
The background features several large, stylized, overlapping swirls in shades of purple, green, and light blue. Interspersed among these swirls are numerous small, yellow, triangular shapes that resemble sun rays or confetti, scattered across the white background.

Работа с файлами в Си- шарп.

- 
- **Файл** – это набор данных, который хранится на внешнем запоминающем устройстве.
 - Файл имеет имя и расширение.
 - Расширение позволяет идентифицировать, какие данные и в каком формате хранятся в файле.

Под работой с файлами подразумевается:

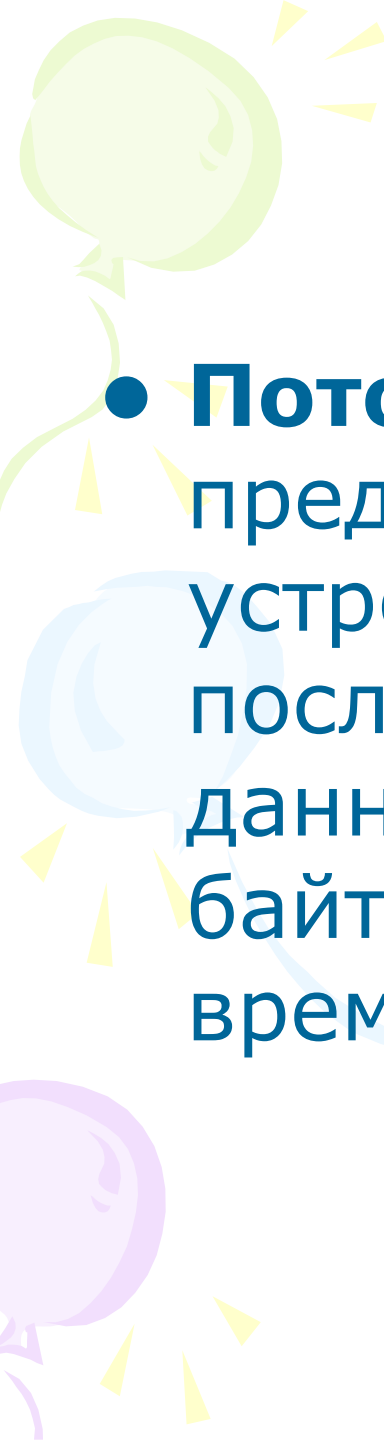
- - создание файлов;
 - удаление файлов;
 - чтение данных;
 - запись данных;
 - изменение параметров файла (имя, расширение...);
 - другое.

Потоковая архитектура

- В основе потоковой архитектуры .NET лежат три понятия:
 - *опорное хранилище (backing store)*
 - *декоратор (decorator)*
 - *адаптер (adapter)*

Потоковая архитектура

- **Опорное хранилище** - это конечная точка ввода-вывода: файл, сетевое подключение и т. д. Оно может представлять собой либо источник, из которого последовательно считываются байты, либо приемник, куда байты последовательно записываются, либо и то и другое вместе.
- Чтобы использовать опорное хранилище его нужно открыть. Этой цели и служат **потоки**, которые в .NET представлены классом `System.IO.Stream`, содержащий методы для чтения, записи и позиционирования потоков.

- 
- **Поток** — абстрактное представление последовательного устройства, облегчающее последовательное хранение данных и доступ к ним (по одному байту в каждый конкретный момент времени).

Потоки делятся на две категории:

- *потоки опорных хранилищ* - потоки, жестко привязанные к конкретным типам опорных хранилищ, такие как FileStream или NetworkStream
- *потоки-декораторы* - наполняют другие потоки, трансформируя данные тем или иным способом, такие как DeflateStream или CryptoStream



ПОТОКИ-ДЕКОРАТОРЫ

- Декораторы освобождают потоки опорных хранилищ от необходимости самостоятельно реализовывать такие вещи, как сжатие и шифрование. Декораторы можно подключать во время выполнения, а также соединять их в цепочки (т.е. использовать несколько декораторов в одном потоке).

Как создать файл?

• Для создания пустого файла, в классе **File** есть метод **Create()**. Он принимает один аргумент – путь. Ниже приведен пример создания пустого текстового файла `new_file.txt` на диске D:

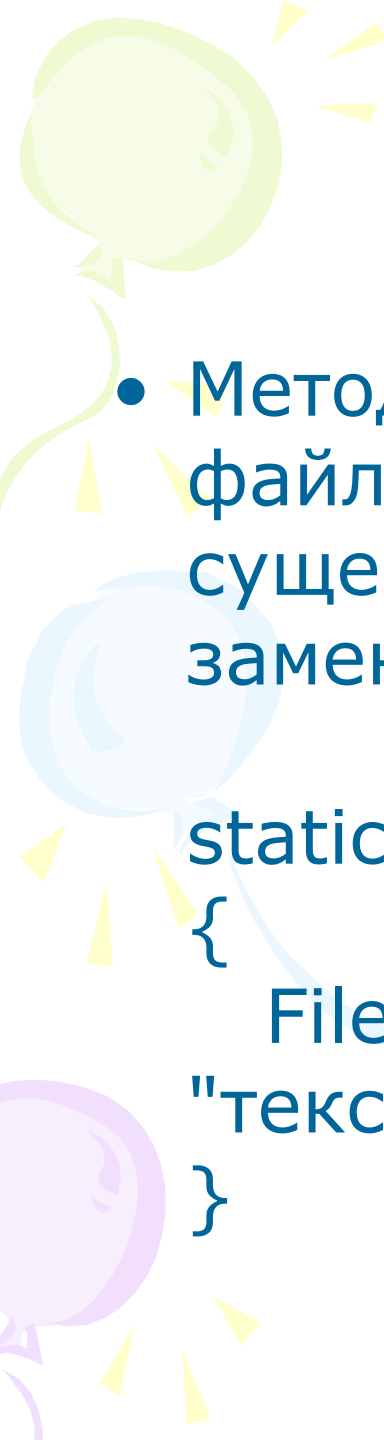
```
static void Main(string[] args)
{
    File.Create("D:\\new_file.txt");
}
```



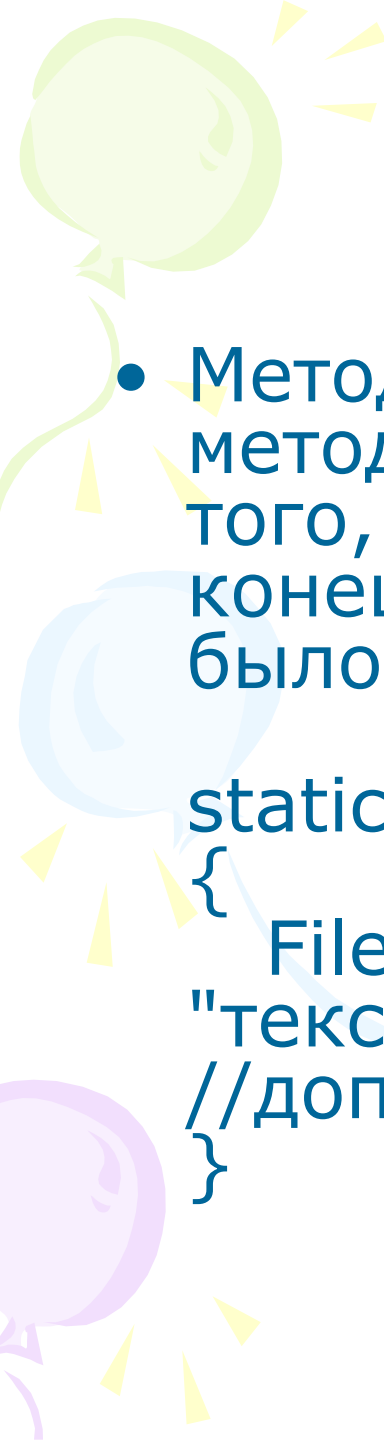
Как удалить файл?

- Метод **Delete()** удаляет файл по указанному пути:


```
static void Main(string[] args)
{
    File.Delete("d:\\test.txt");
    //удаление файла
}
```

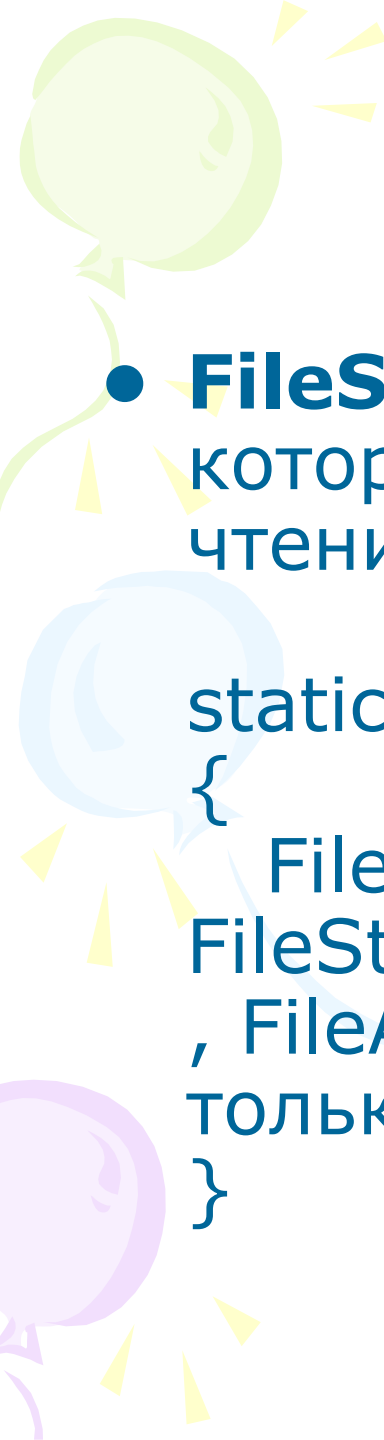
- 
- Метод **WriteAllText()** создает новый файл (если такого нет), либо открывает существующий и записывает текст, заменяя всё, что было в файле:

```
static void Main(string[] args)
{
    File. WriteAllText("D:\\new_file.txt",
"текст");
}
```

- 
- Метод **AppendAllText()** работает, как и метод `WriteAllText()` за исключением того, что новый текст дописывается в конец файла, а не переписывает всё что было в файле:

```
static void Main(string[] args)
{
    File.AppendAllText("D:\\new_file.txt",
"текст метода AppendAllText (");
//допишет текст в конец файла
}
```

- 
- Класс **Stream** является абстрактным базовым классом для всех потоковых классов в Си-шарп. Для работы с файлами нам понадобится класс **FileStream** (файловый поток).

- 
- **FileStream** - представляет поток, который позволяет выполнять операции чтения/записи в файл.

```
static void Main(string[] args)
```

```
{
```

```
    FileStream file = new  
    FileStream("d:\\test.txt", FileMode.Open  
    , FileAccess.Read); //открывает файл  
    только на чтение
```

```
}
```

Режимы открытия **FileMode**:

- *Append* – открывает файл (если существует) и переводит указатель в конец файла (данные будут дописываться в конец), или создает новый файл. Данный режим возможен только при режиме доступа `FileAccess.Write`.
- *Create* - создает новый файл(если существует – заменяет)
- *CreateNew* – создает новый файл (если существует – генерируется исключение)
- *Open* - открывает файл (если не существует – генерируется исключение)
- *OpenOrCreate* – открывает файл, либо создает новый, если его не существует
- *Truncate* – открывает файл, но все данные внутри файла затирает (если файла не существует – генерируется исключение)



- Режим доступа **FileAccess**:

- *Read* – открытие файла только на чтение. При попытке записи генерируется исключение

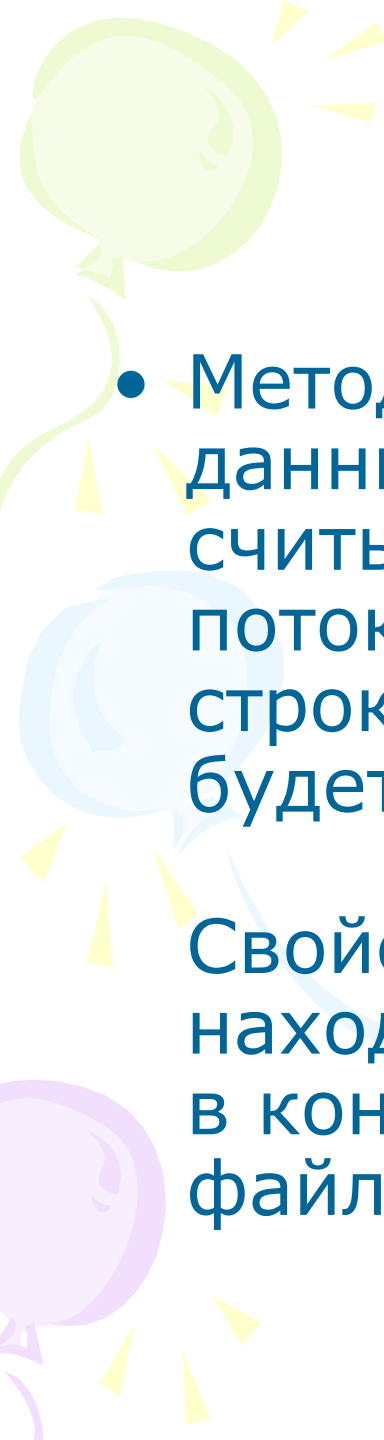
- *Write* - открытие файла только на запись. При попытке чтения генерируется исключение

- *ReadWrite* - открытие файла на чтение и запись.

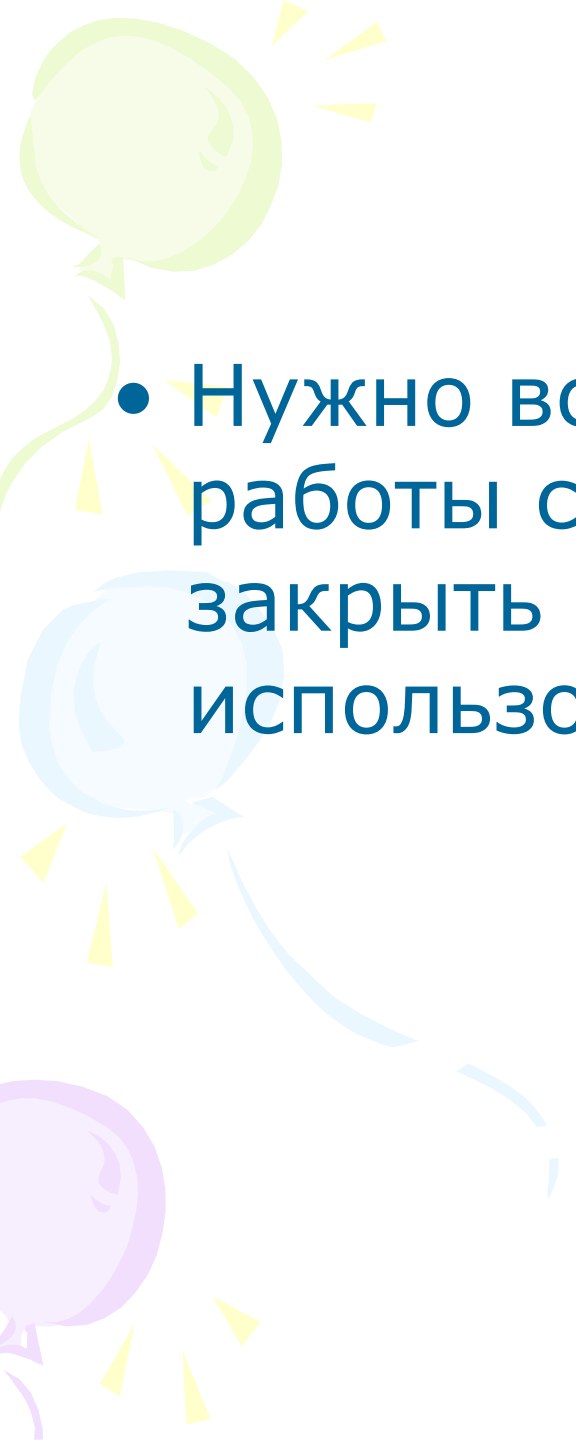
Чтение из файла

Для чтения данных из потока нам понадобится класс **StreamReader**. В нем реализовано множество методов для удобного считывания данных. Ниже приведена программа, которая выводит содержимое файла на экран:

```
static void Main(string[] args)
{
    FileStream file1 = new FileStream("d:\\test.txt",
    FileMode.Open); //создаем файловый поток
    StreamReader reader = new StreamReader(file1); //
    создаем «поточковый читатель» и связываем его с
    файловым потоком
    Console.WriteLine(reader.ReadToEnd()); //считываем
    все данные с потока и выводим на экран
    reader.Close(); //закрываем поток
    Console.ReadLine();
}
```

- 
- Метод **ReadToEnd()** считывает все данные из файла. **ReadLine()** – считывает одну строку (указатель потока при этом переходит на новую строку, и при следующем вызове метода будет считана следующая строка).

Свойство **EndOfStream** указывает, находится ли текущая позиция в потоке в конце потока (достигнут ли конец файла). Возвращает *true* или *false*.

- 
- Нужно всегда помнить, что после работы с потоком, его нужно закрыть (освободить ресурсы), используя метод **Close()**.

Запись в файл

- Для записи данных в поток используется класс **StreamWriter**.
Пример записи в файл:

```
static void Main(string[] args)
{
    FileStream file1 = new FileStream("d:\\test.txt",
    FileMode.Create); //создаем файловый поток
    StreamWriter writer = new StreamWriter(file1); //создаем
    «поточный писатель» и связываем его с файловым потоком
    writer.Write("текст"); //записываем в файл
    writer.Close(); //закрываем поток. Не закрыв поток, в файл
    ничего не запишется
}
```

- Метод **WriteLine()** записывает в файл построчно (то же самое, что и простая запись с помощью `Write()`, только в конце добавляется новая строка).


Создание и удаление папок

- С помощью статического метода **CreateDirectory()** класса **Directory**:

```
static void Main(string[] args)
{
    Directory.CreateDirectory("d:\\new_folder");
}
```

- Для удаления папок используется метод **Delete()**:

```
static void Main(string[] args)
{
    Directory.Delete("d:\\new_folder"); //удаление пустой папки
}
```



Задача 1. Создайте файл numbers.txt и запишите в него натуральные числа от 1 до 500 через запятую.

Задача 2. Дан массив строк: "red", "green", "black", "white", "blue". Запишите в файл элементы массива построчно (каждый элемент в новой строке).

Задача 3. Возьмите любой текстовый файл, и найдите в нем размер самой длинной строки.