**№ 13 Работа с потоковыми классами и файловой системой**

**Курносенко Софья**

Задание

https://docs.microsoft.com/en-us/dotnet/api/system.io?view=netframework-4.8

Каждый класс в данном проекте должен начинаться (Префикс) с ваших инициалов ФИО (*AVF, JK*,….). Предусмотреть обработку ошибок.

1. Создать класс ***XXXLog***. Он должен отвечать за работу с текстовым файлом *xxxlogfile*.*txt*. в который записываются все действия пользователя и соответственно методами записи в текстовый файл, чтения, поиска нужной информации.

a. Используя данный класс выполните запись всех последующих действиях пользователя с указанием действия, детальной информации (имя файла, путь) и времени (дата/время)

2. Создать класс ***XXXDiskInfo*** c методами для вывода информации о

a. свободном месте на диске

b. Файловой системе

c. Для каждого существующего диска - имя, объем, доступный объем, метка тома.

d. Продемонстрируйте работу класса

3. Создать класс ***XXXFileInfo*** c методами для вывода информации о конкретном файле

a. Полный путь

b. Размер, расширение, имя

c. Дата создания, изменения

d. Продемонстрируйте работу класса

4. Создать класс ***XXXDirInfo*** c методами для вывода информации о конкретном директории

a. Количестве файлов

b. Время создания

c. Количестве поддиректориев

d. Список родительских директориев

e. Продемонстрируйте работу класса

5. Создать класс ***XXXFileManager***. Набор методов определите самостоятельно. С его помощью выполнить следующие действия:

a. Прочитать список файлов и папок заданного диска. Создать директорий ***XXXInspect***, создать текстовый файл *xxxdirinfo*.*txt* и сохранить туда информацию. Создать копию файла и переименовать его. Удалить первоначальный файл.

b. Создать еще один директорий *XXXFiles*. Скопировать в него все файлы с заданным расширением из заданного пользователем директория. Переместить *XXXFiles* в *XXXInspect*.

c. Сделайте архив из файлов директория *XXXFiles*. Разархивируйте его в другой директорий.

6. Найдите и выведите сохраненную информацию в файле *xxxlogfile*.*txt* о действиях пользователя за определенный день/ диапазон времени/по ключевому слову. Посчитайте количество записей в нем. Удалите часть информации, оставьте только записи за текущий час.

7. Обязательно обрабатывайте возможные ошибки. В случае с потоками необходимо использовать конструкцию ***using***. Если необходимо «построить» путь, то следует использовать методы класса ***Path***

**Вопросы**

1. Какие классы содержаться с *System*.IO?

2. Для чего используются классы *Directory* и *DirectoryInfo*? В чем отличие?

3. Для чего используются классы *File* и *FileInfo*? Какие методы они содержат.

4. Для чего используются классы *StreamReader* и *StreamWriter*?

5. Для чего используются классы *BinaryWriter* и *BinaryReader*?

6. Как можно сжимать и восстанавливать файлы?

7. Расскажите алгоритм сжатия *GZip*.

8. Для чего служит класс *Path*?

9. Что такое тестовый и бинарный файл. Как можно выполнить чтение и запись в эти файлы.

10. Что такое произвольный доступ к файлу? Приведите пример.

11. Как применяется конструкция ***using*** (не директива) при работе с файловыми потоками? Для чего ее используют.

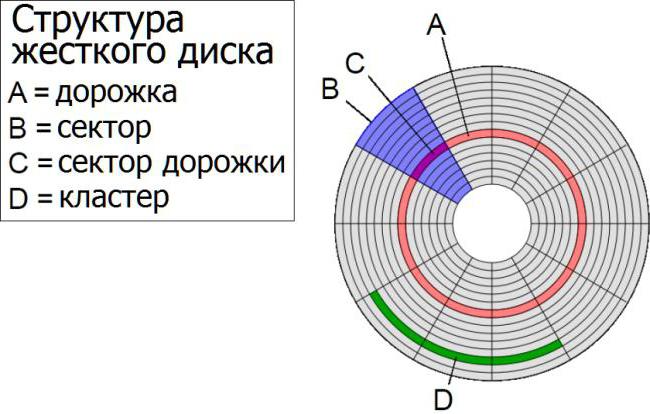
## Что такое файловая система?

Вся информация записывается на носитель в виде [файлов](https://www.pc-school.ru/chto-takoe-fajl-svojstva-fajla/), которые должны располагаться в определенном порядке, иначе операционная система и программы не смогут оперировать с данными. Этот порядок и организует файловая система с помощью определенных алгоритмов и правил размещения файлов на носителе.

Когда программе требуется файл, записанный на диске, ей нет необходимости знать, как и где он хранится. Все, что от программы требуется – это знать имя файла, его размер и атрибуты, чтобы передать эти данные файловой системе, которая обеспечит доступ к нужному файлу. То же самое происходит  и при записи данных на носитель: программа передает информацию о файле (имя, размер, атрибуты) файловой системе, которая сохраняет его по своим определенным правилам.

Для лучшего понимания представьте библиотекаря, который выдает клиенту книгу по ее названию. Или в обратном порядке: клиент сдает прочитанную книгу библиотекарю, который размещает ее обратно на хранение. Клиенту совсем нет необходимости знать, где и как хранится книга, это обязанность служащего заведения. Библиотекарь знает правила каталогизации библиотеки и согласно этим правилам разыскивает издание или размещает его обратно, т.е. выполняет свои служебные функции. В данном примере библиотека – это носитель информации, библиотекарь – файловая система, клиент – программа.

Информация, записываемая на жесткий диск или любой другой носитель, размещается в нем на основе кластерной организации. Кластер представляют собой своего рода ячейку определенного размера, в которую помещается весь файл или его часть.



Если файл имеет размер кластера, то он занимает только один кластер. Если размер файла превышает размер ячейки, то он размещается в нескольких ячейках-кластерах. Причем свободные кластеры могут находиться не рядом с другом, а быть разбросанными по физической поверхности диска. Такая система позволяет наиболее рационально использовать место при хранении файлов. Задача файловой системы  — разложить файл при записи по свободным кластерам оптимальным образом, а также собрать его при чтении и выдать программе или операционной системе.

## Основные функции файловой системы

Основными функциями файловой системы являются:

* размещение и упорядочивание на носителе данных в виде файлов;
* определение максимально поддерживаемого объема данных на носителе информации;
* создание, чтение и удаление файлов;
* назначение и изменение атрибутов файлов (размер, время создания и изменения, владелец и создатель файла, доступен только для чтения, скрытый файл, временный файл, архивный, исполняемый, максимальная длина имени файла и т.п.);
* определение структуры файла;
* [поиск файлов](https://www.pc-school.ru/poisk-fajlov-na-kompyutere/);
* организация каталогов для логической организации файлов;
* защита файлов при системном сбое;
* защита файлов от несанкционированного доступа и изменения их содержимого.

**► Работа с потоками и файловой системой**

Фреймворк .NET предоставляет большие возможности по управлению и манипуляции файлами и каталогами, которые по большей части сосредоточены в пространстве имен **System.IO**. Классы, расположенные в этом пространстве имен (такие как Stream, StreamWriter, FileStream и др.), позволяют управлять файловым вводом-выводом.

## ► Работа с дисками

Работу с файловой системой начнем с самого верхнего уровня - дисков. Для представления диска в пространстве имен System.IO имеется класс **DriveInfo**.

Этот класс имеет статический метод GetDrives, который возвращает имена всех логических дисков компьютера. Также он предоставляет ряд полезных свойств:

* **AvailableFreeSpace**: указывает на объем доступного свободного места на диске в байтах
* **DriveFormat**: получает имя файловой системы
* **DriveType**: представляет тип диска
* **IsReady**: готов ли диск (например, DVD-диск может быть не вставлен в дисковод)
* **Name**: получает имя диска
* **TotalFreeSpace**: получает общий объем свободного места на диске в байтах
* **TotalSize**: общий размер диска в байтах
* **VolumeLabel**: получает или устанавливает метку тома

Получим имена и свойства всех дисков на компьютере:

using System;

using System.IO;

namespace HelloApp

{

class Program

{

static void Main(string[] args)

{

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: {drive.VolumeLabel}");

}

Console.WriteLine();

}

}

}

# }

# 

[IsReady](https://docs.microsoft.com/en-us/dotnet/api/system.io.driveinfo.isready?view=net-6.0) указывает, готов ли диск. Например, он указывает, находится ли компакт-диск в приводе компакт-дисков или готово ли съемное запоминающее устройство для операций чтения / записи. Если вы не проверяете, готов ли диск, и он не готов, запрос диска с помощью [DriveInfo](https://docs.microsoft.com/en-us/dotnet/api/system.io.driveinfo?view=net-6.0) вызовет [исключение](https://docs.microsoft.com/en-us/dotnet/api/system.io.driveinfo?view=net-6.0)[IOException](https://docs.microsoft.com/en-us/dotnet/api/system.io.ioexception?view=net-6.0) .

Не полагайтесь на [IsReady,](https://docs.microsoft.com/en-us/dotnet/api/system.io.driveinfo.isready?view=net-6.0) чтобы избежать [перехвата](https://docs.microsoft.com/en-us/dotnet/api/system.io.driveinfo.isready?view=net-6.0) исключений от других членов, таких как [TotalSize](https://docs.microsoft.com/en-us/dotnet/api/system.io.driveinfo.totalsize?view=net-6.0) , [TotalFreeSpace](https://docs.microsoft.com/en-us/dotnet/api/system.io.driveinfo.totalfreespace?view=net-6.0) и [DriveFormat](https://docs.microsoft.com/en-us/dotnet/api/system.io.driveinfo.driveformat?view=net-6.0) . Между тем, когда ваш код проверяет [IsReady,](https://docs.microsoft.com/en-us/dotnet/api/system.io.driveinfo.isready?view=net-6.0) а затем получает доступ к одному из других свойств (даже если доступ происходит сразу после проверки), возможно, диск был отключен или диск был удален.

# *TotalFreeSpace VS AvailableFreeSpace*

TotalFreeSpace указывает общий объем свободного места, доступного на диске, а не только то, что доступно текущему пользователю.

AvailableFreeSpace указывает объем свободного места, доступного на диске. Обратите внимание, что это число может отличаться от числа TotalFreeSpace , поскольку это свойство учитывает дисковые квоты.

## Что такое дисковые квоты в Windows и для чего они нужны

Дисковые квоты – это инструмент, который позволяет разделять дисковое пространство между пользователями таким образом, чтобы конфликтов по поводу размещения личных файлов ни у кого не возникало.

Разбивать диск на разделы для этого не придётся, достаточно будет наличия нескольких учётных записей (по одной на пользователя) и прав администратора для хозяина компьютера. Перейдите в раздел «Этот компьютер», откройте свойства раздела, ставшего предметов споров и переключитесь на вкладку «Квота». Нажмите «Показать параметры квоты, отметьте галочками пункты «Включить управление квотами» и «Не выделять места на диске при превышении квоты», а затем нажмите «Записи квот».

Далее в окне управления квотами выберите в главном меню Квота → Создать запись квоты.

В открывшемся диалоговом окошке, в поле выбора объектов укажите имя пользователя,

для которого будет создаваться квота и нажмите «ok», в следующем мини-окошке установите объём выделяемого дискового пространства, а также порог выдачи предупреждений.

## ► Работа с каталогами

## Класс Directory

Объявлен и определен в корне System.IO. Содержит в себе группу статических членов для создания, удаления, перемещения и переименования каталогов и подкаталогов.

Запечатанный и **статический** тип: не может иметь наследников, объект класса создать нельзя.

Методы:

**CreateDirectory(String)** и **CreateDirectory(String, DirectorySecurity)**: первый создает каталог по указанному в качестве параметра пути, второй (перегруженный) – делает то же, но с параметрами безопасности Windows.

**Delete(String)** и **Delete(String, Boolean)**: первый удаляет пустой каталог, путь к которому передан в качестве параметра, второй (перегруженный) позволяет удалить каталог со всеми подкаталогами.

**Move(String, String)** – перемещает содержимое каталога, заданного в первом параметре в каталог, заданный вторым параметром.

**GetFiles(String)**, выдают имена содержащихся в каталоге файлах.

**GetCreationTime(String)**, **GetLastWriteTime(String)**, **GetLastAccessTime(String)** – выдают информацию о датах и времени создания, последней записи в каталог или доступа к нему.

**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

**GetParent(path)**: получение родительского каталога

Использовать функционал класса Directory рекомендуется для одиночных операций с каталогом. Поскольку каждое использование сопряжено либо с проверкой введенного пользователем путем, либо с файлами внутри каталога рекомендуется включать его в блок try\catch. Если путь будет задан неправильно, это вызовет исключение.

**Класс DirectoryInfo**

Данный класс предоставляет функциональность для создания, удаления, перемещения и других операций с каталогами. Во многом он похож на Directory.

Некоторые из его свойств и методов:

* **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

Поскольку DirectoryInfo – ссылочный тип (класс), прежде чем воспользоваться его членами, нужно создать его объект с помощью конструктора. В качестве параметра конструктора необходимо передать строку с путем и именем каталога. Каталог будет назначен текущим рабочим каталогом для созданного экземпляра DirectInfo.

//Создаем объект типа и привязываем его к каталогу Temp, находящимся в корне диска С.

DirectoryInfo firstDir = new DirectoryInfo(@"C:\Temp");

//Создаем объект типа и привязываем к каталогу из которого будет запущена программа.

DirectoryInfo secondDir = new DirectoryInfo(".");

Конструктор не создает каталогов, а лишь привязывает к ним. Если каталог, переданный вместе с путём к нему параметром в конструктор, отсутствует, его нужно создать с помощью метода Create() перед его использованием, иначе использование вызовет исключение.

DirectoryInfo firstDir = new DirectoryInfo(@"C:\Temp");

firstDir.Create();

Еще лучше, это воспользоваться свойством Exist перед использованием обьекта внутри блока try/catch:

public static void Main()

{

DirectoryInfo firstDir = new DirectoryInfo(@"c:\Temp");

try

{

if (firstDir.Exists)

{

Console.WriteLine("That path exists already.");

return;

}

// Пытаемся создать каталог.

firstDir.Create();

Console.WriteLine("Каталог был успешно создан.");

// Теперь пытаемся удалить.

firstDir.Delete();

Console.WriteLine("Каталог был успешно удален.");

}

catch (Exception e)

{

Console.WriteLine("Процесс был прерван: {0}", e.ToString());

}

finally { }

}

Посмотрим на примерах применение этих классов

### Получение списка файлов и подкаталогов

string dirName = "D:\\";

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);

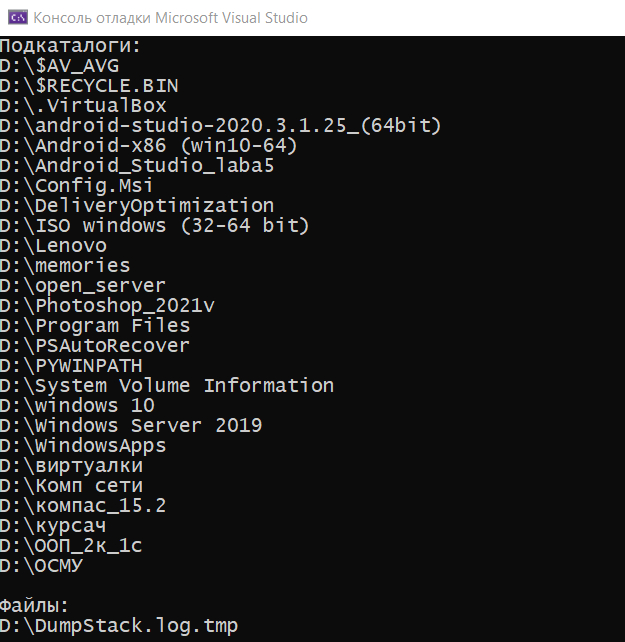
foreach (string s in files)

{

Console.WriteLine(s);

}

}



Обратите внимание на использование слешей в именах файлов. Либо мы используем двойной слеш: "C:\\", либо одинарный, но тогда перед всем путем ставим знак @: @"C:\Program Files"

### Создание каталога

string path = @"C:\SomeDir";

string subpath = @"program\avalon";

DirectoryInfo dirInfo = new DirectoryInfo(path);

if (!dirInfo.Exists)

{

dirInfo.Create();

}

dirInfo.CreateSubdirectory(subpath);

Вначале проверяем, а нету ли такой директории, так как если она существует, то ее создать будет нельзя, и приложение выбросит ошибку. В итоге у нас получится следующий путь: "C:\SomeDir\program\avalon"

### 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

Если мы просто применим метод Delete к непустой папке, в которой есть какие-нибудь файлы или подкаталоги, то приложение нам выбросит ошибку. Поэтому нам надо передать в метод Delete дополнительный параметр булевого типа, который укажет, что папку надо удалять со всем содержимым:

string dirName = @"C:\SomeFolder";

try

{

DirectoryInfo dirInfo = new DirectoryInfo(dirName);

dirInfo.Delete(true);

Console.WriteLine("Каталог удален");

}

catch (Exception ex)

{

Console.WriteLine(ex.Message);

}

Или так:

string dirName = @"C:\SomeFolder";

Directory.Delete(dirName, true);

### Перемещение каталога

string oldPath = @"C:\SomeFolder";

string newPath = @"C:\SomeDir";

DirectoryInfo dirInfo = new DirectoryInfo(oldPath);

if (dirInfo.Exists && Directory.Exists(newPath) == false)

{

dirInfo.MoveTo(newPath);

}

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

## ► Работа с файлами. Классы File и FileInfo

Подобно паре Directory/DirectoryInfo для работы с файлами предназначена пара классов **File** и **FileInfo**. С их помощью мы можем создавать, удалять, перемещать файлы, получать их свойства и многое другое.

Классы File и FileInfo рекомендуется применять, для операций с файлами общего типа, таких как:

◦ Создание и удаление;

◦ Копирование или перемещение;

◦ Открытие и переименование;

◦ Соединение двух файлов в одном;

◦ Изменение атрибутов файла для файловой системы.

Если необходима работа со строками и отдельными символами, члены классов возвращают специальный объект для этого (например, StreamWriter).

Различие между классами кроется в механизме взаимодействия с ними. Так, все члены класса **File** – статические, поэтому он позволяет производить файловые операции без необходимости создавать объект типа. Все его методы требуют указания файлового пути, то есть адресной ссылки внутри файловой системы.

Статичность методов класса File делает его использование предпочтительным в большинстве случаев, когда файловая операция – одиночная.

**Основные методы File**:

* Create – создает новый файл и возвращает объект FileStream для взаимодействия с вновь созданным файлом;
* Delete – удаляет файл;
* Copy – копирует файл;
* Move – перемещает файл;
* Exists(file\_path) – проверяет, существует ли файл;
* CreateText() – создает объект StreamWriter, записывающий новый текстовый файл.

Все эти методы вызываются через имя типа, как при использовании любого статического класса.

Когда существует необходимость множественных действий с конкретным файлом, то безопасней и удобней создать объектную ссылку. Такой подход заключен в классе FileInfo. Функционал этого класса заметно шире благодаря более низкому положению в иерархии наследования типов внутри **System.IO.**

**Так, FileInfo кроме аналогичных методов класса File имеет ряд полезных свойств:**

* **Length** – возвращает размер файла;
* **Directory** – возвращает ссылку на родительский каталог файла (объект типа DirectoryInfo);
* **DirectoryName** – возвращает строку-путь к родительской дериктории;
* **Exist** – проверяет существует ли файл и возвращает значение типа bool;
* Свойство **Extension** – возвращает расширение файла;
* **Name** и **FullName**: возвращают имя или полное имя файла.

Оба класса являются потокобезопасными, однако сохранность значений внутри объектов, которые используют классы не гарантируются.

Один из способов создания дескриптора файла предусматривает использование метода FileInfo.Create():

// Создаем новый файл

FileInfo f = new FileInfo(@"C:\Test.dat");

FileStream fs = f.Create();

// Закрыть файловый поток

fs.Close();

Метод FileInfo.Create() возвращает тип FileStream, который предоставляет синхронную и асинхронную операции записи/чтения лежащего в его основе файла. Имейте в виду, что объект FileStream, возвращенный FileInfo.Create() открывает полный доступ по чтению и записи всем пользователям.

Также имейте в виду, что после окончания работы с текущим объектом FileStream следует закрыть его дескриптор, чтобы освободить лежащие в основе потока неуправляемые ресурсы.

С помощью метода FileInfo.Open() можно открывать существующие файлы, а также создавать новые файлы с гораздо более высокой точностью, чем FileInfo.Create(), учитывая, что Open() обычно принимает несколько параметров для описания общей структуры файла, с которым будет производиться работа. В результате вызова Open() получается возвращенный им объект FileStream. Взгляните на следующую логику:

FileInfo f2 = new FileInfo(@"C:\Test1.dat");

using (FileStream fs2 = f2.Open(FileMode.OpenOrCreate,

FileAccess.ReadWrite, FileShare.None)) { };

Эта версия перегруженного метода Open() требует трех параметров. Первый параметр указывает общий тип запроса ввода-вывода (т.е. создать новый файл, открыть существующий файл и дописать в файл), указываемый в виде перечисления FileMode:

public enum FileMode

{

CreateNew,

Create,

Open,

OpenOrCreate,

Truncate,

Append

}

Второй параметр метода Open() — значение перечисления FileAccess — используется для определения поведения чтения/записи лежащего в основе потока. И, наконец, третий параметр метода Open() — FileShare — указывает, как файл может быть разделен с другими файловыми дескрипторами.

Хотя метод FileInfo.Open() позволяет получить дескриптор файла довольно гибким способом, в классе FileInfo также предусмотрены для этого члены OpenRead() и OpenWrite(). Как и можно было ожидать, эти методы возвращают объект FileStream, соответствующим образом сконфигурированный только для чтения или только для записи, без необходимости применять различные значения перечислений.

Еще один член типа FileInfo, связанный с открытием файлов — OpenText(). В отличие от Create(), Open(), OpenRead() и OpenWrite(), метод OpenText() возвращает экземпляр типа StreamReader, а не FileStream.

## Примеры использования

Допустим, у нас есть тривиальная задача – создание файла и запись в него, разделим ее на 3 этапа:

1. Проверим, существует ли уже такой файл.
2. Если нет, то создадим его и запишем в него строку.
3. Откроем созданный файл и прочитаем строку, записанную нами ранее.

Поскольку задача одиночная используем функционал класса File.

class Program

{

public static void Main()

{

string pathOfFile = @"c:\Examples\FileTestApp.txt"; // Переменная для хранения пути к файлу.

if (!File.Exists(pathOfFile)) // Проверка на существование файла.

{

// Если его еще не существует то создаем и записываем в него.

using (StreamWriter sw = File.CreateText(pathOfFile))

{

sw.WriteLine("Первая строка в файле");

sw.WriteLine("Вторая строка в файле");

sw.WriteLine("Последняя строка в файле");

}

}

// Теперь откроем файл и выведем его содержимое на консоль.

using (StreamReader sr = File.OpenText(pathOfFile))

{

string str = ""; //Переменная в которую построчно считывается содержание файла.

while ((str = sr.ReadLine()) != null) //до тех пор пока не кончатся строки, считываем.

{

Console.WriteLine(str);

}

}

}

}

В качестве пути к файлу мы использовали строковую переменную со значением: *c:\Examples\FileTestApp.txt*. Прежде чем использовать этот код, необходимо, чтобы директория *c:\Examples* была создана. Проверить есть ли она невозможно средствами класса File, для этого используется класс Directory. Если каталога с таким именем не обнаружится, будет вызвано исключение.

В следующем примере описано использование функционала класса FileInfo.

using System;

using System.IO; // Включаем пространство имён Ввода/вывода в приложение.

class Test

{

public static void Main()

{

string pathOfFile = Path.GetTempFileName(); // используем генерацию имени файла.

FileInfo FIO = new FileInfo(pathOfFile); // Создаем объект класса FileInfo.

//Создаем файл и записываем в него.

using (StreamWriter sw = FIO.CreateText())

{

sw.WriteLine("Первая строка");

sw.WriteLine("Вторая строка");

sw.WriteLine("Последняя строка");

}

//Открываем файл и читаем из него.

using (StreamReader sr = FIO.OpenText())

{

string str = "";

while ((str = sr.ReadLine()) != null)

{

Console.WriteLine(str);

}

}

try

{

string pathOfFile2 = Path.GetTempFileName();

FileInfo FIO2 = new FileInfo(pathOfFile2);

//Удостоверяемся что файл назначения отсутствует.

FIO2.Delete();

//Копируем информация в файл.

FIO.CopyTo(pathOfFile2);

Console.WriteLine("{0} скопирован в файл {1}.", pathOfFile, pathOfFile2);

//Удаляем ранее созданный файл.

FIO2.Delete();

Console.WriteLine("{0} удален.", pathOfFile2);

}

catch (Exception e)

{

Console.WriteLine("Процедура прервана: {0}", e.ToString());

}

}

}

Как видно из примера выше, использование ссылки на объект работы с фалом гораздо удобнее в случае если после записи или чтения из фала с ним нужно еще работать. Пример этой работы находится в блоке try/catch (копирование информации из файла в файл).

Поскольку любое безопасное решение обращения к файлам – это выполнение его внутри блока обработки исключений, не придется создавать для каждой одиночной операции новый блок, если есть ссылка на объект.

## ► Архивация и сжатие файлов

### ZipFile

Статический класс **ZipFile** из простанства имен System.IO.Compression предоставляет дополнительные возможности для создания архивов. Он позволяет создавать архив из каталогов. Его основные методы:

* void CreateFromDirectory(string sourceDirectoryName, string destinationFileName): архивирует папку по пути sourceDirectoryName в файл с названием destinationFileName
* void CreateFromDirectory(string sourceFileName, string destinationDirectoryName): извлекает все файлы из zip-файла sourceFileName в каталог destinationDirectoryName

Оба метода имеют ряд дополнительных перегруженных версий. Рассмотрим их применение.

using System;

using System.IO.Compression;

namespace HelloApp

{

class Program

{

static void Main(string[] args)

{

string sourceFolder = "D://test/"; // исходная папка

string zipFile = "D://test.zip"; // сжатый файл

string targetFolder = "D://newtest"; // папка, куда распаковывается файл

ZipFile.CreateFromDirectory(sourceFolder, zipFile);

Console.WriteLine($"Папка {sourceFolder} архивирована в файл {zipFile}");

ZipFile.ExtractToDirectory(zipFile, targetFolder);

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

Console.ReadLine();

}

}

}

В данном случае папка "D://test/" методом ZipFile.CreateFromDirectory архивируется в файл test.zip. Затем метод ZipFile.ExtractToDirectory() распаковывает данный файл в папку "D://newtest" (если такой папки нет, она создается).