Справочный материал

Про byte и sbyte здесь.

В C# есть два типа данных, чтобы хранить целое число в одном байте (8 бит). Отличие между этими типами данных - **ЗНАК**.

1. byte (Беззнаковый, Положительный)

- **Что это такое?**
  8-битное **беззнаковое** целое число. Ключевое слово - "беззнаковое".
- **Диапазон: 0 до 255**
  Всего 256 возможных значений.
- **Почему так?**
  У него нет бита, отвечающего за знак "минус". Все 8 бит работают на то, чтобы показать величину числа. Данный тип данных не знает про отрицательные числа. Его одометр может идти только вперед, от 0 до 255.
- **Где может использоваться? (99% случаев)**
  - **ЦВЕТА (R, G, B, A):** Наш главный пример. Яркость цветового канала не может быть отрицательной. Она либо есть (0-255), либо ее нет. Поэтому `Pixel { public byte R, G, B; }` — это *единственно верный путь*.
  - **РАБОТА С ФАЙЛАМИ/СЕТЬЮ:** Когда мы читаем файл или получаем данные из сети это будет поток **сырых байтов**. Это просто данные, у них нет знака.
  - Любые данные, где отрицательное значение не имеет смысла: количество патронов, процент здоровья, номер символа в кодировке ASCII.

Идеален, когда нам нужно число от 0 до 255. Сомнения? Берем `byte`.

---

2. sbyte (Знаковый)

- **Что это такое?**
  8-битное **знаковое** целое число. s~~(как доллар)~~ в начале — это signed, "знаковый".
- **Диапазон: -128 до 127**
  Всего те же 256 возможных значений, но диапазон "сдвинут" в отрицательную область.
- **Почему так?**
  Этот тип данных тратит один из своих восьми бит (старший бит) на то, чтобы хранить запись, положительный он или отрицательный. Из-за этого на саму величину числа у него остается только 7 бит.
  (А то, что он уходит до -128, а не -127 — это "магия" дополнительного кода).
- **Где используется?**
  - **Изменения/Дельты:** "изменение температуры на -5 градусов", "смещение координат джойстика на +10 по оси X".
  - **Игровые параметры:** "бонус к силе -2", "модификатор урона +5".
  - Взаимодействие с другими языками (Java, например), где byte по умолчанию знаковый.
  Узкоспециализированный инструмент.

Примерный код "Черного ящика" (чтобы вы понимали, с чем работаете)

Вы этот код не пишете, вы его используете. Готовые файлы находятся в курсе ЭИОС.

```
// Структура для хранения цвета
public struct Pixel
{
    public byte R, G, B;
}

// Класс для работы с изображением
public class Image
{
```

```
    private Pixel[,] _pixels;
    public int Width
    {
        get
        {
            return _pixels.GetLength(0);
        }
    }
    public int Height
    {
        get
        {
            return _pixels.GetLength(1);
        }
    }

    // Конструктор и Save() будут сложными, с работой с файлами.
    // Это ваша работа - предоставить им рабочую реализацию.
    public Image(string filePath) { /* ... магия загрузки BMP ... */ }
    public void Save(string filePath) { /* ... магия сохранения BMP ... */ }

    // Индексатор для удобного доступа
    public Pixel this[int x, int y]
    {
        get { return _pixels[x, y]; }
        set { _pixels[x, y] = value; }
    }
}
```

Проход по всем пикселям (базовый шаблон)

```
public static void ApplyFilter(Image image) // Помейте название метода на название вашего фильтра для картинки
{
```

```
    for (int y = 0; y < image.Height; y++) // Цикл по
высоте картинки (аналог строк в двумерном массиве)
    {
        for (int x = 0; x < image.Width; x++) // Цикл по
ширине картики (аналог столбцов в двумерном массиве)
        {
            Pixel currentPixel = image[x, y]; // Получаем
конкретный пиксель x и y

            // ... тут ваша логика изменения R, G, B ...

            byte newR = /* ... */;
            byte newG = /* ... */;
            byte newB = /* ... */;

            image[x, y] = new Pixel { R = newR, G = newG,
B = newB }; // Записываем значение нового цвета в пиксель
        }
    }
}
```

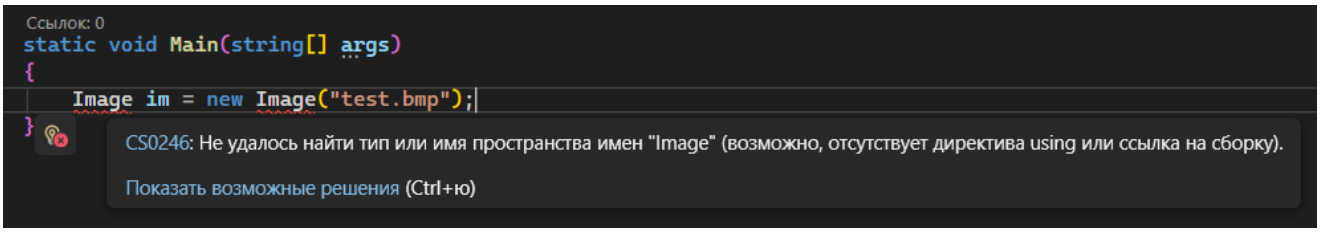
Ограничение значений (Clamping)

Цветовая компонента не может быть меньше 0 или больше 255. Если ваши вычисления выходят за эти рамки, их нужно "обрезать".

```
int newR = currentPixel.R + 50;
if (newR > 255) newR = 255;
if (newR < 0) newR = 0;
image[x, y].R = (byte)newR;
```

Подключение пространства имен

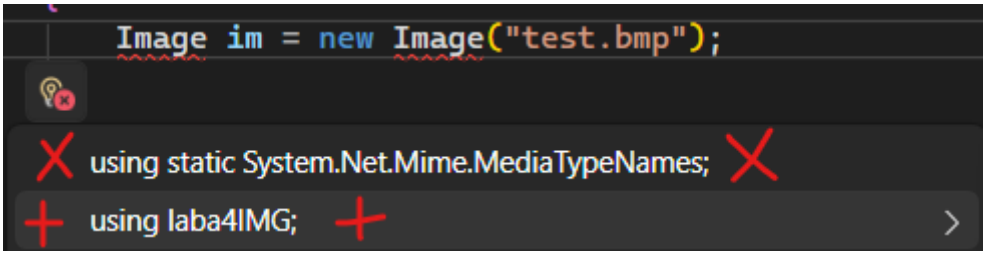
При создании объекта класса `Image` вы можете столкнуться с подобной ошибкой:



laba4_using_sample.png

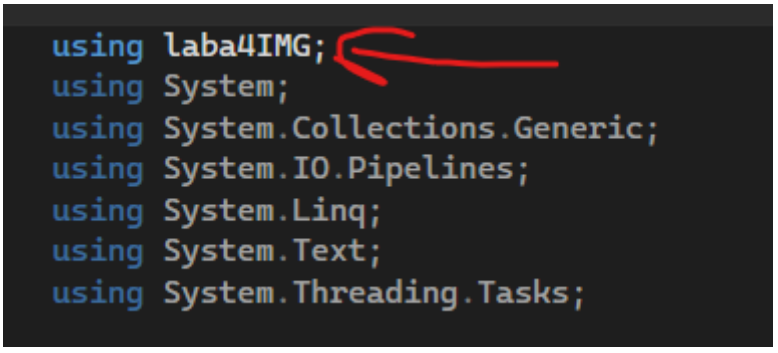
Наша программа не может найти созданный нами тип данных `Image`, нам нужно явно указать пространство имен в котором находится этот класс.

При нажатию на "лампочку с крестиком" вы увидите 2 вариант:



laba4_important_choice.png

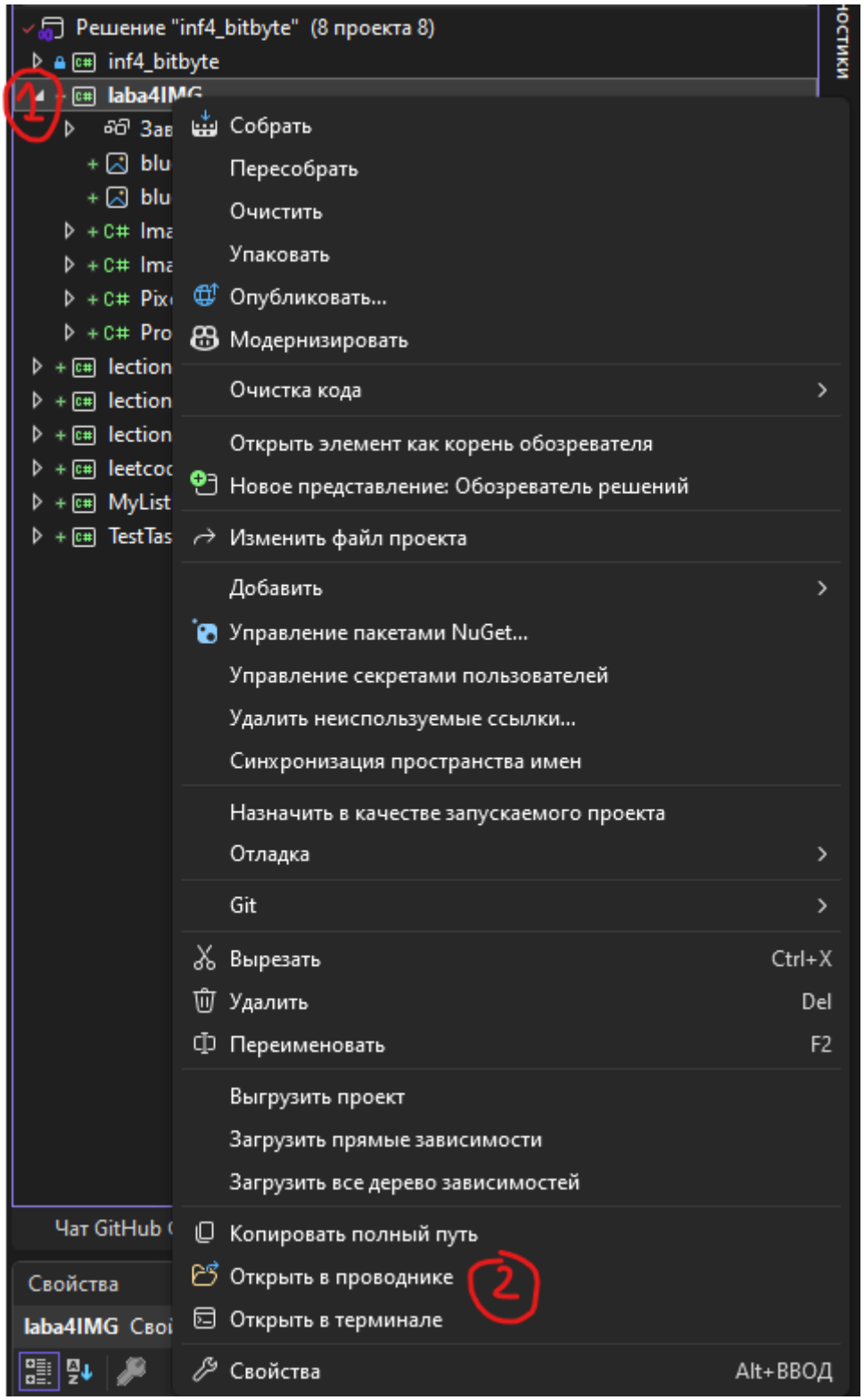
Нужно выбрать вариант с названием вашего проекта. После этого в файле `Program.cs` в самом верху появится строка:



laba4_using_connected.png

Как открыть папку с программой

Для открытия папки с проектом нажмите ПКМ по **ПРОЕКТУ** (зелененькая иконка) => Открыть в проводнике:



laba4_openFolderProj.png

Далее по примерно такому пути:
`laba4IMG\bin\Debug\net9.0`
будут находиться выходные файлы (exe для запуска).

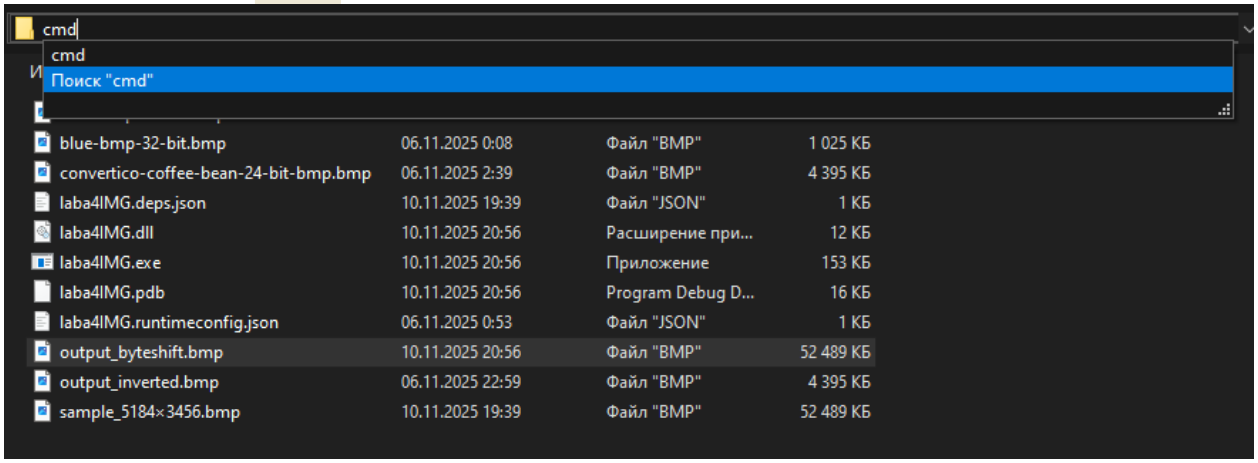
Именно в этой папке должны находиться картинки, именно сюда они будут сохраняться.

Считывание из аргументов командной строки (cmd)

В папке `bin` вашего проекта вы увидите тот самый заветный `file.exe` файл, который и является вашей программой. Вы можете кликнуть по нему 2 раза и он стандартно запустится. Если у вас нет никакой обработки после выполнения программы - программа выполнится и закроется после выполнения.

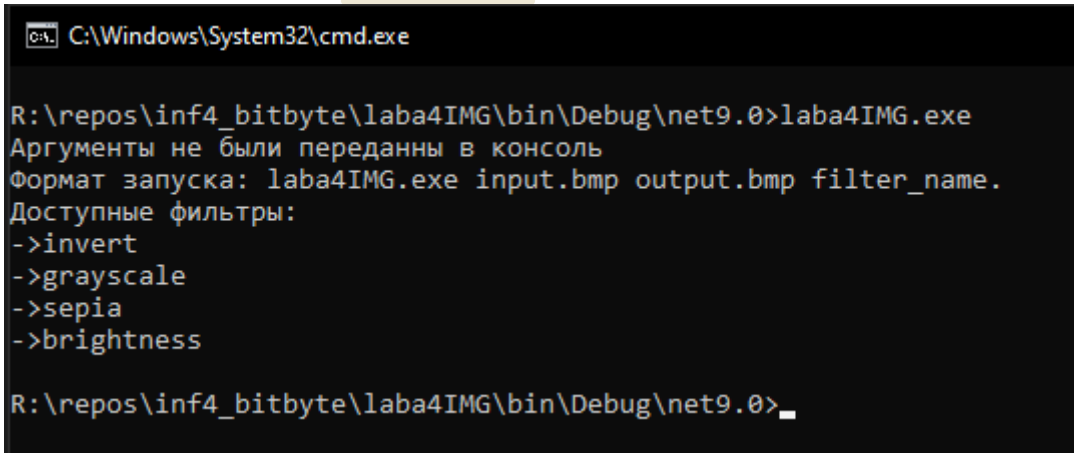
Чтобы увидеть результат мы можем запустить программу из консоли:

- Если ввести `cmd` в проводнике - откроется консоль.



laba4_writecmd_in_explorer.png

- Вводим название `file.exe`



laba4_write_exe_on_cmd.png

- Программа закрылась, но мы видим результат.

Мы можем передать дополнительные аргументы в нашу программу. В коде за это отвечает аргумент `string[] args` в методе `Main`. Данный аргумент является массивом строк, автоматически при запуске программы он преобразует введеные

аргументы через пробел `proga.exe filename.jpg outputfile.jpg invert` в массив. Соответственно:

- `filename.jpg` будет записан в `args[0]`
- `outptfile.jpg` будет записан в `args[1]`
- `invert` будет записан в `args[2]`

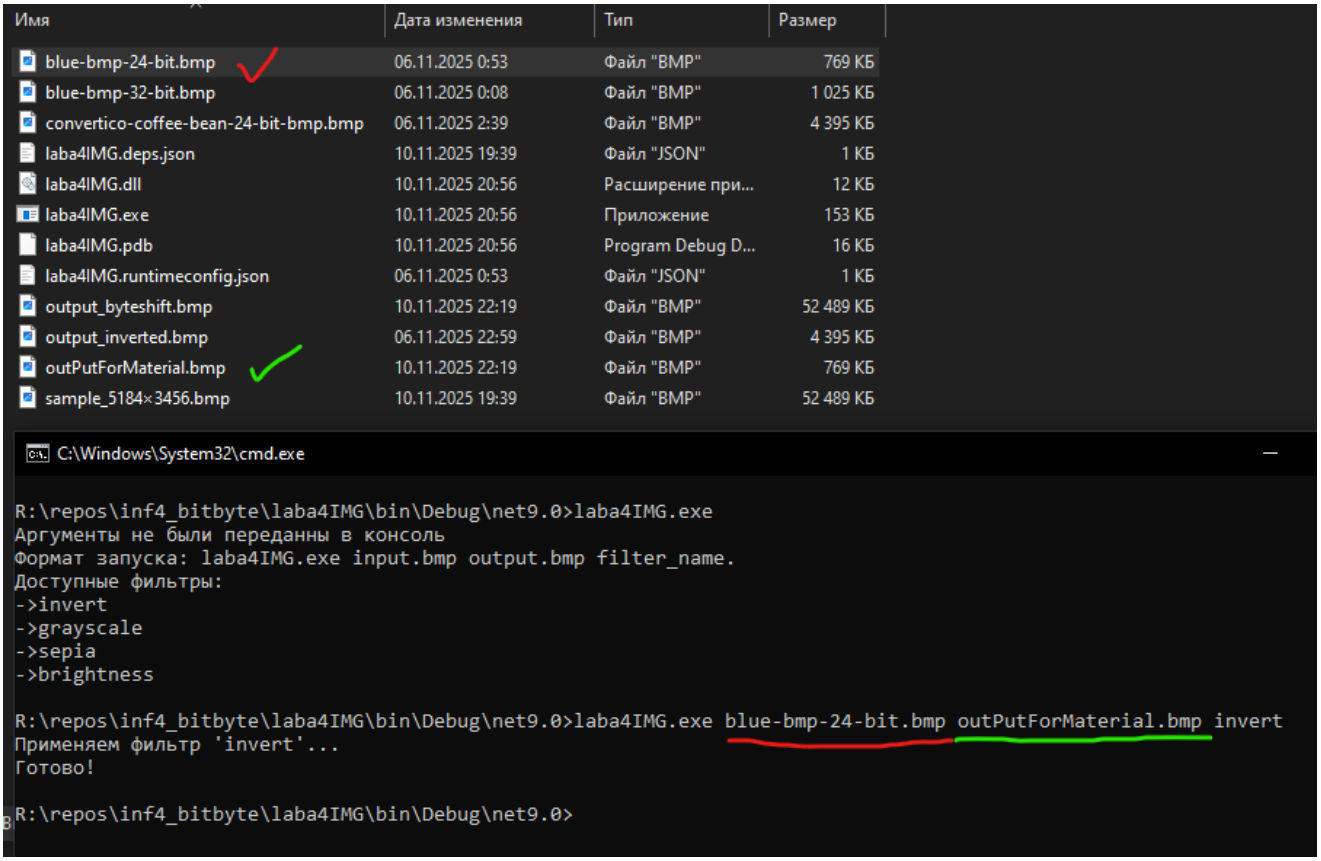
Таким образом мы получаем еще один способ ввода данных в нашу программу на этапе выполения.

Важно: все эти аргументы приходят в виде строки. Если вы хотите ввести и использовать число - его нужно сконвертировать в число.

```
static void Main(string[] args)
{
    // Блок кода срабатывает, если мы передали 0 аргументов.
    // Пример такого вызова программы выше на скрине.
    if (args.Length == 0)
    {
        Console.WriteLine("Аргументы не были переданны в консоль");
        Console.WriteLine("Формат запуска: laba4IMG.exe input.bmp output.bmp filter_name.");
        Console.WriteLine("Доступные фильтры:");

        Console.WriteLine($"->invert");
        Console.WriteLine($"->grayscale");
        Console.WriteLine($"->sepia");
        Console.WriteLine($"->brightness");

        return;
    }

    // Если передано меньше минимально нужного количества аргументов.
    if (args.Length < 3)
    {
```

```
            Console.WriteLine("Вы передали мало аргументов");
        }

        // Парсим переданные аргументы в путь до картинки,
    куда сохранять и какой фильтр выбиран.
        string inputPath = args[0];
        string outputPath = args[1];
        string filterName = args[2].ToLower();
    }
```

Пример выполнения программы через консоль:



laba4_usage_from_cmd.png

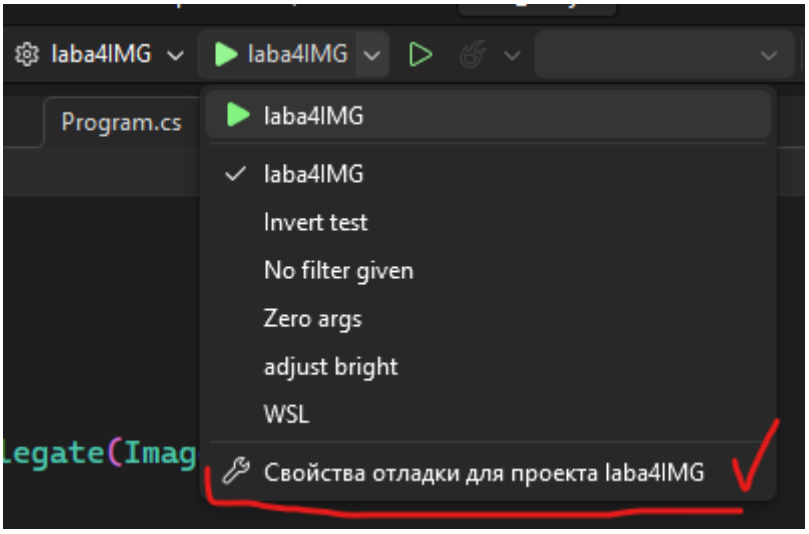
Добавление профилей отладки консоли

Может возникнуть ситуация: мы отлаживаем нашу программу и передачу аргументов из консоли. В таком случае мы не сможем отлаживать и смотреть значения переменных и где ломается программа в `Visual Studio`.

Для этого мы можем создать "профиль отладки" и запускать нашу программу через `Visual Studio` и с переданными в консоль

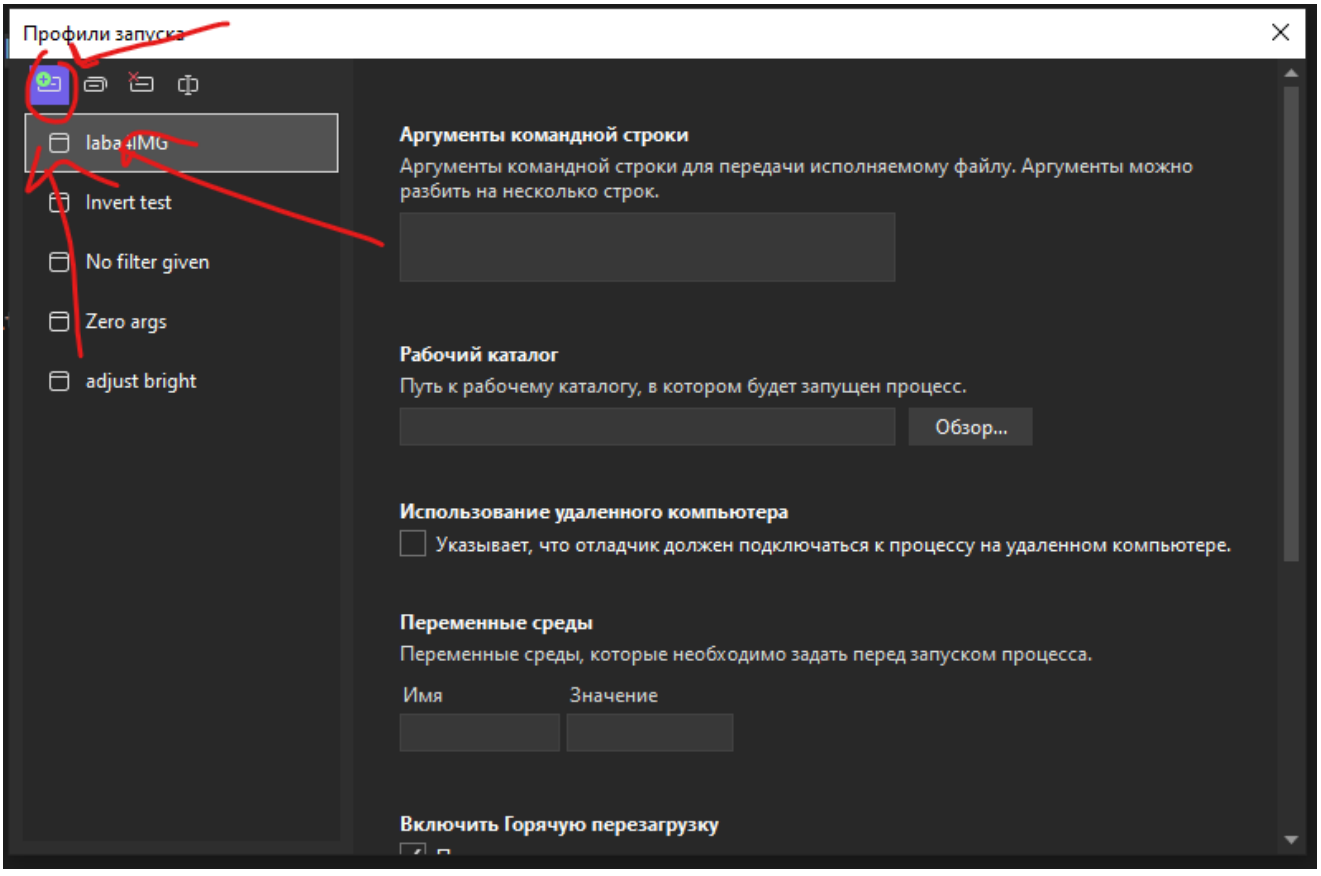
аргументами.
Для этого:

1. Нужно задать на стрелочку вниз рядом с кнопкой запуска проекта. Выбрать "свойства отладки для проекта `laba4IMG`".



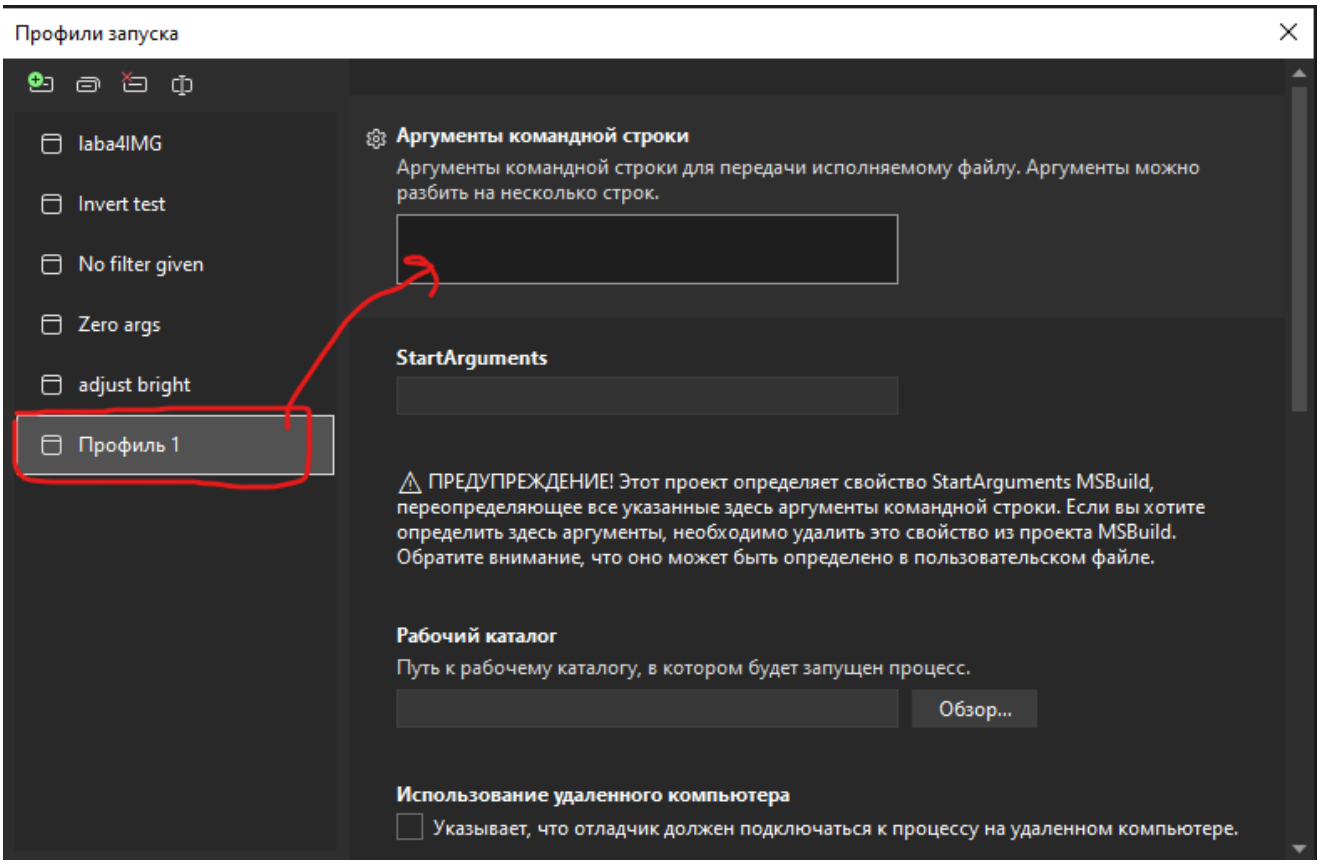
laba4_profile_create_s1.png

Теперь нужно `создать новый профиль` -> `Проект`:



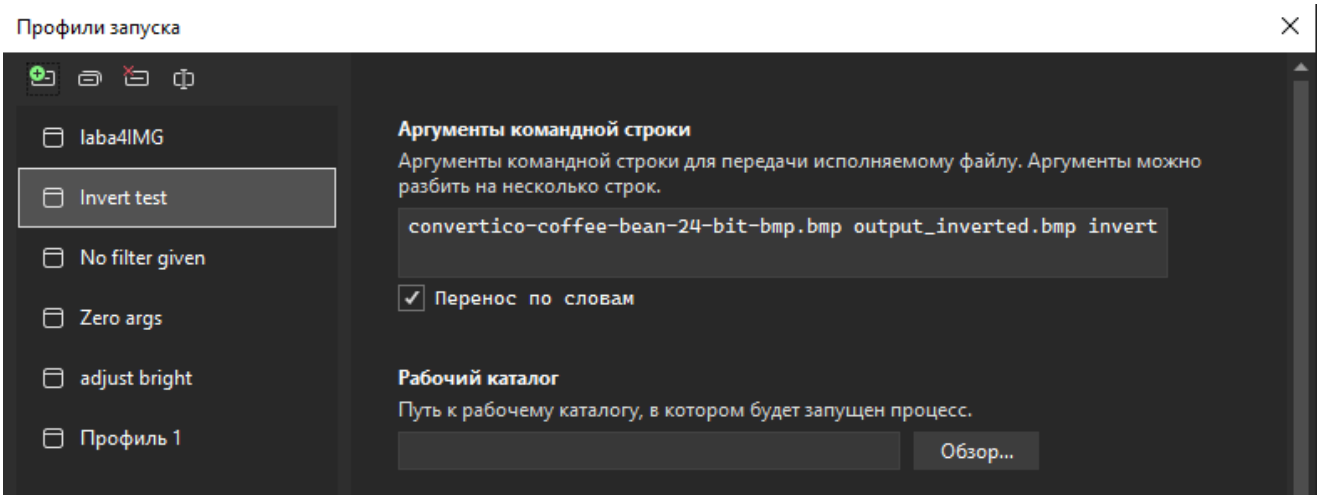
laba4_create_profile_s2.png

Теперь мы можем изменить название этого профиля и вписать аргументы, которые передадутся в консоль при запуске.



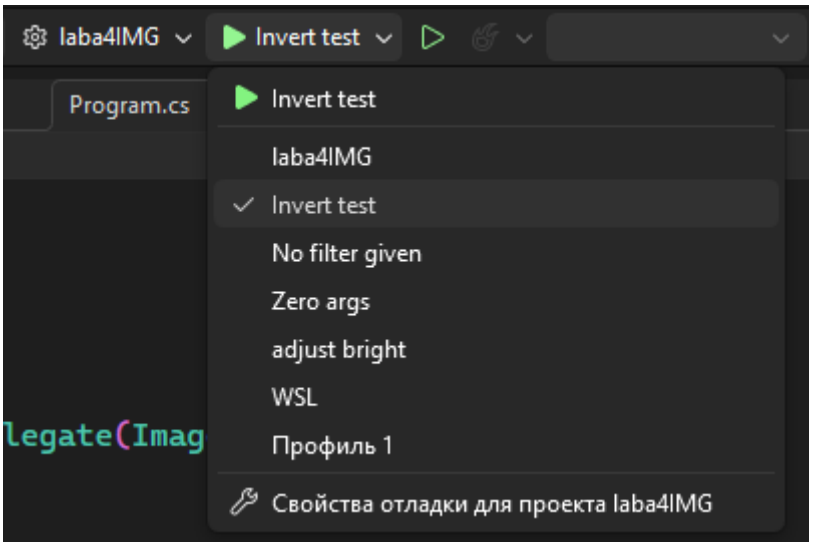
laba4_create_profile_s3.png

Пример переданных аргументов:



laba4_profile_create_s4.png

Теперь мы можем отладить нашу программу, передавая аргументы в консоль во время отладки.



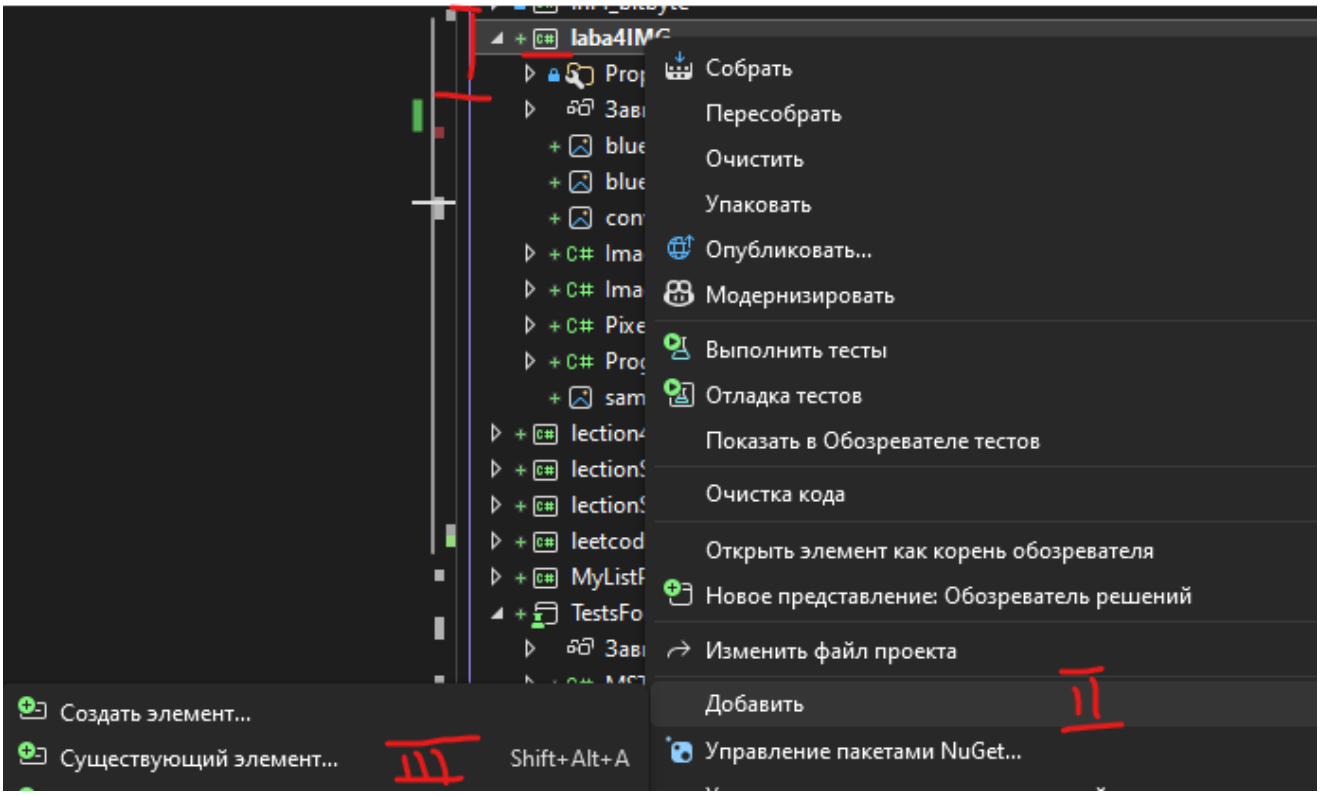
laba4_create_profile_s5.png

Использование созданных классов и структур

Добавление классов в проект (создание/импорт)

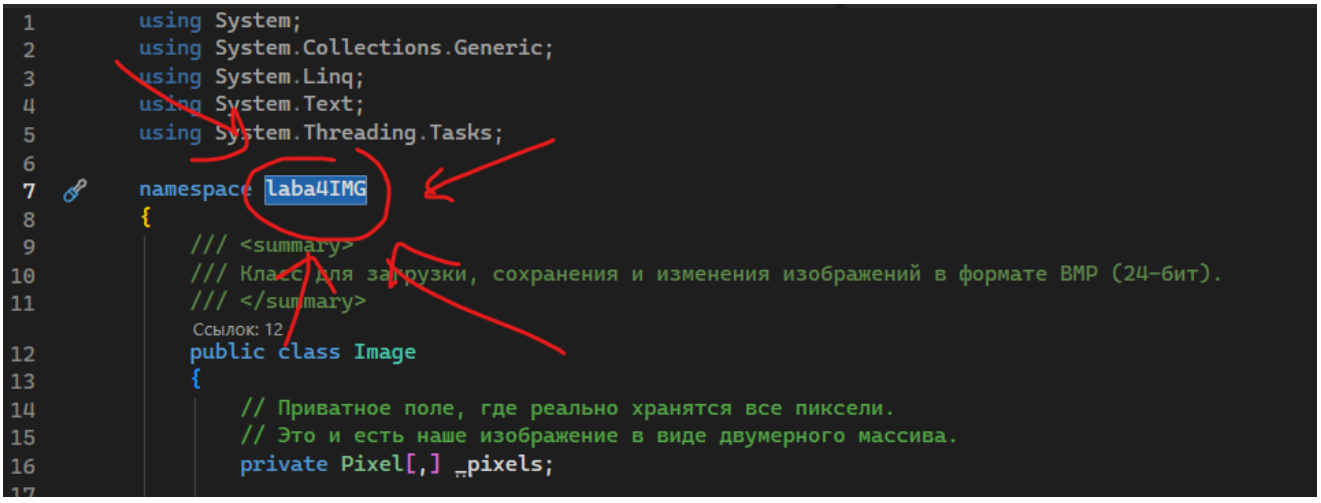
Для добавление **существующего** файла в проект нужно:

- Кликнуть ПКМ по названию проекта (с зеленой иконкой в обозревателе решений слева)
- Кликнуть по "Добавить"
- Выбрать "Существующий элемент"
  Далее просто выбираем файл (`Image.cs` или `Pixel.cs`).
  После в нашем проекте внутри **обозревателя решений** отобразится файл.



laba4_add_existing_file_cs.png

Внутри этого файла нужно поменять пространство имен на ваше.



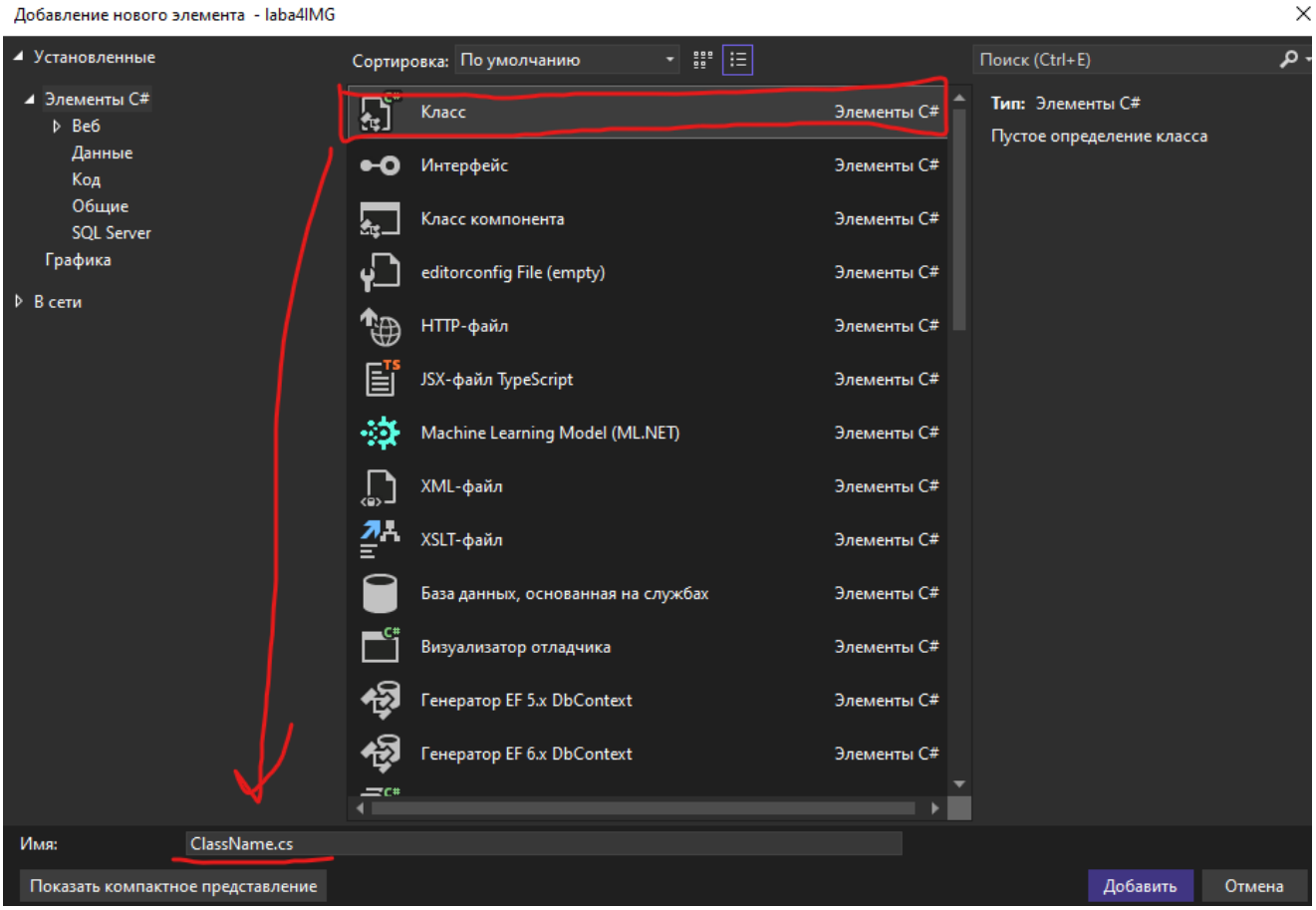
laba4_change_namespace_into_file.png

Обведенную строку нужно заменить на название вашего проекта.

Для создания нового файла путь примерно такой же:

- Кликнуть ПКМ по названию проекта (с зеленой иконкой в обозревателе решений слева)
- Кликнуть по "Добавить"
- Выбрать "Создать элемент..."

Выбираем "Класс", даем ему название. Класс создастся с таким же названием, как файл.



laba4_set_class_name.png

Для создания структуры путь такой же, только нужно в созданном классе поменять `class` на `struct`.

Пространства имен

Пространство имен (namespace) — еще один способ организации кода, который заключается в возможности классы, типы и другие пространства имен объеденять в однин "контейнер".

Простая аналогия: Думайте о пространствах имен как о папках на вашем компьютере. В папке "Фотографии" вы храните изображения, а в папке "Документы" — текстовые файлы. Точно так же в пространстве имен `MyProject.UI` могут храниться классы, отвечающие за пользовательский интерфейс, а в `MyProject.Data` — классы для работы с данными.

> Пример скриншотом тут

Зачем нужны пространства имен?

У пространств имен есть две основные цели:

1. **Организация кода:** Они помогают структурировать большие проекты, делая код более понятным и читаемым. Вы всегда знаете, где искать нужный класс.
2. **Предотвращение конфликтов имен:** В большом проекте или при использовании сторонних библиотек могут появиться классы с одинаковыми именами. Пространства имен решают эту проблему. Например, у вас может быть класс `Logger` в вашем проекте, и библиотека, которую вы используете, тоже может иметь класс `Logger`. Поместив их в разные пространства имен (например, `MyProject.Logging` и `ThirdPartyLibrary.Logging`), вы избежите путаницы.

Как объявить и использовать пространство имен?

Объявление

Для создания пространства имен используется ключевое слово `namespace`, за которым следует его имя.

```
namespace MyProject
{
    // Здесь находятся ваши классы, структуры и т.д.
    class MyClass
    {
        // ...
    }
}
```

Пространства имен могут быть вложенными, образуя иерархию, подобную папкам.

```
namespace MyProject.Data
{
    class DatabaseReader
    {
        // ...
    }
```

```
}
```

Не обязательно пространство имен должно называться как папка или проект в котором оно находится.

Использование

Есть два способа получить доступ к классу внутри пространства имен:

1. **Полное (уточненное) имя:** Указать всю цепочку пространств имен до имени класса через точку.
   ```
   MyProject.Data.DatabaseReader reader = new MyProject.Data.DatabaseReader();
   ```
2. **Директива `using`:** Чтобы не писать каждый раз длинное имя, можно "подключить" пространство имен в начале файла с помощью директивы `using`. Это говорит компилятору, где искать классы, которые вы используете.
   ```
   using System;
   using MyProject.Data; // Подключаем наше пространство имен

   class Program
   {
       static void Main(string[] args)
       {
           // Теперь можно использовать короткое имя
           DatabaseReader reader = new DatabaseReader();
           Console.WriteLine("Доступ к классу получен!");
       }
   }
   ```

Стандартные пространства имен .NET

Когда вы создаете проект, вы уже используете множество пространств имен из библиотеки классов .NET Framework. Вот

несколько ключевых примеров:

- `System`: Содержит фундаментальные и базовые классы, такие как `Console`, `String`, `Int32`.
- `System.IO`: Предоставляет классы для работы с файлами и потоками данных (например, `File`, `StreamReader`).
- `System.Collections.Generic`: Содержит классы для работы с коллекциями, такие как `List<T>` и `Dictionary<TKey, TValue>`.

Начиная с .NET 6 и C# 10, многие из этих часто используемых пространств имен подключаются к вашему проекту автоматически (неявно), поэтому вам не всегда нужно писать для них директиву `using`.