

# **Лекция 1:**

# **Обзор языка C#**

# Литература

- <http://msdn.microsoft.com/ru-ru/vstudio/default.aspx>
- Спецификация языка C#. Версия 4.0. – 567 с.
- **Герберт Шилдт** C# 4.0: полное руководство. : Пер. с англ. — М. : ООО "И.Д. Вильямс", 2011. — 1056 с.
- **Трей Неш** C# 2010: ускоренный курс для профессионалов. — М.:ООО "И.Д. Вильямс". 2010. — 592 с.
- **ЭндрюТроелсен** Язык программирования C# 2010 и платформа .NET 4.0, 5-е изд. : Пер. с англ. — М. : ООО "И.Д. Вильяме", 2011. — 1392 с.
- **Фаронов В.В.** Программирование на языке C#. – СПб.: Питер, 2007. – 240с.
- **Карли Уотсон, Кристиан Нейгел, Якоб Хаммер и др.** Visual C# 2008: базовый курс. Visual Studio® 2008

# **.Net Framework**

- **открытость для языков программирования**
- **принципиально новый подход к построению каркаса среды - Framework .Net:**
  - FCL (Framework Class Library) - библиотеку классов каркаса;
  - CLR (Common Language Runtime) - общезыковую исполнительную среду

# CLR

Среда CLR отвечает за обслуживание процесса выполнения приложений, которые разрабатываются с помощью .NET

## Функции CLR:

### ■ двухшаговая компиляция:

- преобразование исходного кода в управляемый код на промежуточном языке ***Intermediate Language*** (IL),
- преобразование IL-кода в машинный код конкретного процессора, который выполняется с помощью  **JIT-компилятора** (*Just In Time compiler* – оперативное компилирование);

### ■ управление кодом: загрузка и выполнение уже готового IL-кода с помощью JIT-компилятора;

### ■ управление памятью при размещении объектов с помощью сборщика мусора (***Garbage Collector***);

### ■ обработка исключений.

# FCL

## FCL – библиотека классов платформы

- библиотека разбита на несколько модулей таким образом, что имеется возможность использовать ту или иную ее часть в зависимости от требуемых результатов
- FCL включает в себя:
  - Common Language Specification (CLS – общая языковая спецификация)
    - устанавливает основные правила языковой интеграции
  - Описание базисных типов
    - Common Type System (CTS — единая система типов)

# Правила CLS

- Все типы, фигурирующие в прототипе метода, должны быть CLS-совместимыми.
- Элементы массивов должны иметь CLS-совместимые типы. Индексация элементов массива должна начинаться с нуля.
- CLS-совместимый класс должен быть унаследован от CLS-совместимого класса.
- Имена методов CLS-совместимых классов нечувствительны к регистру.
- Допустимыми типами перечислителей являются `Int16`, `Int32` и `Int64`. Перечисления других типов не являются CLS-совместимыми.

# Что собой представляет язык C#

- C# является строго типизированным объектно-ориентированным языком
- Эволюционировал из языков C и C++ и был создан Microsoft специально для работы с платформой .NET
- C# непосредственно связан с языками Си, C++ и Java
- От языка Си – унаследовал синтаксис, многие ключевые слова и операторы
- C# построен на улучшенной объектной модели, определенной в C++
- Подобно Java язык C# предназначен для создания переносимого кода

# Приложения, которые можно писать на C#

Наиболее распространенные:

- Консольные приложения позволяют выполнять вывод на "консоль", то есть в окно командного процессора.
- Windows-приложения, использующие элементы интерфейса Windows, включая формы, кнопки, флажки и т.д.
- Web-приложения – web-страницы, которые могут просматриваться любым web-браузером.
- Web-сервисы – распределенные приложения, которые позволяют обмениваться по Интернету данными с использованием единого синтаксиса



# Создание приложений на языке C#

- Комплект инструментов для разработки программного обеспечения

## *.NET Framework SDK (Software Development Kit)*

- Компилятор csc.exe и текстовый редактор (блокнот)
- Бесплатная IDE-среда SharpDevelop
- Visual C# 2019 Express (распространяется бесплатно)
- Visual Studio 2019

# Компилятор csc.exe

- Список аргументов командной строки, которые может принимать компилятор csc

csc -?

- Компиляция TestApp.cs в консольное приложение TestApp.exe

```
csc /target:exe TestApp.cs
```

- или

```
csc /t:exe TestApp.cs
```

- Для создания другого имени исполняемого файла используется флаг /out:

```
csc /target:exe /out: TestApplication.exe TestApp.cs
```

- Добавление ссылок на внешние сборки:

```
csc /r:System.Windows.Forms.dll TestApp.cs
```

# Понятия проекта и решения

- Для разработки приложений требуется создавать *решения (solutions)*.
- Приложение, находящееся в процессе разработки, называется *проектом*.
- Несколько приложений могут быть объединены в *решение (solution)*.
  - решения могут содержать несколько проектов,
  - связанный между собой код можно группировать в одном месте, даже если впоследствии он будет компилироваться в несколько сборок

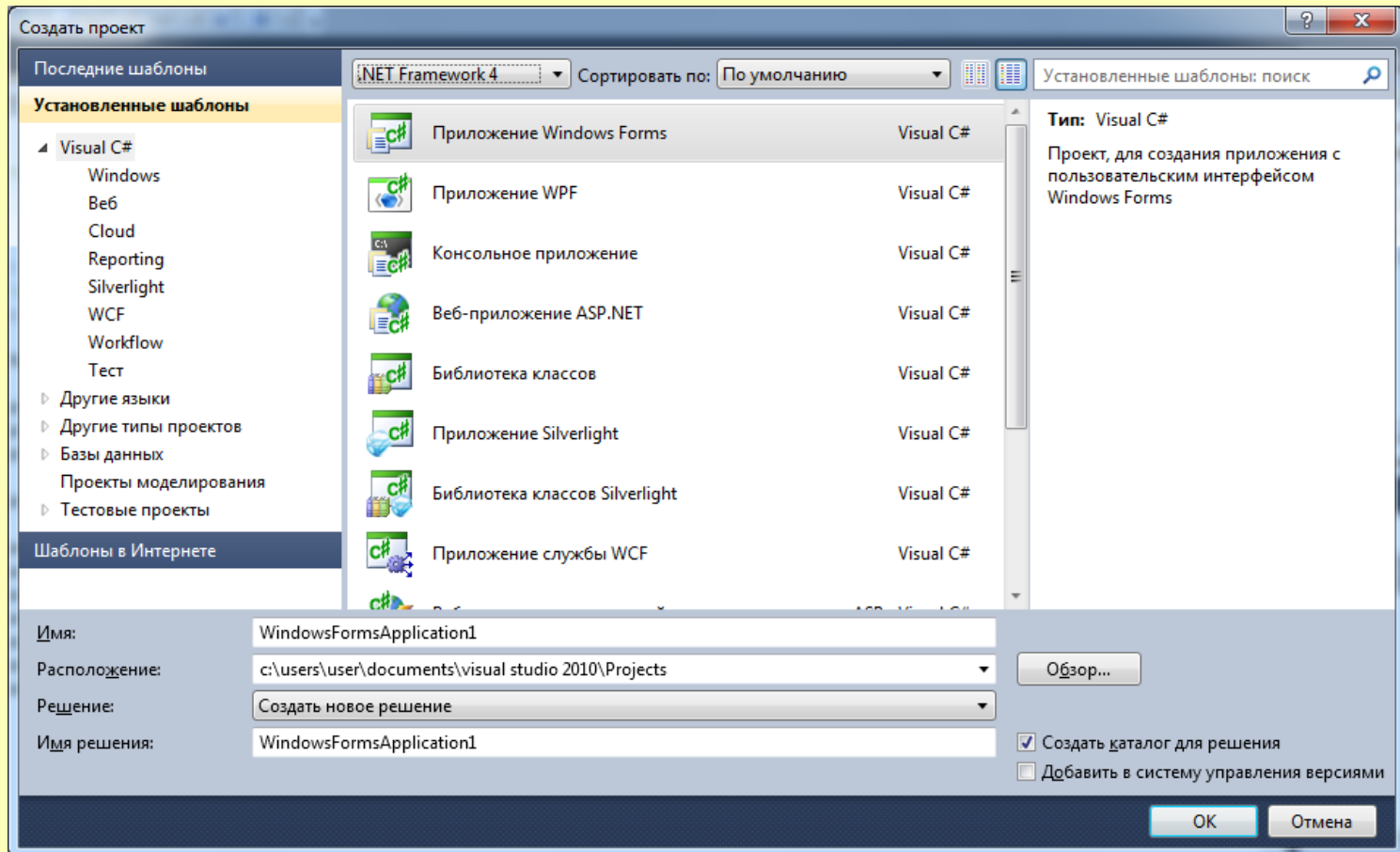
# Среда разработки Visual Studio .Net

## Функциональные возможности Visual Studio:

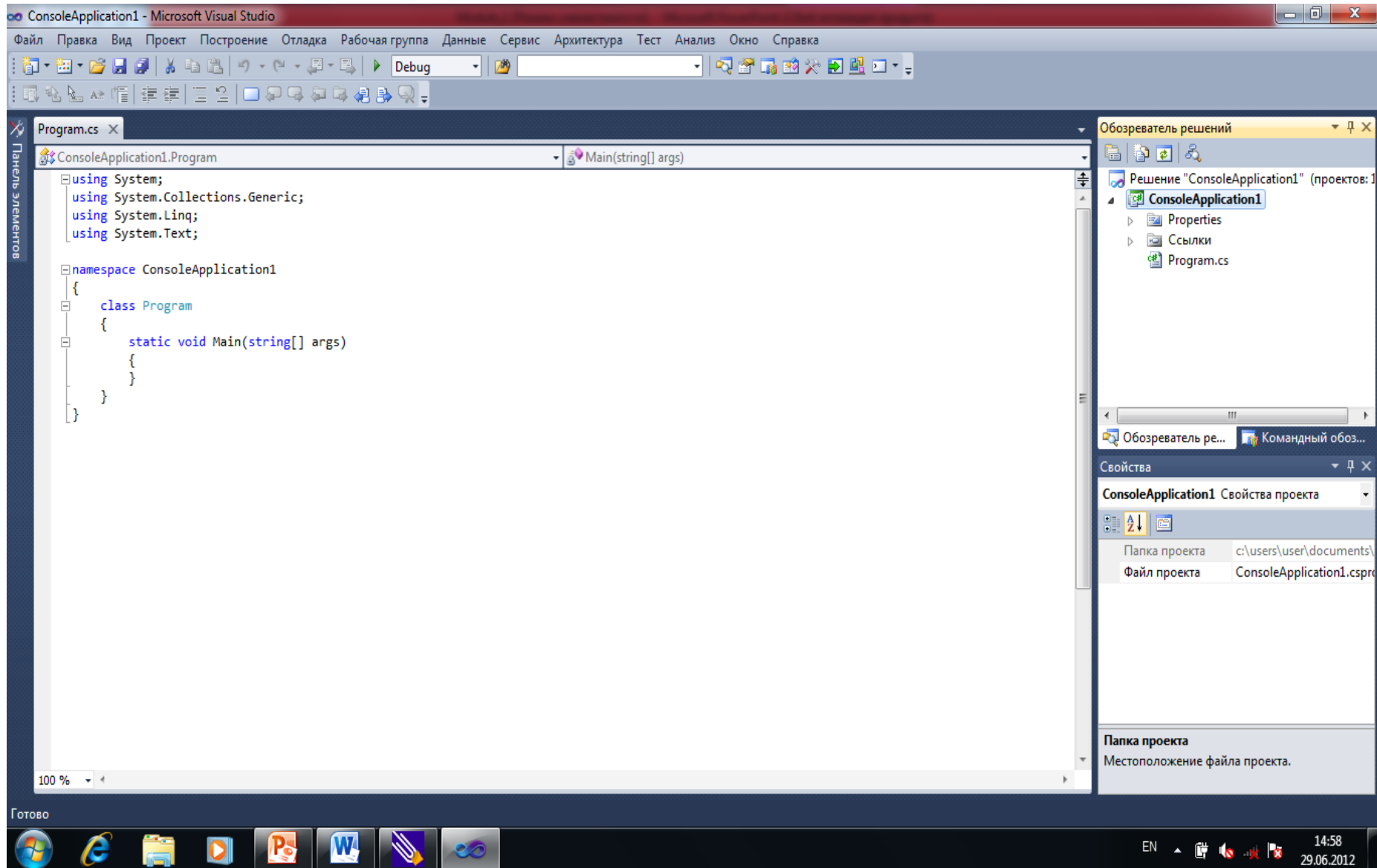
- Автоматизация шагов, требуемых для компиляции исходного кода
- Текстовый редактор может интеллектуальным образом обнаруживать ошибки и предлагать код
- В состав VS входят конструкторы для приложений типа Windows Forms и Web Forms
- Мастера автоматизируют выполнение наиболее распространенных задач
- Средства для визуализации и навигации по элементам проекта
- Усовершенствованные приемы отладки

# Создание проекта

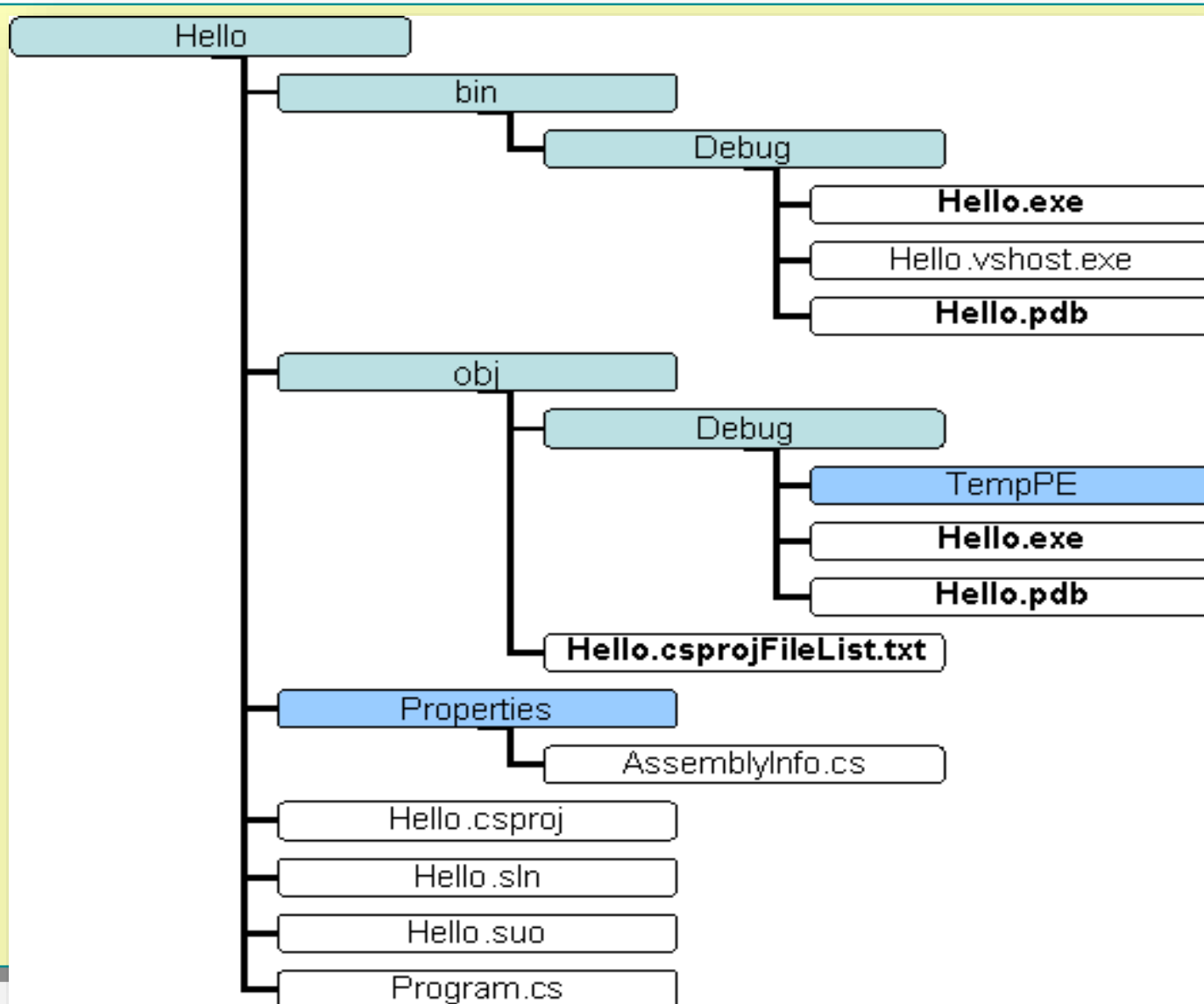
- **File – New – Project:** откроется диалоговое меню New Project



# Интегрированная среда разработки (Integrated Development Environment) — IDE



# Структура проекта



# ◆ Структура программы на C#

- Hello, World
- Класс
- Метод Main
- Директива using и пространство имен System
- Демонстрация: Создание C# программы в Visual Studio



# Hello, World

```
using System;

class Hello
{
    static void Main()
    {
        Console.WriteLine("Hello, World");
    }
}
```

# Класс

- C# приложение – это коллекция классов, структур и типов
- Класс – это набор данных и методов
- Синтаксис

```
class name  
{  
    ...  
}
```

- C# приложение может состоять из нескольких файлов

# Метод Main

- При написании Main необходимо:
  - Использовать прописную “M”, как в “Main”
  - Назначить метод Main точкой входа в приложение
  - Объявить **Main** как **static void Main**
- В одном приложении может использоваться несколько классов, имеющих метод Main
- Приложение выполняется до тех пор, пока не будет достигнуто окончание метода Main или не выполнится оператор return

# Варианты метода Main()

- **Возвращаемый тип int и массив строк в качестве параметра:**

```
static int Main(string[] args)
{
    // Должен обязательно возвращать значение перед выходом!
    return 0;
}
```

- **Ни возвращаемого типа, ни параметров:**

```
static void Main()
{ }
```

- **Возвращаемый тип int, но никаких параметров:**

```
static int Main ()
{ // Должен обязательно возвращать значение перед выходом!
    return 0;
}
```

# Директива using и пространство имен System

- В .NET Framework есть много полезных классов
  - Организованных в пространства имен
- System – наиболее часто используемое пространство имен
- Обращайтесь к классам через их пространства имен

```
System.Console.WriteLine("Hello, World");
```

- Директива using

```
using System;  
...  
Console.WriteLine("Hello, World");
```

# Составные части управляемого модуля

## ■ Заголовок PE32 или PE32+:

- Заголовок показывает тип файла

## ■ Заголовок CLR:

- Содержит информацию, которая превращает этот модуль в управляемый

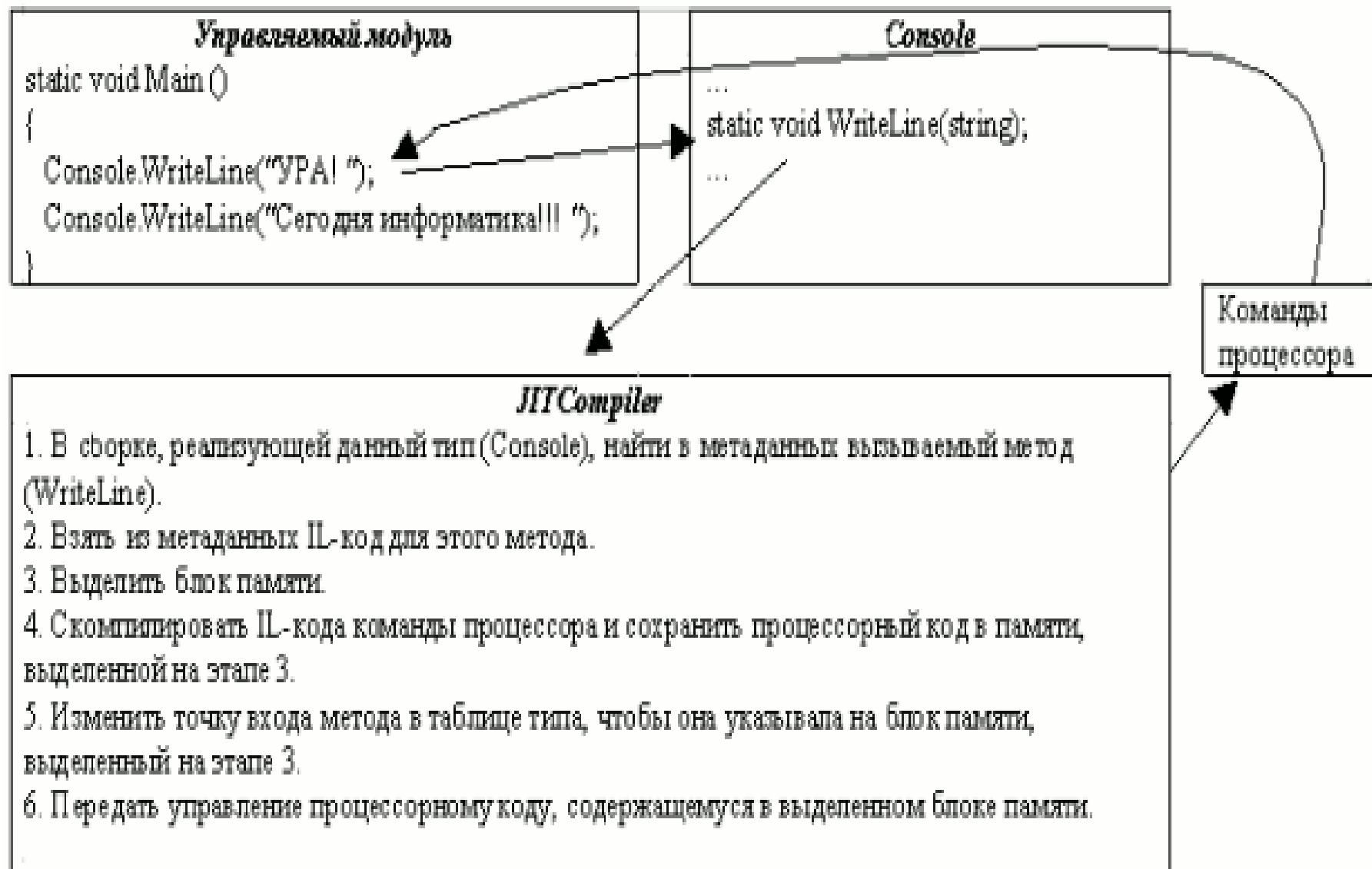
## ■ Метаданные:

- набор таблиц данных, описывающих то, что определено в модуле

## ■ IL-код:

- управляемый код, создаваемый компилятором при компиляции исходного кода

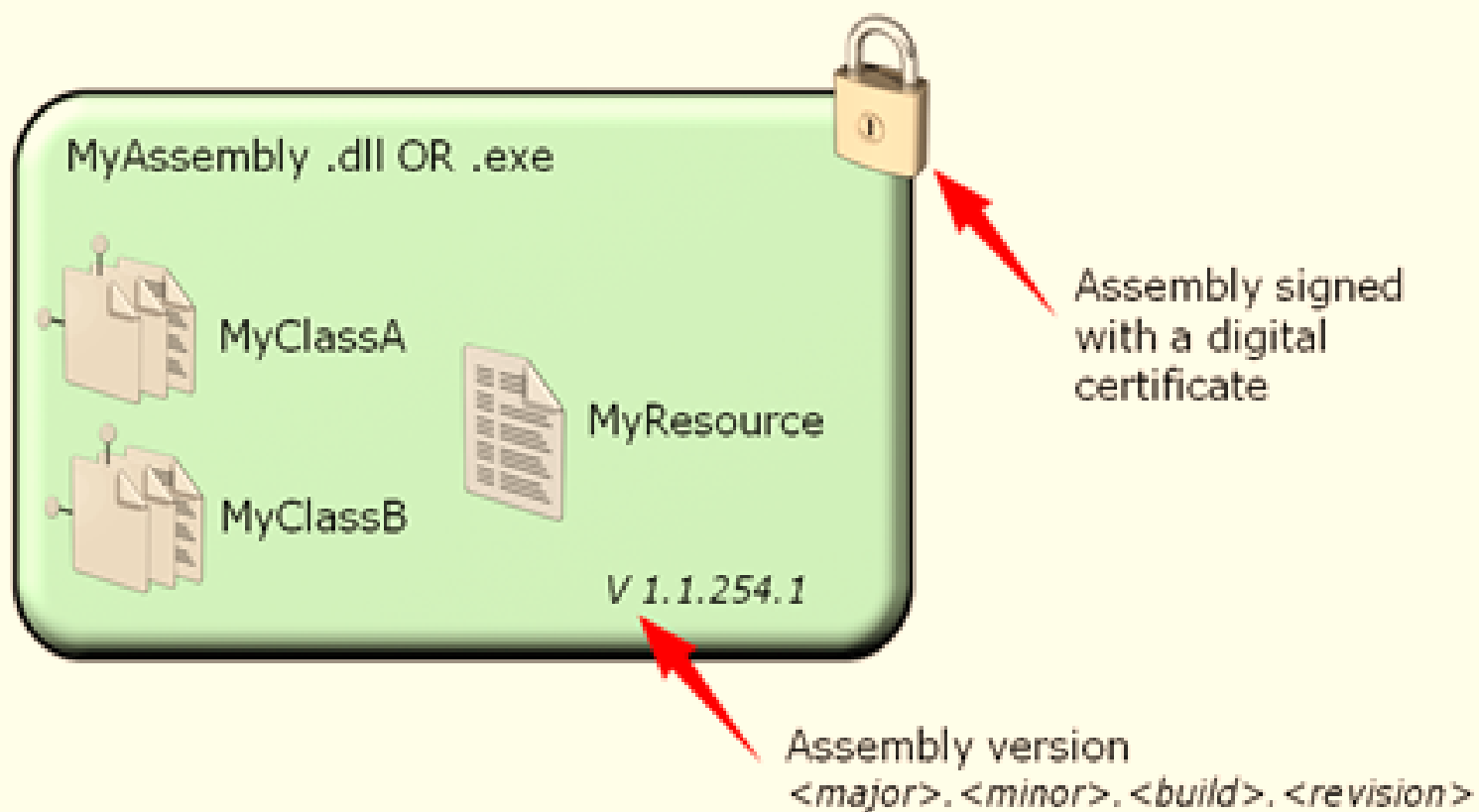
# Выполнение программы в среде CLR



# Сборка (Assembly)

Building blocks of .NET Framework applications

Collection of types and resources that form a logical unit of functionality





# ◆ Базовые операции ввода-вывода

- Класс Console
- Методы Write и WriteLine
- Методы Read и ReadLine

# Класс Console

- Позволяет осуществлять стандартный ввод-вывод и получать доступ к стандартному потоку ошибок
- Используется только для консольных приложений
  - Стандартное устройство для ввода - клавиатура
  - Стандартное устройства для вывода - экран
  - Стандартное устройство для вывода ошибок - экран

# Свойства консоли

- **ForegroundColor** – цвет текста

```
Console.ForegroundColor = ConsoleColor.Green;
```

- **BackgroundColor** – определяет цвет фона текста

```
Console.BackgroundColor = ConsoleColor.Green;
```

- Для изменения цвета всего окна :

```
Console.BackgroundColor = ConsoleColor.White;  
Console.Clear();
```

- **CapsLock** – свойство только для чтения, имеет тип bool и возвращает истину, если нажата соответствующая клавиша  

```
if (Console.CapsLock)  
    Console.WriteLine("Отключите <Caps Lock>!!");
```

- **NumberLock** – определяет, нажата ли клавиша *Number Lock*.

- **Title** – текстовая строка, которая отображает заголовок окна

```
Console.Title = "Новая жизнь";
```

# Методы Write и WriteLine

- Методы `Console.Write` и `Console.WriteLine` позволяют отображать информацию на консоль
  - Метод **WriteLine** добавляет в конце символ перевода строки
- Оба метода перегружены
- Можно использовать информацию о форматировании и список параметров
  - Форматирование текста
  - Форматирование чисел

# Использование управляющих последовательностей

- Управляющей последовательностью называют определенный символ, предваряемый обратной косой чертой

- \a            Звуковой сигнал
- \b            Возврат на шаг назад
- \f            Перевод страницы
- \n            Перевод строки
- \r            Возврат каретки
- \t            Горизонтальная табуляция
- \v            Вертикальная табуляция
- \\            Обратная косая черта
- \'            Апостроф
- \"            Кавычки

# Управление размером поля вывода

- Первым аргументом `WriteLine` указывается строка вида `{n, m}`
  - `n` определяет номер идентификатора из списка аргументов метода `WriteLine`,
  - `m` – количество позиций (размер поля вывода), отводимых под значение данного идентификатора.

```
static void Main()  
{  
    double x= Math.E;  
    Console.WriteLine("E={0,20}", x);  
    Console.WriteLine("E={0,10}", x);  
}
```

**Значение идентификатора выравнивается по правому краю**

# Управление размещением вещественных данных

- Первым аргументом `WriteLine` указывается строка вида `{n: ##.###}`
  - где `n` определяет номер идентификатора из списка аргументов метода `WriteLine`,
  - `##.###` определяет формат вывода вещественного числа.

```
static void Main()
{
    double x= Math.E;
    Console.WriteLine("E={0:##.###}", x);
    Console.WriteLine("E={0:.#####}", x);
}
```

# Управление форматом числовых данных

- Первым аргументом `WriteLine` указывается строка вида

**{n: <спецификатор>m}**

- n определяет номер идентификатора из списка аргументов метода `WriteLine`,
- <спецификатор> - определяет формат данных,
- m – количество позиций для дробной части значения идентификатора.
  - Параметры указаны в табл.1.1



# Методы Read и ReadLine

- Методы `Console.Read` и `Console.ReadLine` читают символы потока ввода
  - `Read` читает по одному символу
  - `ReadLine` читает строку символов

# Получение числовых значений

```
static void Main()  
{  
    string s = Console.ReadLine();  
    int x = int.Parse(s); //преобразование строки в число  
    Console.WriteLine(x);  
}
```

## ■ Или сокращенный вариант:

```
static void Main()  
{  
    //преобразование введенной строки в число  
    int x = int.Parse(Console.ReadLine());  
    Console.WriteLine(x);  
}
```

## ■ Для преобразования строкового представления в вещественное: методы `float.Parse()` ИЛИ `double.Parse()`.

## ◆ Практические рекомендации

- Комментарии
- Создание XML документации
- Обработка исключительных ситуаций
- Демонстрация: Создание и просмотр XML документации

# Комментарии

## ■ Комментарии необходимы

- Правильно документированное приложение помогает разработчику разобраться в структуре приложения

## ■ Комментирование строки текста

```
// Get the user's name  
Console.WriteLine("What is your name? ");  
name = Console.ReadLine( );
```

## ■ Комментирование блока текста

```
/* Find the higher root of the  
   quadratic equation */  
x = (...);
```

# Создание XML документации

```
/// <summary> The Hello class prints a greeting
/// on the screen
/// </summary>
class Hello
{
    /// <remarks> We use console-based I/O.
    /// For more information about WriteLine, see
    /// <seealso cref="System.Console.WriteLine"/>
    /// </remarks>
    public static void Main( )
    {
        Console.WriteLine("Hello, World");
    }
}
```

# Построение XML-отчета

- В командной строке: с параметром /doc :

```
csc XMLsample.cs /doc:XMLsample.xml
```

- Для просмотра созданного XML-кода:

```
type XMLsample.xml
```

- В среде разработки:

- в окне **Solution Explorer** для строки с именем проекта в контекстном меню выбрать **Properties**,
- в окне свойств, перейти на вкладку **Build**,
- в области **Output** активировать **XML documentation file** и в поле ввода указать имя XML-файла, например, hello.xml.

## Обработка исключительных ситуаций

[illegible]

# ◆ Компиляция, запуск и отладка

- Компиляция приложения
- Запуск приложения
- Демонстрация: компиляция и запуск C# программы
- Отладка



# Компиляция приложения

- **Опции командной строки компилятора**
- **Компиляция из командной строки**
- **Компиляция из оболочки Visual Studio.NET**
- **Поиск ошибок**

# Запуск приложения

- **Запуск из командной строки**

- Наберите имя запускаемого приложения

- **Запуск из Visual Studio**

- **Debug→Start Without Debugging**

# Отладка

- **Исключения и оперативная (Just-in-Time) отладка**
- **Отладчик Visual Studio**
  - Установка точек останова (breakpoints) и наблюдения за переменными
  - Пошаговое выполнение кода
  - Наблюдение за переменными и их изменение

# Команды для пошаговой отладки кода

## Начало отладки

- В меню Отладка (Debug)
  - Запуск (Start Debugging) – F5
  
- Шаг с заходом (Step Into) – F11
- Шаг с обходом (Step Over) – F10
- Шаг с выходом (Step Out) – Shift+F11

# Лабораторная работа 1: Создание простой программы на C#



## ◆ Организация ввода-вывода

- С#-программы выполняют операции ввода-вывода посредством потоков, которые построены на иерархии классов.
- Поток (stream) - это абстракция, которая генерирует и принимает данные.
- Потоки:
  - байтовые,
  - символьные,
  - двоичные

# Понятие о потоках

- **Класс Stream пространства имен System.IO - представляет байтовый поток и является базовым для всех остальных потоковых классов.**
- **Из класса Stream выведены байтовые классы потоков:**
  - FileStream - байтовый поток, разработанный для файлового ввода-вывода,
  - BufferedStream - заключает в оболочку байтовый поток и добавляет буферизацию;
  - MemoryStream - байтовый поток, который использует память для хранения данных.

# Байтовый поток

- Конструктор, который открывает поток для чтения и/или записи:

```
FileStream(string filename, FileMode mode)
```

- Версия конструктора позволяет ограничить доступ только чтением или только записью:

```
FileStream(string filename, FileMode mode,  
            FileAccess how)
```



# Байтовый поток. Пример

```
FileStream fileIn = new FileStream("text.txt",  
                                   FileMode.Open,  
                                   FileAccess.Read);  
FileStream fileOut = new FileStream("newText.txt",  
                                    FileMode.Create,  
                                    FileAccess.Write);  
  
int i;  
while ((i = fileIn.ReadByte()) != -1)  
{  
    fileOut.WriteByte((byte)i);  
}  
fileIn.Close();  
fileOut.Close();
```

# СИМВОЛЬНЫЙ ПОТОК

- **StreamReader** – содержит свойства и методы, обеспечивающие считывание символов из байтового потока

```
StreamReader fileIn = new StreamReader(new FileStream("text.txt",  
FileMode.Open, FileAccess.Read));
```

- **StreamWriter** – содержит свойства и методы, обеспечивающие запись символов в байтовый поток

```
StreamWriter fileOut=new StreamWriter(new FileStream("text.txt",  
FileMode.Create, FileAccess.Write));
```

```
StreamWriter(string name, bool appendFlag);
```

# Символьный поток. Пример

```
StreamReader fileIn = new StreamReader("text.txt",  
                                       Encoding.GetEncoding(1251));  
StreamWriter fileOut=new StreamWriter("newText.txt", false);  
string line;  
while ((line=fileIn.ReadLine())!=null) //пока поток не пуст  
{  
    fileOut.WriteLine(line);  
}  
fileIn.Close();  
fileOut.Close();
```