**Классовая и иерархическая структура**

**О проекте**

В среде программирования **MS Visual Studio** и конкретно в C# вся информация (исходный код, ресурсы и т.п.) организуется в виде классовой и иерархической структуры.

**Классовость** подразумевает оформление всей информации в виде независимых, автономных блоков, которые называются классами и имеют некоторые характеристики. В частности, имя (программное имя) и файл.

**Иерархическая структура** подразумевает объединение этих классов, наличие отношений принадлежности между ними.

На вершине этой структуры находится **Решение** (Solution), которое соответствует решению поставленной задачи, как единого целого. Примером Solution может служить задача разработки нового текстового редактора или задача разработки службы для мониторинга сети.

Решение может состоять из одного или нескольких **Проектов** (Project). Если решение - это задача, то можно сказать, что проект соотвествует подзадаче. Для примера текстового редактора можно было бы создать два проекта (подзадачи): сам текстовый редактор с функционалом - ввод и редактирование текста, форматирование, операции с файлами и т.п. и второй проект - компонент, реализующий сервисные операции (проверка орфографии, грамматики, расстановка переносов и т.п.).

Каждый проект может состоять из одного или нескольких **Классов** (Class). Если очень крупно, то классы можно разделить на программные и другие. Программные классы являются контейнерами кода (объявлений или\и операторов). К другим типам классов можно отнести классы с ресурсами (текстовый файл, пиктограмма), модули, выполняющие служебные функции (модуль подключения к источнику данных).

Каждый класс может состоять из области деклараций и области кода. В области кода может быть одна или более **подпрограмм** (subroutine).

Подпрограмма является практической реализацией определенного алгоритма.

Каждый из перечисленных выше элементов (кроме подпрограмм) сохраняется в отдельном файле определенного типа.

Каждый из перечисленных выше элементов имеет свое имя, тип и настройки.

|  |  |
| --- | --- |
| Тип | Расширение файла |
| Solution | \*.SLN |
| Project | \*.CSPROJ ... |
| Class | \*.CS ... |

**Основы синтаксиса**

***О синтаксисе***

Для успешной реализации примеров этого урока Вам необходимо знать следующие синтаксические правила и конструкции.

Техника ввода кода

1. Все операторы в программе на C# записываются на отдельных строках.
2. Окончанием оператора (строки) служит точка с запятой.
3. Есть возможность разместить несколько операторов на одной строке, так же разделяя их точкой с запятой. Пример:  
   X = 10 ; Y = 25 ;

**Объявление переменных**

Существует несколько способ объявления переменных. В примерах ниже мы будем использовать следующий вариант:

type name ;

Здесь **name** - имя (идентификатор) переменной. **type** - тип переменной. Например, **short** - целочисленная знаковая переменная, размером 2 байта.

**Оператор присвоения (записи)**

Для записи данных в переменную используется оператор присвоения. В C# это знак равенства. Такая запись

shrX = 100;

означает, что в переменную shrX записывается значение 100. Т.е. значение из правой части записывается в переменную в левой части.

**Запись строк и операция конкатенации**

Строковые константные значения (тип string) записываются в двойных кавычках. Например, "Hello, world!”

Для объединения строк используется операция конкатенации +. Приведенный пример записывает приветствие в переменную strGreet

strName = "John”;

strGreet = "Hello, " + strName;

**Операторы ввода и вывода**

Для организации ввода и вывода в консольном приложении используется класс Console . Применяемые в уроке методы этого класса:

**Readline** - ввод значения. Завершение ввода - Enter.

**Writeline** - вывод значения с переводом каретки.

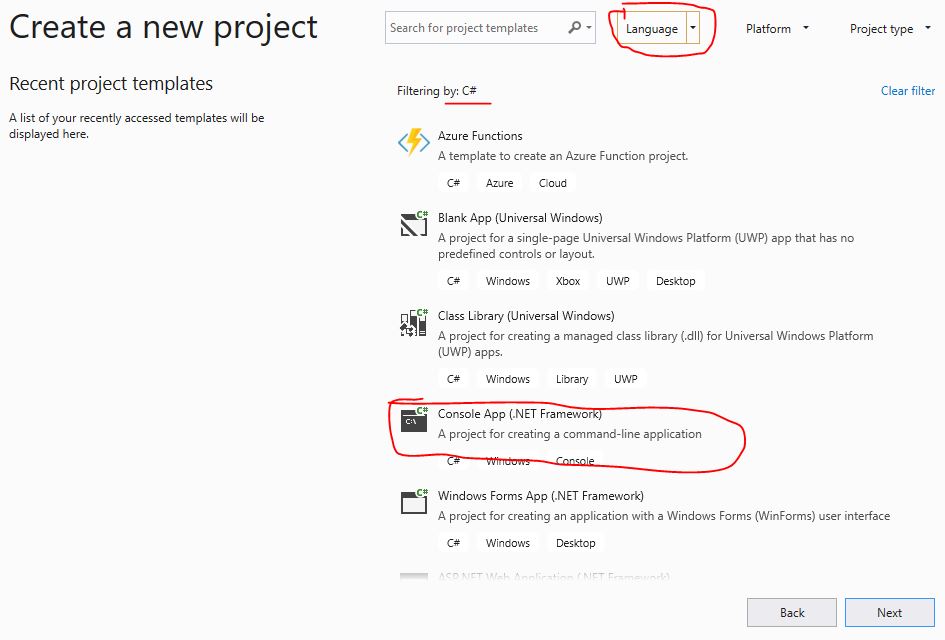
**ReadKey** - используется для организации паузы в программе до нажатия любой клавиши.

Для перехода к следующей части лекции нужно ответить на контрольный вопрос - Выберите метод, который НЕ может быть использован справа от оператора присвоения.

### Запуск IDE

Запустите среду программирования MS Visual Studio (иллюстрации ниже соответствуют версии 2019). В первом диалоговом окне выберите вариант Create a new project для создания нового проекта.

В следующем диалоговом окне необходимо выбрать тип шаблона для будущего проекта. Установите (для упрощения) фильтр по языку программирования - C#. Прокрутите список шаблонов и выберите Console App (.Net Framework).

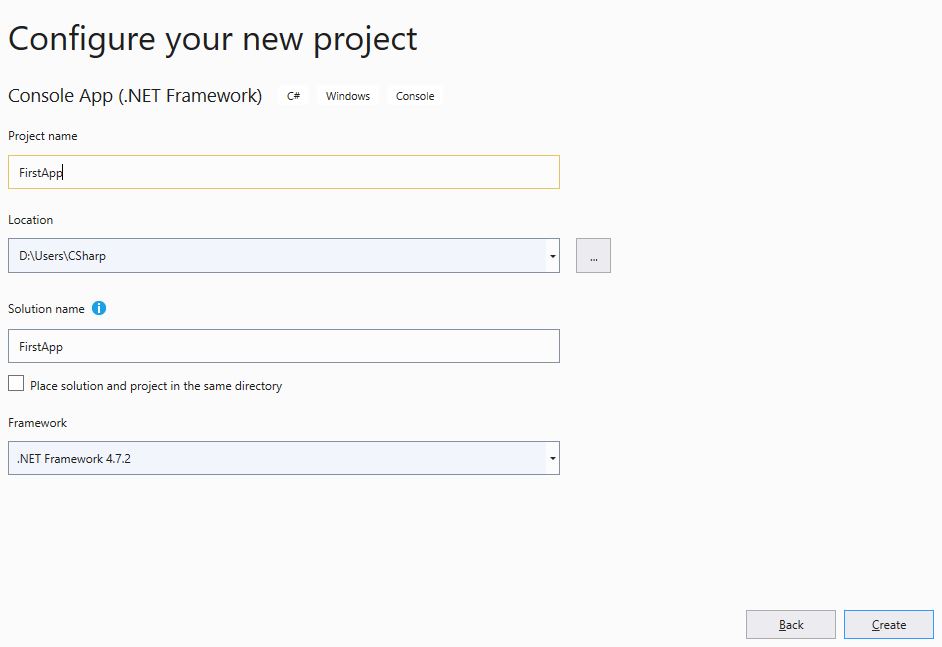


Выбранный шаблон влияет в частности на формат конечного результата, откомпилированного файла.

Нажмите кнопку Next.

В поле Project Name введите имя проекта: FirstApp. Это же имя будет именем Solution.

Укажите правильное местоположение проекта на диске. Лучший вариант для занятий на практиках в классе - D:\Users\username.



Нажмите кнопку **Create**.

Будет создано новое решение с одним проектом и одним классом.

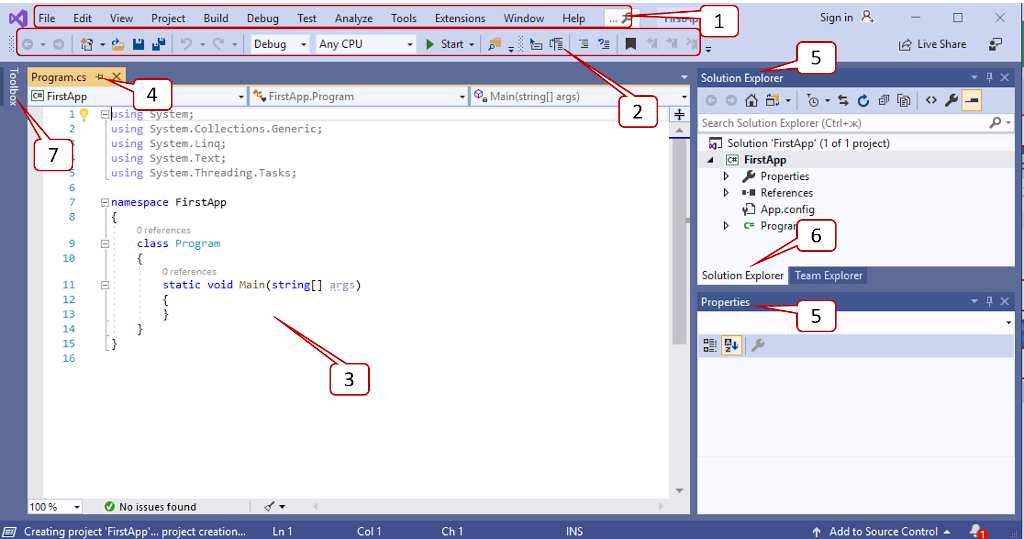
### Знакомство с интерфейсом и его настройка

Для начала познакомимся с интерфейсом среды программирования.

Интегрированная среда разработки (**IDE**) VS содержит меню (**menu**), панели инструментов (**toolbars**), рабочие окна или окна документов (**windows**) и служебные окна (**tool windows**).

1. Строка **Меню**.

2. **Панели инструментов** предоставляют возможность выбирать нужные кнопки для выполнения определенных операций. Стандартно мы будем использовать две панели инструментов **Standard** , **Debug** и **Text Editor**. Однако, общее количество панелей намного больше трех. Включать или выключать панели можно с помощью **View \ Toolbars**.



3. **Рабочее окно документа**. В нашем случае, стандартного класса Program.cs, в котором мы и будем писать код. В общем случае, окон может быть больше одного. Переключение между ними можно выполнять с помощью вкладок 4 или **CTRL+TAB**.

4. **Вкладки** открытых окон.

5. **Служебные окна** предназначены для различных целей. Располагаться они могут с любой стороны от рабочих окон. Их можно перетаскивать мышью за заголовок, делать закрепленными (dockable) или плавающими (floating). Если нужное служебное окно закрыто, то открыть его можно с помощью меню View (в этом меню перечислены практически все окна, например, Solution Explorer, Output и т.п.).

6. Служебные окна объединяются в **группы**, переключение между ними выполняется с помощью вкладок 6.

7. Служебные окна могут быть в **скрытом состоянии**. В этом случае, видны только вкладки этих окон. Для их отображения достаточно нацелится на вкладку.

Нам в первую очередь понадобятся следующие служебные окна:

1. **Solution Explorer** - позволяет просмотреть структуру проекта, выполнять операции с классами, с подключаемыми библиотеками. Доступные операции: добавление, удаление, переименование класса.

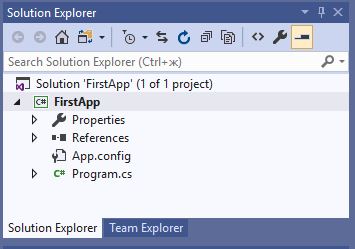
2. **Properties** - окно свойств. Позволяет просматривать и редактировать свойства выделенного объекта. Например, класса.

3. **Output** - окно вывода служебной информации. Позволяет следить за сообщениями компилятора. А также читать свои собственные служебные сообщения, выдаваемые из кода.

4. **Error List** - окно с информацией об ошибках в коде или предупреждениях компилятора.

### Структура проекта и класса

Наше решение и его составляющие отображаются в окне **Solution Explorer** (SE).



На вершине иерархии сейчас находится решение FirstApp за ним находится проект FirstApp.

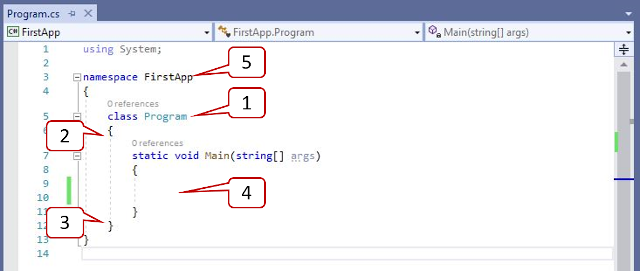
Program.cs - модуль класса, класс кода, в котором мы и будем вводить операторы и декларации нашей первой программы.

Не путайте класс и файл, в котором он сохраняется. В одном файле может быть несколько классов. В настоящее время, файл называется Program.cs и внутри имеет только один класс, с именем Program. Но мы можем организовать сколько угодно классов в одном файле. В общем случае, имя файла и имя класса могут не совпадать.

Псевдопапка References позволяет посмотреть подключенные на текущий момент библиотеки, на которые могут быть ссылки в тексте программы.

Первое о чем нужно позаботится - о сохранении данных на диске. Имя и расположение папки мы задали при создании проекта, поэтому в процессе работы можно пользоваться командой  Save All (кнопка на панели инструментов) или выполните через меню: File \ Save All...

Класс Program имеет следующую структуру:



1. Название и начало класса.

2. Открывающая скобка класса. Между открывающейся скобкой и первой подпрограммой область (часть кода) часто называют область деклараций. Сюда можно поместить различные объявления (декларации), например переменных или констант.

3. Закрывающая скобка класса.

4. Подпрограмма Main и ее область кода. Именно она получает управление при запуске программы.

5. Название и начало пространства имен.

Давайте переименуем класс. Удалите в тексте Program и напишите StartClass.

Теперь переименуем файл, в котором этот класс сохраняется. Для этого вызовите контекстное меню для модуля в окне SE и щелкните по команде Rename. Введите имя файла StartClass.cs.

#### Задание

В папке вашего проекта, на диске найдите следующие файлы:

· MyFirstApp.sln - файл решения

· MyFirstApp.csproj - файл проекта

· FirstClass.cs - файл класса.

Помните, что все операции с файлами необходимо выполнять в IDE, в окне SE. Например, такие операции как: переименование файлов, удаление, добавление новых...

### Написание программы. Первый вариант.

## *Первый вариант*

Метод **Main** первая получает управление в консольном проекте, именно в ней мы и будем вводить код. Пусть наша программа осуществляет ввод двух числовых значений от пользователя и обменивает эти значения между собой.

Введите следующий код в методе Main, при этом обратите внимание на следующее:

1. Регистр символов - при объявлении имена переменных необходимо вводить в том регистре, который Вам необходим. При вводе имен операторов регистр не важен, его автоматически исправляет редактор.

2. Раскраска синтаксических форм.

3. Автоматические отступы.

4. Выпадающие списки при вводе типа данных или члена класса (после точки). С раскрывающими списками необходимо работать так: ввести первые символы (иногда достаточно двух-трех символов) нужного Вам слова и выбрать необходимый элемент из списка пробелом или табуляцией (не Enter). Для удобства, можно подгонять маркер в списке нажимая на стрелки вверх-вниз на клавиатуре.

5. Комментарии - они должны помогать воспринимать программу.

6. При щелчке - автоматическое выделение синтаксических конструкций (например, при щелчке по оператору IF) и всех вхождений идентификатора (например, при щелчке по названию переменной).



Как Вы поняли, объект **Console** позволяет работать с консолью: осуществлять ввод-вывод информации, а также выполнять еще ряд полезных операций.

Используемые методы в данном примере:

· **WriteLine** - вывод сообщения с автоматическим переводом каретки

· **ReadLine** - чтение строки из консоли (до нажатия Enter )

· **ReadKey** - осуществляет чтение символа, введенного пользователем с отображением его на экране.

### Запуск программы

В процессе редактирования, набора кода, программа не работает, не запущена. Такое состояние называется **Edit mode**.

Для запуска программы нажмите **F5** или щелкните по кнопке Start Debugging или выполните меню **Debug \ Start Debugging**. Вы оказались в режиме **Run mode**.

Появится окно консоли, в котором необходимо ввести два числа и проверить результат.

После завершения работы процедуры (в конце нужно нажать любую клавишу) окно автоматически закроется.

Если работу программы необходимо прервать досрочно (например, при ее зацикливании), то это можно сделать, нажав на **CTRL+BREAK** или просто закрыв окно, или нажав на кнопку **Stop Debugging** в IDE.



Запускать программу можно и из файловой системы. Зайдите в папку с проектом, в подкаталог **obj\Debug**. Здесь Вы увидите исполняемый файл **MyFirstApp.exe**, который можно запустить на исполнение.

Другой вариант, работа через консоль, командную строку:

1. Выполните **ПУСК \ Выполнить**. Введите cmd и нажмите ОК.

2. Перейдите на диск и в Вашу папку с файлом EXE.   
d:\

cd users\yourname\MyFirstApp\MyFirstApp\Obj\Debug

3. Далее, запускаем программу:   
MyFirstApp.exe

Этот вариант отличается тем, что после завершения работы программы окно командной строки не закрывается, и Вы можете наблюдать результат, выведенный на экран и без паузы, организуемой программно.

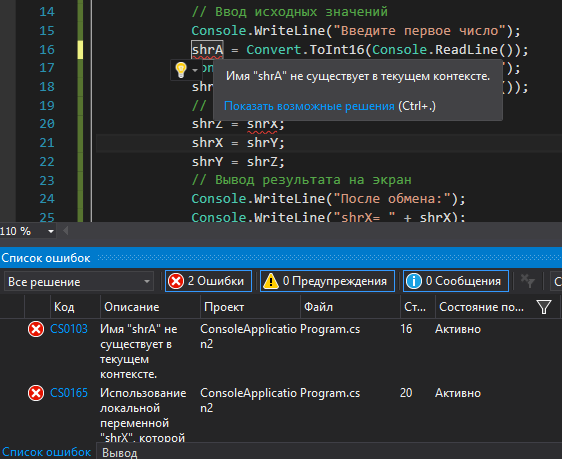
### Обработка синтаксических ошибок

## *Обработка синтаксических ошибок*

Вы можете столкнуться с синтаксическими ошибками, логическими ошибками и ошибками времени выполнения.

Давайте искусственно сделаем ошибку в теле программы.

Например, вместо имени переменной shrX в строке ввода первого числа укажите shrA.



Обратите внимание, что редактор проверяет синтаксис «на лету» и определенные виды ошибок Вы обнаружите сразу, еще до запуска процесса компиляции. Вы можете:

1. наблюдать фразу с ошибкой по синему волнистому подчеркиванию;

2. получить подсказку об ошибке при наведении курсора мыши (в данном случае **Имя "shrA" не существует в текущем контексте**);

3. получить описание и местоположение ошибки в специальном окне **Error List** (отобразить его можно через меню View).

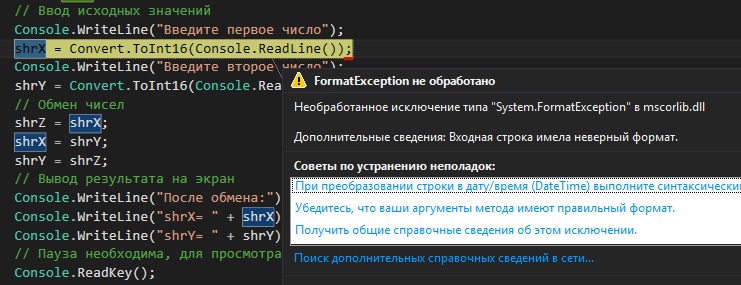
Верните верное имя переменной в текст программы.

### Обработка run-time ошибок

## *Обработка run - time ошибок*

Run - time ошибки или исключительные ситуации, или exception возникают в процессе работы программы.

В нашей программе Вы можете поэкспериментировать с ошибками подобного типа, если попробовать ввести вместо числа текст, который нельзя конвертировать в значение типа Short. В примере, проиллюстрированном ниже, была попытка ввести "X”.



Обратите внимание, на выделение строки, где ошибка произошла, сообщение об ошибке.