Системы программирования: основные понятия. Базовые понятия технологии .NET Framework.

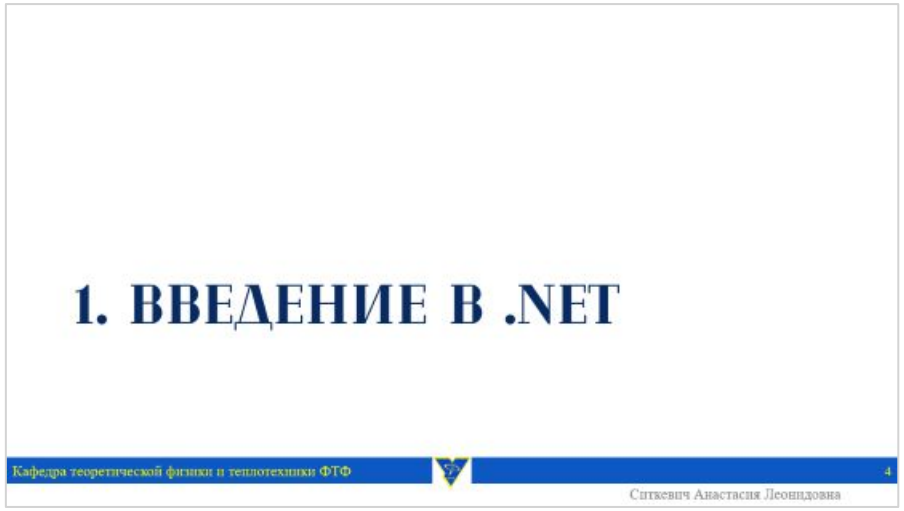
Лекция 3: Базовые понятия технологии .NET Framework.

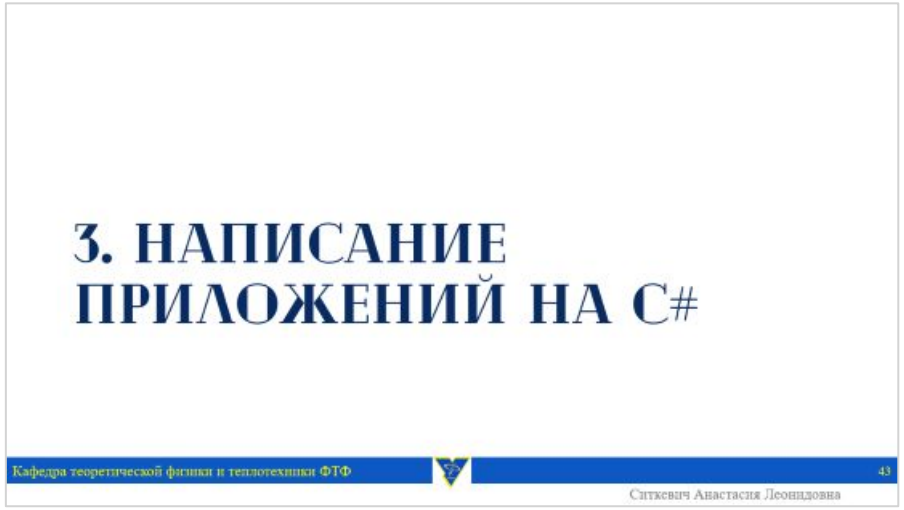
Ситкевич Анастасия Леонидовна

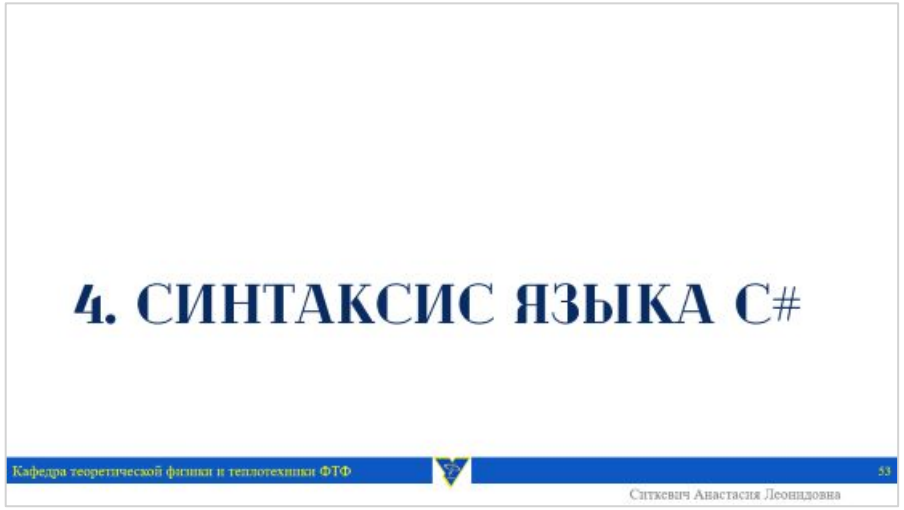
Полезные источники информации

- Как работает .NET и зачем он нужен
- Общеязыковая исполняющая среда CLR
- Что такое .NET? Введение и обзор
- Особенности работы CLR в .NET framework
- C # - Базовый синтаксис

Вопросы лекции







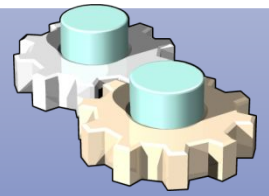
1. Введение в .Net

Введение в .Net

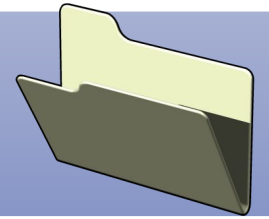
- .NET — это бесплатная кроссплатформенная платформа разработчика с открытым исходным кодом для создания различных типов приложений.
- Приложения и библиотеки .NET создаются из исходного кода и файла проекта с помощью интерфейса командной строки .NET или интегрированной среды разработки (IDE), такой как Visual Studio
- .NET является бесплатным, открытый код и проектом .NET Foundation.
- Платформа .NET поддерживается корпорацией Майкрософт и сообществом на GitHub в нескольких репозиториях.

Платформа .NET Framework

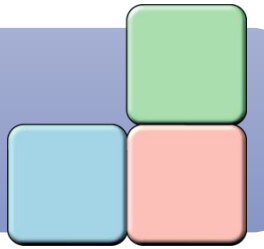
Среда CLR
(Common Language Runtime, CLR)



Библиотека классов .NET Framework
(.NET Framework Class Library)



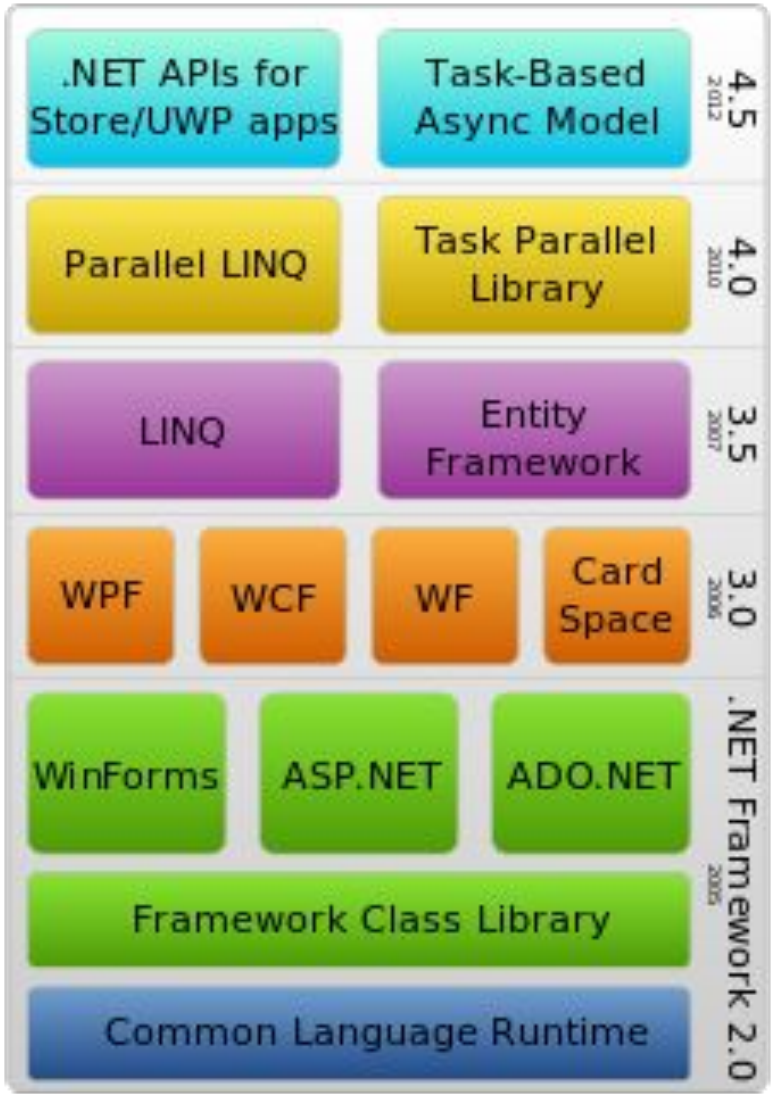
Фреймворки для разработки приложений
(Development Frameworks)



Стандартная библиотека

- Base Class Library
  - System (Int32, String, DateTime, ...)
  - System.IO
  - System.Text
  - ...
- Framework Class Library
  - System.Net
  - System.Xml
  - ...
- Domain-specific
  - ASP.NET — веб-приложения
  - Entity Framework — ORM, работа с базами данных
  - ...

Стек технологий .Net framework



Версия	CLR	Дата выхода	Visual Studio
1.0	1.0	1 мая 2002 года	Visual Studio .NET
1.1	1.1	1 апреля 2003 года	Visual Studio .NET 2003
2.0	2.0	11 июля 2005 года	Visual Studio 2005
3.0	2.0	6 ноября 2006 года	Visual Studio 2005 + расширения
3.5	2.0	9 ноября 2007 года	Visual Studio 2008
4.0	4	12 апреля 2010 года	Visual Studio 2010
4.5	4	15 августа 2012 года	Visual Studio 2012
4.5.1	4	17 октября 2013 года	Visual Studio 2013
4.5.2	4	5 мая 2014 года	н/д
4.6	4	20 июля 2015 года	Visual Studio 2015
4.6.1	4	17 ноября 2015 года	Visual Studio 2015 Update 1

Стек технологий .Net framework и .Net

Версия	CLR	Дата выхода	Visual Studio
4.6.2	4	20 июля 2016	
4.7	4	5 апреля 2017	Visual Studio 2017
4.7.1	4	17 октября 2017	Visual Studio 2017
4.7.2	4	30 апреля 2018	Visual Studio 2017
4.8	4	18 апреля 2019	Visual Studio 2019
4.8.1	4	11 октября 2021	Visual Studio 2019

Версия	Дата выхода	Версия Visual Studio	Конец поддержки
.NET Core 1.0	27 июня 2016	Visual Studio 2015 Update 3	27 июня 2019 г
.NET Core 1.1	16 ноября 2016	Visual Studio 2017, версия 15.0	27 июня 2019 г
.NET Core 2.0	14 августа 2017	Visual Studio 2017, версия 15.3	1 октября 2018 г
.NET Core 2.1 (LTS[7])	30 мая 2018	Visual Studio 2017, версия 15.7	21 августа 2021 г
.NET Core 2.2	4 декабря 2018	Visual Studio 2019, версия 16.0	23 декабря 2019 г
.NET Core 3.0	23 сентября	Visual Studio 2019, версия 16.3	3 марта 2020 г
.NET Core 3.1 (LTS[7])	3 декабря 2019	Visual Studio 2019, версия 16.7	13 декабря 2022 г
.NET 5	20 ноября 2020	Visual Studio 2019, версия 16.11	10 мая 2022 г
.NET 6 (LTS[7])	8 ноября 2021	Visual Studio 2022, версия 17.0	12 ноября 2024 г
.NET 7	8 ноября 2022	Visual Studio 2022, версия 17.4	
.NET 8 (LTS[7])	ноябрь 2023 (запланировано)		

Ситкевич Анастасия Леонидовна

Основные черты платформы .NET Framework

- **Поддержка нескольких языков.** Основой платформы является общеязыковая среда исполнения Common Language Runtime (CLR), благодаря чему .NET поддерживает несколько языков: наряду с **C#** это также **VB.NET, C++, F#**, а также различные диалекты других языков, привязанные к .NET, например, **Delphi.NET.**
- **Мощная библиотека классов.** .NET представляет единую для всех поддерживаемых языков библиотеку классов. И какое бы приложение мы не собирались писать на C# - текстовый редактор, чат или сложный веб-сайт - так или иначе мы задействуем библиотеку классов .NET.
- **Разнообразие технологий**. Общеязыковая среда исполнения CLR и базовая библиотека классов являются основой для целого стека технологий, которые разработчики могут задействовать при построении тех или иных приложений. ***ADO.NET и Entity Framework Core, WPF и WinUI, Windows Forms, Xamarin/MAUI, ASP.NET и т.д.***
- **Производительность.** Согласно ряду тестов веб-приложения на .NET 7 в ряде категорий сильно опережают веб-приложения, построенные с помощью других технологий. Приложения на .NET 7 в принципе отличаются высокой производительностью.

Среда выполнения

Среда CLR (Common Language Runtime) является основой для всех приложений .NET. Ниже перечислены основные функции среды выполнения .

- Управление памятью;
- Загрузка сборок;
- Безопасность;
- Обработка исключений;
- Синхронизация.

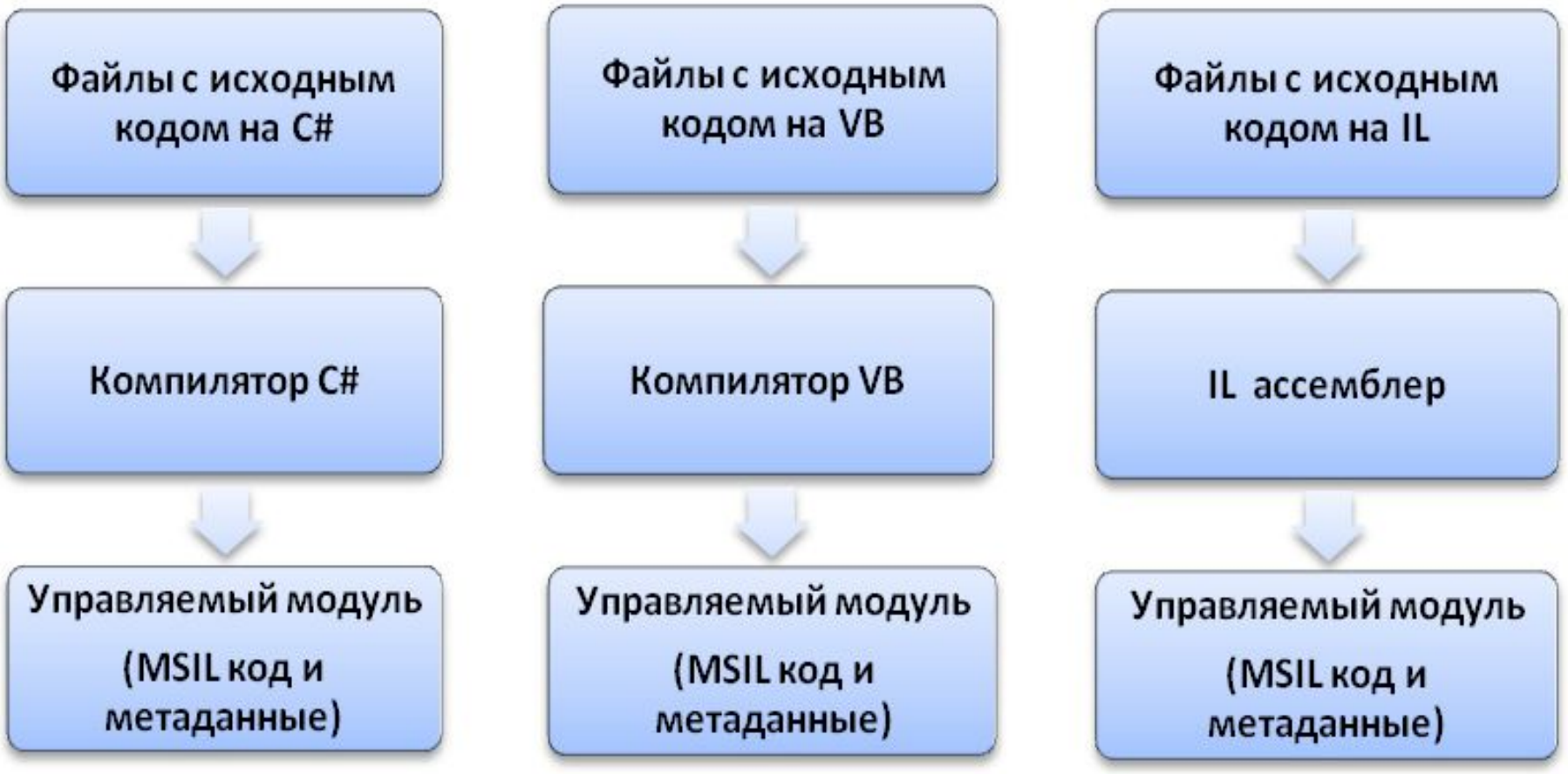
Microsoft:

- C++/CLI, C#, Visual Basic, F#, Iron Python, Iron Ruby и ассемблер Intermediate Language (IL).

Прочие компании и университеты:

- Ada, APL, Caml, COBOL, Eiffel, Forth, Fortran, Haskell, Lexico, LISP, LOGO, Lua, Mercury, ML, Mondrian, Oberon, Pascal, Perl, Php, Prolog, RPG, Scheme, Smalltalk и Tcl/Tk.

Управляемые модули, MSIL код и метаданные



управляемый модуль (managed module) — стандартный переносимый исполняемый (portable executable, PE) файл 32-разрядной (PE32) или 64-разрядной Windows (PE32+), который требует для своего выполнения CLR.

Части управляемого модуля

Часть	Описание
Заголовок PE32 или PE32+	Стандартный заголовок PE-файла Windows, аналогичный заголовку Common Object File Format (COFF). Файл с заголовком в формате PE32 может выполняться в 32- и 64-разрядной версиях Windows, а с заголовком PE32+ — только в 64-разрядной. Заголовок обозначает тип файла: GUI, CUI или DLL, он также имеет временную метку, показывающую, когда файл был собран. Для модулей, содержащих только IL-код, основной объем информации в заголовке PE32(+) игнорируется. В модулях, содержащих машинный код, этот заголовок содержит сведения о машинном коде
Заголовок CLR	Содержит информацию (интерпретируемую CLR и утилитами), которая превращает этот модуль в управляемый. Заголовок включает нужную версию CLR, некоторые флаги, метку метаданных MethodDef точки входа в управляемый модуль (метод Main), а также месторасположение/размер метаданных модуля, ресурсов, строгого имени, некоторых флагов и пр.
Метаданные	Каждый управляемый модуль содержит таблицы метаданных. Есть два основных вида таблиц — это таблицы, описывающие типы данных и их члены, определенные в исходном коде, и таблицы, описывающие типы данных и их члены, на которые имеются ссылки в исходном коде.
Код Intermediate Language (IL)	Код, создаваемый компилятором при компиляции исходного кода. Впоследствии CLR компилирует IL в машинные команды

метаданные

Метаданные устраняют необходимость в заголовочных и библиотечных файлах при компиляции, так как все сведения об упоминаемых типах/членах содержатся в файле с реализующим их IL-кодом. Компиляторы могут читать метаданные прямо из управляемых модулей.

Среда Microsoft Visual Studio использует метаданные для облегчения написания кода. Ее функция IntelliSense анализирует метаданные и сообщает, какие методы, свойства, события и поля предпочтительны в данном случае и какие именно параметры требуются конкретным методам.

В процессе верификации кода CLR использует метаданные, чтобы убедиться, что код совершает только «безопасные по отношению к типам» операции.

Метаданные позволяют сериализовать поля объекта, а затем передать эти данные по сети на удаленный компьютер и там провести процесс десериализации, восстановив объект и его состояние на удаленном компьютере.

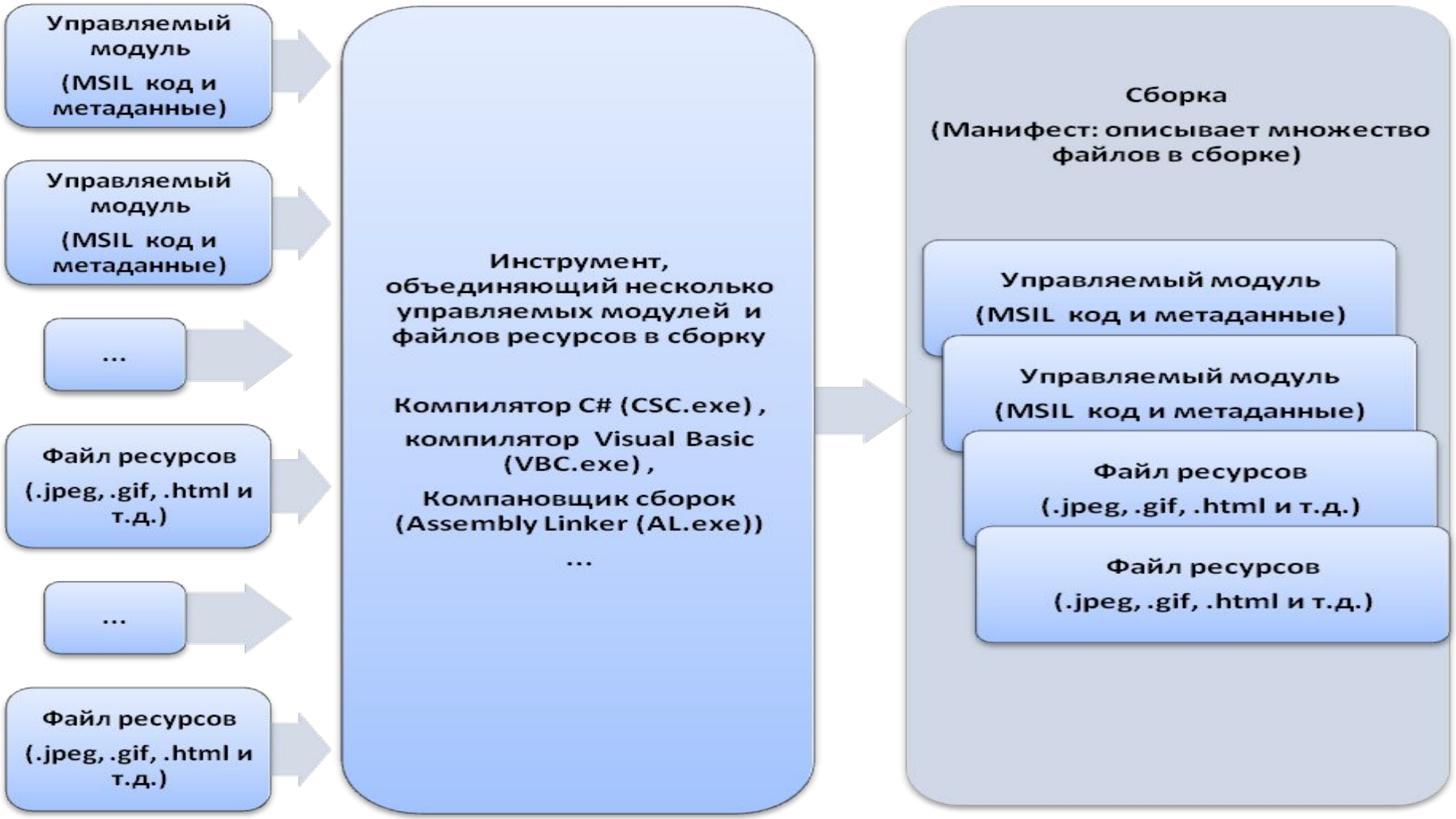
Метаданные позволяют сборщику мусора отслеживать жизненный цикл объектов. При помощи метаданных сборщик мусора может определить тип объектов и узнать, какие именно поля в них ссылаются на другие объекты.

Сборки в .NET

Логическая группировка одного или нескольких управляемых модулей и файлов ресурсов

Самая маленькая единица с точки зрения повторного использования, безопасности и управления версиями

Сборки в .NET


Управляемый модуль
(MSIL код и метаданные)
Управляемый модуль
(MSIL код и метаданные)
...
Файл ресурсов
(.jpeg, .gif, .html и т.д.)
...
Файл ресурсов
(.jpeg, .gif, .html и т.д.)
Инструмент, объединяющий несколько управляемых модулей и файлов ресурсов в сборку
Компилятор C# (CSC.exe),
компилятор Visual Basic (VBC.exe),
Компановщик сборок (Assembly Linker (AL.exe))
...
Сборка
(Манифест: описывает множество файлов в сборке)
Управляемый модуль
(MSIL код и метаданные)
Управляемый модуль
(MSIL код и метаданные)
Файл ресурсов
(.jpeg, .gif, .html и т.д.)
Файл ресурсов
(.jpeg, .gif, .html и т.д.)

Сборки в .NET

Некоторые характеристики сборки:

- В сборке определены повторно используемые типы
- Сборка помечена номером версии
- Со сборкой может быть связана информация безопасности

Сборки в .NET

Сборки реализованы как файлы *EXE* или *DLL*.

Для библиотек, предназначенных для .NET Framework, сборки можно совместно использовать в нескольких приложениях, поместив их в глобальный кэш сборок (GAC).

Сборки загружаются в память только в том случае, если они реально используются. Если они не используются, то они не загружаются. Благодаря этому свойству сборки могут быть эффективным средством для управления ресурсами в крупных проектах.

Сведения о сборке можно получить программным путем.

Сборку можно загрузить только для ее проверки.

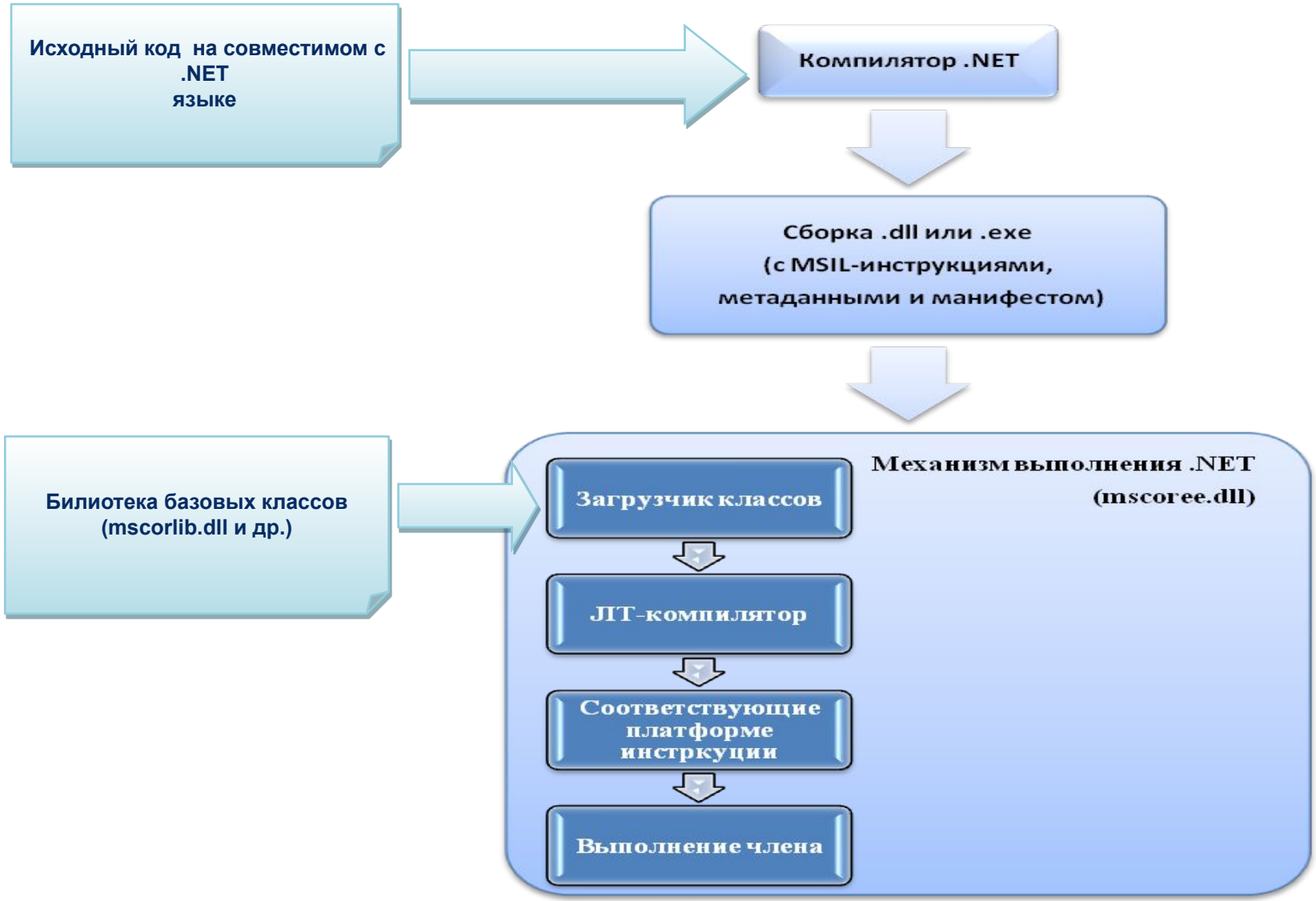
Сборки в .NET

```
NameAssembly,Version=1.1.0.0,Culture=en,PublicKeyToken=874e23ab874e23ab
```

Номер версии сборки состоит из следующих компонент:

- Главный номер версии (Major version number)
- Второстепенный номер версии (Minor version number)
- Номер сборки (Build number)
- Номер версии (Revision number)

Как CLR загружает, компилирует и запускает сборки



Как CLR загружает, компилирует и запускает сборки

CLR содержит несколько компонентов, которые при запуске .NET Framework приложений выполняют следующие задачи:

- **Загрузчик классов (Class Loader)** находит и загружает все сборки, которые требуются приложению. Сборки к этому моменту будут уже скомпилированы в IL
- **IL-to-native компилятор** проверяет IL код, а затем компилирует все сборки в машинный код, готовый к исполнению
- **Code Manager** загружает исполняемую сборку и запускает метод Main
- **Garbage Collector** обеспечивает автоматическое управление памятью жизни всех объектов, которые создает приложение
- **Exception Manager** предоставляет структурированную обработку исключений для .NET приложений, которая интегрирована с структурированной обработкой исключений Windows

Инструменты, предоставляемые .NET Framework

Caspol.exe

Ngen.exe

Makecert.exe

Sn.exe

Gacutil.exe

Ildasm.exe

Инструменты, предоставляемые .NET Framework

Инструмент	Описание
Code Access Security Policy Tool (Caspol.exe)	Позволяет пользователям и администраторам изменять политику безопасности на уровне компьютера, пользователя и предприятия Может включать определение пользовательского набора разрешений и добавления сборки в полный список доверия.
Certificate Creation Tool (Makecert.exe)	Позволяет пользователям создавать сертификаты X.509, предназначенные исключительно для тестирования. Этот инструмент создает пару из открытого и закрытого ключей для цифровой подписи и помещает ее в файл сертификата. Он также привязывает пару ключей к указанному имени издателя и создает сертификат X.509, который связывает заданное пользователем имя с открытым ключом пары. Как правило, эти сертификаты можно использовать для подписания сборки и определения Secure Sockets Layer (SSL) соединений.

Инструменты, предоставляемые .NET Framework

Инструмент	Описание
Global Assembly Cache Tool (Gacutil.exe)	Позволяет пользователям просматривать содержимое глобального кэша сборок и кэша загрузки, а также управлять ими. С помощью этого инструмента можно добавлять и удалять сборки в GAC, для того, чтобы приложения могли получать к ним доступ.
Native Image Generator (Ngen.exe)	Генератор образов в машинном коде (Native Image Generator) — это средство повышения быстродействия управляемых приложений. Ngen.exe создает образы в машинном коде, представляющие собой файлы, содержащие компилированный специфический для процессора машинный код, и устанавливает их в кэш образов в машинном коде на локальном компьютере. Среда выполнения может использовать образы в машинном коде, находящиеся в кэше, вместо использования JIT-компилятора для компиляции исходной сборки.

Инструменты, предоставляемые .NET Framework

Инструмент	Описание
MSIL Disassembler (Ildasm.exe)	MSIL Disassembler является парным инструментом к ассемблеру MSIL (Ilasm.exe). Ildasm.exe принимает входной исполняемый файл (PE-файл). Содержащий код на языке MSIL, и создает на его основе текстовый файл, который може служить входным для программы Ilasm.exe. Можно использовать Ildasm.exeдля просмотра промежуточного языка MSIL в файле. Если анализируемый файл является сборкой, то эти данные могут включать в себя атрибуты сборки, а также ссылки на другие модули и сборки. Эти данные полезны для определения того, является ли файл сборкой или частью сборки и имеет ли он ссылки на другие модули и сборки.
Strong Name Tool (Sn.exe)	Позволяет пользователям подписывать сборки строгими именами. Strong Name Tool включает в себя команды для создания новой пары ключей, извлечения открытого ключа из пары ключей и верификации сборки.

Компиляция

2. Знакомство со средой разработки Visual Studio 2022 и ее возможностями при разработке .NET

Основные возможности Visual Studio 2022

Разработка

- IntelliCode
- Горячая перезагрузка для .Net сразу применит ваши правки к запущенному приложению

Отладка

- Простая отладка, профилирование и диагностика кода
- Интеллектуальные точки останова кода

Тестирование

- Комплексные инструменты тестирования помогают писать высококачественный код

Сотрудничество

- Используйте систему управления версиями для гибкой и эффективной совместной работы

Расширение

- Тысячи расширений позволят вам настроить IDE под себя

Шаблоны в Visual Studio 2022

Шаблоны:

обеспечивают стартовый код, который можно использовать для быстрого создания функционирующего приложения

включают поддержку компонентов и элементов управления, относящихся к типу проекта

обеспечивают настройку Visual Studio 2022 IDE согласно типу разрабатываемого приложения

обеспечивают добавление ссылки на любую начальную сборку, обычно требующуюся соответствующему типу приложения

Шаблоны в Visual Studio 2022

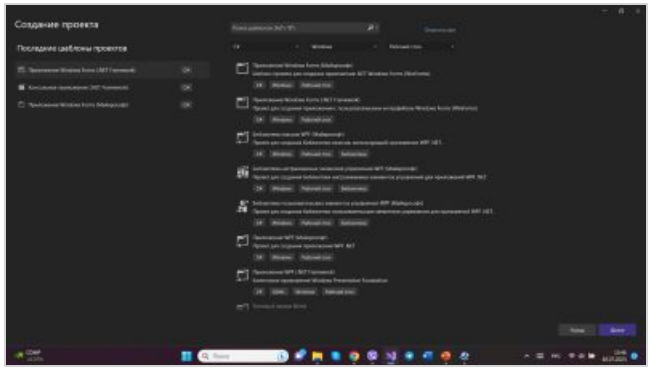
Шаблон	Описание
Console Application	Предоставляет параметры среды, инструменты, ссылки на проекты и стартовый код для разработки приложения, выполняемое в интерфейсе командной строки. Этот тип приложения считается простым по сравнению с шаблоном приложения Windows Forms, потому что отсутствует графический интерфейс пользователя.
WPF Application	Предоставляет параметры среды, инструменты, ссылки на проекты и стартовый код для создания богатых графических приложений Windows. Приложения WPF позволяет создавать новое поколение приложений Windows, с гораздо большим контролем над дизайном пользовательского интерфейса.
Class Library	Предоставляет параметры среды, инструменты и стартовый код для построения .dll сборок. Этот тип файла можно использовать для хранения функциональностей, на которые можно ссылаться из других приложений.

Шаблоны в Visual Studio 2022

Шаблон	Описание
Windows Forms Application	Предоставляет параметры среды, инструменты, ссылки на проекты и стартовый код для построения графических приложений Windows Forms.
ASP.NET Web Application	Предоставляет параметры среды, инструменты, ссылки на проекты, и стартовый код для создания серверных, компируемых веб-приложений ASP.NET.
ASP.NET MVC 2 Application	Предоставляет параметры среды, инструменты, ссылки на проекты и стартовый код, чтобы создать Model-View-Controller (MVC) веб-приложение. ASP.NET MVC веб-приложение отличается от стандартного веб-приложений ASP.NET тем, что архитектура приложения позволяет отделить уровень представления (presentation layer), слой бизнес-логики (business logic layer) и уровень доступа к данным (data access layer).

Шаблоны в Visual Studio 2022

Шаблон	Описание
Silverlight Application	Предоставляет параметры среды, инструменты, ссылки на проекты и стартовый код для создания богатого графического веб-приложения.
WCF Service Application	Предоставляет параметры среды, инструменты, ссылки на проекты и стартовый код для построения Service Orientated Architecture (SOA) сервисов.



Структура проектов и решений Visual Studio

Visual Studio 2022 использует решения и проекты как концептуальные контейнеры для организации исходных файлов в процессе разработки. Классификация исходных файлов таким образом, упрощает компановку и развертывание процесса для приложений .NET Framework

ASP.NET project

.aspx .csproj

.aspx.cs .config

WPF project

.xaml .csproj

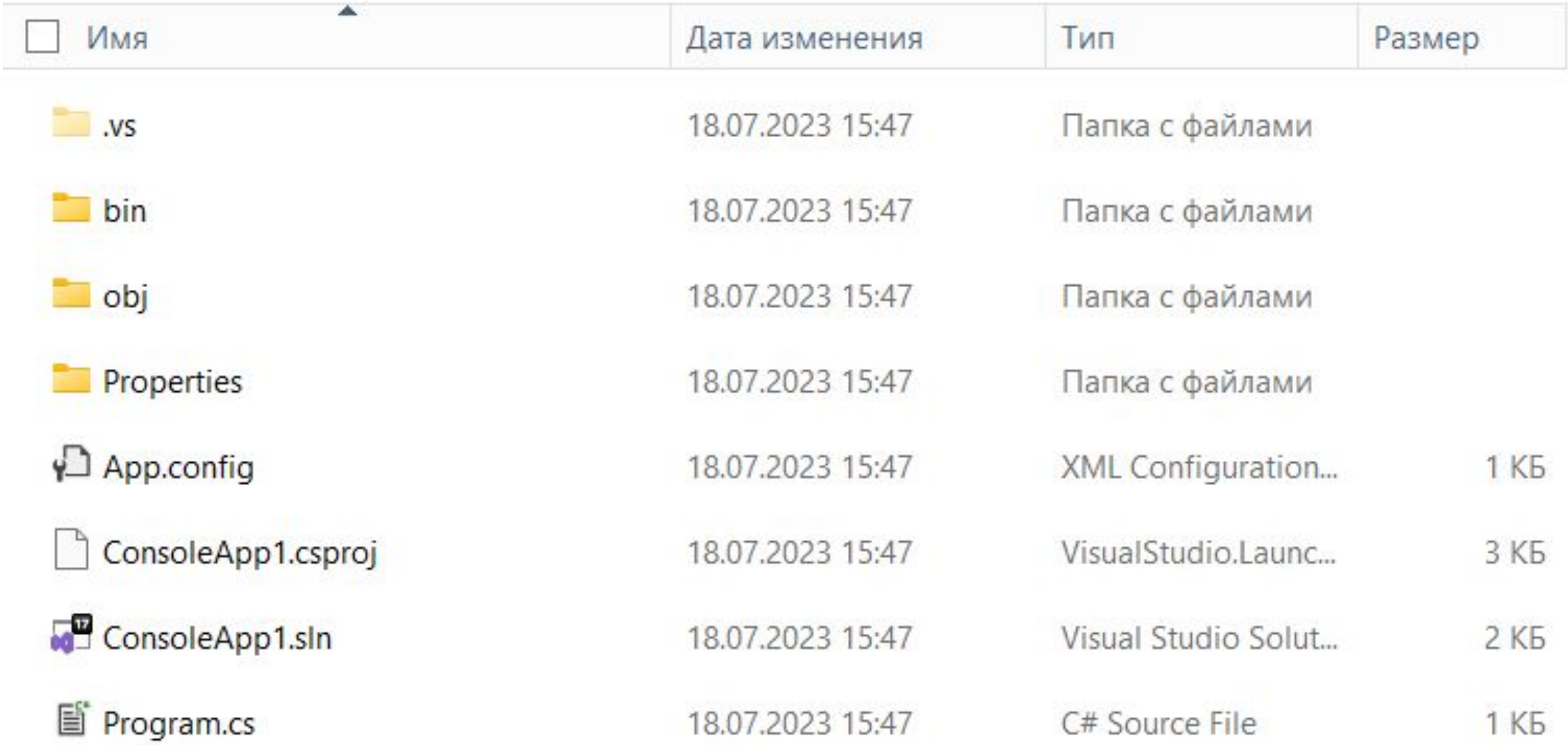
.xaml.cs .config

Console project

.cs .csproj

.config

Структура проектов и решений Visual Studio



Имя	Дата изменения	Тип	Размер
.vs	18.07.2023 15:47	Папка с файлами	
bin	18.07.2023 15:47	Папка с файлами	
obj	18.07.2023 15:47	Папка с файлами	
Properties	18.07.2023 15:47	Папка с файлами	
App.config	18.07.2023 15:47	XML Configuration...	1 КБ
ConsoleApp1.csproj	18.07.2023 15:47	VisualStudio.Launc...	3 КБ
ConsoleApp1.sln	18.07.2023 15:47	Visual Studio Solut...	2 КБ
Program.cs	18.07.2023 15:47	C# Source File	1 КБ

Структура проектов и решений Visual Studio

Файл	Описание
.cs	Файлы кода, которые могут принадлежать к одному проектному решению. Этот тип файла может быть одним из следующих: • модуль • файл Windows Forms • файл классов
.csproj	Файлы проекта, которые могут принадлежать к нескольким проектным решениям. Файл .csproj также хранит параметры проекта.
.aspx	Файлы, представляющие веб-страницы ASP.NET. Файл ASP.NET может содержать код Visual C# или использоваться сопровождающим .aspx.cs файлом для хранения кода в дополнение к разметке страницы.
.config	Файлы конфигурации это XML-файлы, использующиеся для хранения настроек на уровне приложения, например, таких как строки подключения к базе данных, которые затем можно изменять без повторной компиляции приложения.
.xaml	XAML файлы используются в WPF и Silverlight Microsoft® приложениях для определения элементов пользовательского интерфейса.

Структура проектов и решений Visual Studio

Файл	Описание
.sln	Файл решения Visual Studio 2022, обеспечивающий единую точку доступа к нескольким проектам, элементам проекта и элементам решения. Файл .sln стандартный текстовый файл, который не рекомендуется изменяять извне Visual Studio 2022.
.suo	Файл параметров пользователя решения, который хранит все настройки, которые изменяются при настройке Visual Studio 2022 IDE.

Структура проектов и решений Visual Studio

Разбиение на несколько проектов в одном решении Visual Studio обеспечивает следующие преимущества:

Позволяет работать с несколькими проектами в рамках одной Visual Studio 2022 сессии

Позволяет применять параметры конфигурации в глобальном масштабе для нескольких проектов

Позволяет развернуть несколько проектов в рамках единого решения

Visual Studio 2022

Открыть последние

Поиск в недавнем (ALT+"B")

На этой неделе

BallisticTask.sln — 12.07.2023 18:09
C:\Users\girl-\source\repos\BallisticTask

В этом месяце

WindowsFormsApp2.sln — 27.06.2023 11:21
C:\Users\girl-\Desktop\WindowsFormsApp2

Ранее

Laba2(CorC).sln — 01.04.2023 08:31
C:\Users\girl-\source\repos\Laba2(CorC)

2semestr.sln — 18.03.2023 09:30
C:\Users\girl-\source\repos\2semestr

DBChemical.sln — 20.01.2023 10:32
C:\Users\girl-\source\repos\DBChemical

Calculator.sln — 09.12.2022 14:15
C:\Users\girl-\Desktop\Calculator-Win-Forms\Calculator

calcul.sln — 03.12.2022 10:26
C:\Users\girl-\Desktop\calcul\calcul

Calc.sln — 03.12.2022 09:58
C:\Users\girl-\source\repos\Calc

CRISTALL.sln — 03.12.2022 08:53

Начало работы

Клонирование репозитория
Получить код из интернет-репозитория, например, GitHub или Azure DevOps

Открыть проект или решение
Открыть локальный проект Visual Studio или SLN-файл

Открыть локальную папку
Перейти и изменить код в любой папке

Создание проекта
Выберите шаблон проекта с формированием шаблонов кода, чтобы начать работу

Продолжить без кода →

Создание проекта

Последние шаблоны проектов

- Приложение Windows Forms (.NET Framework) — C#
- Консольное приложение (.NET Framework) — C#
- Приложение Windows Forms (Майкрософт) — C#

Поиск шаблонов (ALT+"B")

Очистить все

C# | Windows | Рабочий стол

- **Приложение Windows Forms (Майкрософт)**
  Шаблон проекта для создания приложения .NET Windows Forms (WinForms).
  C# Windows Рабочий стол
- **Приложение Windows Forms (.NET Framework)**
  Проект для создания приложения с пользовательским интерфейсом Windows Forms (WinForms)
  C# Windows Рабочий стол
- **Библиотека классов WPF (Майкрософт)**
  Проект для создания библиотеки классов, использующей приложение WPF .NET.
  C# Windows Рабочий стол Библиотека
- **Библиотека настраиваемых элементов управления WPF (Майкрософт)**
  Проект для создания библиотеки настраиваемых элементов управления для приложений WPF .NET.
  C# Windows Рабочий стол Библиотека
- **Библиотека пользовательских элементов управления WPF (Майкрософт)**
  Проект для создания библиотеки пользовательских элементов управления для приложений WPF .NET.
  C# Windows Рабочий стол Библиотека
- **Приложение WPF (Майкрософт)**
  Проект для создания приложения WPF .NET
  C# Windows Рабочий стол
- **Приложение WPF (.NET Framework)**
  Клиентское приложение Windows Presentation Foundation
  C# XAML Windows Рабочий стол
- Тестовый проект NUnit

Назад | Далее

Настроить новый проект

Консольное приложение (.NET Framework)

C# Windows Консоль

Имя проекта

ConsoleApp1

Расположение

C:\Users\girl-\source\repos

...

Имя решения

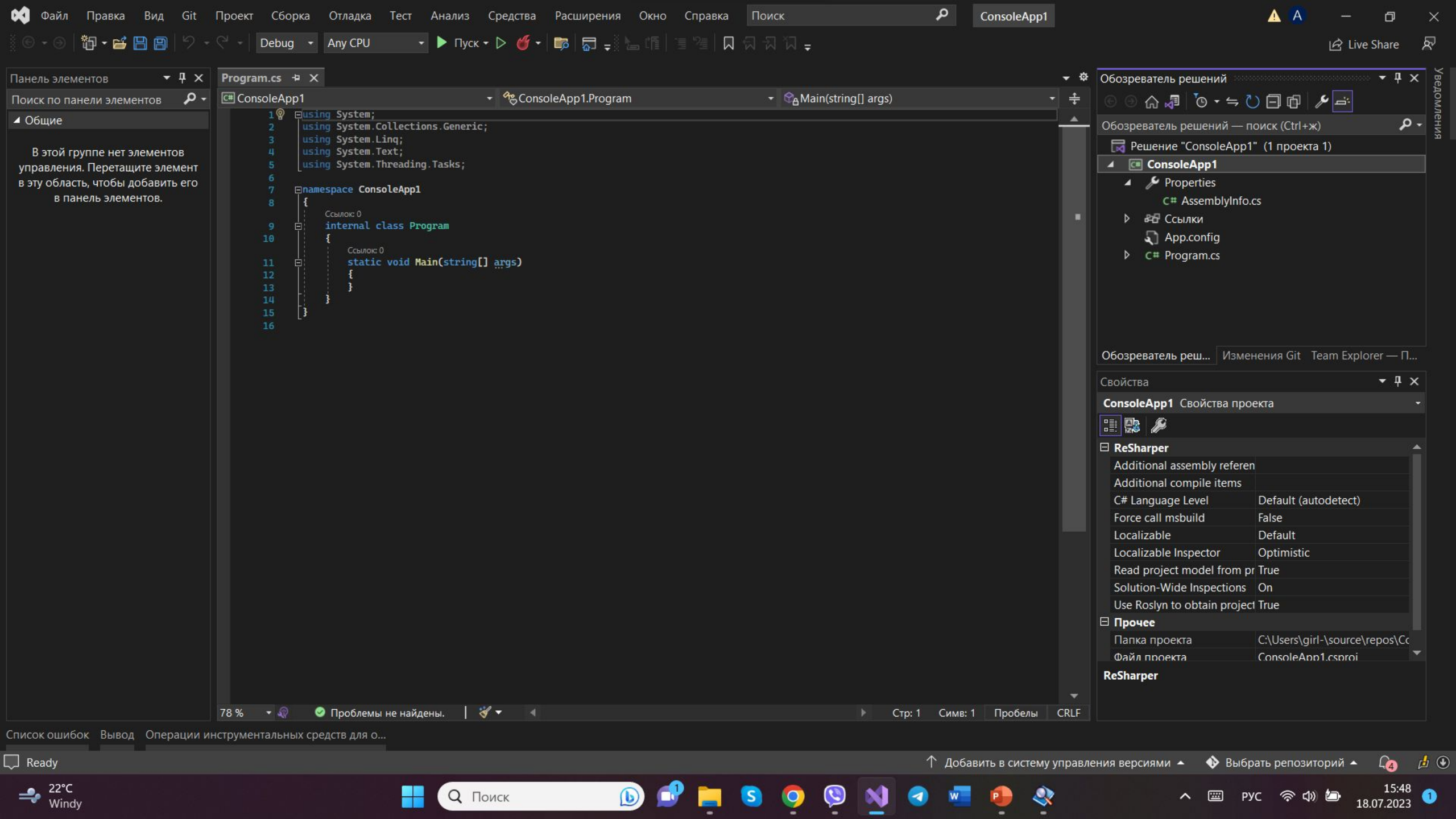
ConsoleApp1

Поместить решение и проект в одном каталоге

Платформа

.NET Framework 4.7.2

Назад Создать


Файл Правка Вид Git Проект Сборка Отладка Тест Анализ Средства Расширения Окно Справка
Поиск
ConsoleApp1
Debug
Any CPU
Пуск
Live Share
Панель элементов
Поиск по панели элементов
Общие
В этой группе нет элементов управления. Перетащите элемент в эту область, чтобы добавить его в панель элементов.
Program.cs
ConsoleApp1
ConsoleApp1.Program
Main(string[] args)
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
namespace ConsoleApp1
{
Ссылок: 0
internal class Program
{
Ссылок: 0
static void Main(string[] args)
{
}
}
}
78 %
Проблемы не найдены.
Стр: 1
Симв: 1
Пробелы
CRLF
Обозреватель решений
Обозреватель решений — поиск (Ctrl+ж)
Решение "ConsoleApp1" (1 проекта 1)
ConsoleApp1
Properties
AssemblyInfo.cs
Ссылки
App.config
Program.cs
Обозреватель реш...
Изменения Git
Team Explorer — П...
Свойства
ConsoleApp1 Свойства проекта
ReSharper
Additional assembly referen
Additional compile items
C# Language Level Default (autodetect)
Force call msbuild False
Localizable Default
Localizable Inspector Optimistic
Read project model from pr True
Solution-Wide Inspections On
Use Roslyn to obtain project True
Прочее
Папка проекта C:\Users\girl-\source\repos\Cc
Файл проекта ConsoleApp1.csproj
ReSharper
Уведомления
Список ошибок
Вывод
Операции инструментальных средств для о...
Ready
Добавить в систему управления версиями
Выбрать репозиторий
22°C
Windy
Поиск
РУС
15:48
18.07.2023

3. Написание приложений на C#

Классы и пространства имен

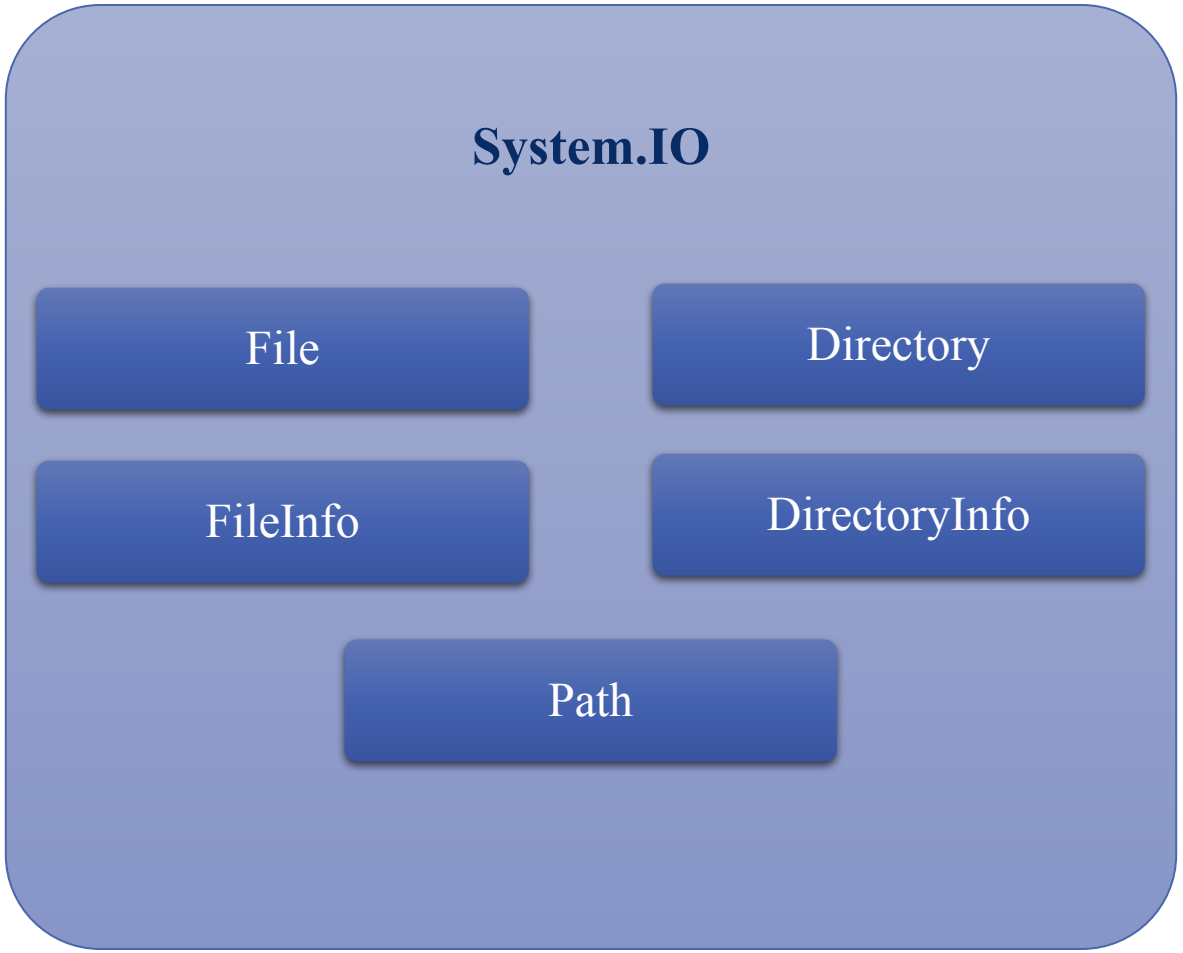
Класс

- Класс по существу чертеж, определяющий характеристики сущности, включает в себя свойства, определяющие типы данных, которые может содержать объект, и методы, описывающие поведение объекта
- Классы хранятся в сборках

Пространство имен

- Пространство имен представляет собой логический набор классов
- Пространство имен является средством для устранения неоднозначности классов, которые могут иметь одинаковые имена в различных сборках

Классы и пространства имен



Классы и пространства имен

Для использования класса, определенного в .NET Framework, следует выполнить следующие задачи:

Добавить ссылку на сборку, которая содержит скомпилированный код для класса

Импортировать пространство имен, которое содержит класс

```
using System;
using System.IO;
using System.Collections;
```

Структура консольного приложения

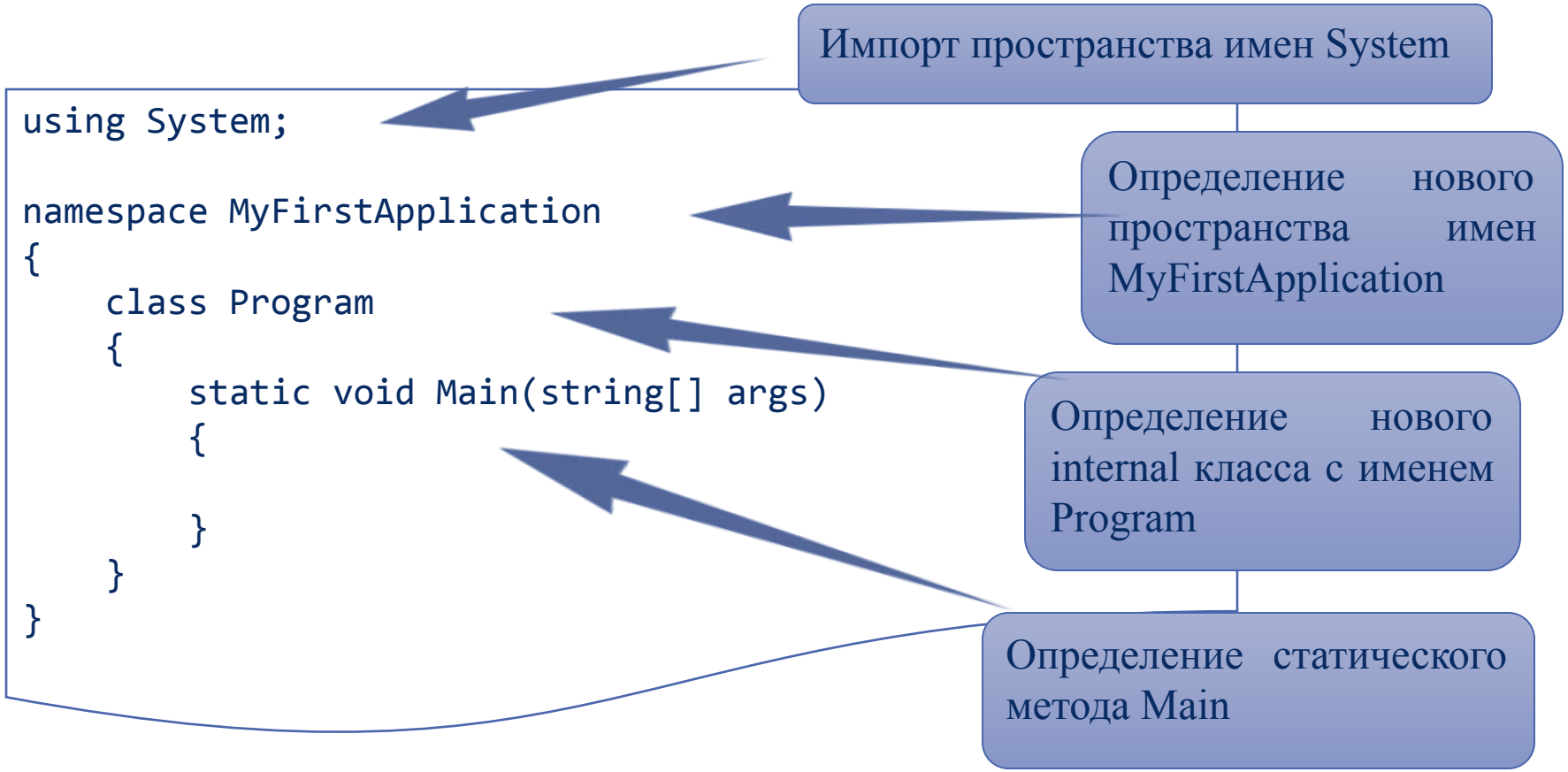
При создании нового консольного приложения с помощью шаблона Console Application, Visual Studio 2022 выполняет следующие задачи:

Создает новый файл с расширением .csproj для представления консольного проекта и структуры всех компонентов по умолчанию в консольном проекте

Добавляет ссылки на сборки библиотеки классов .NET Framework, которые обычно требуются консольным приложениям. Этот набор включает в себя сборку System

Создает файл Program.cs с методом Main, предоставляющий точку входа в консольное приложение

Структура консольного приложения



Выполнение ввода и вывода с использованием консольного приложения

```
using System;
...
Console.Clear();
```

```
using System;
...
int nextCharacter = Console.Read();
```

```
using System;
...
ConsoleKeyInfo key = Console.ReadKey();
```

```
using System;
...
string line = Console.ReadLine();
```

```
using System;
...
Console.Write("Hello there!");
```

```
using System;
...
Console.WriteLine("Hello there!");
```

Выполнение ввода и вывода с использованием консольного приложения

Метод	Описание
Clear()	Очищает окно и буфер консоли от данных. кода. using System; ... Console.Clear(); // clears the console display
Read()	Читает следующий символ из консоли. using System; ... int nextCharacter = Console.Read();
ReadKey()	Читает следующий символ или клавишу из окна консоли. using System; ... ConsoleKeyInfo key = Console.ReadKey();

Выполнение ввода и вывода с использованием консольного приложения

Метод	Описание
ReadLine()	Считывает следующую строку символов из окна консоли. using System; ... string line = Console.ReadLine();
Write()	Пишет текст в окне консоли. using System; ... Console.Write("Hello there!");
WriteLine()	Пишет текст в следующую строку в окне консоли. using System; ... Console.WriteLine("Hello there!");

Рекомендации по комментированию приложений C#

1. В начале процедуры следует поместить блок комментария, который должен включать информацию о цели процедуры, возвращаемом значении, аргументах и так далее
2. В длинных процедурах комментарии используются для того, чтобы выделить единицы работы в рамках процедуры
3. При объявлении переменных комментарии используются для указания того, как переменная будет использоваться
4. При написании структурного решения комментарии используются для указания, как решение будет выполнено и что оно означает

4. Синтаксис языка C#

Состав языка

- **Символы**:
  - буквы: A-Z, a-z, _, буквы нац. алфавитов
  - цифры: 0-9, A-F
  - спец. символы: +, *, {, …
  - пробельные символы

Лексема (token, токен) – минимальная единица языка, имеющая самостоятельный смысл

- **Лексемы**:
  - константы 2 0.11 “Вася”
  - имена Vasia a _11
  - ключевые слова double do if
  - знаки операций + <= new
  - разделители ; [ ] ,
- **Выражения**
  - выражение - правило вычисления значения: a + b
- **Операторы**
  - исполняемые: c = a + b;
  - описания: double a, b;

```
static void Main()
      { Самолёт Боинг =
                  new Самолёт();
        Боинг.Полетели();
      }
```

Синтаксис языка

```
1   using System;
2
3   namespace _Class4Lecture
4   {
        Ссылок: 0
5       internal class Program
6       {
            Ссылок: 0
7           static void Main(string[] args)
8           {
9               int x = 12 * 30;
10              Console.WriteLine
11                  (x);
12              int a = 3; // Комментарий относительно присваивания 3 переменной x
13              int b = 3; /* Это комментарий, который
14  занимает две строки */
15
16
17
18
19          }
20      }
21  }
22
```

Идентификаторы и ключевые слова

Идентификаторы представляют собой имена, которые программисты выбирают для своих классов, методов, переменных и т.д.

- Идентификатор должен быть целостным словом, которое по существу состоит из символов Unicode и начинается с буквы или символа подчеркивания.
- Идентификаторы в C# чувствительны к регистру символов.
- По соглашению для параметров, локальных переменных и закрытых полей должен применяться “верблюжий ” стиль (вроде myVariable), а для всех остальных идентификаторов — стиль Pascal (наподобие MyMethod).
- Идентификаторы имеют пространства имен, классы, методы, поля классов, различные переменные и константы (о которых мы поговорим чуть позже), а так же, другие сущности программ.
- Если вы действительно хотите применять идентификатор с именем, которое конфликтует с ключевым словом, то к нему необходимо добавить префикс @.
  - int using = 123; // Не допускается
  - int @using = 123; // Разрешено

Ключевые слова являются именами, которые имеют для компилятора особый смысл.

- Большинство ключевых слов зарезервированы, а это означает, что их нельзя использовать в качестве идентификаторов.

Ключевые слова

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

Контекстные ключевые слова

Неоднозначность с контекстными ключевыми словами не может возникать внутри контекста, в котором они используются.

Некоторые ключевые слова являются контекстными, т.е. их можно использовать также в качестве идентификаторов — без символа @:

add	equals	nameof	set
alias	from	not	unmanaged
and	get	on	value
ascending	global	or	var
async	group	orderby	with
await	in	partial	when
by	into	record	where
descending	join	remove	yield
dynamic	let	select	

Блоки кода, знаки пунктуации и операции

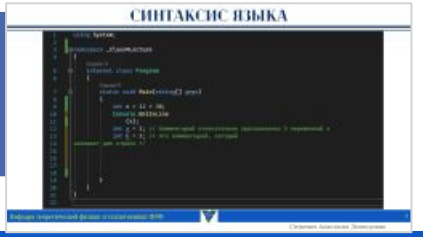
Блоки кода

- Любой блок кода является неким контейнером, который создает локальную область видимости. Это значит, что в создавая блок кода, мы можем оградить его содержимое от внешних вмешательств и различных конфликтов. И так, в C# границы блока кода обозначаются фигурными скобками. Начало блока открывающей, т.е. «{», а конец — закрывающей, т. е. «}».

Знаки пунктуации помогают размечать структуру программы.

- Операторы могут записываться в нескольких строках:

Операция преобразует и объединяет выражения.



Ситкевич Анастасия Леонидовна

Комментарии

- Комментарии используются для объяснения кода. Компиляторы игнорируют записи комментариев. Многострочные комментарии в программах на C # начинаются с / * и заканчиваются символами * /. Однострочные комментарии обозначаются символом «//».

```
int x = 3; // Комментарий относительно присваивания 3 переменной x
int x = 3; /* Это комментарий, который
занимает две строки */
```

Константы и переменные только для чтения

Константы

Используются только для хранения неизменяемых данных

Объявляются с помощью ключевого слова const

Значение можно инициализировать только во время разработки

```
const DataType variableName = Value;
const double PI = 3.14159;
int radius = 5;
double area = PI * radius * radius;
double circumference = 2 * PI * radius;
```

Константы и переменные только для чтения

Переменные только для чтения (read-only)

Используются только для хранения неизменяемых данных

Объявляются с помощью ключевого слова readonly

Значение можно инициализировать во время выполнения

```
readonly DataType variableName = Value;
readonly string currentDateTime = DateTime.Now.ToString();
```

летералы

Литералы

Литералы представляют неизменяемые значения (иногда их еще называют константами). Литералы можно передавать переменным в качестве значения.

- Литералы
  - Логические
  - Целочисленные
  - Вещественные
  - Символьные
  - Строчные
  - Ключевое слово null

Ситкевич Анастасия Леонидовна

Литералы

```
Console.WriteLine(true);
Console.WriteLine(false);
```

Логические литералы

Есть две логических константы - true (истина) и false (ложь):

Вещественные литералы

Вещественные литералы представляют дробные числа.

Этот тип литералов имеет две формы.

- вещественные числа с фиксированной запятой, при которой дробную часть отделяется от целой части точкой.
- могут определяться в экспоненциальной форме MEp, где M — мантисса, E - экспонента

```
3.14
100.001
-0.38
```

```
Console.WriteLine(3.2e3);
Console.WriteLine(1.2E-1);
```


Ситкевич Анастасия Леонидовна

Литералы

Целочисленные литералы

Целочисленные литералы представляют положительные и отрицательные целые числа

Целочисленные литералы могут быть выражены в

- десятичной,
- шестнадцатеричной
- двоичной форме.

```
Console.WriteLine(-11);
Console.WriteLine(5);
Console.WriteLine(505);
```

Числа в двоичной форме предваряются символами 0b, после которых идет набор из нулей и единиц:

```
Console.WriteLine(0b11);        // 3
Console.WriteLine(0b1011);      //11
Console.WriteLine(0b100001);    // 33
```

Для записи числа в шестнадцатеричной форме применяются символы 0x, после которых идет набор символов от 0 до 9 и от A до F, которые собственно представляют число:

```
Console.WriteLine(0x0A);    // 10
Console.WriteLine(0xFF);    // 255
Console.WriteLine(0xA1);    // 161
```

Литералы

Символьные литералы

Символьные литералы представляют одиночные символы. Символы заключаются в одинарные кавычки.

обычные символы:

```
Console.WriteLine('2’);
Console.WriteLine('A’);
Console.WriteLine('T');
```

управляющие последовательности

```
Console.WriteLine('\x78');    //x - из таблицы ASCII
Console.WriteLine('\x5A');    //Z - из таблицы ASCII
Console.WriteLine('\u0420');    //P - Unicode
Console.WriteLine('\u0421');    // C – Unicode
'\n' - перевод строки
'\t' – табуляция
'\\' – слеш
```

Литералы

Строковые литералы

Строковые литералы представляют строки. Строки заключаются в двойные кавычки

```
Console.WriteLine("hello");
Console.WriteLine("фыва");
Console.WriteLine("hello word");
```

Если внутри строки необходимо вывести двойную кавычку, то такая внутренняя кавычка предваряется обратным слешем:

```
Console.WriteLine("Компания \"Рога и копыта\"");
```

null

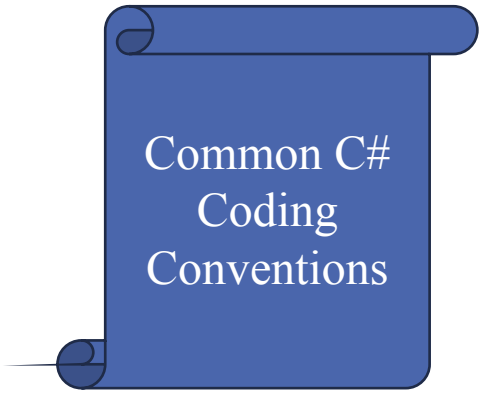
null представляет ссылку, которая не указывает ни на какой объект. То есть по сути отсутствие значения.

Нотации

Понятные и согласованные между собой имена — основа хорошего стиля. Существует несколько *нотаций* — соглашений о правилах создания имен.

В C# для именования различных видов программных объектов чаще всего используются две нотации:

- *Нотация Паскаля* - каждое слово начинается с прописной буквы:
  - MaxLength, MyFuzzyShooshpanchik
- *Camel notation* - с прописной буквы начинается каждое слово, составляющее идентификатор, кроме первого:
  - maxLength, myFuzzyShooshpanchik

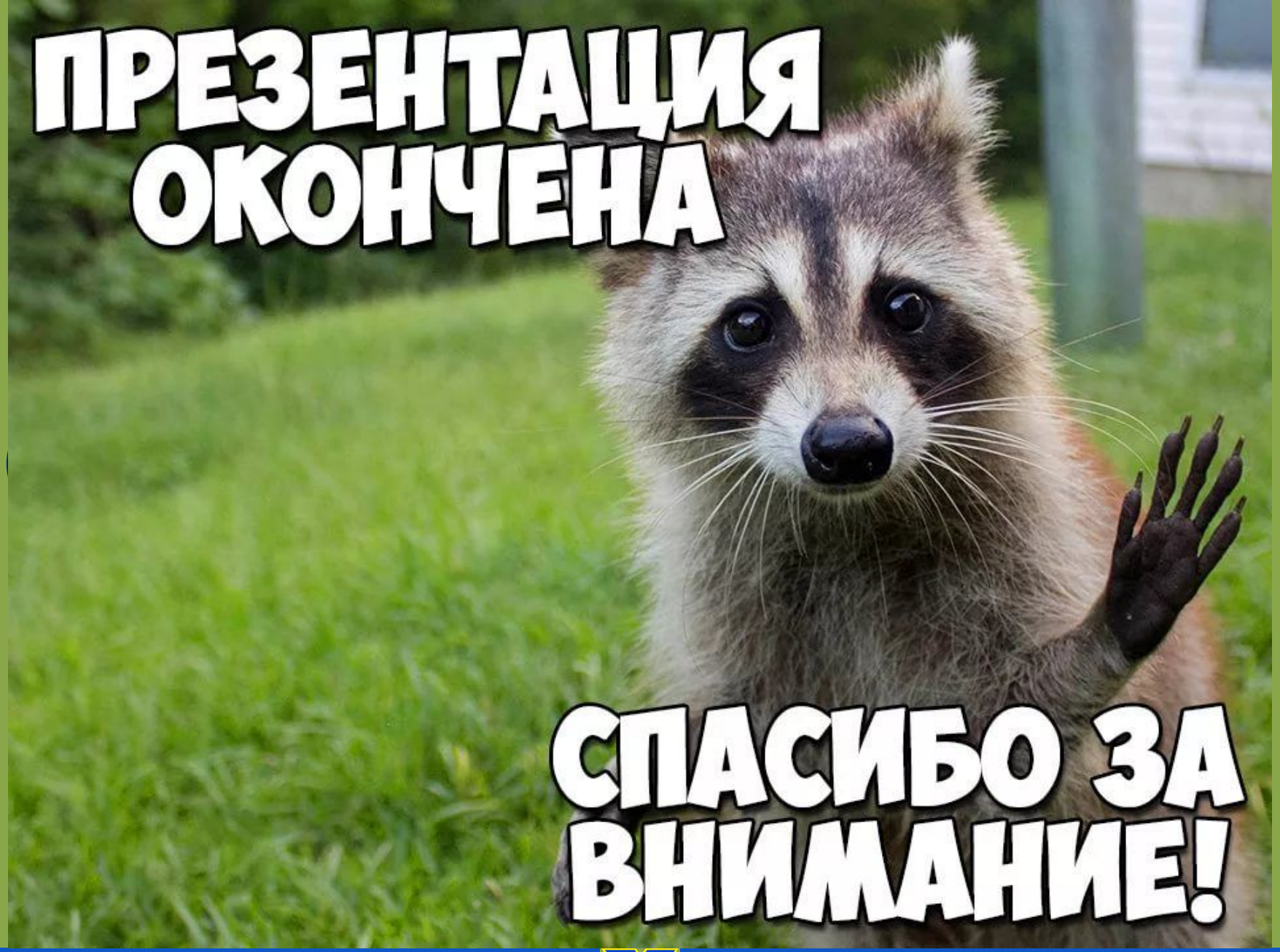


Common C# Coding Conventions

Object Name	Notation	Example
Namespace name	PascalCase	namespace DBChemicalElements
Class name	PascalCase	public class ChemicalElementModal
Interface name	PascalCase	public class IChemicalElement
Constructor name	PascalCase	public FormInsertcs()
Method name	PascalCase	private SqlDataAdapter CreateAdapter(string sqlCommand)
Method arguments	camelCase	private SqlDataAdapter CreateAdapter(string sqlCommand)
Local variables	camelCase	var dataAdapter = new SqlDataAdapter(command);
Constants name	PascalCase	public const string ElementType = "Metal";
Field name Public	PascalCase	public string ElementName;
Field name Private	_camelCase	private DataSet _dataSet = null;
Properties name	PascalCase	public ChemicalElementModal ChemicalElement { get; private set; }
Delegate name	PascalCase	delegate void InfoMessage();
Enum type name	PascalCase	enum ElementType { }

Общие рекомендации по именованию

	правильно	ошибка	исключение
Не используйте в имени тип переменной в виде префикса	int counter; string name;	int iCounter; string strName;	-
Не используйте Caps при именовании констант	public const string ShippingType = "DropShip";	public const string SHIPPINGTYPE = "DropShip";	-
В качестве имен используйте слова, передающие смысл переменной / метода / класса и т.д. (не допускается использования транслитерации)	Private void GetData() {} string elementName;	Private void Method1 () {} Private void PolychitDannye () {} string a ;	
Используйте существительные для именования классов / полей и т.д. Для именования методов используйте глаголы	public class Employee {} Private void GetData() {} string elementName	public class Get {} Private void Data() {}	
Избегайте использования аббревиатур	UserGroup userGroup;Assignment employeeAssignment;	UserGroup usrGrp;Assignment empAssignment;	CustomerId customerId;XmlDocument xmlDocument;FtpHelper ftpHelper;UriPart uriPart;
Не используйте нижнее подчеркивание	public DateTime clientAppointment;public TimeSpan timeLeft;	public DateTime client_Appointment;public TimeSpan time_Left;	private DateTime _registrationDate;
Используйте предопределенные имена типов (псевдонимы C#), такие как int, float, string для локального объявления параметров и членов. Используйте имена .NET Framework, такие как Int32, Single, String, при доступе к статическим членам типа, таким как Int32.TryParse или String.Join.	string firstName;int lastIndex;bool isSaved;string commaSeparatedNames = String.Join(", ", names);int index = Int32.Parse(input);	String firstName;Int32 lastIndex;Boolean isSaved;string commaSeparatedNames = string.Join(", ", names);int index = int.Parse(input);	
При именовании интерфейсов используйте префикс I	public interface IShape {}public interface IShapeCollection{}		


ПРЕЗЕНТАЦИЯ
ОКОНЧЕНА
СПАСИБО ЗА
ВНИМАНИЕ!

Файл Правка Вид Git Проект Сборка Отладка Тест Анализ Средства Расширения Окно Справка Поиск ConsoleApp1

Debug Any CPU Пуск Live Share

Панель элементов

Поиск по панели элементов

Общие

В этой группе нет элементов управления. Перетащите элемент в эту область, чтобы добавить его в панель элементов.

Program.cs*

ConsoleApp1 ConsoleApp1.Program width

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace ConsoleApp1
{
    Ссылок: 0
    internal class Program
    {
        //Поля класса
        private double length;
        private double width;

        //Метод, задающий параметрам объекта определенные значения
        Ссылок: 0
        public void SetParams()
        {
            length = 4;
            width = 3;
        }

        //Метод, возвращающий вычисленную по длине и ширине площадь
        Ссылок: 1
        public double GetArea()
        {
            return length * width;
        }

        //Метод, возвращающий описание объекта
        Ссылок: 0
        public void Display()
        {
            Console.WriteLine("Length: {0}", length);
            Console.WriteLine("Width: {0}", width);
            Console.WriteLine("Area: {0}", GetArea());
        }


        Ссылок: 0
        static void Main(string[] args)
        {

        }
    }
}
```

78 % Проблемы не найдены. Стр: 42 Симв: 2 Пробелы CRLF

Обозреватель решений

Обозреватель решений — поиск (Ctrl+;)

Решение "ConsoleApp1" (1 проекта 1)

ConsoleApp1

Properties

AssemblyInfo.cs

Ссылки

App.config

Program.cs

Обозреватель реш... Изменения Git Team Explorer — П...

Свойства

Уведомления

Список ошибок Вывод Операции инструментальных средств для о...

Готово Добавить в систему управления версиями Выбрать репозиторий

23°C Near record Поиск ENG 18:08 18.07.2023