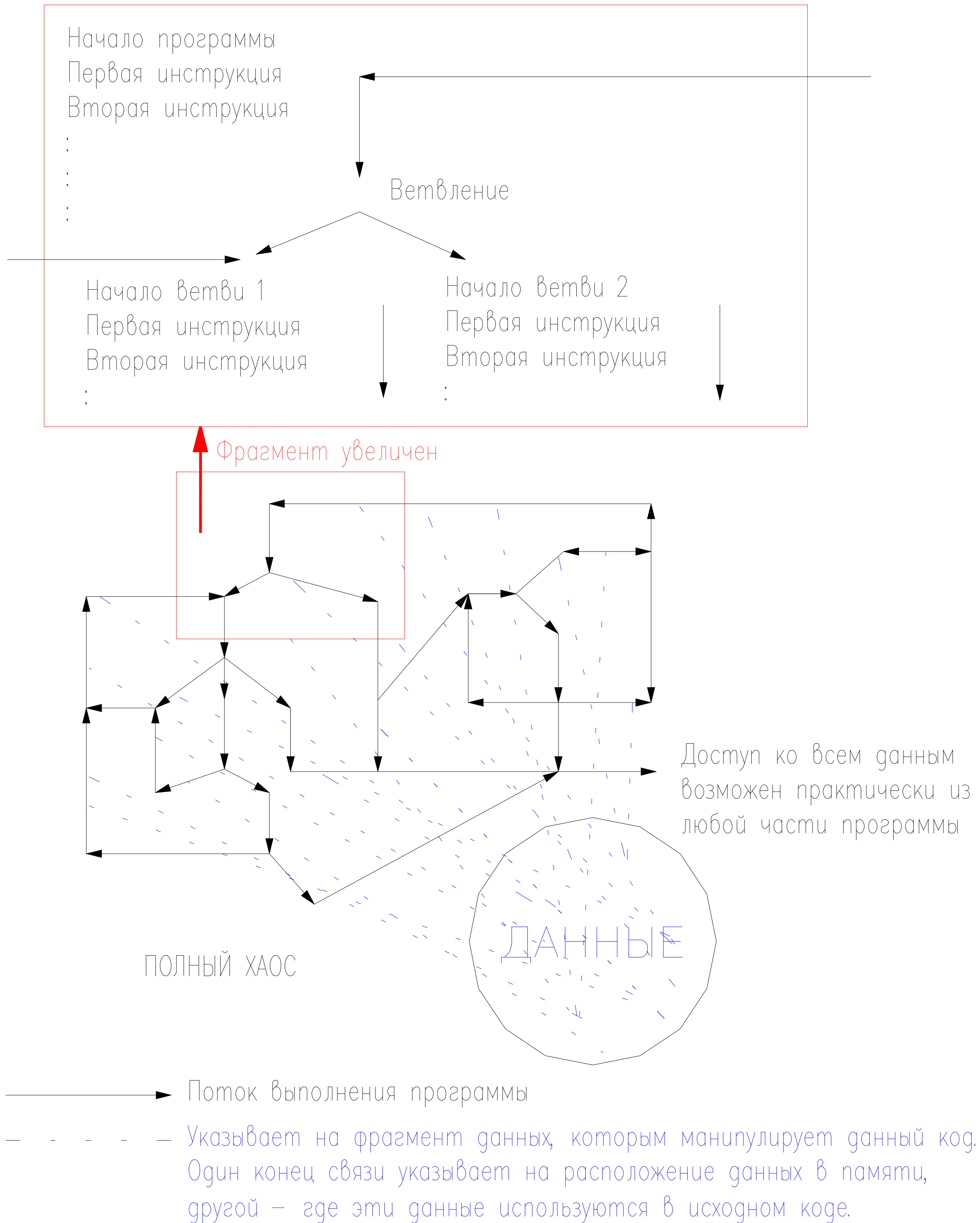
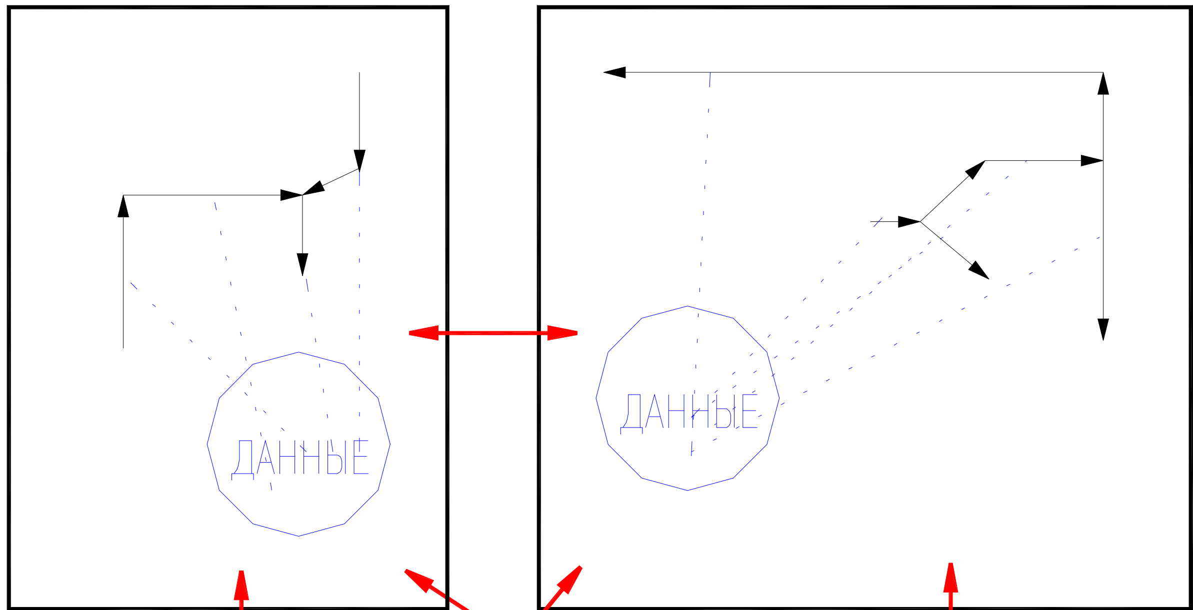
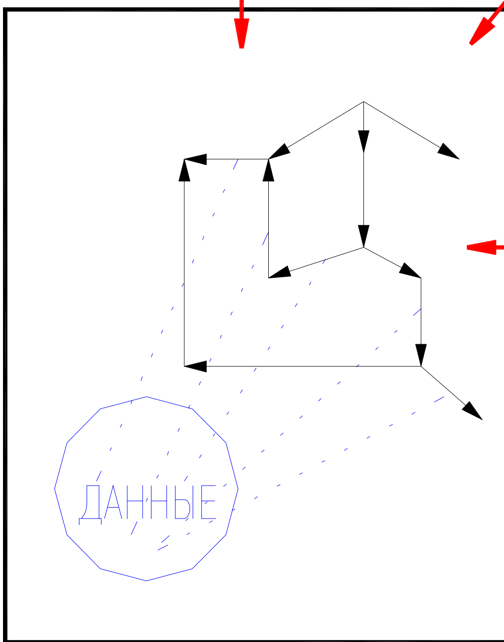


Рис. 1.1. Процедурно-ориентированная программа

Модуль (или обьект) A

Модуль (или обьект) B





ДАННЫЕ

Модуль (или обьект) C

Модуль (или обьект) D

УКАЗЫВАЕТ НА ОБРАБАТЫВАЕМЫЙ ЭЛЕМЕНТ ДАННЫХ

ПОТОК ВЫПОЛНЕНИЯ

СООБЩЕНИЯ, ПОСЫЛАЕМЫЕ МЕЖДУ ОБЪЕКТАМИ

Рис. 1.2. Разделение процедурно-ориентированной программы на самодостаточные модули

2. Терминология ООП:

1) **объект** - относится к **модулю**;
2) **метод** – это по существу программа-функция, или программа-процедура. Метод – это действие, которое способен выполнить объект;
3) **переменные экземпляра** - являются частью данных объекта.

ООП **— это методология, в которой при моделировании конкретных ситуаций используются объекты.**

В повседневной жизни нас окружают объекты: *книги, дома, машины, собаки, люди, столы, пауки, тарелки, чашки...*

Часто объекты способны принимать участие в определенных действиях. **Автомобиль, например, способен** *"открывать дверь", "закрывать дверь", "запускать двигатель", "двигаться вперед", "набирать скорость", "разворачиваться" и "тормозить".*

Каждый объект взаимодействует со своим окружением и влияет на другие объекты.

Примерами подобных взаимодействий могут быть:

1) *объект "человек",* подходящий к *объекту "автомобиль"* и открывающий его дверь, или

2) *объект "автомобиль"*, содержащий *объект "человек"* и транспортирующий его из точки А в точку Б.

При написании программы с участием автомобилей и людей с использованием *процедурно-ориентированной методологии* все **внимание будет уделено *действиям*** (*"открыть дверь", "закрыть дверь"* и т.д.).

С другой стороны, при использовании объектно-ориентированного программирования (ООП*)* ***программа будет строиться вокруг объектов*** **"автомобиль" и "человек".**

// **НОВЫЙ ТЕРМИН *КОМПЬЮТЕРНОЕ* МОДЕЛИРОВАНИЕ**

Компьютерное моделирование пытается имитировать процессы, происходящие в *реально* существующей или *теоретической* системе. Собирая и анализируя данные, полученные этой искусственной системе, можно получить ценные сведения о внутренних особенностях системы реальной. Для успешного моделирования необходимо разработать модель и реализовать ее в компьютере. Для этого созданы объектно-ориентированные языки программирования и специальная среда программирования. О них далее.

Чтобы заполнить брешь между *реальным миром* и *моделированием его на компьютере*, необходимо, прежде всего, *идентифицировать участвующие в данном процессе объекты*. Обычно, это один из первых шагов при разработке программы в ООП.

Попытаемся представить лифтовую систему в действии:

1-й шаг. КАКИЕ ОБЪЕКТЫ В ЭТОЙ СИСТЕМЕ УЧАСТВУЮТ? Примите во внимание, что ***объекты прямо соответствуют существительным.*** Реальные объекты далее выделены полужирным.

Несколько **лифтов** расположено в **здании** с десятью **этажами**. Каждый лифт имеет доступ ко всем десяти этажам. Когда **человек** желает вызвать лифт, ему необходимо нажать **кнопку** на этаже, где он находится в данный момент, и т. д.

2-й шаг. Каждый *объект* (модуль) выполняет определенные *функции* (методы) и имеет определенные данные (т. е. характеристики или атрибуты)

Например, объект "Лифт" может иметь следующие данные: *"максимальная скорость", "текущее местоположение", "текущая скорость", "максимальное число людей, которое может вместить лифт", и т.д.*

В ООП данные также называются переменные экземпляра. То есть переменные экземпляра эквивалентны данным

Функции объекта "Лифт" могут быть следующими: *"подниматься", "опускаться", "остановиться" и "открыть дверь".* **Определив правильные атрибуты и функции каждого объекта в реальном мире, их можно представить в компьютерной модели.**

Итак -

1 *Поведение объекта* реального мира представлено в объекте ОО-программы в форме *методов*. Заметьте, что методы (действия) соответствуют **глаголам**.

2. Методы объекта выполняют действия с переменными экземпляра этого же объекта.

// **ПРИМЕЧАНИЕ**

Объект в объектно-ориентированной программе не обязательно представляет объект **физический**. Он с таким же успехом может представлять и **концептуальный** объект. Примеры концептуальных объектов — *праздник, компьютерный курс, путешествие* и т.д.

Расширим запас терминологии ООП

Еще один важный термин ООП — класс. *Класс определяет общие черты (переменные экземпляра и методы) группы подобных друг другу объектов.* Следовательно, *все объекты одного класса имеют те же переменные экземпляра и методы.* Какие переменные экземпляра и методы включать в создаваемый класс, программист выбирает сам; все зависит от потребностей создаваемой программы.

Проиллюстрируем значение класса примером. Рассмотрим автомобиль с концептуальной точки зрения. В реальной жизни люди видят, трогают и водят конкретные автомобили. Примерами могут быть стоящий на стоянке голубой VOLVO, максимальная скорость которого 100 миль в час, или черный BMW с максимальной скоростью 150 миль в час. Оба этих реально существующих, осязаемых автомобиля можно считать объектами. **(рис. 1.3 – Класс служит шаблоном для своих объектов).**

Концепции объекта и класса представлены на столь ранней стадии обсуждения, чтобы студент привык к идее ООП и подготовился к более подробному и практическому представлению этих тем.

3. Несколько слов о компонентно-ориентированном программировании

Оно связано с повторным использованием программного кода, который написан опытными программистами, z.B. фирмы Microsoft. Существенным достоинством ООП является **повторное** использование программного кода. Оказывается, что класс, то есть шаблон для своих объектов, оказался очень хорошим способом повторно использовать код.

Программа, при создании которой была предусмотрена возможность ее повторного использования, называется **компонентом** (**программным компонентом**).

Класс Автомобиль

Переменные экземпляра:

Марка	Заполнить поле
Текущее местоположение	Заполнить поле
Максимальная скорость	Заполнить поле
Текущая скорость	Заполнить поле

Методы:
Открыть дверь
Закрыть дверь
Двигаться вперед
Двигаться назад
Ускоряться
Тормозить
Поворачиваться

Переменные экземпляра класса Автомобиль выполняют роль незаполненных ШАБЛОНОВ...

...для каждого объекта Автомобиль, в котором эти переменные заполнены уникальными значениями, в то время как...

Объект Volvo класса Автомобиль

Переменные экземпляра:

Марка	Volvo
Текущее местоположение	Стоянка
Максимальная скорость	100 миль/час
Текущая скорость	0 миль/час

Методы:
Открыть дверь
Закрыть дверь
Двигаться вперед
Двигаться назад
Ускоряться
Тормозить
Поворачиваться

Объект BMW класса Автомобиль

Переменные экземпляра:

Марка	BMW
Текущее местоположение	Соседский гараж
Максимальная скорость	150 миль/час
Текущая скорость	0 миль/час

Методы:
Открыть дверь
Закрыть дверь
Двигаться вперед
Двигаться назад
Ускоряться
Тормозить
Поворачиваться

...методы, указанные в классе Автомобиль, идентичны во всех объектах Автомобиль

Рис. 1.3. Класс служит шаблоном для своих объектов

4. Настало время сказать о тех языках программирования, которые являются объектно-ориентированными

Рассмотрим генеалогическое дерево основных языков высокого уровня – **рис. 1.4**. (genealogia - греч. – родословная)

К основным объектно-ориентированным языкам программирования относятся: **C#** (**C Sharp**) – это последнее достижение Microsoft, **Java**, **C++**, **SmallTalk**, **Visual Basic**, **Simula** и некоторые др. Язык C# наиболее соответствует парадигме ООП. Именно его мы будем изучать в контексте ООАиП.

Sharp (*англ.*) – отточенный.

- нотный знак, обозначающий повышение звука

Язык C#, вышедший из C и C++, **был создан для программистов корпоративных приложений**. C# - это современный, простой, объектно-ориентированный и безопасный при приведении типов язык. Он многое унаследовал от C и C++, но в определенных областях, таких как пространства имен, классы, методы и обработка исключений, он существенно отличается от этих языков.

Язык C# предоставляет в ваше распоряжение множество удобных средств, таких как сборка мусора, обеспечение безопасности типов, поддержка контроля версий и др. По умолчанию код выполняется, в безопасном режиме в котором запрещено применение указателей.

5. О программных продуктах фирмы Microsoft, предназначенных для реализации парадигмы ООП

Несколько лет назад Microsoft разработала с этой целью

платформу .NET (произносится дот нэт)

Эта платформа, написана на языке C#. Комплекс предназначен для разработки различных типов программ: Web-серверы, оконные программы, консольные приложения и др.

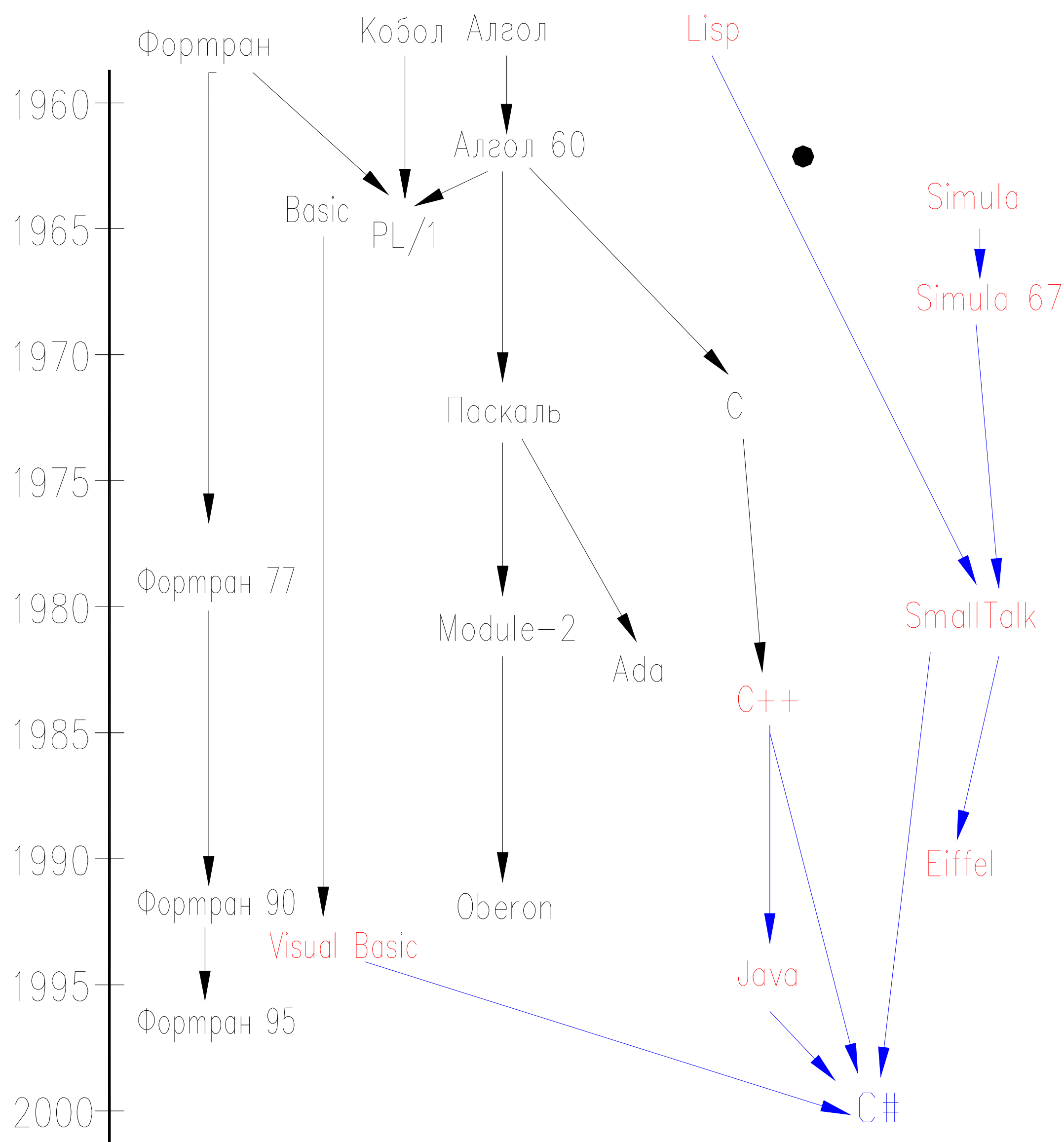
.

1. В состав **.NET-**платформы входит библиотека классов, содержащая массу готовых программных компонентов, которые можно использовать в собственных программах. Она экономит немало времени, так как программист может воспользоваться готовыми фрагментами кода. Фактически, он повторно использует код, созданный и тщательно проверенный профессиональными программистами Microsoft. Эта библиотека классов называется **.NET Framework** или *библиотека базовых классов* (**Base Class Library, BCL**).

2. **.NET** обеспечивает перекрестное взаимодействие программ, написанных на разных языках. Любой язык, поддерживаемый **.NET**, может взаимодействовать с другими языками этой платформы. На платформу **.NET** было перенесено около 15 языков.

Для исполнения кода, написанного на любом из поддерживающих платформу **.NET** языков, используется одна и та же среда исполнения, ее часто называют *единой средой исполнения* (***Common Language Runtime, CLR***).

При создании платформы **.NET** реализованы многие современные программные технологии. *О достоинствах платформы* **.NET** *можно много говорить* [1]*, но мы будем ей пользоваться и по ходу изложения материала характеризовать.*



● Данная стрелка указывает на то, что механизм сборки мусора (впервые примененный в LISP) был в дальнейшем использован во многих языках, таких как SmallTalk, Eiffel, Java, Visual Basic и C#.

Рис. 1.4. Генеалогическое дерево наиболее известных языков высокого уровня

Прекрасной иллюстрацией возможностей **.NET** и библиотеки **.NET Framework** является поддержка оконных графических интерфейсов пользователя (на занятии демонстрирует-ся результат выполнения C#-программы, реализующей графический интерфейс).

Таким образом, на ПК должна быть установлена платформа **.NET**.

Microsoft представляет платформу в двух вариантах:

5.1. **Платформа .NET** 1-й вариант - устанавливаются два файла:

1) файл dotnetfx.exe 23,7Mb - Microsoft .NET Framework v1.1 Redistributable
этот файл инсталлируется *первым*

2) файл setup.exe 108,8 Мб .NET SDK v1.1
этот файл инсталлируется *вторым*
SDK – Software Development Kit – набор для разработки ПО.

.NET распространяется Microsoft **бесплатно**.
.NET можно скачать с Web-сайта http://msdn.microsoft.com/downloads .

В первом варианте есть все, что необходимо для разработки ООП. Нужен толь-ко редактор, z.B. модифицированный Notepad. Желательны нумерация строк и под-светка операторов C#.

Фирма **Borland** приобрела лицензию на право использования **.NET и разрабо-тала интегрированную среду C#Builder** [2]**.** Первая версия среды C#Builder (впол-не достаточная) также распространяется **бесплатно –**.
файл csb10_per_noncommercial.exe 28,9 Мбайта**. Скачивается с ftp –**

ftp://ftpd.borland.com/download/csharpbuilder/csb1/csb10_per_noncommercial.exe .

5.2. Среда VISUAL STUDIO .NET 2-й вариант. Распространяется Microsoft на ком-мерческой основе. Эта разработка включает .NET, а также интегрированную среду про-граммирования, разработанную Microsoft и др.[3].

C#-программы можно запускать на выполнение: 1) из среды VISUAL STUDIO .NET, 2) из среды Borland (**C#Builder**), 3) из командной строки.

Отметим, что имя файла с C#-программой имеет расширение **.cs** .

6. Настало время написать программу на C#

6.1. Семь шагов создания программы на C#

После того как успешно установлена среда VISUAL STUDIO .NET, все готово к тому, чтобы писать и выполнять программы на C#. Создать программу на C# можно за семь ша-гов:

1. Загрузить среду VISUAL STUDIO .NET.

2. Осуществить набор и редактирование исходного кода на C#; сохранить текст про-граммы.

3. Использовать компилятор языка C# для преобразования этого исходного кода в PE-файл (**.exe**) или DLL (**.dll**).

PE – Portable Executable – исполнимый exe-файл.

DLL – Dynamic Link Library – файл непосредственно **не** выполняется. Это компо-нент, предназначенный исключительно для повторного использования в составе какого-либо

приложения. Опция компилятора для формирования dll-файла будет приведена на одном из следующих занятий.

4. Если компилятор обнаруживает синтаксические ошибки на шаге 3, перейти к шагу 2 и исправить их. Далее вновь перейти к шагам 3, 4 и т.д.

5. Выполнить программу.

6. Проверить соответствует ли ее поведение ожидаемому. Если нет вернуться к шагу 2 и исправить ошибки в коде C#.

7. Отпраздновать написание первой программы на C#.

П р и м е ч а н и е: при компиляции C#-программы (создание exe-файла) из командной строки целесообразно воспользоваться следующим bat-файлом (его имя, z.B. – **1.bat**)

Текст bat-файла: 1.bat

```
@echo off
C:\WINDOWS\MICROSOFT.NET\FRAMEWORK\V1.1.4322\CSC.EXE %1.cs
echo off
```

6.2. Каркас простейшей C#-программы

```
1:  using System;
2:  class имя_класса
3:  {
4:              static void Main ()
5:              {
            ............
             предложения  на языке C#
            .............
10:             }
11: }
```

Пояснение семантики:

1: using System; - объявление об использовании классов, принадлежащих разделу System в библиотеке классов .NET Framework.

2: class имя_класса - начало определения класса имя_класса

3: { - начало блока класса имя_класса

4: static void Main () - Main указывает откуда начинается выполнение программы

5: { - *начало блока Main*

....

10: } - *конец блока Main*

11: } - конец блока класса имя_класса

6.3. C#-программа №1 – выводит на экран строки У. Шекспира, которые соответствуют тому, что чувствуют многие люди, когда начинают изучать язык программирования.

Программа №1 **-** набрана в среде C#Builder - Project_Shakespeare – файл class.cs

Листинг 1.

```
 1: using System;
 2: // Вывод на консоль текста:  цитата из У. Шекспира
 3: namespace Project_Shakespeare
 4: {
 5:    public class Shakespeare
 6:   {
 7:      public static void Main()
 8:      {
 9:          Console.WriteLine("Though this be madness");
10:          Console.WriteLine("yet there is method in it");
11:          Console.WriteLine("William Shakespeare");
12:          Console.WriteLine();
13:          Console.WriteLine("Перевод на русский язык");
14:          Console.WriteLine("Это кажется безумным,");
15:          Console.WriteLine("и, вместе с тем, в этом имеется и логика и смысл");
16:          Console.ReadLine();
17:       }
18:    }
19: }
```

---

7. C#-Программа №2 - интерактивная программа дружественная пользователю.

Программа №2 **-** набрана в среде C#Building - Project_Mic3_1c80 – файл class.cs .

Листинг 2.1

```
01: using System;
02: // Простая программа на C#
03: namespace Project_Mic3_1c80
04: {
05:     class Hello
06:     {
07:           // Программа начинается с вызова метода Main()
08:              public static void Main()
09:            {
10:        string answer;
11:
12:        Console.WriteLine("Do you want me to write the famous words?");
13:        Console.WriteLine("Вы хотите, чтобы я написал знаменитые слова?");
14:        Console.WriteLine("Type Y for YES; N for NO. Then <Enter>");
15:        answer = Console.ReadLine();
16:        if (answer == "Y")
17:                Console.WriteLine("Hello World!");
18:        Console.WriteLine();
19:        Console.WriteLine("Bye, bye!");
20:        Console.ReadLine();
21:              }
22:       }
23: }
```

7.1. Псевдокод C#-программы №2 (Project_Mic3_1c80 – файл class.cs. Листинг 2.2)

01: Ключевое слово **using** с последующим названием предопределенного пространства имен System: позволяет указывать в ссылках краткие имена классов

02: Комментарий: Простая программа на C#

03: Ключевое слово **namespace**: объявление собственного пространства имен Project_Mic3_1c80

04: { - граница *пространства имен* Project_Mic3_1c80

05: Ключевое слово **class**: начало определения класса Hello

06: **{** - начало блока класса Hello

07: Комментарий: Программа начинается с вызова метода Main()

08: Ключевые слова **public static void:** начало определения метода Main()

09: **{ -** начало блока метода Main()

10: Ключевое слово **string:** объявление переменной answer для хранения текста

11: Пустая строка

12: Ключевые слова Console.WriteLine: Вывести Do you want me to write the two words? Перейти на строку вниз

13: См. пояснение к строке 12

14: См. пояснение к строке 12

15: Сохранить ожидаемый ответ пользователя (ключевые слова Console.ReadLine) в переменной answer. Перейти на строку вниз.

16, 17: Ключевое слово **if:** если в answer хранится '**Y**', вывести: Hello World!
Если в answer не хранится '**Y**'. пропустить строку 17 и продолжить выполнение со строки 18.

18: Ключевые слова Console.WriteLine(): вывести “пустую” строку. Перейти на строку вниз.

19: Ключевые слова Console.WriteLine: вывести: Bye Bye! Перейти на строку вниз.

20: Ключевые слова Console.ReadLine: вывод программы сохраняется на экране до нажатия на клавишу <Enter>

21: **} -** конец блока метода Main()

22: **} -** конец блока класса Hello

23: } - граница *пространства имен* Project_Mic3_1c80

// **ЕСЛИ ВЫВОД ИСЧЕЗАЕТ С ЭКРАНА БЫСТРЕЕ, ЧЕМ ЕГО МОЖНО ПРОЧЕСТЬ?**

Эта проблема имеет простое решение: в конце метода Main (перед закрывающей фигурной скобкой - **}**) нужно разместить следующую команду (см. строку 20 в листинге 2.1):

```
Console.ReadLine();
```

Этот вызов ожидает, пока пользователь нажмет клавишу Enter, что позволяет изучить вывод программы.

Программа на листинге 2.1 довольно проста, но содержит важные компоненты типичной C#-программы. **Рассмотрим каждую часть программы подробно**. *Концепции, представленные в этом разделе, применимы к большинству программ на C#.*

Пространство имен

Чтобы понять назначение строки 01 (см. Программу №2, листинг 2.1) необходимо обратиться к концепции пространства имен

```
01: using System;
```

Как удается придать дому опрятный вид? Ответ прост: решением **проблемы являются контейнеры.** Люди формируют вложенные иерархии контейнеров для хранения предметов одинакового типа. Например, кухонный нож (объект) размещается на кухне (рассматриваемой в данном случае как контейнер) в левом верхнем ящике (контейнере) в лотке для режущих инструментов (контейнере) в отделении для ножей (контейнере) вместе с другими более-менее подобными ножами.

Контейнеры позволяют не только держать дом в чистоте и порядке, но и находить различные предметы, сохраняемые в нем. В качестве разделителя между именами контейнеров используется **точка**. Гостю, который хочет найти нож достаточно вывести сообщение

Кухня.ЛевыйВерхнийЯщик.ЛотокДляРежущихИнструментов.ОтделениеДляНожей

В заключение, можно избежать и совпадения имен. Например, можно различать нож, применяющийся для рыбалки (хранящийся в гараже), и нож для еды (хранящийся на кухне) , хотя оба объекта и являются ножами. Различение достигается просто указанием местонахождения объектов. Таким образом, кухонный нож

Кухня.ЛевыйВерхнийЯщик.ЛотокДляРежущихИнструментов.ОтделениеДляНожей.Нож

будет отличаться от ножа для рыбалки

Гараж.ШкафСоСнастями.ПравыйВерхнийЯщик.КоробкаДляНожей.Нож

Кухня содержит не только ящики и коробки, но и такие объекты, как стулья и столы. Аналогично, *каждый контейнер может содержать не только другие контейнеры, но и объекты*.

Система контейнеров, описанная здесь, повторяет *концепцию пространства имен.*

Пространство имен **представляет собой контейнер для конструируемых классов, которые требуются программе.** *Пространства имен позволяют организовать исходный код при разработке программ (держа "домашние" классы в "чистоте и порядке").* Кроме того, они позволяют сообщать "гостям", где хранятся классы, чтобы они (другие программисты) могли с легкостью обращаться и повторно использовать их. Совпадение имен классов, созданных различными программистами или даже компаниями, исключено, поскольку каждый класс можно сослаться уникальным образом посредством имени его пространства имен.

> Обратившись к документации по **.NET**-платформе, можно встретить огромное количество классов (фактически, их несколько тысяч). *Поэтому эта платформа очень сильно и зависит от пространств имен*, необходимых для структурной организации и доступа к классам.

.NET-платформа содержит важное пространство имен под названием **System**. В нем содержатся классы, фундаментальные для любой программы на C#. Оно включает **класс Console**, используемый в листинге 2.1 (строки 12-14 и 17-20).

Следует избегать имен классов, похожих на такие важные и часто используемые идентификаторы пространств имен, как System, Collections, Forms и IO.

Листинг 2.1 иллюстрирует две различных возможности доступа к классам пространства имен **System** (или любого другого) в исходном коде.

- **Без** using System — тогда при всяком обращении к классам пространства имен System идентификатор последнего необходимо указывать явно, то есть:

```
System.Console.WriteLine("Bye Bye!");
```

класс — Console; **метод** — WriteLine

Ссылка на **пространство имен** System

- **С** using System, как в строке 01 листинга 2.1 — любой класс пространства имен System можно вызывать без явного указания префикса System. Предыдущий пример кода можно, таким образом, урезать до следующего (как в строках 12-14 и 17-20 листинга 2.1).

```
Console.WriteLine("Bye Bye!");
```

Ссылка на пространство имен System (она удалена, поскольку больше не требуется).

Команда **using** освобождает программистов от необходимости постоянно указывать пространство имен в исходном коде. Она улучшает читаемость кода, сокращая ссылки.

7.2. Основные элементы C#-программы №2 (Project_Mic3_1c80 – файл class.cs)

Номера строк в этом разделе соответствуют номерам строк на листингах 2.1 и 2.2 .

7.2.1.Комментарии

Строка 02 содержит комментарий, содержимое которого игнорируется компилятором. Он используется для описания действий, которые выполняет программа. В данном случае он просто сообщает о том, что программа написана на C#.

02: // Простая программа на C#

// **КОММЕНТАРИИ**
Комментарии важны для исходного кода не меньше, чем любые другие элементы. Они позволяют и другим программистам, и автору (после того, как он долго не обращался к конкретному исходному коду) разобраться в действиях программы. К сожалению, структура и логика исходного кода забываются достаточно быстро.

Двойной символ косой черты (//) заставляет компилятор игнорировать текст до конца строки. Строка 02 содержит только комментарий, однако последний можно разместить и в строке с кодом. Строки 02 и 03 можно объединить следующим образом:

namespace Project_Mic3_1c80 // Простая программа на C#

Другой вариант кода *не*корректен:

// Простая программа на C# **namespace** Project_Mic3_1c80

так как вся строка, включая и **namespace** Project_Mic3_1c80**,** рассматривается компилятором как комментарий.

7.2.2. Определение класса

Для объяснения строки 05 необходимо обратиться к концепции ключевого, или зарезервированного, слова. *Ключевое слово* имеет специальное значение в языке C# и распознается компилятором.

05: **class** Hello

В строке 05 для определения класса используется ключевое слово **class.** Hello — это имя класса, которое располагается непосредственно за **class.**

Каждый язык, включая и разговорные, содержит слова, имеющие специальное значение. Эти слова называют словарем, или терминологией.

Аналогично, язык C# имеет собственный словарь, состоящий из 77 ключевых слов. На листинге 2.1 представлено несколько из них: **class, public, static, void, string, if** и др**.**

Ключевые слова имеют для компилятора специальное значение. Их нельзя использовать для других целей в C# (ключевые слова "*зарезервированы*").

class, например, нельзя использовать как название ни для какого элемента C# (метода или переменной).

Следует отметить, что ключевое слово может быть частью имени, поэтому название **classVariable** вполне корректно.

Технический термин для имен вроде Hello из примера — **идентификатор.** Идентификаторы применяются для именования не только классов, но и методов, и переменных экземпляра.

7.2.3. Идентификаторы (имена)

Имена в исходном коде часто называют идентификаторами. Многие элементы — классы, объекты, методы, переменные экземпляра — должны всегда иметь идентификаторы. В отличие от ключевых слов C#, выбранных разработчиками, выбор всех идентификаторов остается за программистом.

Здесь существует несколько правил. **Идентификатор может состоять только из букв, цифр (0-9) и символа подчеркивания (_).** Идентификатор не может начинаться с цифры и совпадать с одним из ключевых слов (см. приложение).

Примеры:

Допустимые идентификаторы:

Elevator

_ elevator

My2Elevator

My_Elevator

MyElevator

*Не*допустимые идентификаторы:

Ele vator

6Elevator

В C# учитывается регистр, поэтому прописные и строчные буквы считаются разными символами. **Hello** и **hello** так же отличаются для компилятора, как **Hello** и **Bye**.

Рекомендуется *имена переменных начинать со строчной буквы*, а имена классов и методов – с прописной.

7.2.4. Фигурные скобки и блоки исходного кода

Строка 06 содержит открывающую фигурную скобку **{**, которая указывает на начало блока.

Блок — это фрагмент исходного кода C#, заключенный в фигурные скобки.

Закрывающая фигурная скобка **}** указывает на конец блока. Блок является логической единицей кода. Фигурные скобки всегда применяются в парах. Когда в коде встречается {, это значит, что где-то далее обязательно находится отвечающая ей }. Скобка }, соответствующая строке 06, находится в строке 22. Еще одна пара фигурных скобок находится в строках 09 и 21.

06: **{**

Поскольку скобка **{** в 06 расположена сразу после начала определения класса в строке 05, компилятор знает, что блок всего класса Hello содержится между открывающей скобкой **{** в строке 06 и закрывающей скобкой **}** в строке 22. **В блоке класса теперь можно разместить методы и переменные экземпляра** (как показано на рис. 1.5) при условии, что все объявления находятся внутри блока.

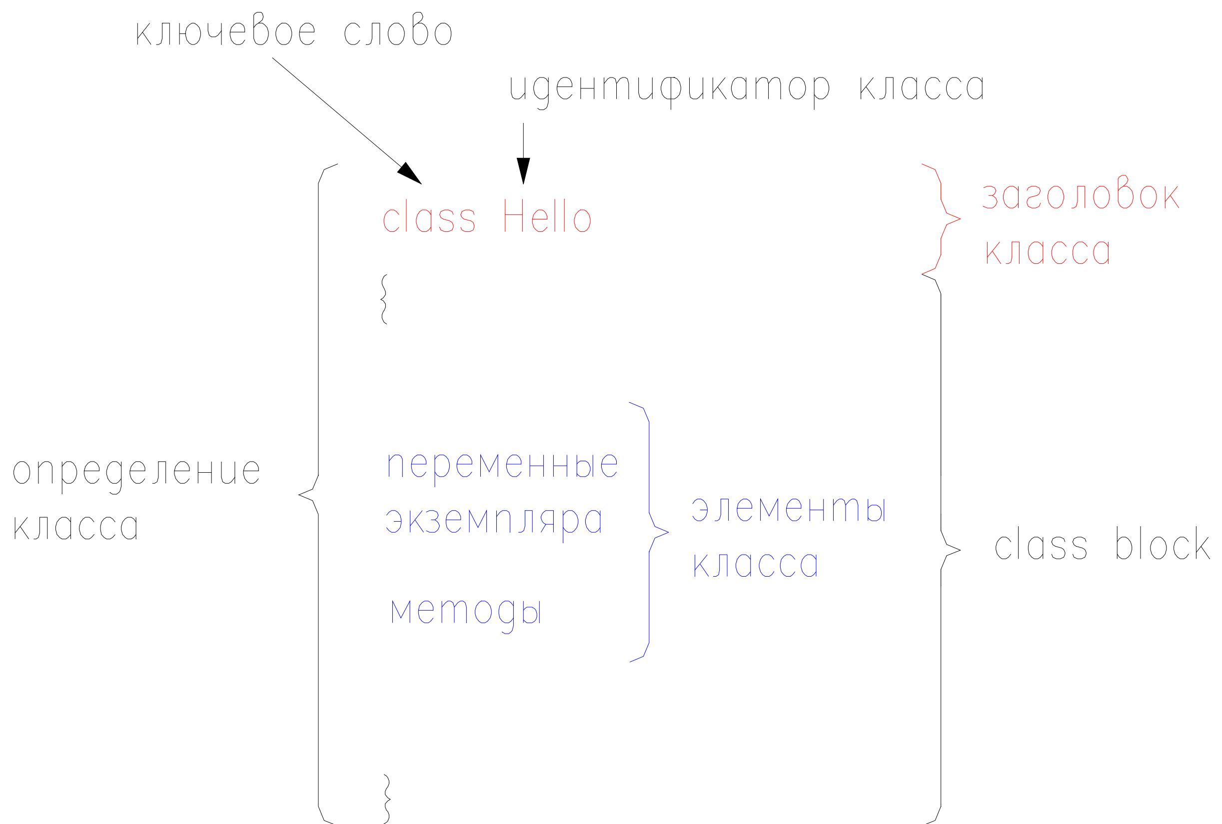


Рис. 1.5. Определение класса

// **СОВЕТ**

Здесь описан *способ, как предотвратить пропуск парных скобок*. Всегда, когда требуется начать блок с открывающей скобки {, *нужно сразу вводить и закрывающую скобку* }. Затем, расположив курсор между скобками, вводить код блока.

7.2.5.Метод Main() и его определение

В строке 08 начинается определение метода под названием Main. В C# нет ключевого слова наподобие "method", указывающего на то, что конструкция является методом. Компилятор распознает метод по круглым скобкам, следующим за его именем, в частности, () после Main.

```
08:     public static void Main()
```

Метод Main *имеет в* C# *специальное значение*. С этого метода начинает выполнение каждое приложение на C# — он *вызывается* средой исполнения **.NET** при запуске программы.

Например, сложное приложение для работы с электронными таблицами, написанное на C#, может содержать тысячи методов с различными идентификаторами, но ***только метод Main вызывается средой исполнения .NET при запуске программы.***

Точное значение всех элементов строки 08 пока что не будем обсуждать, поскольку оно требует более детального понимания определенных объектно-ориентированных принципов C#.

И тем не менее, дать краткую характеристику элементам строки 08 необходимо уже сейчас. Класс состоит из **интерфейса**, реализуемого посредством: ***1)*** **открытых** методов и ***2)*** *скрытой* части, состоящей из **закрытых** методов и переменных экземпляра.

Ключевое слово **public** в строке 08, очевидно, является *специфицикатором доступности*. Это ключевое слово позволяет управлять видимостью элемента класса. В данном случае (перед методом Main) оно указывает, что Main является открытым методом и, таким образом, частью интерфейса класса Hello. В результате метод Main можно вызывать извне объекта Hello

Основные элементы определения метода иллюстрируются рис. 1.6.

// **МЕТОД MAIN**

Каждая программа на C# должна содержать метод Main. При ее запуске среда исполнения **.NET**, в первую очередь, ищет этот метод. Если он найден, с него начинается выполнение, если нет — выводится сообщение об ошибке.

Main на листинге 2.1 расположен внутри класса Hello, а среда исполнения **.NET** — снаружи. При попытке запуска Main среда будет рассматриваться как еще один объект, запрашивающий доступ к методу класса. Поэтому его необходимо открыть, сделав частью интерфейса класса. Чтобы среда **.NET** могла получить доступ к Main, его следует всегда объявлять как **public**.

Обычно Main вызывает методы других объектов, но в этом простом примере имеется только один класс с одним методом.

Для первоначального рассмотрения ключевого слова **static** обратимся вновь к обсуждению различий между: **1)** классом и **2)** объектом. Класс представляет собой спецификацию (шаблон) того, как создать объект, так же, как чертеж является просто планом реального дома. *Класс обычно не может предпринимать каких-либо действий. Ключевое слово* **static** *позволяет отойти от этой схемы и воспользоваться методами класса, не создавая конкретного экземпляра объекта*.

Когда **static** включено в заголовок метода, это сообщает классу о том, что для использования метода не нужно создавать экземпляров за пределами класса. Таким образом, метод Main может использоваться до создания определенного объекта класса Hello.

В данном случае это обязательно, так как Main вызывается средой исполнения **.NET** до того, как создаются какие-либо объекты.

// **ПРИМЕЧАНИЕ**

Таким образом, метод Main **всегда** объявляется как **public и static.**

Концепция ключевого слова **static** будет изложена подробнее в дальнейшем.

Чтобы понять значение ключевого слова **void** в строке 08, необходимо обратиться непосредственно к тому, как работают методы. В этом разделе будет приведено лишь краткое объяснение: **void** указывает, что метод Main() не возвращает значения в точку вызова.

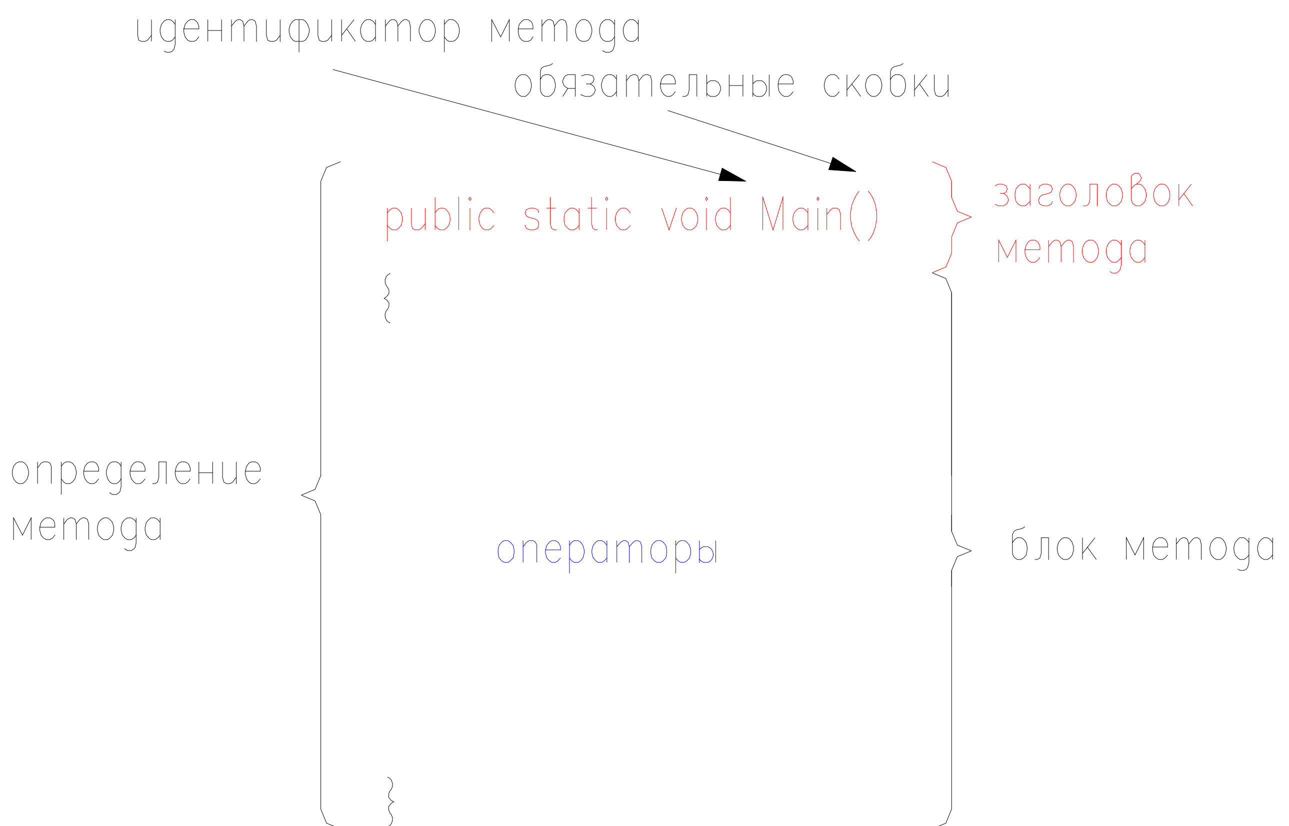


Рис. 1.6. Определение метода

В строке 09 скобка **{** указывает на начало блока Main(), в котором содержится тело метода. Блок заканчивается скобкой **}** в строке 21.

```
09:     {
```

// **СОВЕТ**

Для улучшения читаемости кода следует выбирать значащие (осмысленные) имена переменных и избегать аббревиатур.

В этом случае логику работы исходного кода можно восстановить, просто читая его, а не обращаясь к руководствам и другим вспомогательным материалам.

7.2.6. Переменные: представление данных, хранимых в памяти компьютера

answer в строке 10 является идентификатором переменной.

Переменная является именованной позицией в памяти, представляющей хранимый блок данных. Ключевое слово **string** указывает, что answer принадлежит типу **string.** Тип **string** объединяет отдельные символы в строки**.**

```
10:  string answer;
```

Идентификатор (answer) программист выбирает по своему усмотрению, а **string** является зарезервированным словом.

Размещение answer после **string** в строке 10 на техническом языке означает, что объявлена переменная answer типа **string.**

// **ПРИМЕЧАНИЕ**

Каждая переменная, используемая в программе на C#, должна быть объявлена.

Переменная answer применяется в строках 10 и 16. Переменная типа **string** может содержать текст. "Coco is a dog", "y", "n" являются примерами текста, который может храниться в answer**.** В C# строки текста обозначаются " " (двойные кавычки). Составные части определения переменной показаны на рис. 1.7.

Из рисунка видно, что переменная состоит из трех элементов:

1. *Идентификатора* (в данном случае, answer)**.**

2. *Типа,* т.е. вида информации, которую она может хранить (в данном случае — последовательность символов).

3. *Значения,* т.е. хранимой информации. Текущее значение на рисунке равно "Julian is a boy".

Здесь изложены не все сведения *о типе* **string**. Это будет сделано в дальнейшем. *Фактически, тип* **string** *является ссылочным*: переменная не хранит непосредственно текст, а ссылается на область памяти, в которой он содержится.

Единственное, что осталось рассмотреть в строке 10, — символ точки с запятой (**;**). Любую задачу, выполняемую программой на C#, можно разбить на последовательность инструкций. Простейшая инструкция называется *оператором*. Все операторы заканчиваются символом точки с запятой. Строка 10 содержит оператор объявления переменной, поэтому он, как и другие, заканчивается точкой с запятой.

Строка 11 является пустой.

11:

Компилятор C# игнорирует пустые строки. Однако они могут быть вставлены в исходный код для улучшения его читаемости.

7.2.7. Запуск методов .NET-платформы

Оператор в строке 12

12: Console.WriteLine("Do you want me to write the two words?");

заставляет программу вывести на экран следующее:

Do you want me to write the two words?

На данный момент достаточно рассматривать вызов Console.WriteLine как просто способ вывода, имеющий смысл: "вывести все, что содержится в скобках после WriteLine на экран и перейти на одну строку вниз".

Вот что, вкратце, происходит в строке 12. Console — это класс .NET Framework. **.NET Framework** является библиотекой, содержащей множество полезных классов, созданных разработчиками из Microsoft. Таким образом, для вывода текста на экран повторно используется класс Console. Он содержит метод WriteLine, который и вызывается командой: Console.WriteLine. WriteLine выполняет все действия — выводит на экран текст, заключенный в скобки ("Do you want me to write the two words?").

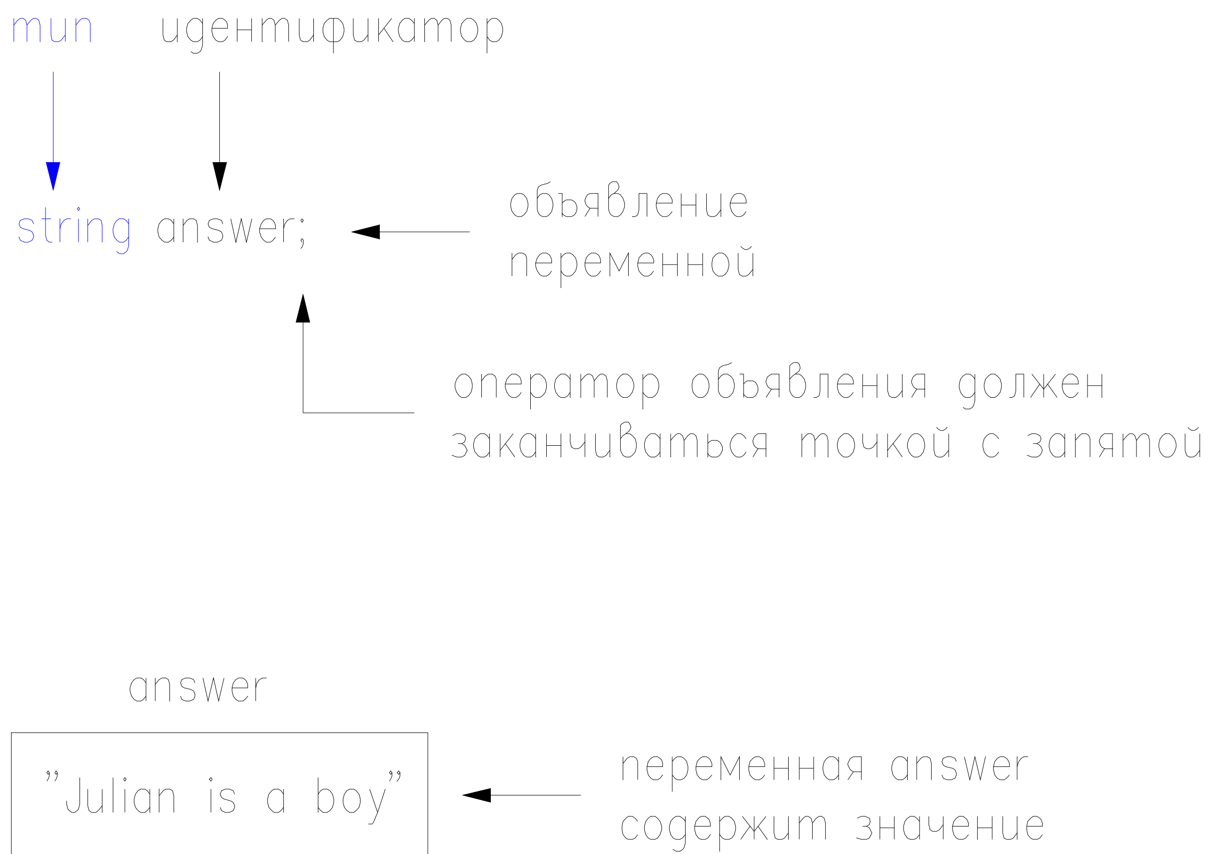


Рис. 1.7. Тип, идентификатор и значение переменной

Когда метод выполняет определенную задачу в программе, это называют *вызовом*. Элемент внутри круглых скобок (текст "Do you want me to write the two words?" в примере) называется аргументом. *Аргумент* содержит информацию, необходимую вызываемому методу для выполнения задачи. *Аргумент* передается методу WriteLine при вызове. После этого метод обращается к данным уже посредством своих внутренних операторов. Действие метода WriteLine можно описать теперь более осмысленно: "При вызове метода WriteLine вывести на экран аргумент, переданный ему".

Строка 12, как и 10, содержит оператор и поэтому заканчивается точкой с запятой.

Подчеркнем еще раз следующее. Классы являются "схемами – шаблонами", а объекты – "исполнителями". Метод – это действие, которое осуществляет объект. *Метод класса можно использовать в том случае, когда в объявлении метода присутствует ключевое слово* static (которое упоминалось ранее).

7.2.8. Общий механизм вызова метода

Инструкции метода содержатся внутри его определения в форме операторов.

Вызвать метод — это просто означает выполнить его инструкции. Выполнение происходит последовательно в том порядке, в котором они написаны в исходном коде.

Метод можно определить только *внутри* класса. Он является действием, которое способен выполнить объект. Вызов метода имеет следующий синтаксис: имя объекта (или класса, если метод объявлен как **static)**, точка **.**, имя метода и завершающая пара круглых скобок (), в которых *могут* содержаться аргументы. Последние представляют собой данные, передаваемые методу.

Вызов не*статического* метода:

ИмяОбъекта.ИмяМетода(Необязательные_аргументы)

Соответственно, вызов *статического* метода: -

ИмяКласса.ИмяМетода(Необязательные_аргументы)

Заменив общие элементы реальными именами, легко получить оператор из строки 19 листинга 2.1.

Console.WriteLine("Bye Bye!");»

По завершении метода поток управления возвращается в точку, из которой произошел вызов. Для вызова метода в строке 19 листинга 2.1 можно выделить следующие шаги:

1. Выполнить операторы в строках, предшествующих 19.
2. Запустить строку 19.
3. Вызвать Console.WriteLine с аргументом "Bye Bye!".
4. Выполнить операторы внутри Console.WriteLine(...).
5. Возвратить управление оператору в строке *после* 19.
6. Выполнить остальные операторы метода Main.

В строке 14 содержится еще один вызов метода WriteLine

14: Console.WriteLine ("Type Y for YES; N for NO. Then <Enter>");

В результате на экран выводится строка:

Type Y for YES; N for NO. Then <Enter>

и курсор перемещается на одну строку вниз.

// **СООБЩЕНИЕ: СИНОНИМ** ВЫЗОВА **МЕТОДА**

Рассмотрим строку 14 листинга 2.1. В ней содержится оператор, вызывающий метод WriteLine. В объектно-ориентированном программировании для обозначения такого вызова часто применяется еще один термин.

Когда метод объекта **A** содержит оператор, вызывающий метод объекта **B**, говорят, что **A** посылает сообщение **B**. В строке 14 класс Hello посылает сообщение классу Console. Сообщением является

WriteLine("Type Y for YES; N for NO. Then <Enter>");

Общая схема ООП подразумевает, что объекты выполняют действия, запускаемые при получении сообщений. В примере выше действием является вывод на консоль

Type Y for YES; N for NO. Then <Enter>

7.2.9. Присваивание значения переменной

В строке 15 вновь повторно используется класс Console. На этот раз применяется другой из его статических методов — ReadLine, который приостанавливает выполнение программы, ожидая ввода от пользователя. Ответом может быть введенный текст, завершае-

мый нажатием клавиши Enter. Как следует из названия, метод ReadLine читает ввод.

15: answer = Console.ReadLine();

При нажатии Enter текст, введенный пользователем, сохраняется в переменной answer. Очевидно, когда пользователь вводит 'Y', answer содержит "Y", когда 'N', — "N" и т.д. За это отвечает знак равенства (=), расположенный после answer.

В C# знак равенства (=) используется несколько иначе, чем в стандартной арифметике. В арифметике он просто обозначает равенство выражений слева и справа от знака. В C# знак равенства имеет значение "Сделать answer равным Console.Readline()" или, другими словами, "Сохранить текст, введенный с клавиатуры в переменной answer ".

Механизм задания нового значения переменной answer называется *присваиванием*. Говорят, что введенный текст присваивается переменной answer. Общее выражение в строке 15 называют оператором присваивания, а сам знак равно (=) называют *операцией присваивания* (в данном контексте). Если знак равно используется в других контекстах, он имеет другие названия.

7.2.10. Ветвление посредством оператора if

Одно из применений знака равно показано в строке 15. В строке 16 он используется в совершенно другом контексте, - на этот раз, - в стандартном арифметическом.

16: **if** (answer == "Y")

17: Controle.WriteLine ("Hello World!")

В C# два последовательных знака равенства (==) обозначают операцию равенства, используемую для сравнения выражений слева и справа от него.

В строке 16 спрашивается: answer == "Y", т.е. "Равно ли значение answer "Y"?" Ответом может быть: истина (**true**) или ложь (**false**).

Выражение, которое может принимать только одно из двух значений (**true** или **false**), называется *логическим*.

Ключевое слово **if** с последующим логическим выражением answer == "Y", заключенным в скобки, имеет следующий смысл: только если answer == "Y" равно **true** (истинно), нужно запустить оператор в строке, следующей за 16. Если же answer == "Y" равно **false** (ложно), поток выполнения должен перейти к строке 18.

В строках 16 и 17 содержится оператор **if**. Он управляет потоком исполнения, позволяя выбрать два различных направления. Такие операторы, как **if**, называются операторами ветвления.

// **ПРЕДУПРЕЖДЕНИЕ**

Пара круглых скобок, в которые заключено выражение в операторе **if** :

if (answer == "Y")

является обязательной. Их отсутствие приводит к синтаксической ошибке.

7.2.11. Завершение метода Main() и класса Hello

Скобка **}** в строке 21 завершает блок метода Main, начатый в строке 09.

21: **}**

// **ПРЕДУПРЕЖДЕНИЕ**

Фигурные скобки {} всегда используются в паре. Невыполнение этого правила приводит к ошибке при компиляции.

В строке 22 скобка **}** завершает блок класса Hello.

22: **}**

8. Компиляция в .NET исходного кода C#

Традиционный процесс компиляции исходного кода, написанного на языке программирования высокого уровня, в исполняемую программу имел несколько недостатков. Два из них рассмотрены ниже.

Проблема I:

Для каждой аппаратной платформы (характеризуемой типом процессора и т.д.) **требуется свой компилятор**, поскольку у каждой из них — свой машинный язык.

Проблема 2:

У большинства программистов есть любимый язык программирования. Когда дело доходит до работы нескольких программистов в рамках одного проекта, начинаются трудности. Возможно, **лучшим решением было бы позволить каждому члену команды писать на своем любимом языке**, но это нелегко. **Разные языки на машинном уровне решают одну и ту же функциональность разными способами — частично это зависит от особенностей компиляторов.** В свою очередь, *это делает невозможным взаимодействие различных языков между собой.*

В **C#** и **.NET** **реализованы интересные решения описанных выше двух проблем**. Рассмотрим все составляющие процесса компиляции программы в **.NET** (рис.1.8).

На **рис. 1.8** представлены еще два языка — **C++** и **Visual Basic**. И вне зависимости от того, на каком языке (из поддерживаемых **.NET**) написана программа, это никак не влияет на процесс ее компиляции в **.NET.** После того как написан *исходный код, его нужно откомпилировать в машинный код*. Однако сначала он компилируется в другой язык, который называется **Microsoft Intermediate Language** (MSIL). Все компиляторы, ориентированные на **.NET**-платформу, должны генерировать на выходе код данного **промежуточного языка MSIL.**

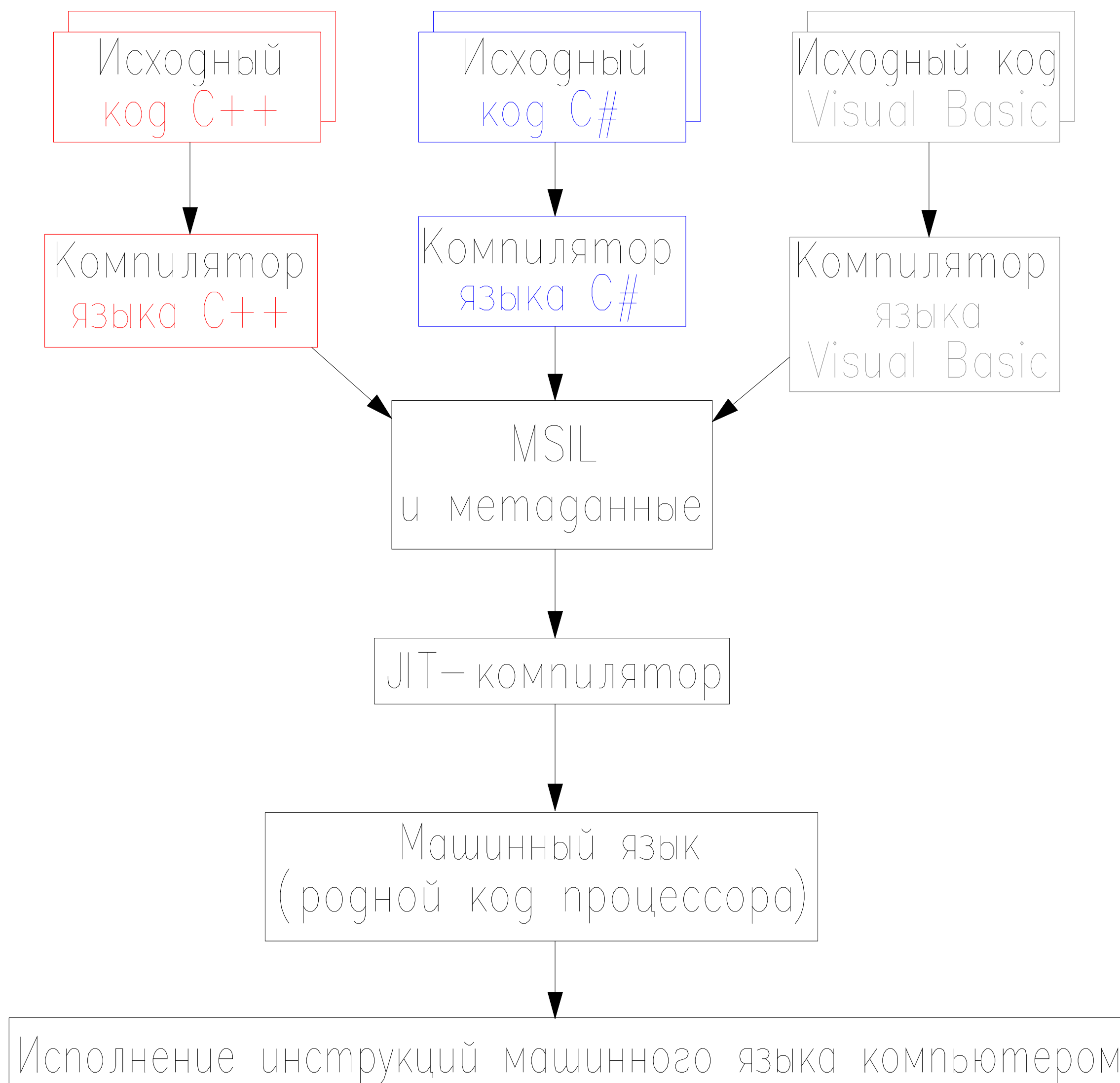
Как ясно из названия, **MSIL** является промежуточным звеном между языками высокого уровня и машинными языками. Затем код **MSIL** можно быстро и эффективно транслировать в машинный язык с помощью **JIT**-компилятора (**Just in Time-Compiler**). Код, генерируемый **JIT**-компилятором, ничем не отличается от машинного кода, генерируемого обычным компилятором, однако **JIT**-компилятор использует несколько иную стратегию. Вместо того чтобы, интенсивно используя память и затрачивая значительное время, преобразовать в машинный код сразу весь код **MSIL**, он компилирует в машинный код лишь те части приложения, которые реально необходимы в данный момент. В *результате код компилируется "на ходу", непосредственно перед исполнением, и **JIT**-компилятор не расходует время на компиляцию неиспользуемого **MSIL** -кода.*

В чем же преимущества архитектуры .NET? Прежде всего, *код **MSIL** остается неизменным, на какой бы аппаратной платформе он ни использовался.* Единственным *машино-зависимым элементом является **JIT**-компилятор*, и при изменении оборудования лишь он нуждается в модификации. На каждом компьютере применяется свой JIT-компилятор, преобразующий код **MSIL** в машинный код, совместимый с данной конкретной конфигурацией. **В результате все, что нужно** — это компилировать код, написанный на языке высокого уровня, в универсальный код **MSIL**, язык которого остается неизменным. *Это и является решением упомянутой выше Проблемы 1.*

Теперь перейдем к упрощенному объяснению того, **как решается Проблема 2**. На **рис.1.8** следует обратить внимание на **Метаданные** непосредственно за **MSIL**-кодом. Вообще, термин "метаданные" можно перевести как "**данные о данных**". Метаданные генери-

руются компилятором языка высокого уровня и *содержат подробное описание всех элементов исходного кода*. Описание это настолько подробное, что исходный код других языков высокого уровня сможет использовать данный исходный код, как если бы он был написан на том же языке. Теперь *программисты, пишущие на C++, C# и Visual Basic, реально смогут работать в рамках одного проекта*. То есть решена проблема 2.

Важно знать о существовании **MSIL**, но в повседневном программировании сталкиваться с ним напрямую не приходится. Обычно применяются две команды: – **1)** одна для компиляции программы в **MSIL**-код и метаданные, а **2)** другая - для исполнения программы (при этом будет вызываться **JIT**-компилятор). Фактически выполнение программы - это исполнение конечного результата работы компиляторов: **MSIL** в этом процессе остается "невидимым" для пользователя.



MSIL – MS Intermediate Language – промежуточный язык.

JIT-компилятор – Just in Time – В процессе выполнения.

Метаданные - "данные о данных".

Рис. 1.8. Процесс компиляции в .NET

Контрольные вопросы

1. Как отличить от других файлов файл с исходным кодом C#?
2. Перечислите основные службы, предоставляемые .NET -платформой.
3. Кратко охарактеризуйте процедурно-ориентированный стиль программирования.
4. Кратко охарактеризуйте объектно-ориентированный стиль программирования. В чем его преимущества в сравнении с процедурным стилем?
5. Что такое объект? Из чего он состоит?
6. Расскажите о том, как соотносятся классы и объекты.
7. В чем разница между классом и его объектами?
8. Как называется часть кода, предназначенная для повторного использования?
9. Какая часть .NET предоставляет программные компоненты для повторного использования?
10. С чего начинается комментарий?
11. Зачем используются комментарии, если компилятор игнорирует их?
12. Что представляют собой ключевые слова и идентификаторы?
13. Как обозначается блок в C#?
14. Можно ли создать программу без метода Main? Объясните почему.
15. Перечислите существенные элементы объявления переменной.
16. Как называется базовая инструкция языка C#? Каким символом она завершается?
17. Какой метод какого класса из библиотеки классов .NET Framework вызывается для вывода текста на консоль? Запишите выражение, которое выводит на экран "My dog is brown".
18. Как вызывается метод? Что при этом происходит?
19. Что такое присваивание? Какой символ применяется для его выполнения?
20. Как можно выбрать один из путей выполнения программы?
21. Что такое MSIL? Какие достоинства MSIL дает архитектуре .NET?

Упражнения по программированию

Упражение №1 (на базе рассмотренной C#-программы №1)

Напишите, откомпилируйте и запустите программу на C#, которая бы вывела на экран командной консоли следующую выдержку из “Фауста” Гете.

Перевод на русский язык:

Не бог ли эти знаки начертал?
Таинственен их скрытый дар!
Они природы силы раскрывают.
И сердце нам блаженством наполняют.

Откомпилируйте и запустите эту C#-программу двумя способами: *1)* из интегрированной среды программирования, *2)* из командной строки (воспользуйтесь bat-файлом 1.bat – текст этого файла – см. с. 13)

// **Совет**. Для работы в командной строке рекомендуется создать папку CSharpFiles и поместить в нее файл 1.bat, а также файлы с C#-программами, расширение их имени .cs . При вызове компилятора C# (csc.exe) указывать имя компилируемого файла без расширения .cs .

Упражение №2 (на базе рассмотренной C#-программы №2)

В следующих упражнениях нужно изменить или добавить части в листинге 2.1 для выполнения требуемых действий.

1. Замените вывод "Bye Bye!" перед выходом из программы на "Bye Bye. Have a good day!"

2. Измените программу так, чтобы вместо ввода Y для вывода на экран "Hello World!" пользователь вводил Yes. Программа должна информировать пользователя об этом (измените строку 14).

3. Измените имя переменной, содержащей пользовательский ввод, с answer на userInput.

4. Добавьте в программу вывод дополнительной строки "The program is terminating" после "Bye Bye. Have a good day!"

5. Объявите переменную userName типа string. До того, как программа выводит "Do you want me to write the famous words?", запросите у пользователя имя. Программа должна сохранить его в переменной userName и вывести после этого "Hello" и ее содержимое. Подсказка: последний вывод осуществляется командой

```
Console.WriteLine ("Hello" + userName);
```

Выполнение программы теперь должно дать примерно такой вывод:

Please type your name
Deborah<enter>
Hello Deborah
Do you want me to write the famous words?
Type Yes for yes; n for no. Then <enter>.
Yes<enter>
Hello World!
Bye Bye. Have a good day!
The program is terminating.

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 с.

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**].

http://books.dore.ru/bs/f6sid16.html - книги по теме C#

Загляни в Интернет-магазин

http://www.ozon.ru

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

Зарезервированные слова языка *C#*

В таблице представлены зарезервированные (ключевые) слова языка C#. Эти слова используются в программе только по своему прямому назначению. Применять их по-другому, например, как идентификаторы, запрещено.

Таблица. **Ключевые слова C#**

abstract	as	base	bool
break	byte	case	catch
char	checked	class	const
continue	decimal	default	delegate
do	double	else	enum
event	explicit	extern	false
finally	fixed	float	for
foreach	goto	if	implicit
in	int	interface	internal
is	lock	long	namespace
new	null	object	operator
out	override	params	private
protected	public	readonly	ref
return	sbyte	sealed	short
sizeof	stackalloc	static	string
struct	switch	this	throw
true	try	typeof	uint
ulong	unchecked	unsafe	ushort
using	virtual	void	volatile
while			