

Лекция 2: Ваша первая программа. Знакомство с C# и .NET.

Добро пожаловать в мир C#!

На прошлой лекции мы говорили о программировании в целом: об алгоритмах, парадигмах и о том, как компьютер понимает наши команды. Сегодня мы переходим от теории к практике и сделаем первый шаг в мир C#.

«Программирование — это не о наборе кода, а о мышлении.» — Рич Хикки.

1. C#: Современный, мощный и универсальный

Давайте поближе познакомимся с языком, который нам предстоит изучать.

Немного истории

- **2000 год:** Компания Microsoft под руководством Андерса Хейлсберга представляет миру язык C#. Его цель — создать современный, простой в изучении и одновременно мощный язык для новой платформы .NET.
- **Название:** "C#" ("Си-шарп"). Символ `#` — это не решетка, а музыкальный диез, обозначающий повышение на полтона. Это символизирует эволюцию по отношению к языкам C и C++.
- **Эволюция:** За более чем 20 лет C# прошел огромный путь от языка для Windows-приложений до одного из самых популярных и востребованных языков в мире, который постоянно развивается и добавляет новые возможности.

Философия C#

C# создавался как язык, в котором сочетаются мощь C++ и простота Visual Basic. Его ключевые принципы:

- **Объектно-ориентированность:** Все сущности в C# — это объекты.
- **Типобезопасность:** Компилятор строго следит за типами данных, чтобы вы не смогли, например, случайно сложить строку и число. Это предотвращает множество ошибок.
- **Универсальность:** На C# можно написать что угодно.
- **Современность и развитие:** Язык постоянно обновляется, добавляя удобные функции для разработчиков.

Где применяется C#?

- **Игры:** Один из лидеров в геймдеве благодаря движку **Unity**. От инди-игр до AAA-хитов.
- **Десктопные приложения:** Приложения для Windows (Windows Forms, WPF).
- **Веб-приложения и сервисы:** Бэкенд для сайтов и API на платформе **ASP.NET Core**.
- **Мобильные приложения:** С помощью **Xamarin (MAUI)**.
- **Облачные сервисы:** Интеграция с **Microsoft Azure**.

«C# во многих отношениях превосходит C++ и Java. Он похож на то, каким мог бы быть Java, если бы разработчики этого языка забыли о обратной совместимости и просто работали над улучшением его.» — Бен Альбахари.

C# и его соседи: ключевые отличия

Чтобы лучше понять C#, давайте кратко сравним его с другими популярными языками.

Аспект	C#	C++	Java	Python
Управление памятью	Автоматическое (сборщик мусора). Можно не беспокоиться.	Ручное. Программист сам выделяет и освобождает память. Сложнее, но дает полный контроль.	Автоматическое (сборщик мусора). Как в C#.	Автоматическое (сборщик мусора).
Производительность	Очень высокая. Близко к C++.	Максимальная. Плата за полный контроль.	Высокая.	Ниже, чем у C#/C++/Java.
Простота изучения	Относительно простой. Строгая типизация помогает избегать ошибок.	Сложный. Множество "подводных камней".	Относительно простой. Похож на C# в основе.	Очень простой. (Но изучение алгоритмизации, а след. и других языков будет сложным)
Кроссплатформенность	Да (благодаря .NET Core/.NET 5+).	Да (но нужно перекомпилировать под каждую ОС).	Да ("write once, run anywhere").	Да.
Основная ниша	Универсальный язык для игр, веба, предприятий.	Системное программирование, игры, высокопроизводительные приложения.	Корпоративные приложения, Android-разработка.	Data Science, AI/ML, веб-бэкенд, скрипты.

Вывод: C# — это золотая середина между производительностью C++ и простотой Python, с мощной экосистемой, не уступающей Java.

«В C++ трудно избежать выстрела себе в ногу, а в C# это сделать практически невозможно, так как вам просто не дадут в руки пистолет.» — Programming folklore

2. .NET: Платформа, на которой все работает

C# — это язык, а .NET — это платформа, среда выполнения для этого языка. Представьте, что C# — это английский язык, а .NET — целая страна (Великобритания или США), со своими законами и инфраструктурой, где на этом языке говорят.

.NET (ранее .NET Framework, сейчас просто .NET или .NET Core) — это:

- **Большая библиотека готовых классов (FCL - Framework Class Library):** Коллекция тысяч полезных "кирпичиков" для всего: работы с файлами, сетью, базами данных, графикой. Зачем писать велосипед, если можно его взять готовым?
- **Common Language Runtime (CLR):** Та самая "виртуальная машина", о которой мы говорили в прошлой лекции. Она управляет выполнением программы: загружает код, компилирует его Just-In-Time (JIT), занимается сборкой мусора.

Важно: Современный .NET (начиная с .NET 5) — **кроссплатформенный**. Ваши программы будут работать на Windows, Linux и macOS.

3. Установка и настройка: Ваш рабочий инструмент — Visual Studio

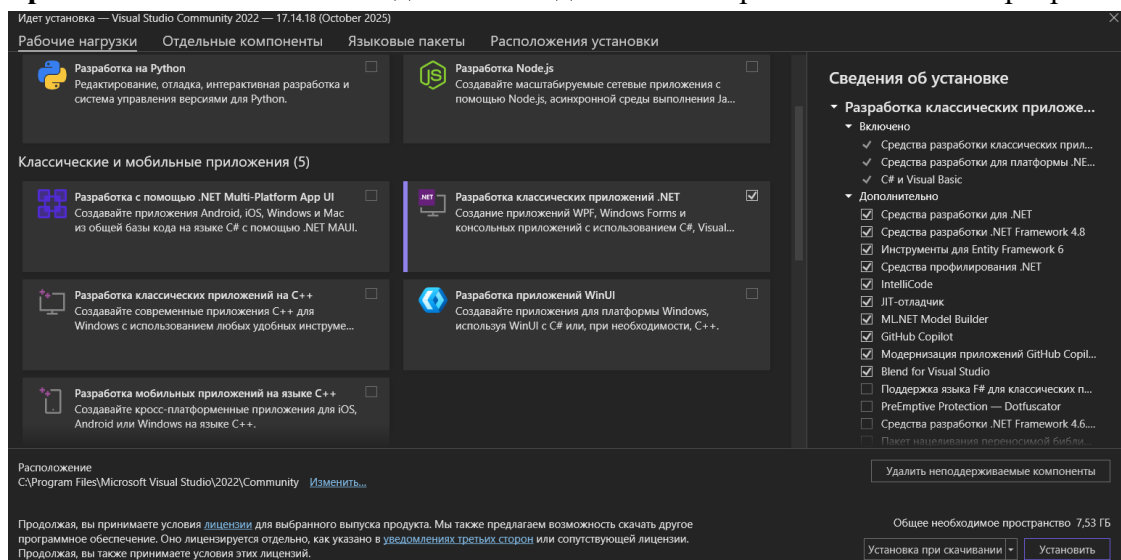
Чтобы начать программировать, нам нужна среда разработки (IDE). Мы будем использовать **Visual Studio** — мощнейшую и самую популярную IDE для .NET.

Шаг 1: Загрузка

1. Зайдите на сайт visualstudio.microsoft.com.
2. Скачайте **бесплатную** версию **Visual Studio**. Её возможностей нам хватит с лихвой.

Шаг 2: Установка

1. Запустите установщик.
2. Вам будет предложено выбрать **рабочие нагрузки**. Это наборы инструментов для разных типов разработки.
3. **ВАЖНО:** Выберите рабочую нагрузку **"Разработка классических приложений .NET"**. Этого достаточно для наших первых консольных программ.



4. Нажмите "Установить" и дождитесь окончания процесса. Это может занять время.

Шаг 3: Первый запуск

1. При первом запуске Visual Studio предложит вам войти с учетной записью Microsoft. Это можно пропустить.
2. Выберите цветовую тему. Это дело вкуса! :)
3. Поздравляю, вы в Visual Studio!

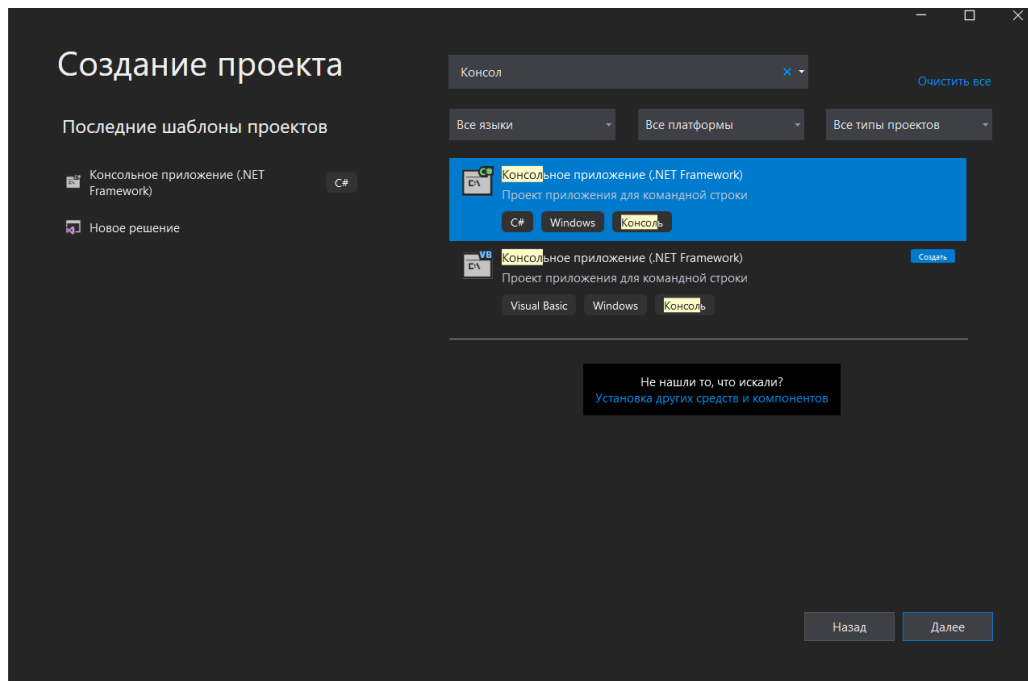
«Давайте сменим традиционный подход к построению программ: будем считать, что наша цель — не дать указания компьютеру о ходе его работы, а объяснить человеку, что именно мы хотим добиться от компьютера.» — Дональд Кнут.

4. Первая программа: "Hello, World!"

Пришло время для священного ритуала всех программистов — написания программы, которая выводит на экран фразу "Hello, World!".

Шаг 1: Создание проекта

1. В стартовом окне Visual Studio выберите **"Создать новый проект"**.
2. В поиске шаблонов введите **"Console"** и выберите шаблон **"Консольное приложение"** (язык C#). Нажмите "Далее".

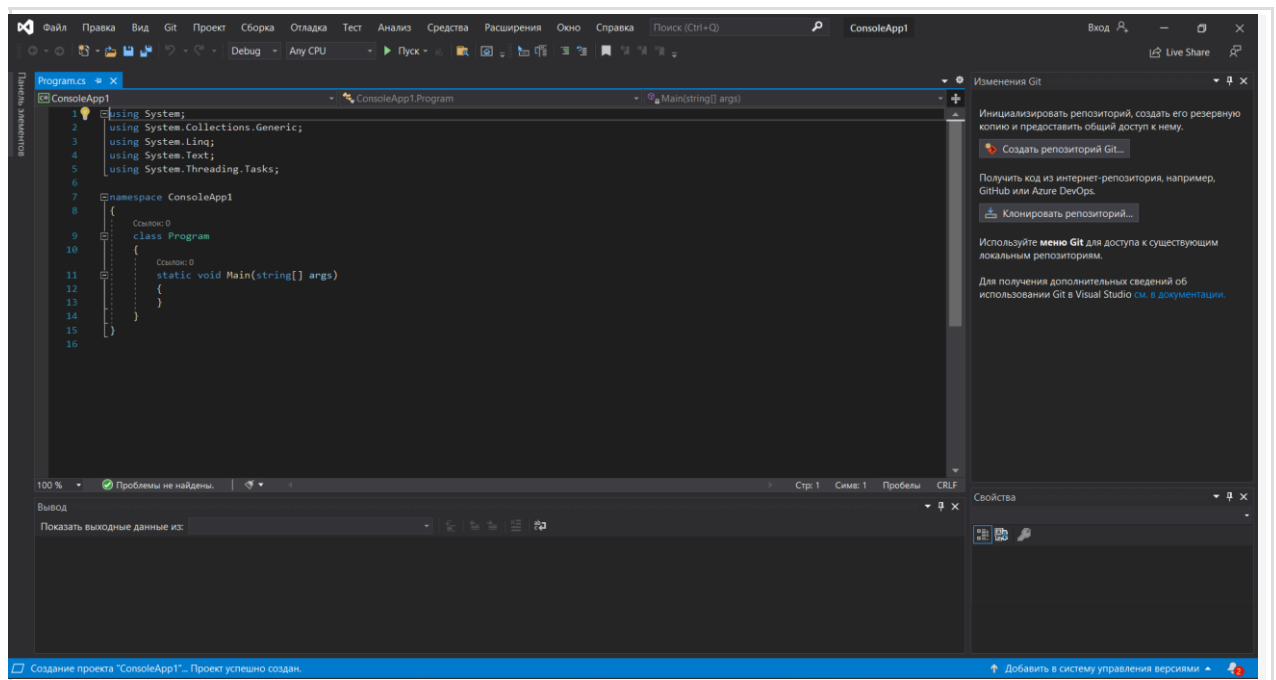


Выбор шаблона проекта

3. Дайте проекту имя, например, `HelloWorld`. Выберите, где его сохранить. Нажмите "Создать".

Шаг 2: Знакомство со средой

Visual Studio создаст для вас проект и откроет файл `Program.cs`. Вы увидите примерно такой код:



Давайте напишем базовый код, с которого начинают все программисты – Hello world!

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

Шаг 3: Запуск программы!

Это самый волнующий момент. Нажмите клавишу **F5** на клавиатуре или зеленую стрелочку "Пуск" вверху окна Visual Studio.

Откроется черное окно консоли, и вы увидите заветные слова:

```
Hello, World!
```

Поздравляю! Вы только что написали и запустили свою первую программу на C#!

«Если начинающий кодер успешно введёт несколько строк, он получит награду — увидит «Hello, World!» на своём экране, и вдруг он будет готов двигаться дальше. Это метафора того, что нужно быть достаточно смелым, чтобы попробовать что-то новое.» — Брайан Керниган (соавтор первого "Hello, world").

Шаг 4: Ничего не заработало...

И вот, Вы написали строчку кода, как и было сказано, а Visual Studio отказался ее компилировать, сказав что-то вроде:

	Код	Описание	Проект
❌	CS8370	Компонент "инструкции верхнего уровня" недоступен в C# 7.3. Используйте версию языка 9.0 или выше.	ConsoleApp1
❌	CS0103	Имя "Console" не существует в текущем контексте.	ConsoleApp1

Дело в том, что такой способ написания кода стал доступен с релиза C# 9.0. Действительно, если запустить код на требуемых версиях, то все сработает.

В рамках данного курса, мы **НЕ** будем писать подобный код. Он, конечно, простой, но наша задача – научиться программировать. В реальности, Вы можете попасть в компанию, где будет использоваться старая версия, и никто не будет менять ее под Вас.

Исправим наш код, что бы он заработал.

```
1  using System;
2
3  Ссылка: 0
   public class Example
4  {
5      Ссылка: 0
       public static void Main()
6      {
7          Console.WriteLine("Hello World");
8          Console.ReadKey();
9      }
10 }
```

Разберем код построчно:

Строка 1: `using System;`

- `using` - директива для подключения пространств имен
- `System` - основное пространство имен в .NET, содержащее фундаментальные классы

Эта строка позволяет использовать классы из пространства имен `System` без полного квалифицированного имени

Строка 3: `public class Example`

- `public` - модификатор доступа, означает, что класс доступен из других частей программы
- `class` - ключевое слово для объявления класса
- `Example` - имя класса

Строка 4: {

- Открывающая фигурная скобка - начало тела класса `Example` (проще говоря, аналог `Begin` в `Pascal`, или “начало”, если спускаться до алгоритмов)

Строка 5: `public static void Main()`

- `public` - метод доступен извне
- `static` - метод принадлежит классу, а не конкретному объекту
- `void` - метод не возвращает значение
- `Main()` - главный метод программы, точка входа при запуске

Строка 6: {

- Открывающая фигурная скобка - начало тела метода `Main`

Строка 7: `Console.WriteLine("Hello World");`

- `Console` - класс для работы с консолью
- `WriteLine` - метод, который выводит текст и переходит на новую строку
- `"Hello World"` - строковый литерал, текст для вывода
- `;` - завершение оператора

Строка 8: `Console.ReadKey();`

- `ReadKey()` - метод, который ожидает нажатия любой клавиши пользователем

Нужен чтобы консольное окно не закрывалось сразу после выполнения программы

Строки 9-10: } и }

- Закрывающие фигурные скобки - завершают метод `Main` и класс `Example`

Более понятно, можно объяснить так:

`Using system` – подключаем основную библиотеку системных команд

Создаем класс `Example` (подробнее о классах узнаете, когда начнется ООП)

В нем создаем основную, главную функцию – `Main` (именно в нее программа зайдет сама при запуске)

В функции выводим наше сообщение. Однако, после вывода, действия программы бы закончилось, и она завершилась. Что бы этого избежать, ждем пустого ввода от пользователя, после которого программа дойдет до своего логического завершения.

5. Игры с консолью

По [ссылке](#) можно узнать весь список доступных методов для Console в .NET. Пока что, мы не будем подробно останавливаться на нем, и разбираться как это работает (подробнее будет рассказано в районе 10 лекции). Давайте разберемся с некоторыми интересными методами, которыми будем пользоваться для работы с консолью на протяжении курса.

! Важное замечание! Сейчас и далее мы используем документацию к .NET 4.7.2

Метод	Описание
<code>Beep()</code>	Воспроизводит звук звукового сигнала через динамик консоли.
<code>Beep(Int32, Int32)</code>	Воспроизводит звук сигнала указанной частоты и длительности через динамик консоли.
<code>Clear()</code>	Очищает буфер консоли и соответствующее окно консоли отображаемых сведений.
<code>Console.ForegroundColor = ConsoleColor.Name</code>	НЕ МЕТОД Задаст цвет текста.
<code>Console.BackgroundColor = ConsoleColor.Name</code>	НЕ МЕТОД Задаст цвет фона.
<code>ResetColor()</code>	Задаст цвета переднего плана и фона консоли по умолчанию.
<code>Read()</code>	Считывает следующий символ из стандартного входного потока.
<code>ReadKey()</code>	Получает следующий символ или функциональную клавишу, нажатую пользователем. Нажатие отображается в окне консоли.
<code>ReadLine()</code>	Считывает следующую строку символов из стандартного входного потока.
<code>Write(...)</code>	Записывает данные в стандартный выходной поток без перехода на новую строку.

Метод	Описание
<code>WriteLine(...)</code>	Записывает данные в стандартный выходной поток с переходом на новую строку.

Примечание: Для `Write` и `WriteLine` существует множество перегрузок (для разных типов: `string`, `int`, `object` и т.д.), поэтому в таблице указано обобщённое описание.

Ниже предоставлен код (приложен и на гитхабе), демонстрирующий работу всех указанных команд.

`Thread.Sleep(Time)` – команда, которая ждет `Time` миллисекунд

`\n` – символ переноса на новую строку

Вопрос: Равнозначен ли вывод

`Console.Write("a")`

`Console.Write("\nb")`

и

`Console.WriteLine("a")`

`Console.WriteLine("b")`?

Ответ:

Нет, эти два фрагмента кода **не являются полностью равнозначными**, хотя визуальный результат на консоли может выглядеть одинаково — две строки:

a

b

Однако есть важное **различие в поведении**:

1. `Console.Write("a"); Console.Write("\nb");`

- Выводит `"a"` без перевода строки.
- Затем выводит `"\nb"`, то есть **символ новой строки** (`\n`) и `"b"`.
- Используется **один символ перевода строки** (`\n`), что соответствует Unix/Linux/macOS стилю.

2. `Console.WriteLine("a"); Console.WriteLine("b");`

- `WriteLine` выводит строку и добавляет **окончание строки**, соответствующее текущей операционной системе:
 - На Windows — это `\r\n` (возврат каретки + новая строка).
 - На Unix/Linux/macOS — это `\n`.

Таким образом, на **Windows** `WriteLine` добавит `\r\n`, а в первом случае — только `\n`.

Практическое следствие

Если вы перенаправляете вывод в файл или анализируете его побайтово (например, для тестирования или логирования), **разница будет заметна**:

- Первый вариант: `a\nb\n` (если последняя строка тоже завершена `\n` — зависит от контекста).
 - Второй вариант на Windows: `a\r\nb\r\n`.
-

Вывод

- **Визуально в консоли** — одинаково.
- **Фактически по содержимому вывода** — **не равнозначно**, особенно при кроссплатформенной разработке или при работе с бинарным/точным текстовым выводом.

Данный код был написан для примера, так что оставим его как есть, но, в дальнейшем использовать `\n` не будем.

```

1  using System;
2  using System.Threading;
3
4  class Program
5  {
6      static void Main()
7      {
8          // === Beep ===
9          Console.WriteLine("1. Beep() – звуковой сигнал по умолчанию");
10         Console.Beep(); // Короткий системный звук
11         Thread.Sleep(500);
12
13         Console.WriteLine("2. Beep(800, 3000) – звук 800 Гц, 3000 мс");
14         Console.Beep(800, 3000);
15         Thread.Sleep(500);
16
17         // === Clear ===
18         Console.WriteLine("3. Clear() – очистка консоли через 2 секунды...");
19         Thread.Sleep(2000);
20         Console.Clear();
21
22
23         // === Цвета и ResetColor ===
24         Console.ForegroundColor = ConsoleColor.Green;
25         Console.BackgroundColor = ConsoleColor.Blue;
26         Console.WriteLine("4. Цветной текст (зелёный на синем)");
27         Console.ResetColor();
28         Console.WriteLine("5. ResetColor() – цвета сброшены к стандартным");
29
30
31         // === Read, ReadKey, ReadLine ===
32         Console.WriteLine("6. Read() – нажмите любую клавишу (символ будет считан):");
33         int charCode = Console.Read();
34         Console.WriteLine($"Вы ввели символ с кодом: {charCode} ('{(char)charCode}')");
35
36         Console.WriteLine("\n7. ReadKey() – нажмите любую клавишу:");
37         ConsoleKeyInfo key = Console.ReadKey();
38         Console.WriteLine($"Вы нажали: {key.Key}");
39
40         Console.WriteLine("\n8. ReadLine() – введите строку и нажмите Enter:");
41         string input = Console.ReadLine();
42         Console.WriteLine($"Вы ввели: \"{input}\"");
43
44         // === Write и WriteLine ===
45         Console.WriteLine("\n9. Write и WriteLine – сравнение:");
46         Console.Write("Это Write – ");
47         Console.Write("всё в одной строке.\n");
48         Console.WriteLine("А это WriteLine –");
49         Console.WriteLine("каждый вызов – с новой строки.");
50
51         // Завершение
52         Console.WriteLine("\nДемонстрация завершена. Нажмите любую клавишу для выхода.");
53         Console.ReadKey();
54     }
55 }

```

Итог

Сегодня мы преодолели огромный путь:

1. Познакомились с языком C#, его историей и философией.
2. Узнали о платформе .NET.
3. Сравнили C# с другими языками.
4. Установили и настроили Visual Studio.
5. Написали, запустили и даже модифицировали свою первую программу!

В следующей лекции мы погрузимся в основы: что такое переменные, какие бывают типы данных и как с ними работать.