**Конспект по C#**

**Оглавление**

Оглавление

[Введение 2](#_Toc135077532)

[Шаблоны проектов в С# 4](#_Toc135077533)

[Шаблон консольного приложения 4](#_Toc135077534)

[Шаблон приложения с GUI 6](#_Toc135077535)

[Среды разработки С# 8](#_Toc135077536)

[Основные типы, особенности типизации, контейнеры 10](#_Toc135077537)

[Общие сведения о классах 15](#_Toc135077538)

[Создание проекта модульного теста 20](#_Toc135077539)

[Литература 22](#_Toc135077540)

Введение

C#, C-sharp, си-шарп - язык программирования, сочетающий объектно-ориентированные и аспектно-ориентированные концепции. Разработан в 1998—2001 годах группой инженеров под руководством Андерсa Хейлсбергa в компании Microsoft как основной язык разработки приложений для платформы Microsoft .NET. Компилятор с C# входит в стандартную установку самой .NET, поэтому программы на нём можно создавать и компилировать даже без инструментальных средств вроде Visual Studio.

C# относится к семье языков с C-подобным синтаксисом, из них его синтаксис наиболее близок к С++ и Java. Язык имеет строгую статическую типизацию, поддерживает полиморфизм, перегрузку операторов, указатели на функции-члены классов, атрибуты, события, свойства, исключения, комментарии в формате XML. Переняв многое от своих предшественников — языков С++, Java, Delphi, Модула и Smalltalk — С#, опираясь на практику их использования, исключает некоторые модели, зарекомендовавшие себя как проблематичные при разработке программных систем: так, C# не поддерживает множественное наследование классов (в отличие от C++).

**Особенности языка**

C# разрабатывался как язык программирования прикладного уровня для CLR и, как таковой, зависит, прежде всего, от возможностей самой CLR. Это касается, прежде всего, системы типов C#, которая отражает FCL. Присутствие или отсутствие тех или иных выразительных особенностей языка диктуется тем, может ли конкретная языковая особенность быть транслирована в соответствующие конструкции CLR. Так, с развитием CLR от версии 1.1 к 2.0 значительно обогатился и сам C#; подобного взаимодействия следует ожидать и в дальнейшем. (Однако эта закономерность была нарушена с выходом C# 3.0, представляющим собой расширения языка, не опирающиеся на расширения платформы .NET.) CLR предоставляет C#, как и всем другим .NET-ориентированным языкам, многие возможности, которых лишены «классические» языки программирования. Например, сборка мусора не реализована в самом C#, а производится CLR для программ, написанных на C# точно так же, как это делается для программ на VB.NET, J# и др.

**Название языка**

Символ # в названии языка можно интерпретировать и как две пары плюсов ++, намекающие на новый шаг в развитии языка по сравнению с C++ (подобно шагу от C к C++), и как музыкальный символ диез, вместе с буквой C составляющий в английском языке название ноты до-диез. Последнее и дало название языку. Несмотря на то, что символ # (октоторп) в действительности является символом для обозначения номера на большинстве клавиатур и отличается от символа диез # (Unicode U+266F), Microsoft, как автор языка, неоднократно обращалась к своим клиентам с просьбой принять это допущение.

**С# последняя версия стандарта**

**C# версии 11**

*Выпущено в ноябре 2022 г.*

В C# 11 были добавлены следующие функции:

* Необработанные строковые литералы
* Поддержка универсальной математики
* Универсальные атрибуты
* Строковые литералы UTF-8
* Новые строки в выражениях интерполяции строк
* Шаблоны списков
* Локальные типы файлов
* Обязательные члены
* Автоматические структуры по умолчанию
* Сопоставление Span<char> шаблонов в константе string
* Расширенная nameof область
* Числовой intPtr
* ref поля и scoped ref
* Улучшенное преобразование групп методов для делегирования.
* Предупреждающий сигнал 7

В C# 11 представлена универсальная математика и несколько функций, поддерживающих эту цель. Числовые алгоритмы можно написать один раз для всех числовых типов. Существует больше возможностей, которые упрощают работу с типами struct , например необходимые члены и автоматические структуры по умолчанию. Работа со строками упрощается благодаря необработанным строковым литералам, новой строке в интерполяциях строк и строковым литералам UTF-8. Такие функции, как локальные типы файлов, упрощают генераторы источников. Наконец, шаблоны списков добавляют дополнительную поддержку для сопоставления шаблонов.

Шаблоны проектов в С#

Чтобы получить доступ к шаблонам проектов UWP C#, когда вы создаете проект в Visual Studio, отфильтруйте язык, выбрав C# , платформу, выбрав Windows, и тип проекта, выбрав значение UWP.

Эти шаблоны проектов можно использовать для создания приложений UWP на C#.

|  |  |
| --- | --- |
| Пустое приложение (универсальные приложения для Windows) | Создает приложение UWP. Созданный проект содержит базовую страницу, производную от класса [Windows.UI.Xaml.Controls.Page](https://learn.microsoft.com/ru-ru/uwp/api/windows.ui.xaml.controls.page), с помощью которой можно приступить к созданию пользовательского интерфейса. |
| Приложение модульных тестов (универсальное приложение для Windows) | Создает проект модульного теста на C# для приложения UWP. Дополнительные сведения см. в статье [Модульное тестирование кода C#](https://learn.microsoft.com/ru-ru/visualstudio/test/unit-testing-visual-csharp-code-in-a-store-app). |

С помощью этих шаблонов проектов можно создавать фрагменты приложений UWP на C#.

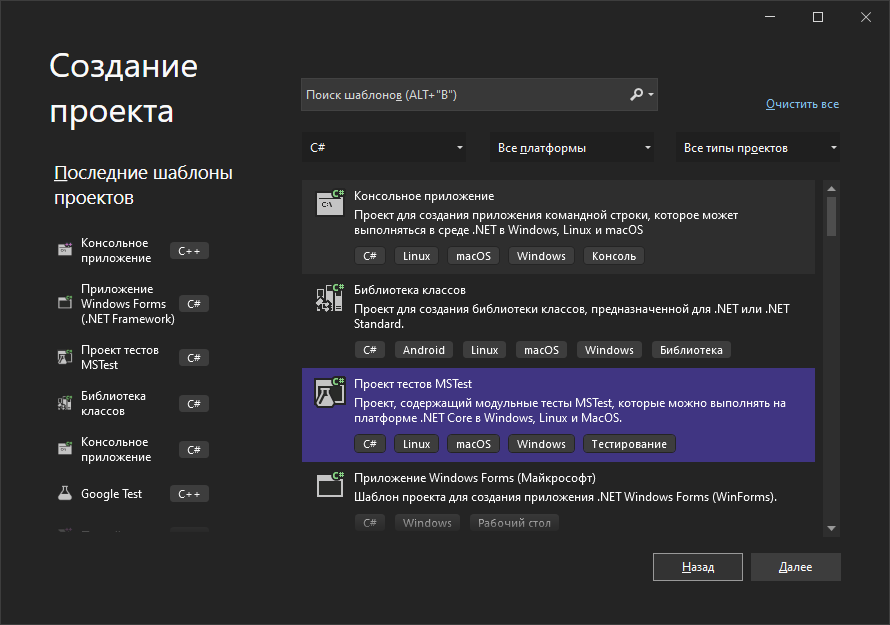
|  |  |
| --- | --- |
| Библиотека классов (универсальные приложения для Windows) | Создает управляемую библиотеку классов (DLL) на языке C#, которую могут использовать другие приложения UWP, написанные с применением управляемого кода. |
| Компонент среды выполнения Windows (универсальные приложения для Windows) | Создает [компонент среды выполнения Windows](https://learn.microsoft.com/ru-ru/windows/uwp/winrt-components/) на C#, который может использоваться в любом приложении UWP, независимо от того, на каком языке программирования такое приложение написано. |
| Необязательный пакет кода (универсальные приложения для Windows) | Создает дополнительный пакет с исполняемым кодом C#, который может загрузить приложение. Подробные сведения см. в статье [Дополнительные пакеты с исполняемым кодом](https://learn.microsoft.com/ru-ru/windows/msix/package/optional-packages-with-executable-code). |

Шаблон консольного приложения

Запустите Visual Studio.

На начальной странице выберите Создать проект.

На странице Создание проекта выберите C# из списка языков, а затем — Все платформы из списка платформ. Выберите шаблон Консольное приложение и нажмите кнопку Далее.



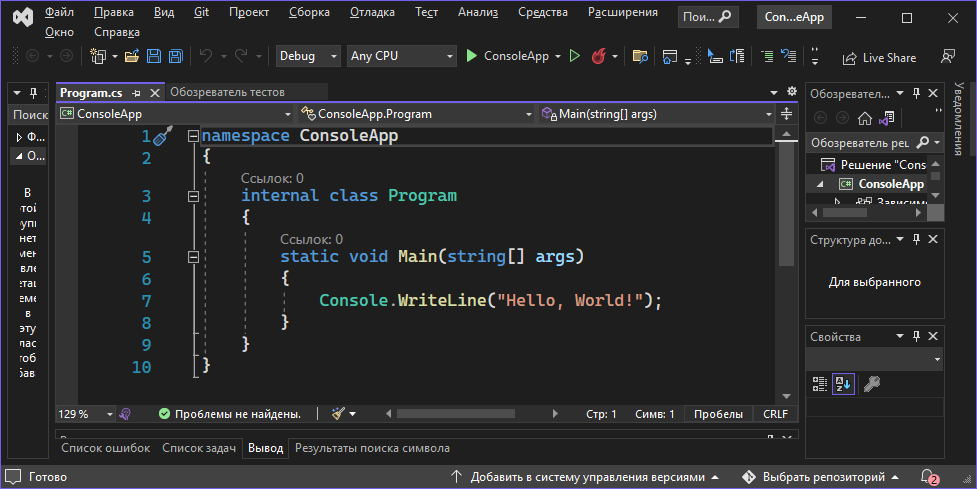
В диалоговом окне Настройка нового проекта введите название проекта в поле Имя проекта . Теперь щелкните Далее.

В диалоговом окне Дополнительные сведения выполните следующие действия.

Выберите .NET нужной версии.

Выберите «Не использовать [операторы верхнего уровня](https://learn.microsoft.com/ru-ru/dotnet/csharp/whats-new/tutorials/top-level-statements)».

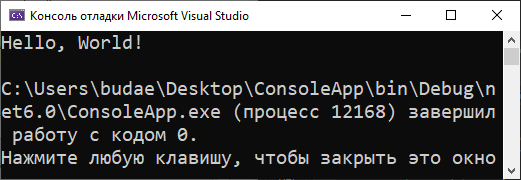
Нажмите кнопку создания.

Шаблон создает простое приложение, которое отображает сообщение "Hello, World!" в окне консоли. Код находится в файле Program.cs:

Код определяет класс Program с одним методом Main, который принимает массив String в качестве аргумента. Main — точка входа в приложение. Это метод, который автоматически вызывается средой выполнения при запуске приложения. Все аргументы, предоставленные в командной строке при запуске приложения, доступны через массив args.

Нажмите клавиши CTRL+F5, чтобы запустить программу без отладки.

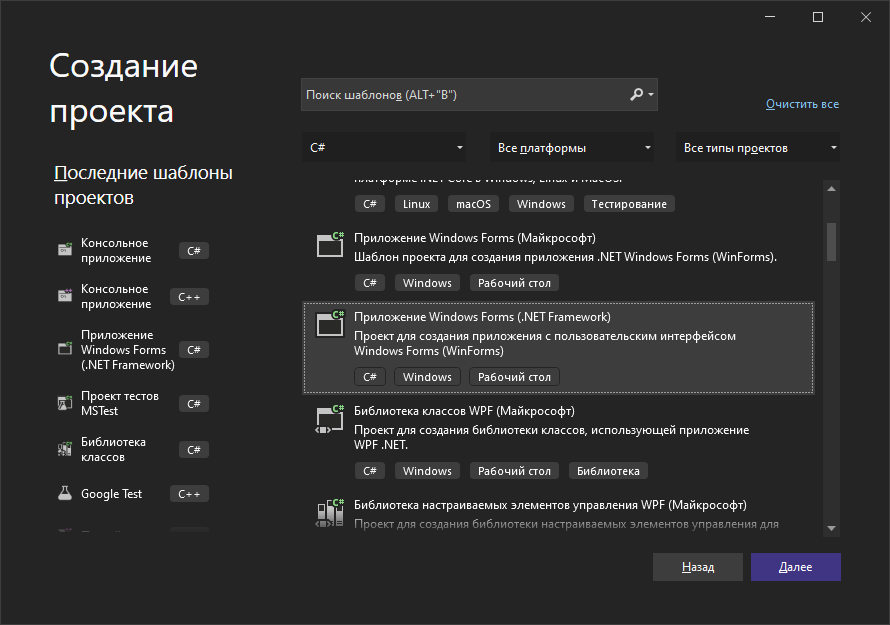
Откроется окно консоли с текстом "Hello, World!" на экране.



Для закрытия консольного окна нажмите любую клавишу.

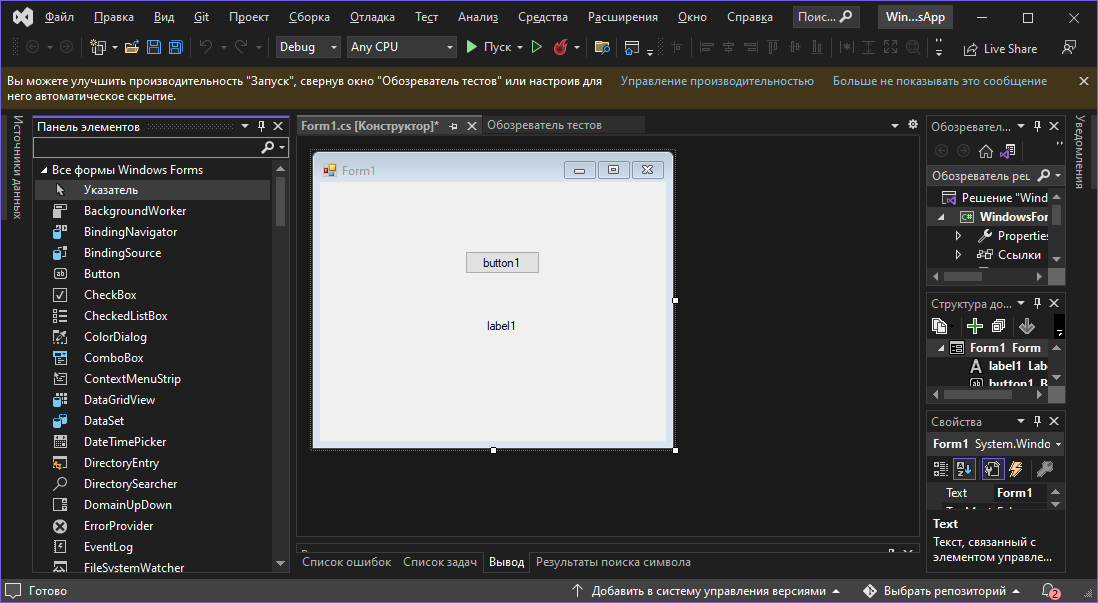
Шаблон приложения с GUI

На странице Создание проекта выберите C# из списка языков, а затем — Все платформы из списка платформ. Выберите шаблон Приложение Windows Forms (.NET Framework) и нажмите кнопку Далее.

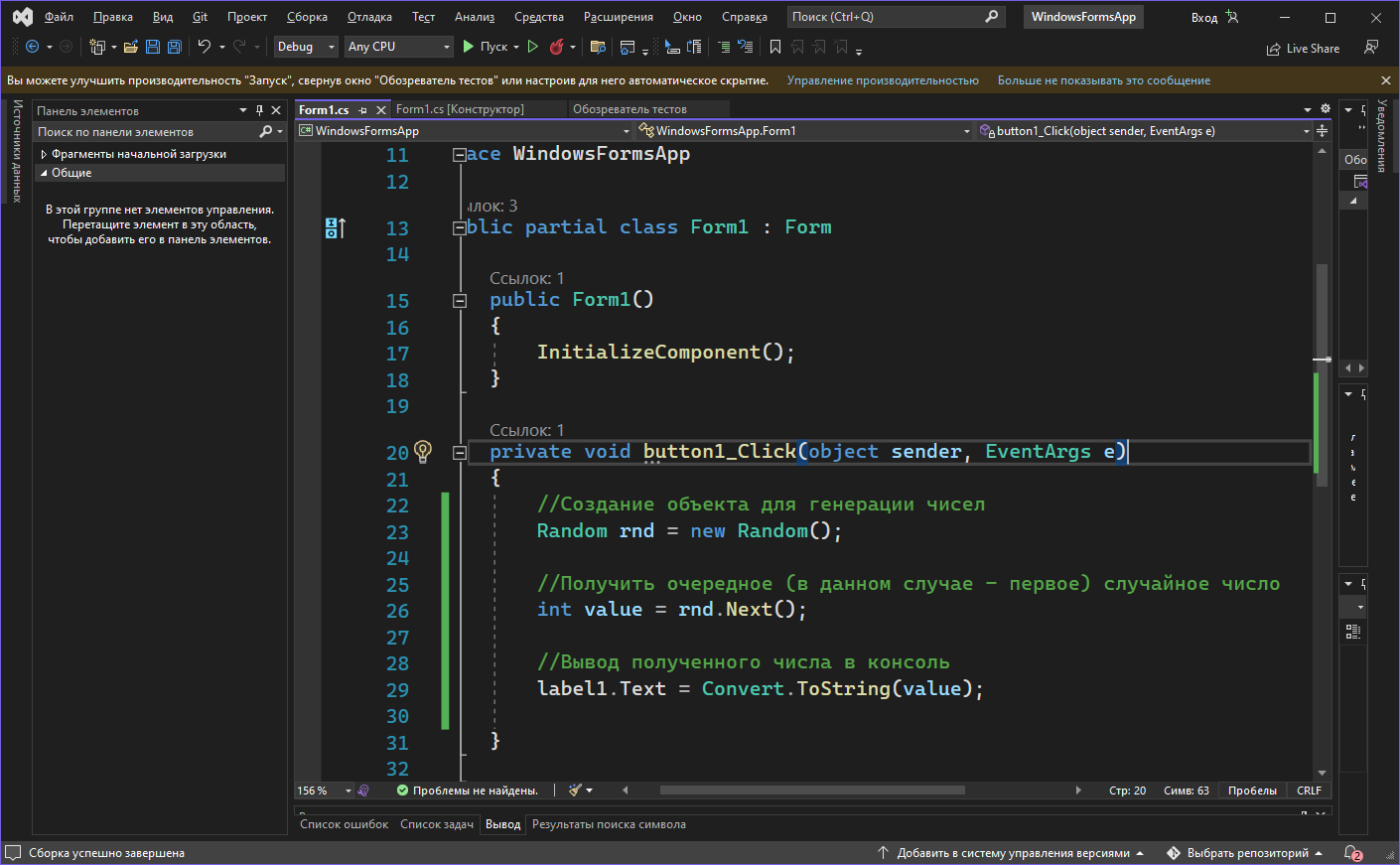


В следующем окне введите имя проекта и выберите необходимую версию .NET Framework

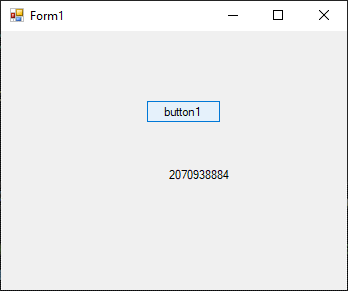
В появившемся окне перетащите из Панели элементов Button и Label.



Нажмите 2 раза на Кнопку. Откроется файл с обработчиками событий, в котором мы создадим вывод случайного числа по нажатии на кнопку.



Нажмите клавиши CTRL+F5, чтобы запустить программу без отладки. При нажатии на кнопку выводится случайное число



Среды разработки С#

***Visual Studio***

Описание: самая «правильная» среда разработки. С Visual Studio многие начинают знакомиться с языком и не расстаются с ней на протяжении всей карьеры программиста.

Плюсы:

* Официальная. Так как и язык, и среда разработки созданы в Microsoft, логично предположить, что ничего более функционального вы не найдете во всем Интернете. В некоторых случаях без Visual Studio не обойтись — например, при использовании технологий UWP и WPF.
* Бесплатная. Версии «Community edition» для рядового пользователя будет достаточно. Тем более, теперь можно подключать плагины (в отличие от старой версии Express).
* Функциональная. В Visual Studio множество качественных плагинов. С их помощью можно расширить функциональность приложения и подключить другие языки.
* Поддерживает платформы .NET. Visual Studio имеет широкие возможности по разработке приложений под Windows, в том числе в .NET-сегменте.
* Облачные хранилища. Зарегистрируйтесь в сообществе Visual Studio — и получите доступ к облачному хранилищу, где сможете располагать файлы проектов.
* Корпоративность. Технология бэклога позволяет членам команды взаимодействовать при гибкой методологии разработки.

Минусы:

* Баги при переходах с триал-версии. При переходе на платную версию могут теряться настройки и нарушаться работа корпоративного сервера.
* Сложность. Самостоятельно освоить Visual Studio новичку будет непросто — слишком много доступных функций, спрятанных в подразделах меню.

***Project Rider***

Описание: среда от JetBrains для работы с платформой .NET. Выпущена в прошлом году, но уже приобрела много поклонников.

Плюсы:

* ReSharper. Это плагин, изначально разработанный для повышения производительности Visual Studio. Теперь на его основе выпущена IDE.
* Поддержка полного цикла. Фирменная черта продуктов JetBrains, воплощенная и в Project Rider. С ним вы сможете организовать весь цикл создания ПО: от идеи до поддержки.
* Функциональность. Project Rider позволяет подключить MSBuild и XBuild, работать с CLI-проектами и организовать отладку приложений .NET and Mono. Множество опций для быстрого создания кода улучшает производительность.
* Multiple runtime. Поддержка нескольких запущенных программ.
* Кроссплатформенность. Project Rider работает с Windows, Linux и MacOS.
* Контроль версий. Встроенный инструмент позволяет напрямую организовать работу с Git, Mercurial и TFS.

Минусы:

* Молодость. Часть функциональности еще в разработке, не все стартовые баги исправлены.
* Стоимость. Самая дешевая версия Project Rider обойдется в 139 долларов за первый год использования. Но есть триал-версия и специальные предложения для студентов и непрофильных организаций.

***Eclipse***

Описание: одна из самых популярных мультиязычных сред. Ориентирована преимущественно на разработку Java-приложений, но полезна и для кодов на C#.

Плюсы:

* Множество плагинов. У Eclipse едва ли не самое большое число надстроек — «на все случаи жизни».
* Активное сообщество. Помогает быстрее освоить среду разработки, выпускает новые плагины.
* Отличные компилятор и отладчик. Первый работает на порядок быстрее, чем у конкурентов, второй — показывает потоки, пересечения, позволяет гибко управлять ходом отладки.
* Кастомизация. Благодаря плагинам и настройкам можно полностью персонализировать Eclipse.
* Бесплатность. Это open-source проект, абсолютно бесплатный.
* Высокая функциональность. Благодаря разработчикам-официалам и членам сообщества с помощью Eclipse можно провести любой C#-продукт по полному циклу разработки.

Минусы:

* Сложность. Как и любой функциональный продукт, Eclipse может показаться новичку слишком сложным.
* Нет гарантий надежности. Так как плагины создаются сообществом, за их качество отвечает только разработчик. Кроме того, сами создатели Eclipse с каждой новой версией плодят баги, не успевая порой исправлять старые.

Основные типы, особенности типизации, контейнеры

***Типы данных***

Как и во многих языках программирования, в C# есть своя система типов данных, которая используется для создания переменных. Тип данных определяет внутреннее представление данных, множество значений, которые может принимать объект, а также допустимые действия, которые можно применять над объектом.

* bool: хранит значение true или false (логические литералы). Представлен системным типом System.Boolean

bool alive = true;

bool isDead = false;

* byte: хранит целое число от 0 до 255 и занимает 1 байт. Представлен системным типом System.Byte

byte bit1 = 1;

byte bit2 = 102;

* sbyte: хранит целое число от -128 до 127 и занимает 1 байт. Представлен системным типом System.SByte

sbyte bit1 = -101;

sbyte bit2 = 102;

* short: хранит целое число от -32768 до 32767 и занимает 2 байта. Представлен системным типом System.Int16

short n1 = 1;

short n2 = 102;

* ushort: хранит целое число от 0 до 65535 и занимает 2 байта. Представлен системным типом System.UInt16

ushort n1 = 1;

ushort n2 = 102;

* int: хранит целое число от -2147483648 до 2147483647 и занимает 4 байта. Представлен системным типом System.Int32. Все целочисленные литералы по умолчанию представляют значения типа int:

int a = 10;

int b = 0b101; // бинарная форма b =5

int c = 0xFF; // шестнадцатеричная форма c = 255

* uint: хранит целое число от 0 до 4294967295 и занимает 4 байта. Представлен системным типом System.UInt32

uint a = 10;

uint b = 0b101;

uint c = 0xFF;

* long: хранит целое число от –9 223 372 036 854 775 808 до 9 223 372 036 854 775 807 и занимает 8 байт. Представлен системным типом System.Int64

long a = -10;

long b = 0b101;

long c = 0xFF;

* ulong: хранит целое число от 0 до 18 446 744 073 709 551 615 и занимает 8 байт. Представлен системным типом System.UInt64

ulong a = 10;

ulong b = 0b101;

ulong c = 0xFF;

* float: хранит число с плавающей точкой от -3.4\*1038 до 3.4\*1038 и занимает 4 байта. Представлен системным типом System.Single
* double: хранит число с плавающей точкой от ±5.0\*10-324 до ±1.7\*10308 и занимает 8 байта. Представлен системным типом System.Double
* decimal: хранит десятичное дробное число. Если употребляется без десятичной запятой, имеет значение от ±1.0\*10-28 до ±7.9228\*1028, может хранить 28 знаков после запятой и занимает 16 байт. Представлен системным типом System.Decimal
* char: хранит одиночный символ в кодировке Unicode и занимает 2 байта. Представлен системным типом System.Char. Этому типу соответствуют символьные литералы:

char a = 'A';

char b = '\x5A';

char c = '\u0420';

* string: хранит набор символов Unicode. Представлен системным типом System.String. Этому типу соответствуют строковые литералы.

string hello = "Hello";

string word = "world";

* object: может хранить значение любого типа данных и занимает 4 байта на 32-разрядной платформе и 8 байт на 64-разрядной платформе. Представлен системным типом System.Object, который является базовым для всех других типов и классов .NET.

object a = 22;

object b = 3.14;

object c = "hello code";

***Использование суффиксов***

При присвоении значений надо иметь в виду следующую тонкость: все вещественные литералы (дробные числа) рассматриваются как значения типа double. И чтобы указать, что дробное число представляет тип float или тип decimal, необходимо к литералу добавлять суффикс: F/f - для float и M/m - для decimal.

float a = 3.14F;

float b = 30.6f;

decimal c = 1005.8M;

decimal d = 334.8m;

Подобным образом все целочисленные литералы рассматриваются как значения типа int. Чтобы явным образом указать, что целочисленный литерал представляет значение типа uint, надо использовать суффикс U/u, для типа long - суффикс L/l, а для типа ulong - суффикс UL/ul:

uint a = 10U;

long b = 20L;

ulong c = 30UL;

***Использование системных типов***

Выше при перечислении всех базовых типов данных для каждого упоминался системный тип. Потому что название встроенного типа по сути представляет собой сокращенное обозначение системного типа. Например, следующие переменные будут эквивалентны по типу:

int a = 4;

System.Int32 b = 4;

***Неявная типизация***

Ранее мы явным образом указывали тип переменных, например, int x;. И компилятор при запуске уже знал, что x хранит целочисленное значение.

Однако мы можем использовать и модель неявной типизации:

var hello = "Hell to World";

var c = 20;

Для неявной типизации вместо названия типа данных используется ключевое слово var. Затем уже при компиляции компилятор сам выводит тип данных исходя из присвоенного значения. Так как по умолчанию все целочисленные значения рассматриваются как значения типа int, то поэтому в итоге переменная c будет иметь тип int. Аналогично переменной hello присваивается строка, поэтому эта переменная будет иметь тип string

Эти переменные подобны обычным, однако они имеют некоторые ограничения.

Во-первых, мы не можем сначала объявить неявно типизируемую переменную, а затем инициализировать:

// этот код работает

int a;

a = 20;

// этот код не работает

var c;

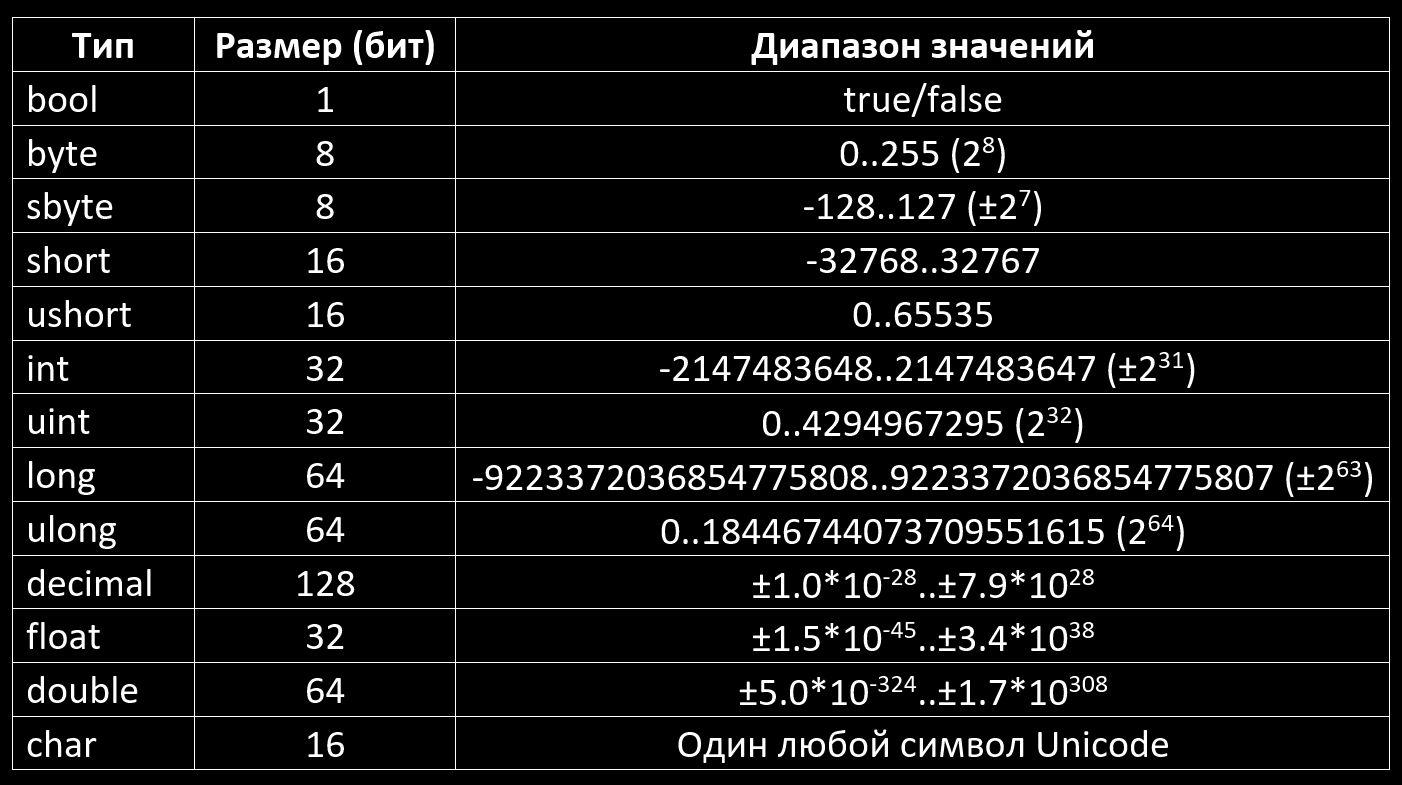
c= 20;

Во-вторых, мы не можем указать в качестве значения неявно типизируемой переменной null:

// этот код не работает

var c=null;

Так как значение null, то компилятор не сможет вывести тип данных.



Общие сведения о классах

**Ссылочные типы**

Тип, определенный как, [class](https://learn.microsoft.com/ru-ru/dotnet/csharp/language-reference/keywords/class) является *ссылочным типом*. Во время выполнения при объявлении переменной ссылочного типа переменная содержит значение [null](https://learn.microsoft.com/ru-ru/dotnet/csharp/language-reference/keywords/null) , пока вы явным образом не создадите экземпляр класса с помощью [new](https://learn.microsoft.com/ru-ru/dotnet/csharp/language-reference/operators/new-operator) оператора или не назначьте ему объект совместимого типа, который мог быть создан в другом месте, как показано в следующем примере:

//Declaring an object of type MyClass.

MyClass mc = new MyClass();

//Declaring another object of the same type, assigning it the value of the first object.

MyClass mc2 = mc;

При создании объекта выделяется достаточный объем памяти для этого объекта в управляемой куче, и переменная хранит только ссылку на расположение данного объекта. Хранение типов в управляемой куче требует дополнительных действий как при выделении памяти, так и при удалении, которое выполняется функцией автоматического управления памятью в среде CLR, известной как *сборка мусора*. Сборка мусора является хорошо оптимизированным процессом и в большинстве случаев не создает помех для производительности. Дополнительные сведения о сборке мусора см. в разделе [Автоматическое управление памятью и сборка мусора](https://learn.microsoft.com/ru-ru/dotnet/standard/garbage-collection/fundamentals).

**Объявление классов**

Классы объявляются с помощью ключевого class слова , за которым следует уникальный идентификатор, как показано в следующем примере:

//[access modifier] - [class] - [identifier]

public class Customer

{

// Fields, properties, methods and events go here...

}

Ключевому слову class предшествует уровень доступа. Поскольку [public](https://learn.microsoft.com/ru-ru/dotnet/csharp/language-reference/keywords/public) используется в этом случае, любой пользователь может создать экземпляры этого класса. За именем класса следует ключевое слово class. Имя класса должно быть допустимым [именем идентификатора](https://learn.microsoft.com/ru-ru/dotnet/csharp/fundamentals/coding-style/identifier-names) C#. Оставшаяся часть определения — это тело класса, в котором задаются данные и поведение. Поля, свойства, методы и события в классе собирательно называются *членами класса*.

**Создание объектов**

Несмотря на то, что они иногда взаимозаменяемы, класс и объект — разные вещи. Класс определяет тип объекта, но не является объектом. Объект — это конкретная сущность, основанная на классе, которую иногда называют экземпляром класса.

Объекты можно создать с помощью ключевого new слова , за которым следует имя класса, на основе которого будет основан объект, следующим образом:

Customer object1 = new Customer();

При создании экземпляра класса ссылка на объект передается программисту. В предыдущем примере object1 представляет собой ссылку на объект, который основан на Customer. Эта ссылка указывает на новый объект, но не содержит данные этого объекта. Фактически, можно создать ссылку на объект без создания собственно объекта:

Customer object2;

Мы не рекомендуем создавать ссылки на объекты, такие как предыдущие, которые не ссылаются на объект, так как попытка получить доступ к объекту через такую ссылку завершится ошибкой во время выполнения. Однако такую ссылку можно сделать указывающей на объект, создав новый объект или назначив ее существующему объекту, как показано далее:

Customer object3 = new Customer();

Customer object4 = object3;

В этом коде создаются две ссылки на объект, которые указывают на один и тот же объект. Таким образом, любые изменения объекта, выполненные посредством object3, отражаются при последующем использовании object4. Поскольку на объекты, основанные на классах, указывают ссылки, классы называют ссылочными типами.

**Наследование классов**

Классы полностью поддерживают *наследование*, фундаментальный механизм объектно ориентированного программирования. При создании класса можно наследовать от любого другого класса, который не определен как [sealed](https://learn.microsoft.com/ru-ru/dotnet/csharp/language-reference/keywords/sealed), а другие классы могут наследовать от класса и переопределять виртуальные методы класса. Кроме того, можно реализовать один или несколько интерфейсов.

При наследовании создается *производный* класс, то есть класс объявляется с помощью *базового класса*, от которого он наследует данные и поведение. Базовый класс задается добавлением после имени производного класса двоеточия и имени базового класса, как показано далее:

public class Manager : Employee

{

// Employee fields, properties, methods and events are inherited

// New Manager fields, properties, methods and events go here...

}

Когда класс объявляет базовый класс, он наследует все члены базового класса, за исключением конструкторов. Дополнительные сведения см. в разделе [Наследование](https://learn.microsoft.com/ru-ru/dotnet/csharp/fundamentals/object-oriented/inheritance).

Класс в C# может напрямую наследовать только от одного базового класса. Тем не менее, поскольку базовый класс может сам наследовать от другого класса, класс может косвенно наследовать от нескольких базовых классов. Кроме того, класс может напрямую реализовать несколько интерфейсов. Дополнительные сведения см. в разделе [Интерфейсы](https://learn.microsoft.com/ru-ru/dotnet/csharp/fundamentals/types/interfaces).

Класс можно объявить [abstract](https://learn.microsoft.com/ru-ru/dotnet/csharp/language-reference/keywords/abstract). Абстрактный класс содержит абстрактные методы, которые имеют определение сигнатуры, но не имеют реализации. Нельзя создавать экземпляры абстрактных классов. Они могут использоваться только через производные классы, реализующие абстрактные методы. И наоборот, [запечатанный](https://learn.microsoft.com/ru-ru/dotnet/csharp/language-reference/keywords/sealed) класс не позволяет другим классам быть от него производными. Дополнительные сведения см. в статье [Абстрактные и запечатанные классы и члены классов](https://learn.microsoft.com/ru-ru/dotnet/csharp/programming-guide/classes-and-structs/abstract-and-sealed-classes-and-class-members).

Определения классов можно разделить между различными исходными файлами. Дополнительные сведения см. в разделе [Разделяемые классы и методы](https://learn.microsoft.com/ru-ru/dotnet/csharp/programming-guide/classes-and-structs/partial-classes-and-methods).

**Пример**

В следующем примере определяется открытый класс, содержащий [автоматически реализуемое свойство](https://learn.microsoft.com/ru-ru/dotnet/csharp/programming-guide/classes-and-structs/auto-implemented-properties), метод и специальный метод, который называется конструктором. Дополнительные сведения см. в статьях [о свойствах](https://learn.microsoft.com/ru-ru/dotnet/csharp/programming-guide/classes-and-structs/properties), [методах](https://learn.microsoft.com/ru-ru/dotnet/csharp/programming-guide/classes-and-structs/methods) и [конструкторах](https://learn.microsoft.com/ru-ru/dotnet/csharp/programming-guide/classes-and-structs/constructors) . Затем создаются экземпляры этого класса с помощью ключевого слова new.

using System;

public class Person

{

// Constructor that takes no arguments:

public Person()

{

Name = "unknown";

}

// Constructor that takes one argument:

public Person(string name)

{

Name = name;

}

// Auto-implemented readonly property:

public string Name { get; }

// Method that overrides the base class (System.Object) implementation.

public override string ToString()

{

return Name;

}

}

class TestPerson

{

static void Main()

{

// Call the constructor that has no parameters.

var person1 = new Person();

Console.WriteLine(person1.Name);

// Call the constructor that has one parameter.

var person2 = new Person("Sarah Jones");

Console.WriteLine(person2.Name);

// Get the string representation of the person2 instance.

Console.WriteLine(person2);

}

}

// Output:

// unknown

// Sarah Jones

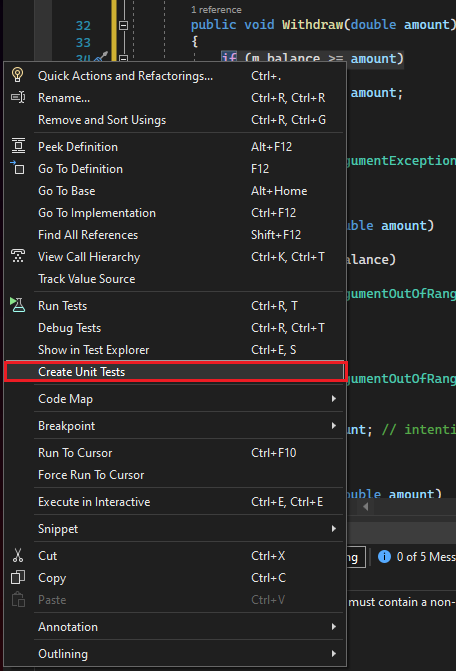
// Sarah Jones

Создание проекта модульного теста

Для C# как правило, проще создать проект модульного теста и заглушки модульных тестов из кода. Кроме того, можно создать проект модульных тестов и тесты вручную в зависимости от потребностей. Если вы хотите создавать модульные тесты из кода на сторонней платформе, вам потребуется установить одно из этих расширений: NUnit или xUnit. Если вы не используете C#, пропустите этот раздел и перейдите к разделу Создание проекта и модульных тестов вручную.

**Создание проекта модульного теста и заглушек модульных тестов**

1. В окне редактора кода выберите в контекстном меню команду Создать модульные тесты.

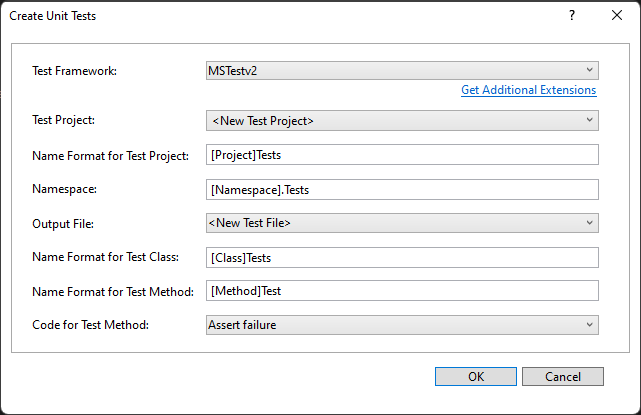


В окне редактора просмотрите контекстное меню

*Примечание*

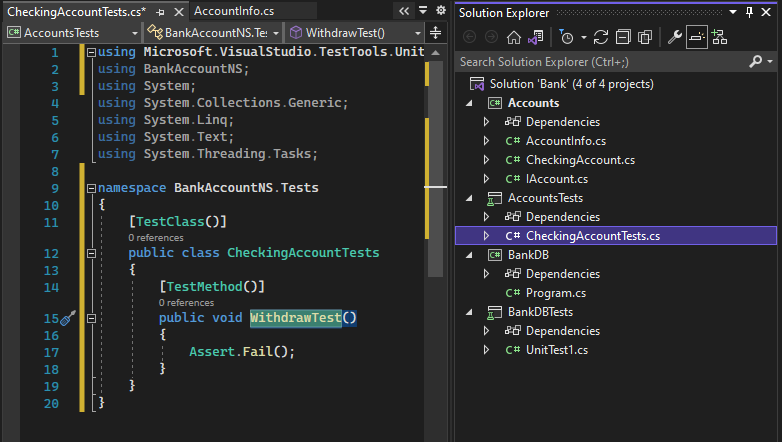
*Команда меню Создать модульные тесты доступна только для кода C#. Чтобы использовать этот метод для .NET Core или .NET Standard, требуется Visual Studio 2019 или более поздней версии.*

1. Нажмите кнопку ОК, чтобы принять значения по умолчанию для создания модульных тестов, или измените значения, которые использовались для создания и назначения имени проекта модульного теста и модульных тестов. Можно выбрать код, который добавляется по умолчанию в методы модульных тестов.



Диалоговое окно

1. Заглушки модульных тестов создаются в новом проекте модульного теста для всех методов в классе.



Модульные тесты созданы

Литература

[**https://learn.microsoft.com/ru-ru/windows/apps/desktop/visual-studio-templates**](https://learn.microsoft.com/ru-ru/windows/apps/desktop/visual-studio-templates)

[**https://learn.microsoft.com/ru-ru/dotnet/csharp/whats-new/csharp-version-history**](https://learn.microsoft.com/ru-ru/dotnet/csharp/whats-new/csharp-version-history)

[**https://learn.microsoft.com/ru-ru/dotnet/csharp/tour-of-csharp/**](https://learn.microsoft.com/ru-ru/dotnet/csharp/tour-of-csharp/)

[**http://bourabai.ru/alg/c-sharp.htm#:~:text=C%23%20%D0%BE%D1%82%D0%BD%D0%BE%D1%81%D0%B8%D1%82%D1%81%D1%8F%20%D0%BA%20%D1%81%D0%B5%D0%BC%D1%8C%D0%B5%20%D1%8F%D0%B7%D1%8B%D0%BA%D0%BE%D0%B2,%D0%B8%D1%81%D0%BA%D0%BB%D1%8E%D1%87%D0%B5%D0%BD%D0%B8%D1%8F%2C%20%D0%BA%D0%BE%D0%BC%D0%BC%D0%B5%D0%BD%D1%82%D0%B0%D1%80%D0%B8%D0%B8%20%D0%B2%20%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%82%D0%B5%20XML**](http://bourabai.ru/alg/c-sharp.htm#:~:text=C%23%20%D0%BE%D1%82%D0%BD%D0%BE%D1%81%D0%B8%D1%82%D1%81%D1%8F%20%D0%BA%20%D1%81%D0%B5%D0%BC%D1%8C%D0%B5%20%D1%8F%D0%B7%D1%8B%D0%BA%D0%BE%D0%B2,%D0%B8%D1%81%D0%BA%D0%BB%D1%8E%D1%87%D0%B5%D0%BD%D0%B8%D1%8F%2C%20%D0%BA%D0%BE%D0%BC%D0%BC%D0%B5%D0%BD%D)**.**

[**https://learn.microsoft.com/ru-ru/visualstudio/test/walkthrough-creating-and-running-unit-tests-for-managed-code?view=vs-2022**](https://learn.microsoft.com/ru-ru/visualstudio/test/walkthrough-creating-and-running-unit-tests-for-managed-code?view=vs-2022)

[**https://habr.com/ru/post/350708/**](https://habr.com/ru/post/350708/)

[**https://metanit.com/sharp/windowsforms/3.1.php**](https://metanit.com/sharp/windowsforms/3.1.php)

[**https://habr.com/ru/post/161205/**](https://habr.com/ru/post/161205/)

[**https://tyapk.ru/blog/post/programming-languages-typing**](https://tyapk.ru/blog/post/programming-languages-typing)

[**https://metanit.com/sharp/tutorial/2.1.php**](https://metanit.com/sharp/tutorial/2.1.php)

**https://gb.ru/posts/c\_sharp\_ides**