



# Разработчик C# + .NET

---

Артём Трофимушкин

# Преподаватель курса

## Артём Трофимушкин

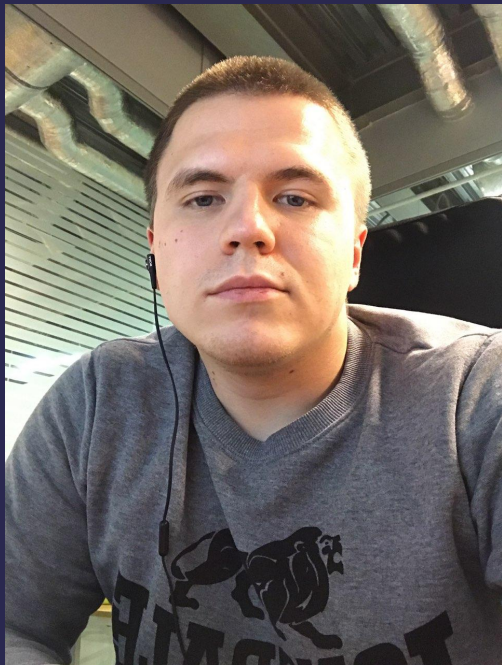
Более 4-х лет опыта коммерческой разработки на языках C# (.NET), JavaScript (TypeScript).

За свою карьеру самостоятельно изучил множество технологий, созданных для разработки классических оконных и веб-приложений. Участвовал в разработке крупных корпоративных веб-проектов.

На текущий момент являюсь старшим разработчиком в компании "Додо Пицца".

Занимаюсь разработкой облачной информационной системы DODO-IS, которая обеспечивает работу всех пиццерий, расположенных по всему миру.

Активно слежу и изучаю облачные технологии и инструменты разработки веб-приложений.



# Особенности курса

---

- 4 месяца (36 занятий или 144 часа)
- Обучение инструментам и методам разработки **с нуля**
- Широкий спектр **практик** для решения различных задач
- Каждые два месяца завершаются **выполнением проекта**
- Сертификат и **уверенность** при поиске работы



# Доступ к материалам курса

---

1. Все занятия записываются
2. Запись, исходный код и презентация будут выкладываться **навечно** в ваш личный кабинет, LMS
3. Доступ будет отправлен вам по **email**



# Как сдавать домашнее задание?

---

1. Домашнее задание необходимо будет загружать в LMS
2. Ваше домашнее задание будет проходить Code Review и высылаться обратно с комментариями преподавателей курса
3. Все домашние задания оцениваются, итоговый балл будет у вас в Сертификате
4. Балл можно исправить, переделав работу с учетом комментариев преподавателя



# Языки программирования

---

Какие виды языков программирования бывают?

1. Компилируемые (C/C++, Rust)
2. Интерпретируемые (JavaScript, Python)
3. Языки с промежуточной компиляцией (C#, Java)



# Язык C# и платформы .NET

---

## Написание кода

1. Мы пишем код на **языке C#**
2. Компилятор языка **C#** преобразует исходный код в промежуточный код платформы .NET, **общий** для всех языков верхнего уровня: **IL** (или CIL: Common Intermediate Language)
3. На выходе, получаем исполняемый файл (**.exe**) или библиотеку компонентов (**.dll**)

## Запуск кода

1. ОС понимает, что код написан с помощью платформы **.NET**
2. ОС запускает платформу **.NET** и передает ей на вход программу на промежуточном языке (**IL**)
3. В момент выполнения, код с промежуточного языка (**IL**) преобразуется в машинный код компилятором **JIT (Just in time)**



# Язык C# и платформы .NET

---

## Какие преимущества?

1. Код можно писать на нескольких совместимых с **.NET** языками
2. Код можно запускать на всех ОС поддерживающих платформу **.NET**
3. Платформа **.NET** изолирует программиста от технических особенностей ОС
4. Платформа **.NET** следит за безопасностью выполнения программы (утечки памяти, ошибки переполнения буфера и тд)
5. Компилятор времени выполнения **JIT** может оптимизировать код с учетом аппаратных возможностей компьютера





# Язык C# и платформы .NET

---

C#

VB

F#

C++

Платформа .NET

Операционная система

Оборудование компьютера



# Язык C# и платформы .NET

---

## .NET Framework

- Возможность использовать специфические функции ОС Windows, однако из-за этого может разворачиваться только на Windows;
- Монолитный компонент с длительным циклом обновления;
- Код доступен только для просмотра.

## .NET Core

- Кроссплатформенность: возможность запускать приложения на Windows, Mac и Linux ОС;
- Модульный компонент, что означает гибкое развертывание и более частые обновления;
- Открытый исходный код



# Средства разработки

---

## Visual Studio Code

- Есть версии для всех ОС Windows Linux and Mac,
- Удобство однообразия среды разработки

## Microsoft Visual Studio

- Имеет большое количество удобных функций для разработки приложений
- Работает на ОС Windows



# Этапы разработки ПО

---

1. Постановка задачи
2. Проектирование
3. Кодирование
4. Отладка / тестирование
5. Сопровождение



# Постановка задачи

---

## Написать программу приветствия пользователя

- Необходимо узнать имя, подождать 5 секунд, а затем вывести ему приветствие.
- Программа должна завершиться, когда пользователь подтвердит прочтение приветствия нажатием любой клавиши на клавиатуре.



# Проектирование

---

- Необходимо узнать имя, а потом вывести ему приветствие.
- Перед вводом приветствия должно пройти 5 секунд.
- Программа должна завершиться, когда пользователь подтвердит прочтение приветствия нажатием любой клавиши на клавиатуре.



# Элементы алгоритмических блок-схем

---

## Терминатор

*(как правило, начало и конец алгоритма)*

## Процесс

*(обработка данных, операция или группа операций)*

## Ввод / Вывод

*(общее обозначение ввода или вывода данных)*



# Проектирование

---

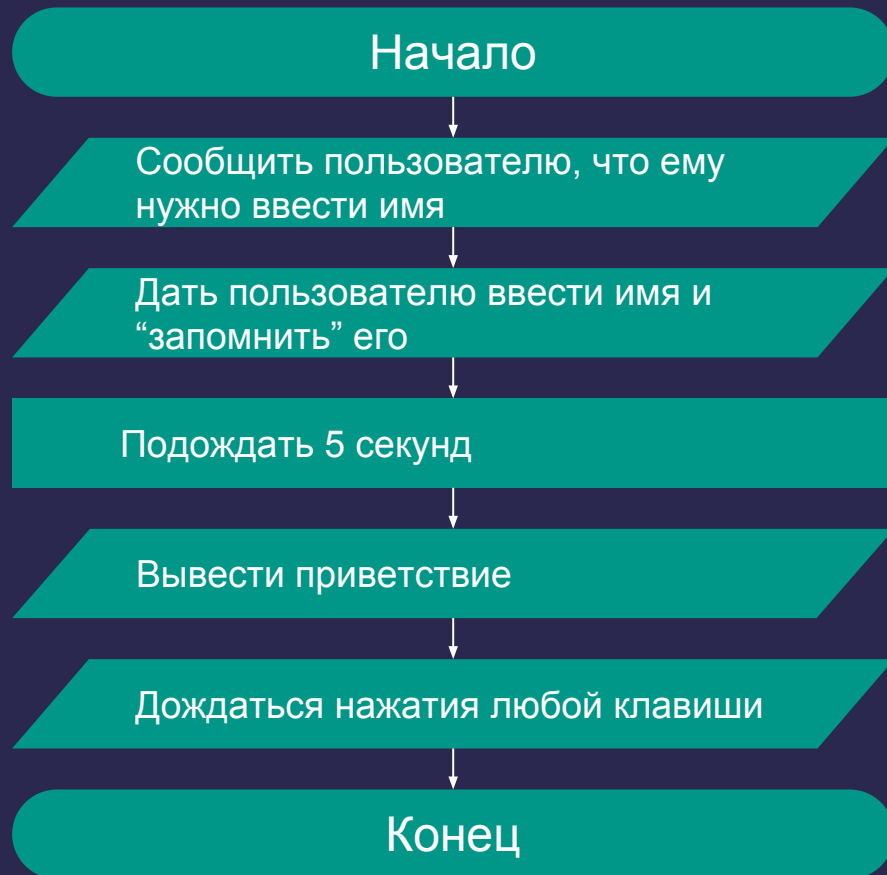
- Необходимо узнать имя, а потом вывести ему приветствие.
- Перед вводом приветствия должно пройти 5 секунд
- Программа должна завершиться, когда пользователь подтвердит прочтение приветствия нажатием любой клавиши на клавиатуре.
- Спросить имя
- Прочитать имя и “запомнить” его
- Подождать 5 секунд
- Вывести приветствие
- Дождаться нажатия любой клавиши





# Проектирование

- Спросить имя
- Прочитать имя и “запомнить” его
- Подождать 5 секунд
- Вывести приветствие
- Дождаться нажатия любой клавиши



# Кодирование

- Спросить имя
- Прочитать имя и “запомнить” его
- Подождать 5 секунд
- Вывести приветствие
- Дождаться нажатия любой клавиши

```
1  using System;
2  using System.Threading;
3
4  namespace DemoApp1
5  {
6      0 references
7      class Program
8      {
9          0 references
10         static void Main(string[] args)
11         {
12             Console.WriteLine("Введите имя");
13             string name = Console.ReadLine();
14             Thread.Sleep(5000);
15             Console.WriteLine($"Здравствуйте, {name}!");
16             Console.ReadKey();
17         }
18     }
19 }
```

# Отладка / Тестирование

Ставим точку остановки в строке номер 12

- **F9** - поставить / убрать точку остановки в режиме редактирования кода

Запуск

- **F5** - запуск в режиме отладки

```
1  using System;
2  using System.Threading;
3
4  namespace DemoApp1
5  {
6      0 references
7      class Program
8      {
9          0 references
10         static void Main(string[] args)
11         {
12             Console.WriteLine("Введите имя");
13             string name = Console.ReadLine();
14             Thread.Sleep(5000);
15             Console.WriteLine($"Здравствуйте, {name}!");
16             Console.ReadKey();
17         }
18     }
```

# Сопровождение (домашняя работа)

---







Модифицировать программу таким образом, чтобы после вывода приветствия программа ожидала ещё 5 секунд и выводила прощание, а уже потом ожидала нажатия клавиши и завершалась.



# Система контроля версий (SVC)

Для чего нужна система контроля версий?

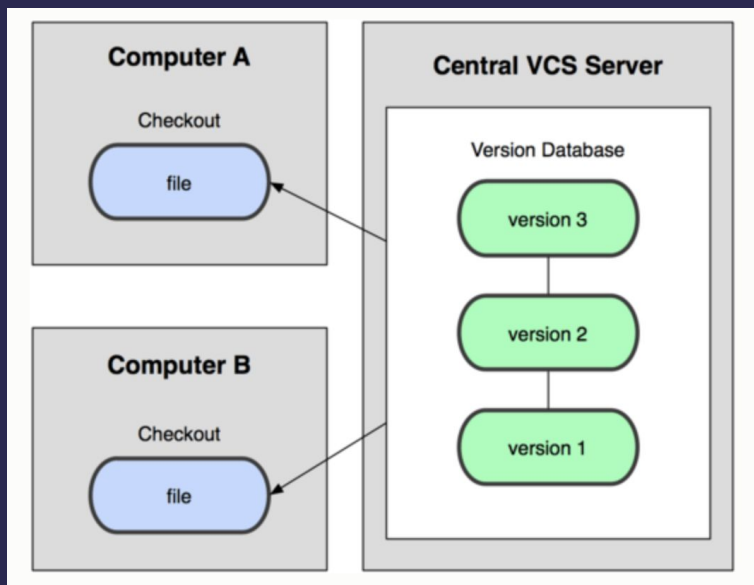
1. Контроль внесения изменений

	Document 1	4/16/2019 5:29 AM	Office Open XML ...
	Document Final (1)	4/16/2019 5:28 AM	Office Open XML ...
	Document Final (2)	4/16/2019 5:28 AM	Office Open XML ...
	Document Final	4/16/2019 5:28 AM	Office Open XML ...
	Document last (2)	4/16/2019 5:28 AM	Office Open XML ...
	Document	4/16/2019 5:27 AM	Office Open XML ...

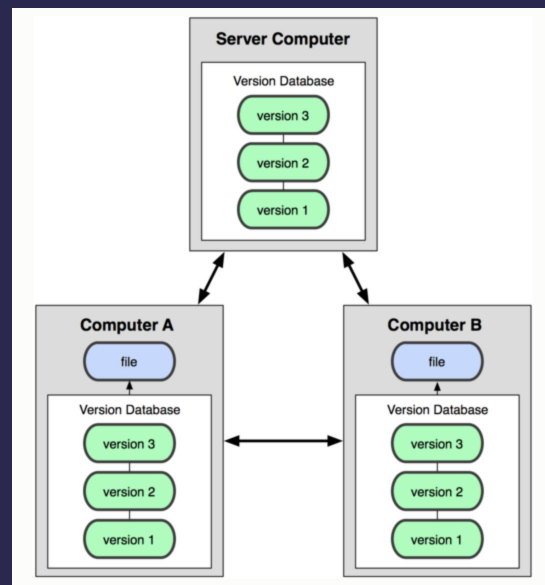
2. Организация совместной работы над проектом

# Системы контроля версий

Централизованные (SVN, SourceSafe)



Распределенные (Git, Mercurial)



# Версионирование исходного кода Git

---

## GitHub

- Зарегистрироваться <https://github.com>

## Git for Windows

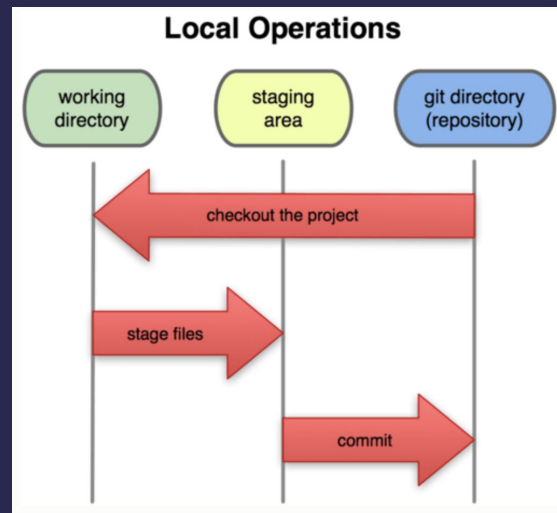
- Скачать <https://git-scm.com/downloads> или <https://gitforwindows.org>
- Подробная справка по Git <https://git-scm.com/doc>



# Версионирование исходного кода Git

## Состояния файлов в Git

- Зафиксированный
- Измененный
- Подготовленный





# Основные команды Git

---

git **clone** - создать локальный репозиторий на основе внешнего

git **init** - создать пустой локальный репозиторий

git **status** - вывести статус файлов в репозитории

git **add** - пометить файлы для последующей операции commit

git **commit** - зафиксировать версию репозитория

git **push** - отправить локальные коммиты на удалённый сервер  
(репозиторий)

git **pull** - получить изменения с удаленного репозитория



# Подробная инструкция по работе с GitHub

---

- Регистрируемся на GitHub <https://github.com>
- Создаем публичный репозиторий nordic-it-cs-q1
- Ставим локальный Git для Windows <https://gitforwindows.org>
- Запускаем Git bash
- `git clone URL-на-репозиторий.git`
- Создаем структуру папок для уроков:
  - 01 (02, 03, и т.д.)
    - ClassWork
    - HomeWork
- Копируем в корневую папку репозитория (nordic-it-cs-q1) файл .gitignore для Visual Studio / C# <https://github.com/github/gitignore/blob/master/VisualStudio.gitignore>
- Добавляем файлы с урока в папку 01/ClassWork



# Подробная инструкция по работе с GitHub

---

- `git status`
- `git add --all`
- `git status`
- `git commit -m "Class work of the 1st lesson added"`
- `git push`



# Подробная инструкция по работе с GitHub

---

## Дома

- Делаем `git clone URL-на-репозиторий.git`
- Создаем новый солюшн в папке `01/HomeWork/`
- Выполняем там домашнюю работу
- `git add --all`
- `git status`
- `git commit -m "Home work of the 1st lesson added"`
- `git push`
- В личном кабинете присылаете мне ссылку на свой GitHub-репозиторий, а также комментарии и вопросы.
- Если что-то не получится, присылайте вопросы, **в самом крайнем случае** просто zip-файл с кодом решения.



# Спасибо за внимание.

