

Лекция 1.

Программирование

Преподаватели

Павел Мокшин

- Разработчик, компания Eastbanc Technologies
- Окончил бакалавриат и магистратуру ФИТ НГУ

Контакты

- **Skype:** pavel.mokshin
- **E-mail:** mokshinpv@gmail.com

Преподаватели

Анна Усова

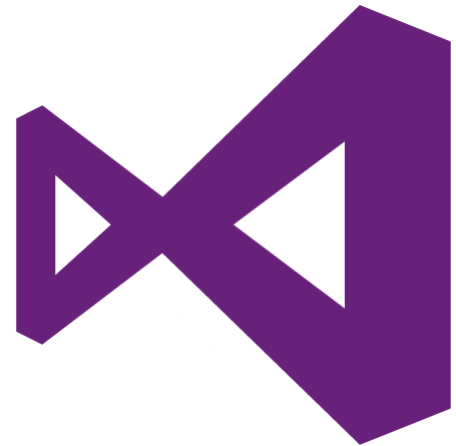
- Разработчик, компания Improve Group
- Опыт репетиторства с 2014 года

Контакты

- **Skype:** anitaand01
- **E-mail:** usova.anna.novosibirsk@gmail.com

Язык Java

- Курс посвящен изучению основ языка C#
- C# является одним из самых распространенных на практике современных языков программирования



Структура курса

- 14 теоретическо-практических занятий:
 - в будний день - 2 часа с перерывом на кофе-брейк
 - в выходной день – 3 часа с перерывом на кофе-брейк
- На каждом занятии изучаем новую теорию и тут же закрепляем ее на практике
- Домашнее задание:
 - небольшие задания к следующему занятию
 - большие задания на курс

Дополнительные задачи

- Часть задач является обязательной
- Звездочкой * помечены дополнительные условия задачи. Выполнение этих задач очень полезно, если вы хотите добиться прогресса быстрее и получить более уверенные знания

Что такое программирование?

- **Программирование** – это процесс написания программ
- **Программа** – последовательность команд для компьютера
- Программы пишутся на **языках программирования**, понятных человеку

Языки программирования

- Языки программирования более строгие и формальные, чем естественные языки
- Например, в них нельзя переставлять слова местами и допускать ошибки
- Машина выполняет в точности то, что написано в программе

Алгоритм

- Программа обычно реализует некоторый **алгоритм**
- **Алгоритм** – последовательность инструкций **исполнителю**, приводящая к желаемому результату за конечное число действий
- У алгоритма обычно есть **входные данные** – что имеем перед выполнением алгоритма; и **выходные данные** – результат выполнения алгоритма
- Примеры алгоритмов:
 - Инструкция как сварить пельмени или как приготовить любое другое блюдо
 - Как вычислить решения квадратного уравнения

Алгоритм варки пельменей

- **Исполнитель** – тот, кто будет готовить, команды предназначаются ему
- **Желаемый результат** – хорошо сваренные пельмени (не разваренные и не сырые)
- **Входные данные** – пельмени, кастрюля, вода, соль, плитка
- **Выходные данные** – сваренные пельмени

Шаги алгоритма варки пельменей

1. Наливаем воду в кастрюлю до середины
2. Ставим кастрюлю на плитку
3. Включаем плитку
4. Добавляем соль по вкусу
5. Ждем, пока вода не закипит, после этого на шаг 6
6. Добавляем пельмени
7. Ждем, пока вода снова не закипит
8. Запоминаем время, когда закипела вода
9. Ждем пока пройдет 5 минут от запомненного времени
10. Выключаем плиту
11. Снимаем пельмени

Алгоритм решения квадратного ур-я

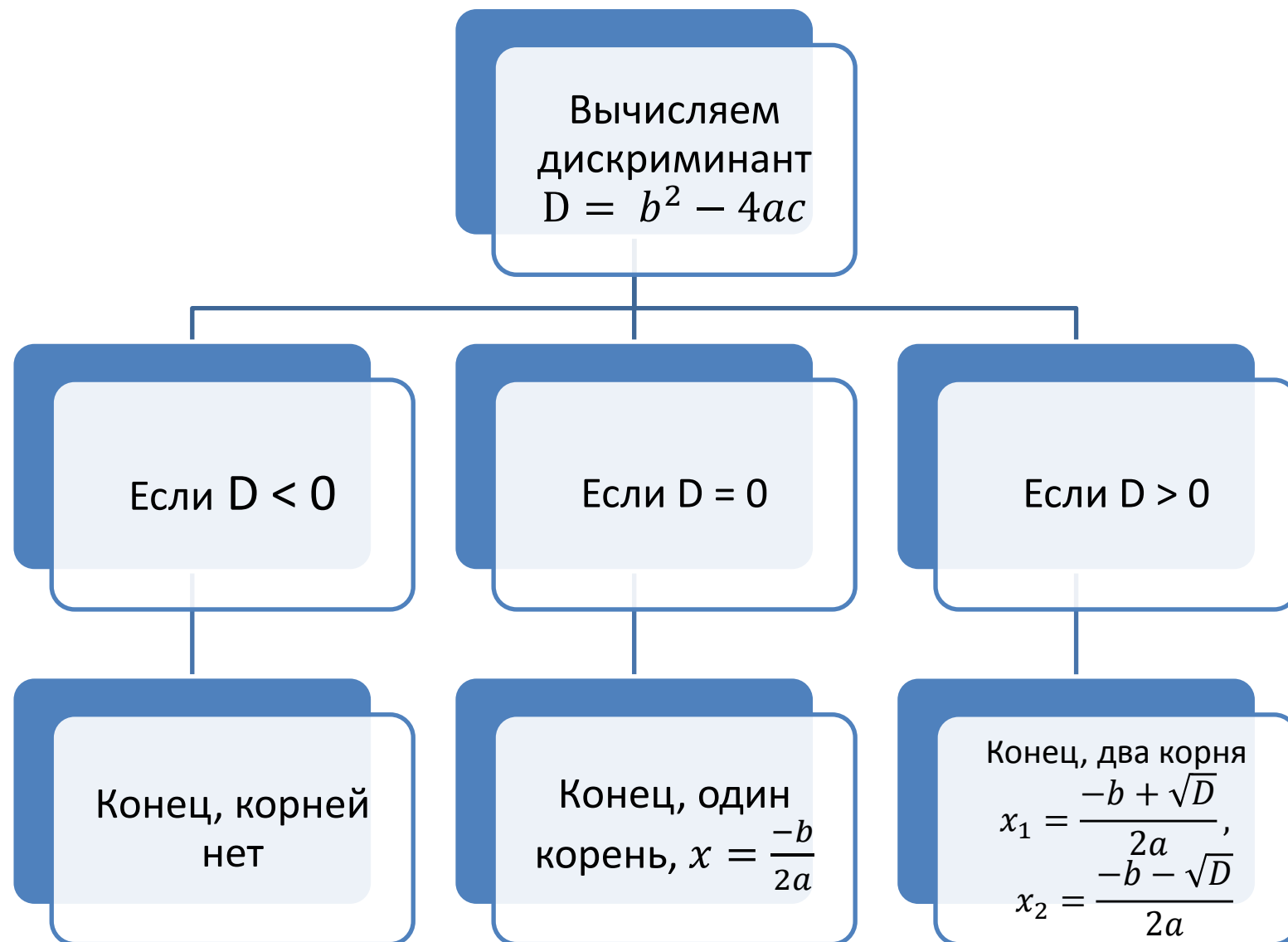
- **Исполнитель** – тот, кто будет решать уравнение
- **Желаемый результат** – количество и значения корней уравнения
- **Входные данные** – квадратное уравнение
- $ax^2 + bx + c = 0$

Шаги решения квадратного ур-ия

1. Вычисляем дискриминант по формуле
 - $D = b^2 - 4ac$
2. Если $D < 0$, то корней нет, конец алгоритма
3. Если $D = 0$, то один корень, $x = \frac{-b}{2a}$, конец алгоритма
4. Иначе ($D > 0$) – два корня,

$$x_1 = \frac{-b + \sqrt{D}}{2a}, \quad x_2 = \frac{-b - \sqrt{D}}{2a}$$

Шаги решения квадратного ур-ия



Что можно увидеть?

- Шаги исполняются последовательно, один за другим
- В некоторых случаях алгоритм завершается досрочно, либо переходит на другой шаг
- Некоторые шаги зависят от условий, например, результат разный в зависимости от дискриминанта
- Некоторые шаги выполняются несколько раз, либо ждут пока выполнится некоторое условие, например, мы периодически смотрим на время и ждем пока пельмени сварятся
- Часто приходится запоминать некоторые данные, чтобы потом их использовать, например, время когда закипела вода, либо значение дискриминанта

Пример программы

- ```
double a = 1; // в переменные положили коэффициенты
double b = -5; // уравнения $ax^2 + bx + c = 0$
double c = 6;
double d = b * b - 4 * a * c; // посчитали дискриминант

if (d < 0)
{
 // код выполнится, если дискриминант < 0
 Console.WriteLine("Корней нет"); // печать в консоль
}
else
{
 // иначе
 double x1 = (-b + Math.Sqrt(d)) / (2 * a);
 double x2 = (-b - Math.Sqrt(d)) / (2 * a);
 Console.WriteLine(x1);
 Console.WriteLine(x2);
}
```



# Пример программы

- Простейшая программа на C#  
`Console.WriteLine("Hi!");`  
`Console.WriteLine(3 + 5);`
- Программа состоит из **команд (инструкций)**
- Команда `Console.WriteLine` печатает в консоль то, что ей передали в скобках (**аргументы/параметры**), а затем вставляет **перевод строки** (Enter)
- Первая команда в этом примере печатает строку `Hi!`, а вторая — число 8

# Пример программы

- Простейшая программа на C#:  
`Console.WriteLine("Hi!");`  
`Console.WriteLine(3 + 5);`
- Общие моменты для многих языков программирования:
  - программы пишутся на английском языке
  - исполнение команд идет **последовательно** – одна за другой сверху вниз
  - командам для работы передается результат вычисления выражений, а не сами выражения (будет напечатано 8, а не 3 + 5)
  - не важно количество пробелов, т.е. один пробел и много пробелов – это одно и то же

# Несколько команд в одной строке

- Допускается писать несколько команд в одной строке, но это плохой стиль – код хуже читается:
- `Console.WriteLine("Hi!"); Console.WriteLine(3 + 5);`  
`Console.WriteLine("OK"); Console.WriteLine(44 - 22);`
- В этом случае команды исполняются слева направо, потом происходит переход к следующей строке
- Hi!  
8  
OK  
22

# Пример программы

- Простейшая программа на C#:  
`Console.WriteLine("Hi!");`  
`Console.WriteLine(3 + 5);`
- Конкретно для C#(и многих других языков):
  - Команды отделяются друг от друга точкой с запятой
  - Важен **регистр** символов: заглавные и строчные буквы считаются различными

# Регистр символов

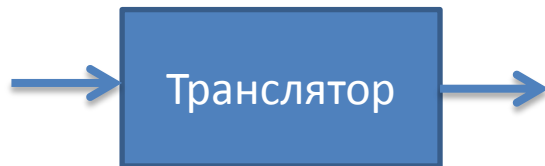
- Команды считаются разными:
  - `Console.WriteLine(3);`
  - `CONSOLE.WRITELINE(3);`
  - `CoNsoLE.WRiteLINE(3);`

# Машинный код

- Компьютер не понимает программы, написанные на языках программирования, а понимает только **машинный код** (например, exe файлы для Windows)
- **Машинный код** – это команды для процессора. А данные, с которыми работает программа, хранятся в оперативной памяти
- Машинный код не понятен человеку, и человек вряд ли сможет писать программы прямо на нем
- Чтобы перевести текст программы (**исходный код**) в машинный код, используются **программы-трансляторы**

Исходный код

```
print(1);
print(2);
```



Машинный код

010101010101

Исполнение  
операционной  
системой

# Виды трансляторов



- Анализирует весь исходный код, проверяет на наличие ошибок, и после этого переводит текст программы целиком в машинный код
- Обычно на выходе получается файл с машинным кодом, т.е. перевод выполняется 1 раз
- Работает с исходным кодом построчно: берет очередную инструкцию из исходного кода, переводит ее в машинный код и тут же исполняет
- На выходе не создается файл с машинным кодом, а при каждом запуске программы, перевод выполняется заново

# Разделение языков по типу транслятора

- По виду используемого транслятора, языки делятся на **компилируемые** и **интерпретируемые**
- С# использует комбинацию этих подходов, рассмотрим это позже



# Среды разработки (IDE)

- Для повышения продуктивности процесса разработки были созданы **среды разработки (IDE)**
- Они включают в себя:
  - текстовый редактор
  - средства для компиляции и запуска программ
  - средства отладки (поиска ошибок) и др.
- Для разработки на C# используется **Visual Studio** от Microsoft
- Собственно, сам язык C# тоже разработали Microsoft

# Немного о C#

- C# – язык для платформы .NET
- Под эту платформу можно писать на многих языках: C#, F#, Visual Basic, C++, Ruby, Python и т.д. Но основной все же C#
- Потенциально язык кроссплатформенный (может запускаться на всех операционных системах), но по факту это не совсем так
- Но Microsoft идет в эту сторону
- C# позволяет разрабатывать приложения любых видов:
  - десктопные (оконные)
  - мобильные (для платформы Android)
  - веб-приложения

# Особенности C#

- На C# не нужно самостоятельно освобождать память, как при использовании C и C++
- Этим занимается **сборщик мусора (Garbage Collector)**, который отслеживает данные, которые стали неиспользуемыми, и освобождает занимаемую ими память
- За счет этого разрабатывать на C# проще и дешевле – меньше возможности написать программу, которая падает или содержит утечки памяти

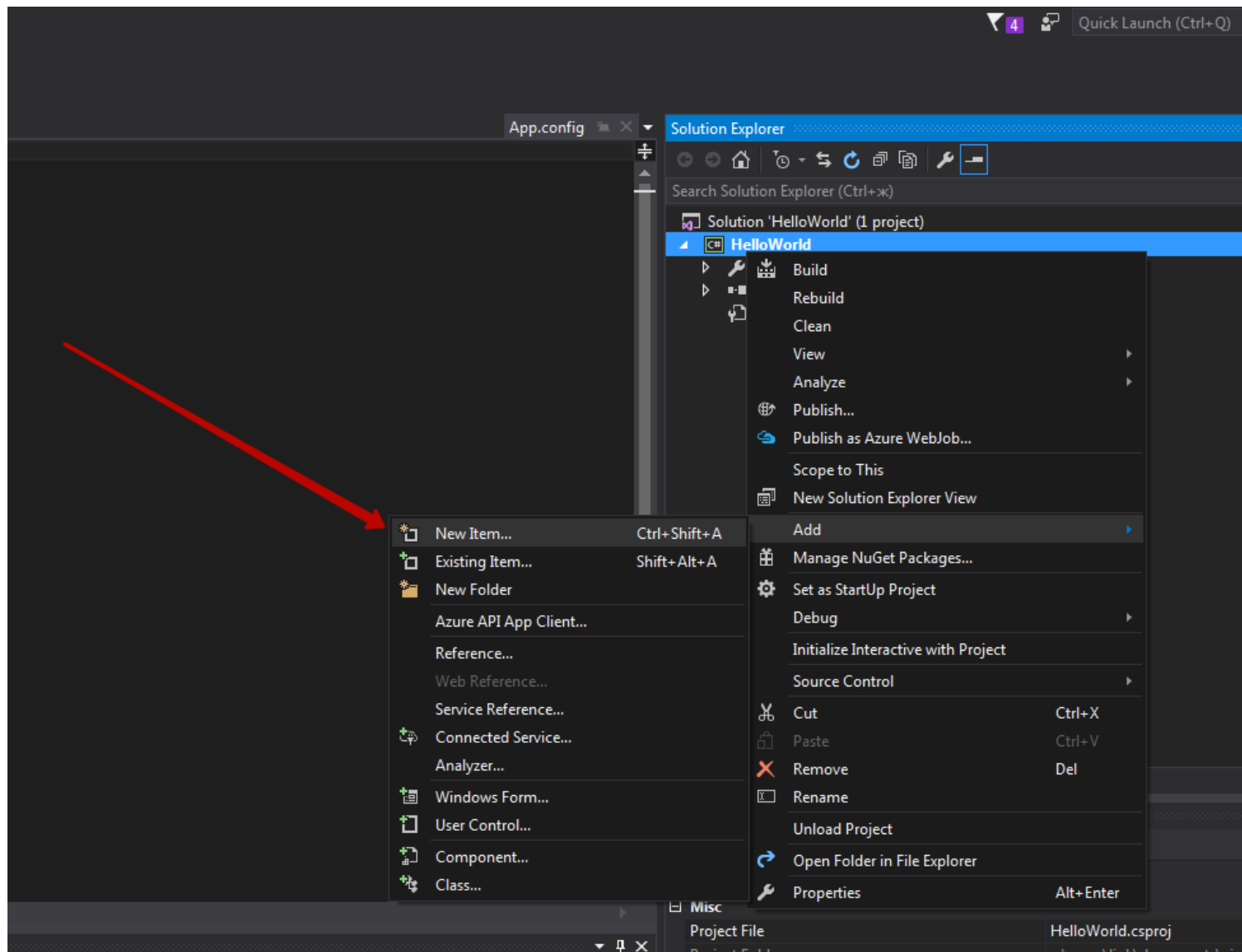
# .NET Framework

- Для того, чтобы можно было запускать программы на C#, нужен **.NET Framework**
- Некоторая его версия есть по умолчанию в Windows, версия зависит от версии ОС
- Любую версию .NET Framework можно скачать с сайта Microsoft
- Текущая версия .NET Framework - 4.6.2
- CLR (Common Language Runtime) - это **среда исполнения** платформы .NET

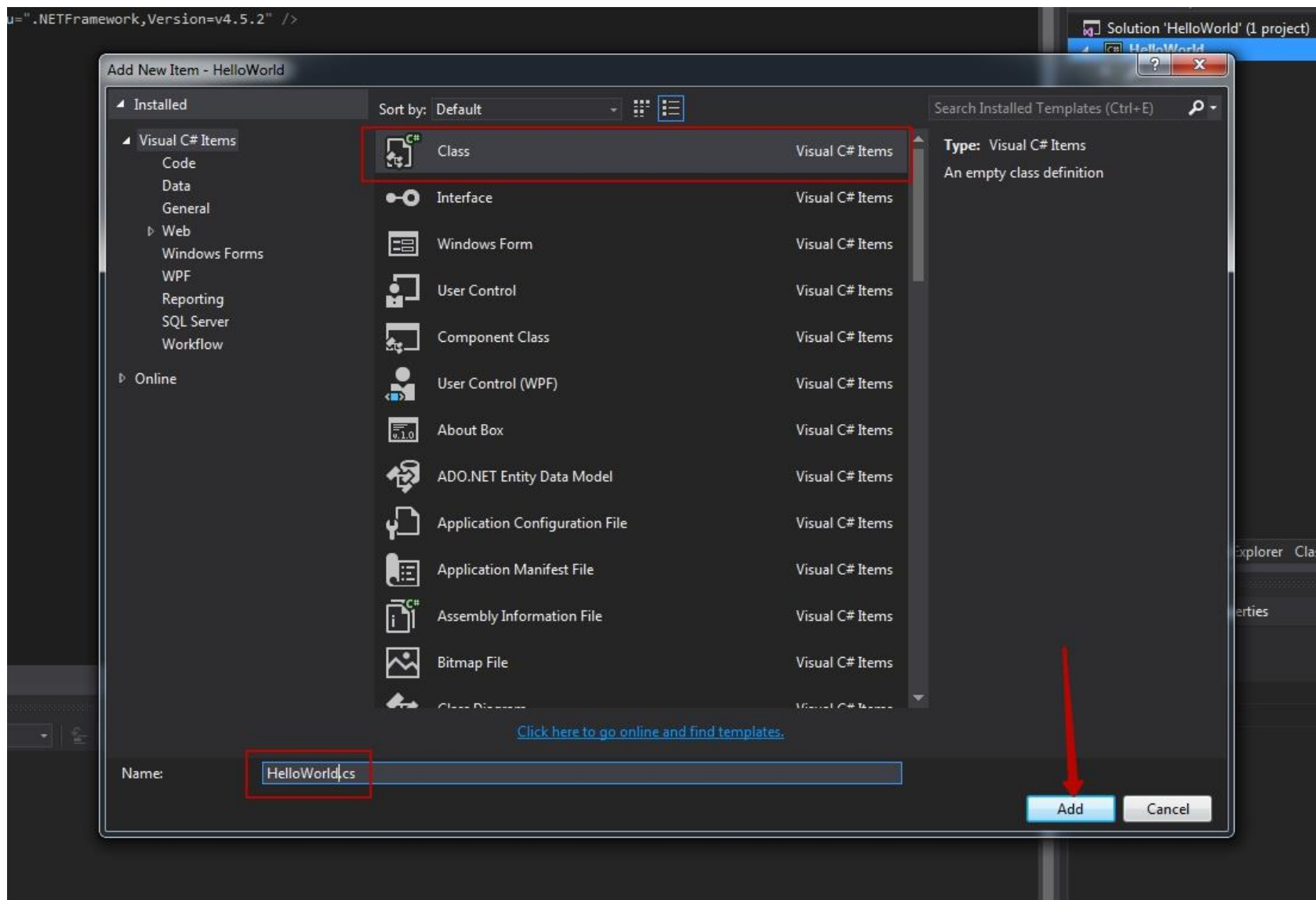
# Код пишется в классах

- Весь код программ на C# находится внутри **классов** (понятие класса рассмотрим позже)
- Обычно каждый класс помещают в отдельном файле
- Добавим в наш проект файл с классом
- Назовем этот класс **HelloWorld**, классы следует называть с заглавной буквы, каждое последующее слово – тоже с заглавной буквы
- Обычно программы состоят из нескольких классов

# Добавление файла с С# кодом – шаг 1



# Добавление файла с С# кодом – шаг 2



# Объявление класса

- `using System;`

```
namespace HelloWorld
{
 class HelloWorld
 {

 }
}
```

- Строк, начинающихся с `using`, может быть больше
- Что такое `using` и `namespace` рассмотрим позже
- Далее идет ключевое слово `namespace` (пространство имен), а после него – его имя. Оно берется само из названия проекта



# Объявление класса

- `using System;`

```
namespace HelloWorld
{
 class HelloWorld
 {

 }
}
```

- Затем идет **блок кода** - он заключается в фигурные скобки {}
- Блоки кода показывают вложенность кода
- Если сравнить с Pascal, то там вместо фигурных скобок используются слова `begin` и `end`

# Объявление класса

- `using System;`

```
namespace HelloWorld
{
 class HelloWorld
 {

 }
}
```

- Внутри скобок `namespace` идет ключевое слово `class`, а после него – имя класса.
- Имя класса следует делать одинаковым с именем файла

# Простейшая программа

- ```
class HelloWorld
{
    static void Main()
    {
        Console.WriteLine("Hello world!");
    }
}
```

Здесь и далее пропускаю
using и namespace

- Внутри класса могут быть объявлены **функции**
- **Функция** – это блок кода, имеющий имя, и который содержит исполняемые команды
- Каждая программа на C# начинается исполнение с функции **Main** некоторого класса
- То есть в каждой программе в одном из классов нужно объявить такую функцию

Простейшая программа

- ```
class HelloWorld
{
 static void Main()
 {
 Console.WriteLine("Hello world!");
 }
}
```
- Внутри функций пишется сам код — последовательность команд через точку с запятой
- Внутри каждой фигурных скобок для форматирования текста программы нужно добавлять по одному TAB
- Это очень сильно повышает читаемость кода, и сразу видно какая фигурная скобка к чему относится

# Простейшая программа

- ```
class HelloWorld
{
    static void Main()
    {
        Console.WriteLine("Hello world!");
    }
}
```
- Вообще, весь код можно разделить на 2 части:
 - Описательная часть (объявления) – например, объявляется класс `HelloWorld` с функцией `Main`. Этот код не исполняется, он описывает программу
 - Исполняемая часть. Это содержимое функции `main` – строка `Console.WriteLine("Hello world!");`

Простейшая программа

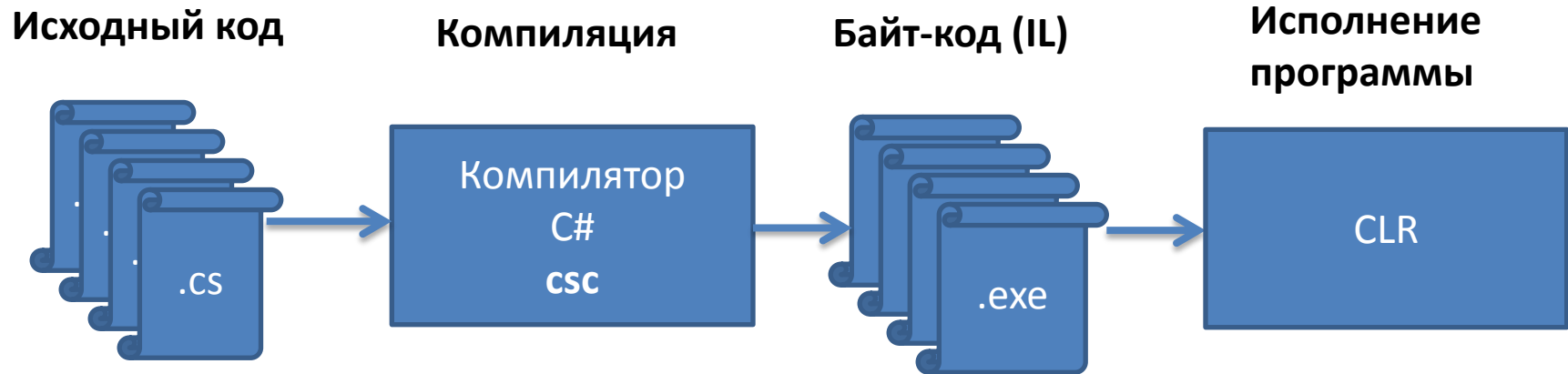
- ```
class HelloWorld
{
 static void Main()
 {

 }
}
```
- Этот пример кода обязателен для любой вашей программы. Свой код нужно писать внутри функции Main
- Этот код должен быть написан в точности так же, как здесь, с таким же регистром символов
- Единственные исключения – имя класса HelloWorld
- Вы можете дать классу любое имя, но тогда и файл желательно переименовать соответствующим образом

# Другие варианты функции Main

- В C# кроме этого варианта допустимы еще другой вариант написания функции Main
- `static void Main()`  
`{`  
`}`
- Другой вариант:
- `static void Main(string[] args)`  
`{`  
`}`
- Еще перед словом `static` бывает слово `public`

# Упрощенная модель исполнения C#



- Файлы с исходным кодом имеют расширение `.cs`
- После компиляции для каждого класса создается файл с расширением `.exe` или `.dll`, в котором содержится байт-код IL
- CLR может исполнять файлы `.exe` и `.dll`
- Заметим, что это не обычные `.exe`, которые получаются, например, при компиляции C++

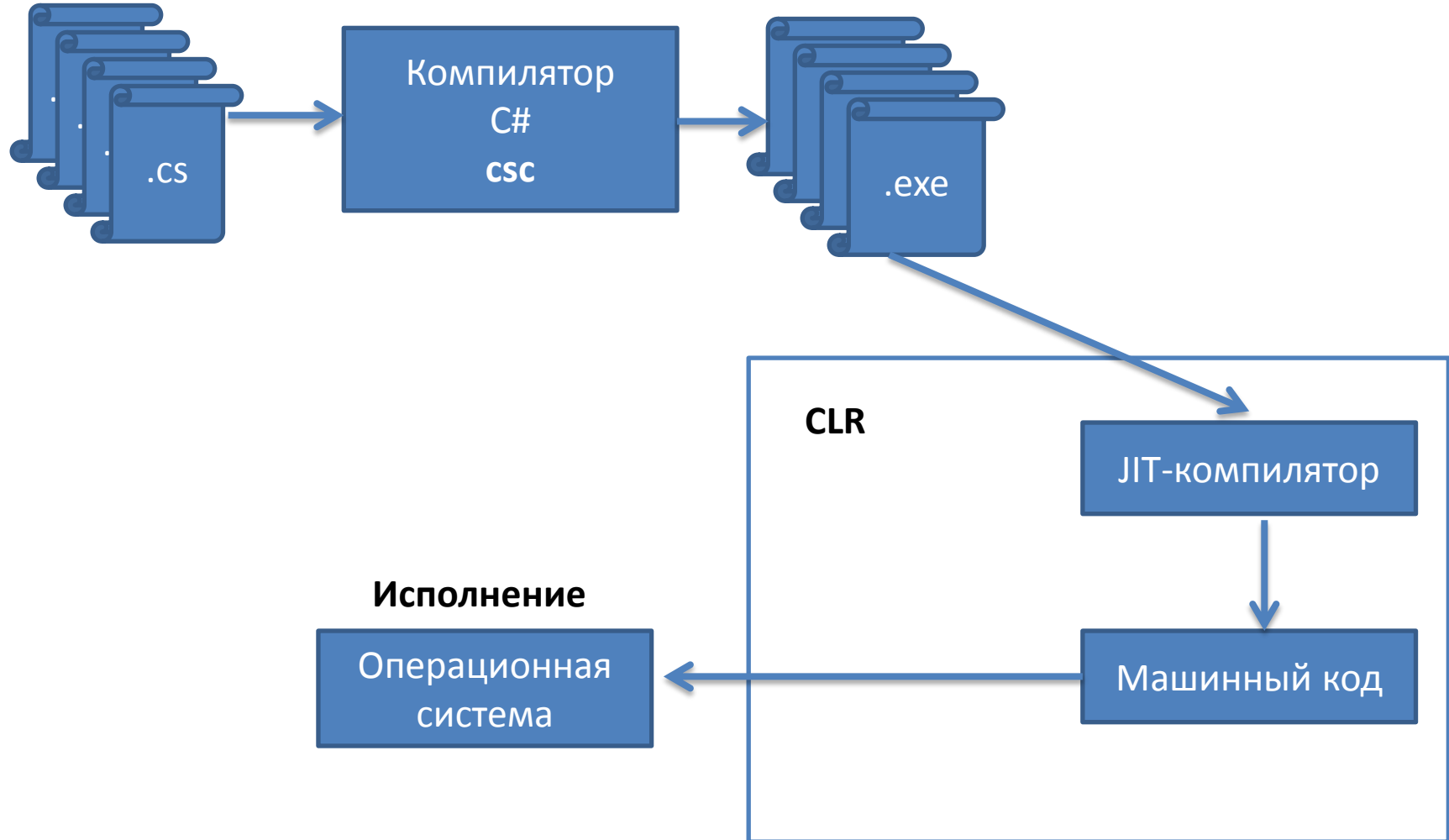


# Подробная модель исполнения C#

Исходный код

Компиляция

Байт-код (IL)

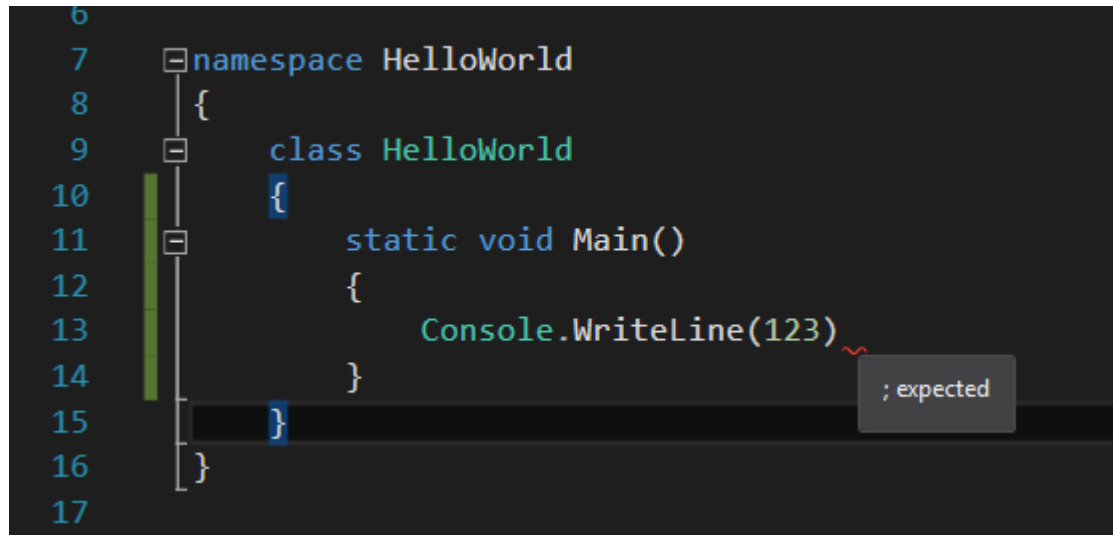


# JIT-компиляция

- JVM не выполняет файлы с байт-кодом напрямую, а сначала компилирует их в машинный код при помощи JIT-компилятора (Just-In-Time compiler) во время исполнения программы
- За счет этого скорость исполнения кода становится сравнимой с C/C++
- За счет того, что JIT-компиляция происходит уже во время исполнения программы, JIT-компилятор может учитывать статистику исполнения программы, и производить более эффективный машинный код по сравнению с языками со статической компиляцией

# Ошибки компиляции

- Если код не будет удовлетворять правилам языка, то компилятор будет подчеркивать часть кода красным и выдавать ошибку
- Настаиваем, чтобы вы учились самостоятельно понимать в чем там ошибка и исправлять её
- Если навести курсор на ошибку, то покажется подсказка с текстом ошибки. Этот текст часто уже понятен. Если вы не знаете, что это значит, попробуйте перевести и погуглить – все ошибки компиляции хорошо гуглятся



```
6
7 namespace HelloWorld
8 {
9 class HelloWorld
10 {
11 static void Main()
12 {
13 Console.WriteLine(123)
14 }
15 }
16 }
17
```

; expected

# Поиск в интернете

- При поиске информации по программированию в интернете лучше всегда пользоваться именно **Google**
- Яндекс и другие поисковики обычно не выдают нужные ссылки по запросам, связанным с программированием



# Типы данных

- **Тип данных** – данные имеющие одинаковую структуру
- Примеры типов:
  - целые числа
  - вещественные числа (десятичные дроби)
  - строки
  - символы
- В C# существует большое количество встроенных типов и можно создавать свои

# Литерал

- **Литерал** – это значение типа, которое встречается непосредственно в коде
- Примеры литералов:
  - 5 // целое число 5
  - 3.4 // вещественное число 3.4
  - “Hello” // строка Hello
- Литерал вещественный, если есть десятичная точка.  
Литералы строк заключаются в двойные кавычки

# Переменные

- В программах (и жизни) часто возникает необходимость запоминать и хранить какие-то значения, которые будут использоваться дальше
- Например, можно запоминать результаты вычислений, а потом читать их
- Для этого используются **переменные**

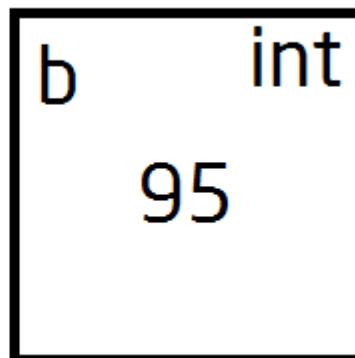
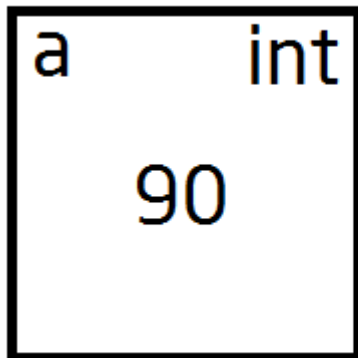
# Переменные

- **Переменная** – ячейка памяти компьютера, которая может хранить в себе одно значение заданного типа
- Переменные в C# имеют **название** и **тип**, а также часто содержат в себе **значение**
- Пример:
- `int a = 3 * 30;      // переменная целого типа int`  
`int b = a + 5;`  
`Console.WriteLine(a + b);`



# Переменные

- **Переменная** – ячейка памяти компьютера, которая может хранить в себе одно значение заданного типа
- Переменные в C# имеют **название** и **тип**, а также часто содержат в себе **значение**
- `int a = 3 * 30;`      `// переменная целого типа int`  
`int b = a + 5;`
- Переменную можно сравнить с коробкой



В переменную нельзя  
«класть» значения  
несовместимого типа

# Переменные

- `int a = 3 * 30;      // переменная целого типа int`  
`int b = a + 5;`  
`Console.WriteLine(a + b);`
- **Объявление** переменной:
- `ТипПеременной` имяПеременной;
- Переменной можно сразу **присвоить** значение при объявлении, как показано в примере. Тогда это называется **определением переменной**

# Переменные

- Можно сначала объявить переменную, а лишь затем задать ей значение
- `int a;`            `// переменная целого типа int`
- `int b = a + 5;`    `// ошибка компиляции –`  
                      `// переменной a еще не присвоено`  
                      `// значение`
- `a = 3;`            `// все ОК`

# Оператор присваивания

- Переменная в левой части оператора получает значение результата выражения в правой части
- Совершенно нормальное дело:

```
int x = 5;
```

```
x = x + 6;
```

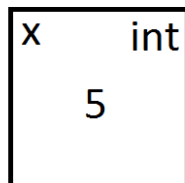
Если переменная используется в левой части от присваивания, то это значит положить в неё

В остальных случаях – это получение копии значения переменной

- Сначала вычисляется правая часть: из переменной `x` вытаскивается 5 и прибавляется к литералу 6, получается 11
- Затем переменной `x` присваивается новое значение 11

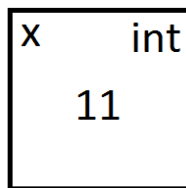
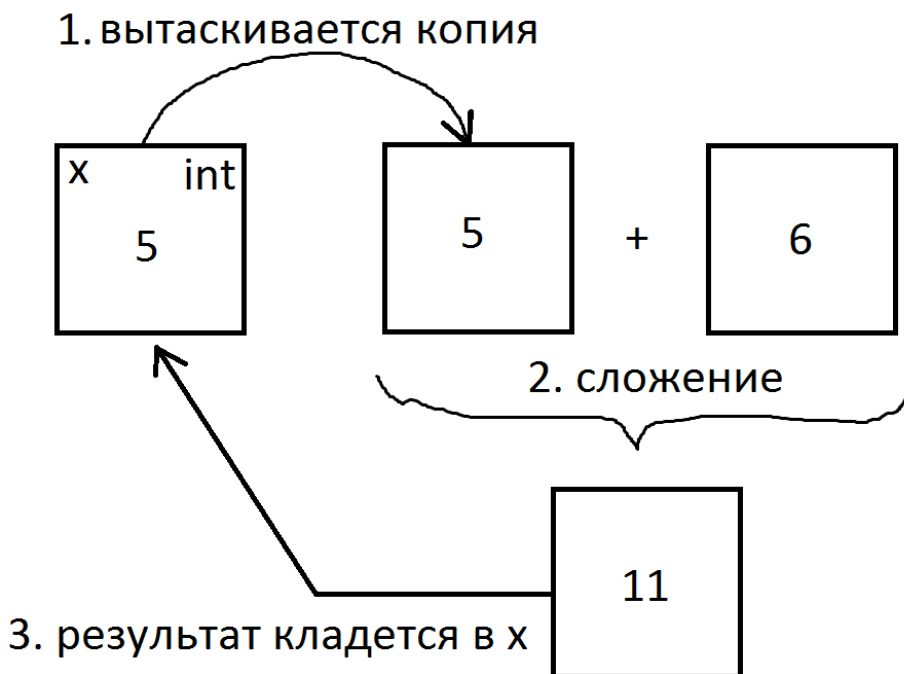
# Оператор присваивания

```
int x = 5;
x = x + 6;
```



После первой строки кода –  
есть переменная int x, в ней  
лежит значение 5

5 и 11 – временные  
переменные.  
Они используются для  
промежуточных  
вычислений



После всего, в переменной x  
будет 11

# Имена переменных

- Важен регистр символов: `variable`, `Variable` и `VARIABLE` – это разные имена переменных
- **Допустимые имена:**
- Первый символ – буква, либо символ подчеркивания `_`
- Последующие символы – буквы, знак подчеркивания или цифры
- Языком допускается использовать русские символы, но этого лучше не делать

# Ключевые слова

- В каждом языке программирования есть **ключевые (зарезервированные) слова** – слова, которые имеют специальный смысл. Эти слова нельзя использовать в качестве имен переменных
- Среда разработки обычно выделяет их другим **цветом** и шрифтом

# Ключевые слова C#

- **Имена типов:** int, short, long, float, double, byte, bool, char
- **Циклы:** for, while, foreach, in, do, break, continue
- **Ветвление:** if, else, switch, case, default
- **Другие:** public, private, protected, internal, readonly, sealed, new, override, virtual, abstract, interface, class, struct, return, void, true, false, null, static, lock, try, finally, catch, volatile, using, throw, this, base, is, as, var
- И некоторые другие



# Хорошие имена переменных

- Важнейшая задача программиста – обеспечить хорошую читаемость своего кода
- Поэтому нужно давать переменным понятные имена, который отражают роль переменной. Не нужно стесняться делать это, даже если название покажется длинным

# Нотация «верблюд»

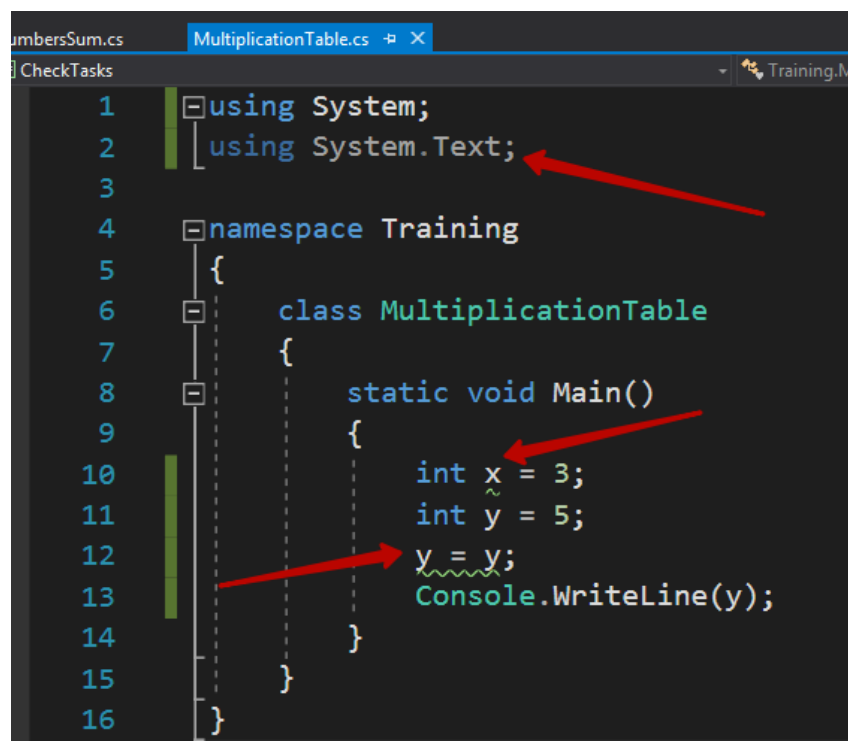
- В C# принято давать переменным имена в соответствии с нотацией «верблюд»
- Т.е. первое слово в названии переменной должны быть со строчной буквы, а последующие слова – с заглавной
- Пример: `numberOfPeople`, `helloWorld`
- Для имен классов и функций используется эта же нотация, только классы и функции начинаются с заглавной буквы
- Пример: `HelloWorld`, `CircleCalculation`

# Предупреждения компилятора

- Бывает, что код компилируется, но при этом все равно не идеален
- Это бывает в следующих случаях:
  - Код написан неоптимально – использовали какую-то длинную конструкцию вместо более простой
  - Есть ненужный код или переменные
  - Код содержит потенциальную логическую ошибку. Например, случайно написали  $x = x$ ; вместо  $x = x1$ ;
  - Или код не соответствует соглашениям именования. Например, использовали не верблужью нотацию

# Предупреждения компилятора

- В таких случаях компилятор выдает **предупреждение (warning)**
- Среда разработки выделяет их зеленым или бледным цветом
- Программа должна быть без warning'ов, поправляйте их



```
1 using System;
2 using System.Text;
3
4 namespace Training
5 {
6 class MultiplicationTable
7 {
8 static void Main()
9 {
10 int x = 3;
11 int y = 5;
12 y = y;
13 Console.WriteLine(y);
14 }
15 }
16 }
```

# Задача 1

- Написать программу, которая вычисляет какое-нибудь сложное выражение, а затем печатает результат в консоль
- Использовать объявление целочисленных переменных типа `int`

# Задача на дом «Создать проект»

- Повторить материал лекции
- Создать новый проект и написать в нем программу – либо из задачи 1, либо какую-нибудь другую