

Работа с объектами файловой системы Потоковые классы

Класс DriveInfo

- ▶ System.IO
- ▶ инкапсулирует информацию о диске
- ▶ GetDrives() - получение массива DriveInfo

AvailableFreeSpace	объем доступного свободного места на диске в байтах
DriveFormat	имя файловой системы
DriveType	представляет тип диска
IsReady	ГОТОВ ЛИ ДИСК
Name	имя диска
TotalFreeSpace	общий объем свободного места на диске в байтах
TotalSize	размер диска в байтах

```
var allDrives = DriveInfo.GetDrives();
foreach (var d in allDrives)
{
    Console.WriteLine("Drive name: {0}", d.Name);
    Console.WriteLine("Drive type: {0}", d.DriveType);
    if (!d.IsReady) continue;
    Console.WriteLine("Volume Label: {0}", d.VolumeLabel);
    Console.WriteLine("File system: {0}", d.DriveFormat);
    Console.WriteLine("Root: {0}", d.RootDirectory);
    Console.WriteLine("Total size: {0}", d.TotalSize);
    Console.WriteLine("Free size: {0}", d.TotalFreeSpace);
    Console.WriteLine("Available: {0}", d.AvailableFreeSpace);
}
```

```
Drive name: C:\
Drive type: Fixed
Volume Label:
File system: NTFS
Root: C:\
Total size: 499581448192
Free size: 202503036928
Available: 202503036928
Drive name: D:\
Drive type: Fixed
Volume Label: Зарезервировано системой
File system: NTFS
Root: D:\
Total size: 104853504
Free size: 72998912
Available: 72998912
Drive name: E:\
Drive type: CDRom
```

Directory и DirectoryInfo

► Работа с каталогами

- выполняют операции при помощи статических методов, при помощи экземплярных методов

► Directory

CreateDirectory(path)	создает каталог по указанному пути path
Delete(path)	удаляет каталог по указанному пути path
Exists(path)	определяет, существует ли каталог по указанному пути path. Если существует, возвращается true , если не существует, то false
GetDirectories(path)	получает список каталогов в каталоге path
GetFiles(path)	получает список файлов в каталоге path
Move(sourceDirName, destDirName)	перемещает каталог
GetParent(path)	получение родительского каталога

```
string dirName = "D:\\\\";

if (Directory.Exists(dirName))
{
    Console.WriteLine("SubDir:");
    string[] dirs = Directory.GetDirectories(dirName);
    foreach (string s in dirs)
    {
        Console.WriteLine(s);
    }
    Console.WriteLine();
    Console.WriteLine("Files:");
    string[] files = Directory.GetFiles(dirName);
    foreach (string s in files)
    {
        Console.WriteLine(s);
    }
}
```

```
SubDir:
D:\$RECYCLE.BIN
D:\Boot
D:\System Volume Information

Files:
D:\bootmgr
D:\BOOTSECT.BAK
```

► DirectoryInfo

Create()	создает каталог
CreateSubdirectory(path)	создает подкаталог по указанному пути path
Delete()	удаляет каталог
Свойство Exists	определяет, существует ли каталог
GetDirectories()	получает список каталогов
GetFiles()	получает список файлов
MoveTo(destDirName)	перемещает каталог
Свойство Parent	получение родительского каталога
Свойство Root	получение корневого каталога

```
string path = @"C:\Template";  
string subpath = @"Today\Note";  
DirectoryInfo dirInfo = new DirectoryInfo(path);  
if (!dirInfo.Exists)  
{  
    dirInfo.Create();  
}  
dirInfo.CreateSubdirectory(subpath);
```

```
string dirName = "C:\\Users";
```

```
DirectoryInfo dirInfo = new DirectoryInfo(dirName);
```

```
Console.WriteLine($"Название каталога: {dirInfo.Name}");
```

```
Console.WriteLine($"Полное название каталога: {dirInfo.FullName}");
```

```
Console.WriteLine($"Время создания каталога: {dirInfo.CreationTime}");
```

```
Console.WriteLine($"Корневой каталог: {dirInfo.Root}");
```

```
Название каталога: Users
```

```
Полное название каталога: C:\Users
```

```
Время создания каталога: 16.07.2016 9:04:24
```

```
Корневой каталог: C:\
```

File и FileInfo

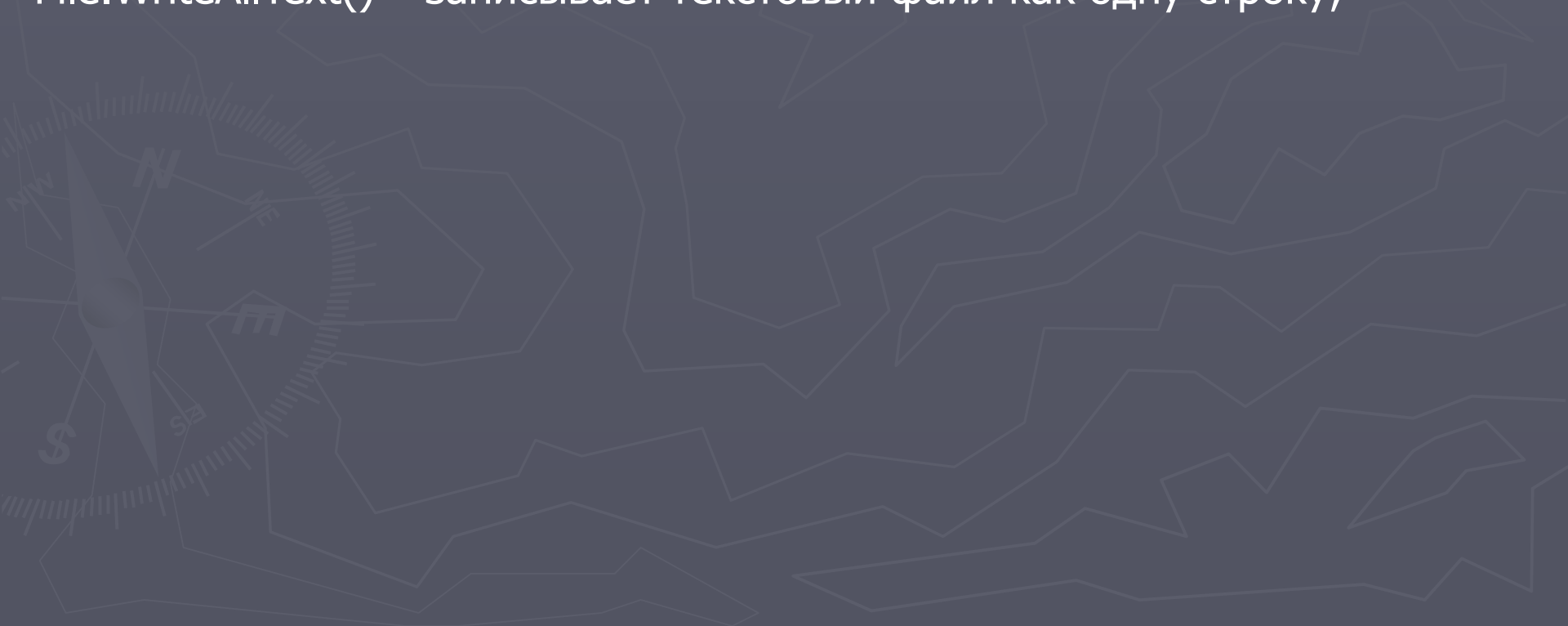


- ▶ Работа с файлами
- ▶ выполняют операции при помощи статических методов, при помощи экземплярных методов

File

Copy()	копирует файл в новое место
Create()	создает файл
Delete()	удаляет файл
Move	перемещает файл в новое место
Exists(file)	определяет, существует ли файл

`File.AppendAllLines()` – добавляет к текстовому файлу набор строк;
`File.AppendAllText()` – добавляет строку к текстовому файлу;
`File.ReadAllBytes()` – возвращает содержимое файла как массив байтов;
`File.ReadAllLines()` – читает текстовый файл как массив строк;
`File.ReadLines()` – читает файл как коллекцию строк, используя отложенные вычисления;
`File.ReadAllText()` – читает содержимое текстового файла как строку;
`File.WriteAllBytes()` – записывает в файл массив байтов;
`File.WriteAllLines()` – записывает в файл массив или коллекцию строк;
`File.WriteAllText()` – записывает текстовый файл как одну строку;



► FileInfo

CopyTo(path)	копирует файл в новое место по указанному пути path
Create()	создает файл
Delete()	удаляет файл
MoveTo(destFileName)	перемещает файл в новое место
Свойство Directory	получает родительский каталог в виде объекта DirectoryInfo
Свойство DirectoryName	получает полный путь к родительскому каталогу
Свойство Exists	указывает, существует ли файл
Свойство Length	получает размер файла
Свойство Extension	получает расширение файла
Свойство Name	получает имя файла
Свойство FullName	получает полное имя файла

```
string path = @"C:\Temp\today.txt";
FileInfo fileInf = new FileInfo(path);
if (fileInf.Exists)
{
    fileInf.Delete();
    // File.Delete(path);
}
```

```
var dir = new DirectoryInfo(@"C:\Users");

// получаем файлы по маске из всех подкаталогов
FileInfo[] f = dir.GetFiles("*.txt", SearchOption.AllDirectories);

// получаем файлы, используя отложенное выполнение
foreach (var fileInfo in dir.EnumerateFiles())
    Console.WriteLine(fileInfo.Name);
```

► DirectoryInfo и FileInfo являются наследниками абстрактного класса FileSystemInfo

Элементы класса `FileSystemInfo`

Имя элемента	Описание
Attributes	Свойство позволяет получить или установить атрибуты объекта файловой системы (тип – перечисление <code>FileAttributes</code>)
CreationTime	Время создания объекта файловой системы
Exists	Свойство для чтения, проверка существования объекта файловой системы
Extension	Свойство для чтения, расширение файла
FullName	Свойство для чтения, полное имя объекта файловой системы
LastAccessTime, LastAccessTimeUtc	Время последнего доступа к объекту файловой системы (локальное или всемирное координированное)
LastWriteTime, LastWriteTimeUtc	Время последней записи для объекта файловой системы (локальное или всемирное координированное)
Name	Свойство для чтения, имя файла или каталога
Delete()	Метод удаляет объект файловой системы
Refresh()	Метод обновляет информацию об объекте файловой системы

Элементы класса FileInfo

Имя элемента	Описание
AppendText()	Создаёт объект StreamWriter для добавления текста к файлу
CopyTo()	Копирует существующий файл в новый файл
Create()	Создаёт файл и возвращает объект FileStream для работы
CreateText()	Создаёт объект StreamWriter для записи текста в новый файл
Decrypt()	Дешифрует файл, зашифрованный методом Encrypt()
Directory	Свойство для чтения, каталог файла
DirectoryName	Свойство для чтения, полный путь к файлу
Encrypt()	Шифрует файл с учётом системных данных текущего пользователя
IsReadOnly	Булево свойство. Указывает, является ли файл файлом только для чтения
Length	Свойство для чтения, размер файла в байтах
MoveTo()	Перемещает файл (возможно, с переименованием)
Open()	Открывает файл с указанными правами доступа
OpenRead()	Создаёт объект FileStream , доступный только для чтения
OpenText()	Создаёт объект StreamReader для чтения информации из текстового файла
OpenWrite()	Создаёт объект FileStream , доступный для чтения и записи

Open()

► режим запроса на открытие файла из перечисления FileMode:

Append – открывает файл, если он существует, и ищет конец файла. Если файл не существует, то он создаётся. Этот режим может использоваться только с доступом FileAccess.Write;

Create – указывает на создание нового файла. Если файл существует, он будет перезаписан;

CreateNew – указывает на создание нового файла. Если файл существует, генерирует исключение IOException;

Open – операционная система должна открыть существующий файл;

OpenOrCreate – операционная система должна открыть существующий файл или создать новый, если файл не существует;

Truncate – система должна открыть существующий файл и обрезать его до нулевой длины.

Open()

- ▶ тип доступа к данным файла - перечисление FileAccess

Read – файл будет открыт только для чтения;

ReadWrite – файл будет открыт и для чтения, и для записи;

Write – файл открывается только для записи, то есть добавления данных

- ▶ ВОЗМОЖНОСТЬ СОВМЕСТНОЙ РАБОТЫ С ОТКРЫТЫМ файлом - FileShare

None – совместное использование запрещено, на любой запрос на открытие файла будет возвращено сообщение об ошибке;

Read – файл могут открыть и другие пользователи, но только для чтения;

ReadWrite – другие пользователи могут открыть файл и для чтения, и для записи;

Write – файл может быть открыт другими пользователями для записи.

```
var file = new FileInfo(@"C:\Test.txt");  
FileStream fs = file.Open(FileMode.OpenOrCreate,  
                           FileAccess.ReadWrite,  
                           FileShare.None);
```



ZipArchive и ZipFile

► System.IO.Compression

► подключить сборки

	System.IdentityModel.Selectors	4.0.0.0
	System.IdentityModel.Services	4.0.0.0
<input checked="" type="checkbox"/>	System.IO.Compression	4.0.0.0
<input checked="" type="checkbox"/>	System.IO.Compression.FileSystem	4.0.0.0
	System.IO.Log	4.0.0.0

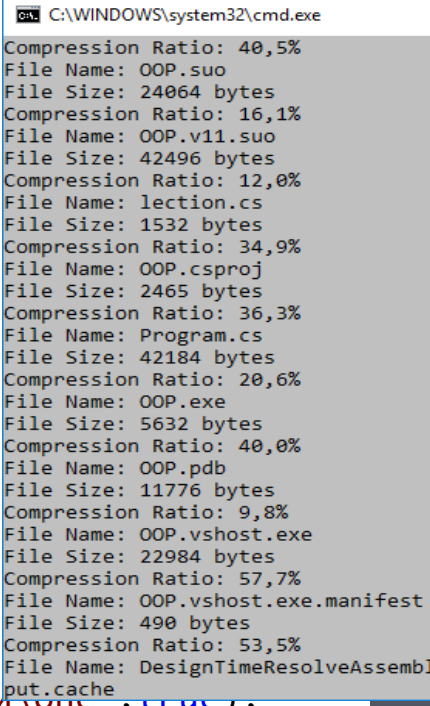
```
using (ZipArchive zip = ZipFile.OpenRead(@"C:\MyArchive.zip"))
{
    foreach (ZipArchiveEntry entry in zip.Entries)
    {
        Console.WriteLine(entry.FullName);

        // предполагается некая работа с потоком данных
        Stream stream = entry.Open();
    }
}
```

► Создание, чтение и извлечение из архива

```
string zipName = @"C:\NATALIA\лекции\ООПС#\ООП_1\лекции\CodeFiles.zip";
string zipFolder = @"C:\NATALIA\лекции\ООПС#\ООП_1\лекции\ООП";
string fileUnzipFullPath = @"C:\NATALIA\лекции\ООПС#\ООП_1\лекции\newООП";
ZipFile.CreateFromDirectory(zipFolder, zipName);
//Открыть zip и прочитать
using (ZipArchive archive = ZipFile.OpenRead(zipName))
{
    //цикл для каждого файла
    foreach (ZipArchiveEntry file in archive.Entries)
    {
        //вывод инфо
        Console.WriteLine("File Name: {0}", file.Name);
        Console.WriteLine("File Size: {0} bytes", file.Length);
        Console.WriteLine("Compression Ratio: {0}",
            ((double)file.CompressedLength / file.Length).ToString("0.0%"));

        //извлечение файла по новому имени
        file.ExtractToFile(@"C:\NATALIA\лекции\ООПС#\ООП_1\лекции\newООП\{0}.cs", true);
    }
}
```



<https://docs.microsoft.com/en-us/dotnet/standard/io/how-to-compress-and-extract-files>

Статический класс Path

- ▶ предназначен для работы с именами файлов и путями в файловой системе

Назначение

- ▶ выделить имя файла из полного пути
- ▶ скомбинировать для получения пути имя файла и имя каталога
- ▶ сгенерировать имя для временного файла или каталога

```
C:\Users\npats\AppData\Local\Temp\tmp3D91.tmp  
C:\Windows
```

```
string tempFile = Path.GetTempFileName();  
Console.WriteLine(tempFile);  
string ext = Path.GetExtension("info.txt");    // .txt  
string win =  
Environment.GetFolderPath(Environment.SpecialFolder.Windows);  
Console.WriteLine(win);
```

Класс FileSystemWatcher

- ▶ позволяет производить мониторинг активности выбранного каталога



Синтаксическая конструкция using

using (*получение-ресурса*)

вложенный-оператор

- ▶ Здесь *получение-ресурса* означает один из вариантов.
 - Объявление и инициализацию локальной переменной (или списка переменных). Тип переменной должен реализовывать **IDisposable**. Такая переменная в блоке **using** доступна только для чтения.
 - Выражение, значение которого имеет тип, реализующий **IDisposable**.

Пример использования using

```
using (ClassWithDispose x = new ClassWithDispose())  
{  
    x.DoSomething();  
    // компилятор C# поместит сюда вызов x.Dispose()  
}
```

```
FileStream fstream = null;
try
{
    fstream = new FileStream(@"D:\file.dat", FileMode.OpenOrCreate);
    // операции с потоком
}
catch(Exception ex)
{
}
finally
{
    if (fstream != null)
        fstream.Close();
}
```

Чтение и запись файлов.

Потоковые классы

- ▶ типы для представления потоков данных
- ▶ адаптеры потоков
- ▶ Поток данных – это абстрактное представление данных в виде последовательности байтов.
 - ассоциируется с неким физическим хранилищем (файлами на диске, памятью, сетью)
 - декорирует другой поток, преобразуя данные тем или иным образом

- ▶ Адаптеры потоков служат оболочкой потока, преобразуя информацию определённого формата в набор байт (сами адаптеры потоками не являются).

System.IO.Stream

- Абстрактный класс, базовый класс для других классов, представляющих потоки

Чтение данных	<code>bool CanRead { get; }</code>
	<code>IAsyncResult BeginRead(byte[] buffer, int offset, int count, AsyncCallback callback, object state)</code>
	<code>int EndRead(IAsyncResult asyncResult)</code>
	<code>int Read(byte[] buffer, int offset, int count)</code>
	<code>Task<int> ReadAsync(byte[] buffer, int offset, int count)</code>
	<code>int ReadByte()</code>
Запись данных	<code>bool CanWrite { get; }</code>
	<code>IAsyncResult BeginWrite(byte[] buffer, int offset, int count, AsyncCallback callback, object state)</code>
	<code>int EndWrite(IAsyncResult asyncResult)</code>
	<code>void Write(byte[] buffer, int offset, int count)</code>
	<code>Task WriteAsync(byte[] buffer, int offset, int count)</code>
	<code>void WriteByte(byte value)</code>
	<code>void CopyTo(Stream destination)</code>
	<code>Task CopyToAsync(Stream destination)</code>

Перемещение	<code>bool CanSeek { get; }</code>
	<code>long Position { get; set; }</code>
	<code>void SetLength(long value)</code>
	<code>long Length { get; }</code>
	<code>long Seek(long offset, SeekOrigin origin)</code>
Закрытие потока	<code>void Close()</code>
	<code>void Dispose()</code>
	<code>void Flush()</code>
	<code>Task FlushAsync()</code>
Таймауты	<code>bool CanTimeout { get; }</code>
	<code>int ReadTimeout { get; set; }</code>
	<code>int WriteTimeout { get; set; }</code>
Другие члены	<code>static readonly Stream Null</code>
	<code>static Stream Synchronized(Stream stream)</code>

- ▶ поддержка асинхронных операций ввода/вывода
- ▶ `ИмяОперацииAsync()`
- ▶ `BeginRead()` и `BeginWrite()` - устарели

Классы для работы с потоками, связанными с хранилищами

- ▶ FileStream – класс для работы с файлами, как с потоками (System.IO).
- ▶ MemoryStream – класс для представления потока в памяти (System.IO).
- ▶ NetworkStream – работа с сокетами, как с потоками (System.Net.Sockets).
- ▶ PipeStream – абстрактный класс из пространства имён System.IO.Pipes, базовый для классов-потоков, которые позволяют передавать данные между процессами операционной системы.

```
using (FileStream fs = new FileStream("test.dat",  
                                       FileMode.OpenOrCreate))  
{  
    for (byte i = 0; i < 100; i++)  
    {  
        fs.WriteByte(i);  
    }  
    fs.Position = 0;  
    while (fs.Position < fs.Length)  
    {  
        Console.Write(fs.ReadByte());  
    }  
}
```

0123456789101112131415161718192021222324252627282930313233343536373839404142434445464748495051525354555657585960616263
6566676869707172737475767778798081828384858687888990919293949596979899100

► Произвольный доступ к файлам

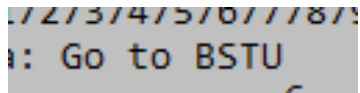
► Seek()

► **SeekOrigin.Begin**: начало файла

► **SeekOrigin.End**: конец файла

► **SeekOrigin.Current**: текущая позиция в файле

```
using (FileStream fstream = new FileStream(@"C:\Users\sometext.dat",
                                           FileMode.OpenOrCreate))
{
    // преобразуем строку в байты
    byte[] input = Encoding.Default.GetBytes(text);
    fstream.Write(input, 0, input.Length);
    // перемещаем указатель
    fstream.Seek(0, SeekOrigin.Begin);
    // считываем 2 символа с текущей позиции
    byte[] output = new byte[2];
    fstream.Read(output, 0, output.Length);
    // декодируем байты в строку
    string textFromFile=Encoding.Default.GetString(output);
    // заменим
    string replaceText = "BSTU";
    fstream.Seek(-3, SeekOrigin.End);
    input = Encoding.Default.GetBytes(replaceText);
    fstream.Write(input, 0, input.Length);
    fstream.Seek(0, SeekOrigin.Begin);
    output = new byte[fstream.Length];
    fstream.Read(output, 0, output.Length);
    textFromFile = Encoding.Default.GetString(output);
    Console.WriteLine("Текст из файла: {0}", textFromFile);
}
```



12/314/516/778/
Go to BSTU

Декораторы потоков

- ▶ DeflateStream и GZipStream – классы для потоков со сжатием данных (System.IO.Compression).
- ▶ CryptoStream – поток зашифрованных данных (System.Security.Cryptography).
- ▶ BufferedStream – поток с поддержкой буферизации данных (System.IO).


```
File.WriteAllBytes("File.bin", new byte[100000]);

// читаем, используя буфер
using (FileStream fsm = File.OpenRead("File.bin"))
{
    using (var bs = new BufferedStream(fsm, 20000))
    {
        bs.ReadByte();
        Console.WriteLine(fs.Position);
    }
}
```

Адаптеры потоков

- ▶ `BinaryReader` и `BinaryWriter` – классы для ввода и вывода примитивных типов в двоичном формате.
- ▶ `StreamReader` и `StreamWriter` – классы для ввода и вывода информации в строковом представлении.
- ▶ `XmlReader` и `XmlWriter` – абстрактные классы для ввода/вывода XML.

StreamReader

Close	закрывает считываемый файл и освобождает все ресурсы
Peek	возвращает следующий доступный символ или -1
Read	считывает и возвращает следующий символ в численном представлении. Read(char[] array, int index, int count),
ReadLine	считывает одну строку в файле
ReadToEnd	считывает весь текст из файла

```
using (StreamReader sr = new StreamReader(@"C:\Users\temp.txt"))
{
    Console.WriteLine(sr.ReadToEnd());
}
```

StreamWriter

Close	закрывает записываемый файл и освобождает все ресурсы
Flush	записывает в файл оставшиеся в буфере данные и очищает буфер
Write	записывает в файл данные простейших типов, как int, double, char, string и т.д.
WriteLine	также записывает данные, добавляет в файл символ окончания строки

```
StreamWriter(writePath, false, System.Text.Encoding.Default).
```

↑
true, добавляются в
конец
false,
перезаписывается

↑
указывает кодировку, в
которой записывается
файл

```
using (StreamWriter sw = new StreamWriter(@"C:\Users\temp.txt",  
                                           false, System.Text.Encoding.Default))  
{  
    sw.WriteLine(text);  
}
```

BinaryWriter

Close()	закрывает поток и освобождает ресурсы
Flush()	очищает буфер, дописывая из него оставшиеся данные в файл
Seek()	устанавливает позицию в потоке
Write()	записывает данные в поток

BinaryReader

Close()	закрывает поток и освобождает ресурсы
ReadBoolean()	считывает значение bool и перемещает указатель на один байт
ReadByte()	считывает один байт и перемещает указатель на один байт
ReadChar()	считывает значение char, то есть один символ, и перемещает указатель на столько байтов, сколько занимает символ в текущей кодировке
ReadDecimal()	считывает значение decimal и перемещает указатель на 16 байт
ReadDouble()	считывает значение double и перемещает указатель на 8 байт
ReadInt16()	считывает значение short и перемещает указатель на 2 байта
ReadInt32()	считывает значение int и перемещает указатель на 4 байта
ReadInt64()	считывает значение long и перемещает указатель на 8 байт
ReadSingle()	считывает значение float и перемещает указатель на 4 байта
ReadString()	считывает значение string. Каждая строка предваряется значением длины строки

```
using (BinaryWriter writer = new
        BinaryWriter(File.Open(@"C:\NATALIA\temp.txt",
                                FileMode.OpenOrCreate)))
{

    writer.Write("43242");
    writer.Write(23);
    writer.Write('s');
    writer.Write(2.4);

}
```

```
string path= @"C:\NATALIA\temp.txt";
using (BinaryReader reader = new
        BinaryReader(File.Open(path,
                                FileMode.Open)))
{

    string name = reader.ReadString();
    int area = reader.ReadInt32();
    char clas = reader.ReadChar();
    double population = reader.ReadDouble();

}

}
```