Работа с объектами файловой системы Потоковые классы

System.IO Класс DriveInfo

- Для представления диска
- GetDrives() - возвращает имена всех логических дисков компьютера.

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

```
DriveInfo[] drives = DriveInfo.GetDrives();

foreach (DriveInfo drive in drives)
{
    Console.WriteLine($"Название: {drive.Name}");
    Console.WriteLine($"Тип: {drive.DriveType}");
    if (drive.IsReady)
    {
        Console.WriteLine($"Объем диска: {drive.TotalSize}");
        Console.WriteLine($"Свободное пространство: {drive.TotalFreeSpace}");
        Console.WriteLine($"Метка диска: {drive.VolumeLabel}");
    }
    Console.WriteLine();
}
```

Получим имена и свойства всех дисков на компьютере

```
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
Drive name: G:\
Drive type: Fixed
Volume Label:
File system: NTFS
Root: G:\
Total size: 500000878592
Free size: 14716362752
Available: 14716362752
```

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
Name	имя каталога
FullName	полный путь к каталогу

Получение списка файлов и подкаталогов

```
string dirName = "C:\\";
// если папка существует
if (Directory.Exists(dirName))
{
    Console.WriteLine("Подкаталоги:");
    string[] dirs = Directory.GetDirectories(dirName);
    foreach (string s in dirs)
    {
        Console.WriteLine(s);
    }
    Console.WriteLine();
    Console.WriteLine("Файлы:");
    string[] files = Directory.GetFiles(dirName);
    foreach (string s in files)
    {
        Console.WriteLine(s);
    }
}
```

Либо мы используем двойной слеш: "C:\\", либо одинарный, но тогда перед всем путем ставим знак @: @"C:\Program Files"

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

var directory = new DirectoryInfo(dirName);

if (directory.Exists)
{
    Console.WriteLine("Подкаталоги:");
    DirectoryInfo[] dirs = directory.GetDirectories();
    foreach (DirectoryInfo dir in dirs)
    {
        Console.WriteLine(dir.FullName);
    }
    Console.WriteLine();
    Console.WriteLine("Файлы:");
    FileInfo[] files = directory.GetFiles();
    foreach (FileInfo file in files)
    {
        Console.WriteLine(file.FullName);
    }
}
```

```
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:\
```

Фильтрация папок и файлов

В качестве фильтра в эти методы передается шаблон, который может содержать два плейсхолдера:

* или символ-звездочка (соответствует любому количеству символов)

? или вопросительный знак (соответствует одному символу)

```
// класс Directory
string[] dirs = Directory.GetDirectories(dirName, "books*.");

// класс DirectoryInfo
var directory = new DirectoryInfo(dirName);
DirectoryInfo[] dirs = directory.GetDirectories("books*.");
```

```
// класс Directory
string[] files = Directory.GetFiles(dirName, "*.exe");

// класс DirectoryInfo
var directory = new DirectoryInfo(dirName);
FileInfo[] files = directory.GetFiles("*.exe");
```

Создание каталога. Класс DirectoryInfo

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

если директория существует, то ее создать будет нельзя, и приложение выбросит ошибку

```
string path = @"C:\SomeDir";
string subpath = @"program\avalon";
if (!Directory.Exists(path))
{
    Directory.CreateDirectory(path);
}
Directory.CreateDirectory($"{path}/{subpath}");
```

Получение информации о каталоге

```
string dirName = "C:\\Program Files";

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}");
```

Удаление каталога

применим метод Delete к непустой папке, в которой есть какие-нибудь файлы или подкаталоги, то приложение нам выбросит ошибку.

перед удалением следует проверить наличие удаляемой папки, иначе приложение выбросит исключение:

```
string dirName = @"C:\SomeDir";

DirectoryInfo dirInfo = new DirectoryInfo(dirName);
if (dirInfo.Exists)
{
    dirInfo.Delete(true);
    Console.WriteLine("Каталог удален");
}
else
{
    Console.WriteLine("Каталог не существует");
}
```

передать в метод Delete дополнительный параметр булевого типа, который укажет, что папку надо удалять со всем содержимым.

```
string dirName = @"C:\SomeDir";
if (Directory.Exists(dirName))
{
    Directory.Delete(dirName, true);
    Console.WriteLine("Каталог удален");
}
else
{
    Console.WriteLine("Каталог не существует");
}
```

Перемещение каталога

При перемещении надо учитывать, что новый каталог, в который мы хотим перемесить все содержимое старого каталога, не должен существовать.

Перемещение каталога в рамках одной папки (как в примере выше) фактически аналогично переименованию папки

```
string oldPath = @"C:\SomeFolder";
string newPath = @"C:\SomeDir";
DirectoryInfo dirInfo = new DirectoryInfo(oldPath);
if (dirInfo.Exists && !Directory.Exists(newPath))
{
    dirInfo.MoveTo(newPath);
    // или так
    // Directory.Move(oldPath, newPath);
}
```

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);
    }
```

Пути к файлам

Для работы с файлами можно применять как абсолютные, так и относительные пути:

```
// абсолютные пути
string path1 = @"C:\Users\eugene\Documents\content.txt";   // для Windows
string path2 = "C:\\Users\\eugene\\Documents\\content.txt";   // для Windows
string path3 = "/Users/eugene/Documents/content.txt";   // для MacOS/Linux

// относительные пути
string path4 = "MyDir\\content.txt";   // для Windows
string path5 = "MyDir/content.txt";   // для MacOS/Linux
```

Получение информации о файле

```
string path = @"C:\Users\eugene\Documents\content.txt";
// string path = "/Users/eugene/Documents/content.txt";  // для MacOS/Linux
FileInfo fileInfo = new FileInfo(path);
if (fileInfo.Exists)
{
    Console.WriteLine($"Имя файла: {fileInfo.Name}");
    Console.WriteLine($"Время создания: {fileInfo.CreationTime}");
    Console.WriteLine($"Размер: {fileInfo.Length}");
}
```

```
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);
```

Удаление файла

```
string path = @"C:\app\content.txt";
FileInfo fileInf = new FileInfo(path);
if (fileInf.Exists)
{
    fileInf.Delete();
    // альтернатива с помощью класса File
    // File.Delete(path);
}
```

Перемещение файла

```
string path =  @"C:\OldDir\content.txt";
string newPath = @"C:\NewDir\index.txt";
FileInfo fileInf = new FileInfo(path);
if (fileInf.Exists)
{
    fileInf.MoveTo(newPath);
    // альтернатива с помощью класса File
    // File.Move(path, newPath);
}
```

Если файл по новому пути уже существует, то с помощью дополнительного параметра можно указать, надо ли перезаписать файл (при значении true файл перезаписывается)

```
string path =  @"C:\OldDir\content.txt";
string newPath = @"C:\NewDir\index.txt";
FileInfo fileInf = new FileInfo(path);
if (fileInf.Exists)
{
    fileInf.MoveTo(newPath, true);
    // альтернатива с помощью класса File
    // File.Move(path, newPath, true);
}
```

Копирование файла. CopyTo (FileInfo)

принимает два параметра:

- путь, по которому файл будет копироваться,
- булевое значение, которое указывает, надо ли при копировании перезаписывать файл (если true, как в случае выше, файл при копировании перезаписывается).

Если же в качестве последнего параметра передать значение false, то если такой файл уже существует, приложение выдаст ошибку.

Copy (File)

принимает три параметра:

- путь к исходному файлу,
- путь, по которому файл будет копироваться,
- булевое значение, указывающее, будет ли файл перезаписываться.

```
string path =  @"C:\OldDir\content.txt";
string newPath = @"C:\NewDir\index2.txt";
FileInfo fileInf = new FileInfo(path);
if (fileInf.Exists)
{
    fileInf.CopyTo(newPath, true);
    // альтернатива с помощью класса File
    // File.Copy(path, newPath, true);
}
```

- 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 `FileAttributes`)
`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);
```

открыть существующий файл или создать новый, если файл не существует;
файл будет открыт и для чтения, и для записи;
совместное использование запрещено, на любой запрос на открытие файла будет возвращено сообщение об ошибке;

Чтение и запись файлов

AppendAllLines(String, IEnumerable<String>) / AppendAllLinesAsync(String, IEnumerable<String>, CancellationToken)

добавляют в файл набор строк. Если файл не существует, то он создается

AppendAllText(String, String) / AppendAllTextAsync(String, String, CancellationToken)

добавляют в файл строку. Если файл не существует, то он создается

Чтение и запись файлов

byte[] ReadAllBytes (string path) / Task<byte[]> ReadAllBytesAsync (string path, CancellationToken cancellationToken)

считывают содержимое бинарного файла в массив байтов

string[] ReadAllLines (string path) / Task<string[]> ReadAllLinesAsync (string path, CancellationToken cancellationToken)

считывают содержимое текстового файла в массив строк

string ReadAllText (string path) / Task<string> ReadAllTextAsync (string path, CancellationToken cancellationToken)

считывают содержимое текстового файла в строку

IEnumerable<string> ReadLines (string path)

считывают содержимое текстового файла в коллекцию строк

void WriteAllBytes (string path, byte[] bytes) / Task WriteAllBytesAsync (string path, byte[] bytes, CancellationToken cancellationToken)

записывают массив байт в бинарный файл. Если файл не существует, он создается. Если существует, то перезаписывается

void WriteAllLines (string path, string[] contents) / Task WriteAllLinesAsync (string path, IEnumerable<string> contents, CancellationToken cancellationToken)

записывают массив строк в текстовый файл. Если файл не существует, он создается. Если существует, то перезаписывается

WriteAllText (string path, string? contents) / Task WriteAllTextAsync (string path, string? contents, CancellationToken cancellationToken)

записывают строку в текстовый файл. Если файл не существует, он создается. Если существует, то перезаписывается

```
string path = @"c:\app\content.txt";

string originalText = "Hello Metanit.com";
// запись строки
await File.WriteAllTextAsync(path, originalText);
// дозапись в конец файла
await File.AppendAllTextAsync(path, "\nHello work");

// чтение файла
string fileText = await File.ReadAllTextAsync(path);
Console.WriteLine(fileText);
```

```
Hello Metanit.com
Hello work
```

перевод на следующую строку.

- Если мы хотим, что в файле изначально шло добавление на новую строку, то для записи стоит использовать метод

WriteAllLines/ WriteAllLinesAsync,

а для добавления - AppendAllLines / AppendAllLinesAsync

- хотим каждую строку файла считать отдельно, то вместо

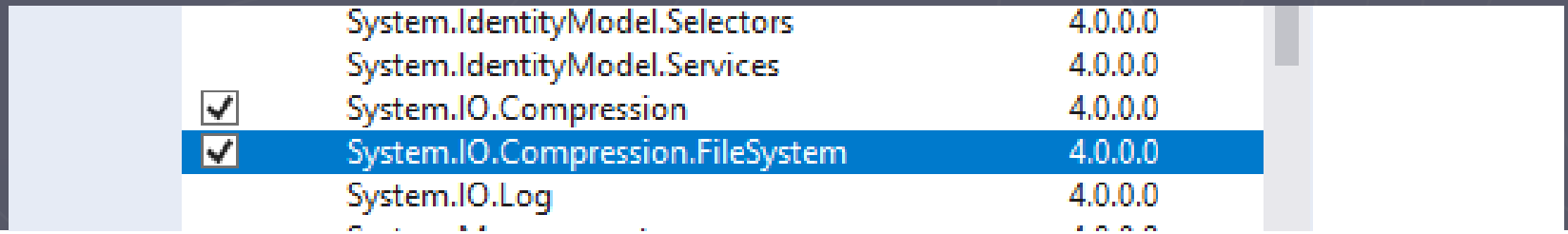
ReadAllText / ReadAllTextAsync применяется ReadAllLines / ReadAllLinesAsync.

Кодировка. System.Text.Encoding

```
using System.Text;

string path = "/Users/eugene/Documents/app/content.txt";

string originalText = "Привет Metanit.com";
// запись строки
await File.WriteAllTextAsync(path, originalText, Encoding.Unicode);
// дозапись в конец файла
await File.AppendAllTextAsync(path, "\nПривет мир", Encoding.Unicode);

// чтение файла
string fileText = await File.ReadAllTextAsync(path, Encoding.Unicode);
Console.WriteLine(fileText);
```

Архивация и сжатие файлов. ZipArchive и ZipFile

- System.IO.Compression
- подключить сборки




```
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();
                }

            }
```

Для создания объекта GZipStream можно использовать один из его конструкторов

GZipStream(Stream stream, CompressionLevel level):

stream представляет данные, а level задает уровень сжатия

GZipStream(Stream stream, CompressionMode mode):

mode указывает, будут ли данные сжиматься или, наоборот, восстанавливаться и может принимать два значения:

CompressionMode.Compress: данные сжимаются

CompressionMode.Decompress: данные восстанавливаются

Если данные сжимаются, то stream указывает на поток архивируемых данных. Если данные восстанавливаются, то stream указывает на поток, куда будут передаваться восстановленные данные.

GZipStream(Stream stream, CompressionLevel level, bool leaveMode):

параметр leaveMode указывает, надо ли оставить открытым поток stream после удаления объекта GZipStream. Если значение true, то поток остается открытым

GZipStream(Stream stream, CompressionMode mode, bool leaveMode)

void CopyTo(Stream destination): копирует все данные в поток destination

Task CopyToAsync(Stream destination): асинхронная версия метода CopyTo

void Flush(): очищает буфер, записывая все его данные в файл

Task FlushAsync(): асинхронная версия метода Flush

int Read(byte[] array, int offset, int count): считывает данные из файла в массив байтов и возвращает количество успешно считанных байтов. Принимает три параметра:

array - массив байтов, куда будут помещены считываемые из файла данные

offset представляет смещение в байтах в массиве array, в который считанные байты будут помещены

count - максимальное число байтов, предназначенных для чтения. Если в файле находится меньшее количество байтов, то все они будут считаны.

ZipFile

предоставляет дополнительные возможности для создания архивов. Он позволяет создавать архив из каталогов.

void CreateFromDirectory(string sourceDirectoryName, string destinationFileName): архивирует папку по пути sourceDirectoryName в файл с названием destinationFileName

void ExtractToDirectory(string sourceFileName, string destinationDirectoryName): извлекает все файлы из zip-файла sourceFileName в каталог destinationDirectoryName

```
using System.IO.Compression;

string sourceFolder = "D://test/"; // исходная папка
string zipFile = "D://test.zip"; // сжатый файл
string targetFolder = "D://newtest"; // папка, куда распаковывается файл
                                     если такой папки нет, она создается

ZipFile.CreateFromDirectory(sourceFolder, zipFile);
Console.WriteLine($"Папка {sourceFolder} архивирована в файл {zipFile}");
ZipFile.ExtractToDirectory(zipFile, targetFolder);

Console.WriteLine($"Файл {zipFile} распакован в папку {targetFolder}");
```

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
```

Создает на диске временный пустой файл с уникальным именем и возвращает полный путь этого файла.

Возвращает расширение из пути файлу

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

Класс FileSystemWatcher

- позволяет производить мониторинг активности выбранного каталога
- Ожидает уведомления файловой системы об изменениях и инициирует события при изменениях каталога или файла в каталоге.

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

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

- 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.Op
        {
            for (byte i = 0; i < 100; i++)
            {
                fs.WriteByte(i);
            }
            fs.Position = 0;
            while (fs.Position < fs.Length)
            {
                Console.Write(fs.ReadByte());
            }
        }
```

если файл существует, он открывается, если нет - создается новый

```
0123456789101112131415161718192021222324252627282930313233343536373839404142434445464748495051525354555657585960616263
6566676869707172737475767778798081828384858687888990919293949596979899100
```

```
FileStream File.Open(string file, FileMode mode);
FileStream File.OpenRead(string file);
FileStream File.OpenWrite(string file);
```

- открывает файл с учетом объекта FileMode и возвращает файловой поток FileStream
- окрывает поток для чтения
- открывает поток для записи.

```
using System.Text;

string path = @"C:\app\note.txt";    // путь к файлу

string text = "Hello METANIT.COM"; // строка для записи

// запись в файл
using (FileStream fstream = new FileStream(path, FileMode.OpenOrCreate))
{
    // преобразуем строку в байты
    byte[] buffer = Encoding.Default.GetBytes(text);
    // запись массива байтов в файл
    await fstream.WriteAsync(buffer, 0, buffer.Length);
    Console.WriteLine("Текст записан в файл");
}

// чтение из файла
using (FileStream fstream = File.OpenRead(path))
{
    // выделяем массив для считывания данных из файла
    byte[] buffer = new byte[fstream.Length];
    // считываем данные
    await fstream.ReadAsync(buffer, 0, buffer.Length);
    // декодируем байты в строку
    string textFromFile = Encoding.Default.GetString(buffer);
    Console.WriteLine($"Текст из файла: {textFromFile}");
}
```

```
Текст записан в файл
Текст из файла: Hello METANIT.COM
```

Произвольный доступ к файлам

- Seek() можем управлять положением курсора потока, начиная с которого производится считывание или запись в файл. Этот метод принимает два параметра: offset (смещение) и позиция в файле. Позиция в файле описывается тремя значениями:
- **SeekOrigin.Begin**: начало файла
- **SeekOrigin.End**: конец файла
- **SeekOrigin.Current**: текущая позиция в файле

```
string path = "note.dat";

string text = "hello world";

using (FileStream fstream = new FileStream(path, FileMode.OpenOrCreate))
{
    // преобразуем строку в байты
    byte[] input = Encoding.Default.GetBytes(text);
    // запись массива байтов в файл
    fstream.Write(input, 0, input.Length);
    Console.WriteLine("Текст записан в файл");
}
// чтение части файла
using (FileStream fstream = new FileStream(path, FileMode.OpenOrCreate))
{
    // перемещаем указатель в конец файла, до конца файла- пять байт
    fstream.Seek(-5, SeekOrigin.End); // минус 5 символов с конца потока

    // считываем четыре символов с текущей позиции
    byte[] output = new byte[5];
    await fstream.ReadAsync(output, 0, output.Length);
    // декодируем байты в строку
    string textFromFile = Encoding.Default.GetString(output);
    Console.WriteLine($"Текст из файла: {textFromFile}"); // world
}
```

```
Текст записан в файл
Текст из файла: world
```

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 fs = File.OpenRead("File.bin"))
    {
using (var bs = new BufferedStream(fs, 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

считывать весь текст или отдельные строки из текстового файла.

StreamReader(string path): через параметр path передается путь к считываемому файлу

StreamReader(string path, System.Text.Encoding encoding): параметр encoding задает кодировку для чтения файла

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

```
string path = "note1.txt";
// асинхронное чтение
using (StreamReader reader = new StreamReader(path))
{
    string text = await reader.ReadToEndAsync();
    Console.WriteLine(text);
}
```

```
string path = "/Users/eugene/Documents/app/note1.txt";

// асинхронное чтение
using (StreamReader reader = new StreamReader(path))
{
    string? line;
    while ((line = await reader.ReadLineAsync()) != null)
    {
        Console.WriteLine(line);
    }
}
```

Считаем текст из файла построчно

StreamWriter

Для записи в текстовый файл

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 конце
`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);
            }
```

```
string path = "note1.txt";
string text = "Hello World\nHello METANIT.COM";

// полная перезапись файла
using (StreamWriter writer = new StreamWriter(path, false))
{
    await writer.WriteLineAsync(text);
}
// добавление в файл
using (StreamWriter writer = new StreamWriter(path, true))
{
    await writer.WriteLineAsync("Addition");
    await writer.WriteAsync("4,5");
}
```