

**Работа с файлами. Классы  
для работы с файлами.**

- Большинство задач в программировании так или иначе связаны с работой с файлами и каталогами. Нам может потребоваться прочитать текст из файла или наоборот произвести запись, удалить файл или целый каталог, не говоря уже о более комплексных задачах, как например, создание текстового редактора и других подобных задачах.
- 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.

# Файл –

это набор данных, который хранится на внешнем запоминающем устройстве (например на жестком диске). 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.

# System.IO

Работа с файлами – наиболее традиционный способ использования постоянной памяти. Для этого в **C#** имеется множество классов, содержащихся в пространстве имен **System.IO**

Возможности классов этого пространства имен можно разбить на 2 класса:

1. Классы, использующие файловую систему.
2. Классы, использующие потоки.

# пространство имен **System.IO**

- В C# есть пространство имен **System.IO**, в котором реализованы все необходимые нам классы для работы с файлами. Чтобы подключить это пространство имен, необходимо в самом начале программы добавить строку `using System.IO`.
- Для использования кодировок еще добавим пространство `using System.Text`;
- `using System`;  
`using System.Collections.Generic`;  
`using System.Linq`;  
**`using System.Text`**;  
**`using System.IO`**;

**Классы, использующие файловую  
систему**

# Работа с дисками

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

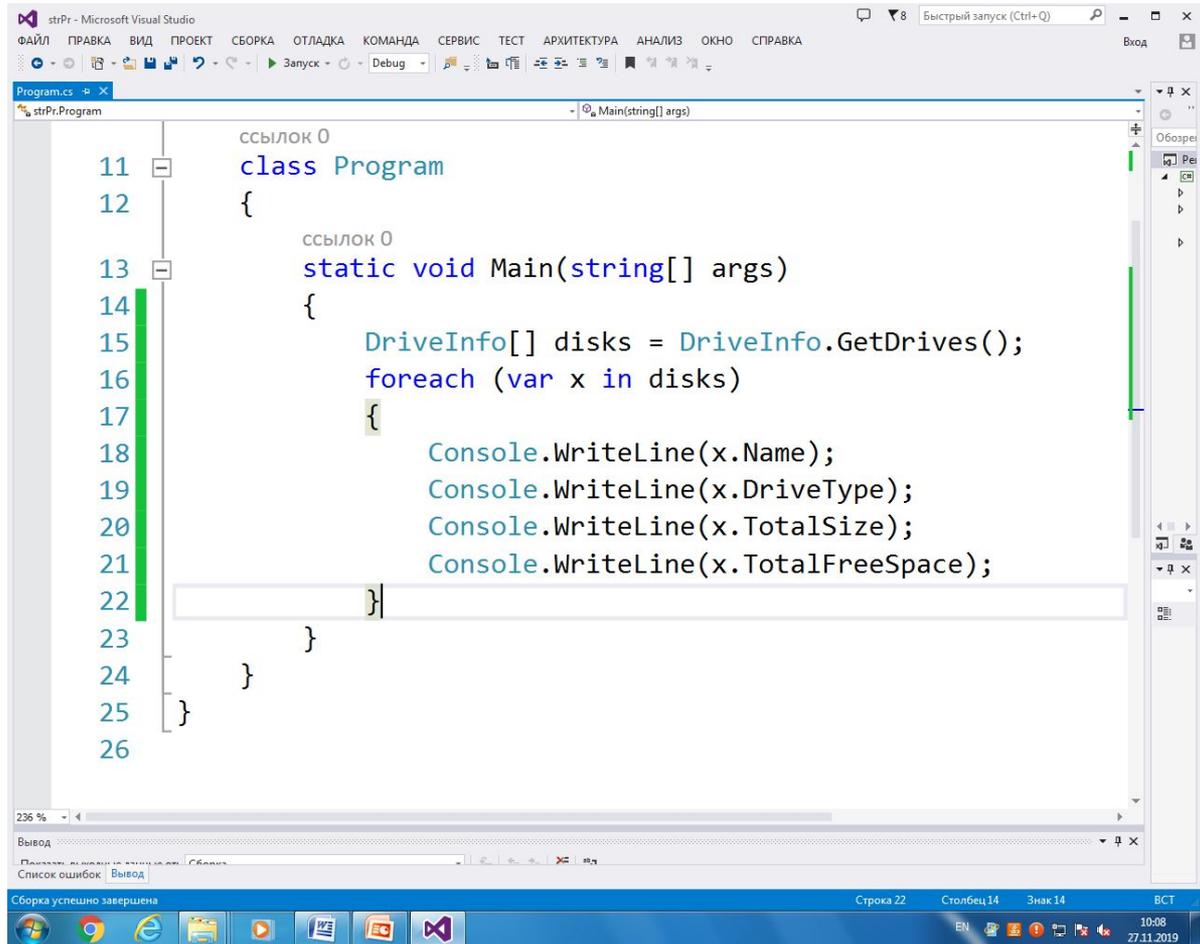
# Класс DriveInfo

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



# Работа с каталогами

- Для работы с каталогами в пространстве имен System.IO предназначены сразу два класса: **Directory** и **DirectoryInfo**.

# Класс Directory

Класс Directory предоставляет ряд статических методов для управления каталогами.

Некоторые из этих методов:

- **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

# Класс DirectoryInfo

**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 DirectoryInfo.

## Например:

```
// Привязаться к текущему рабочему каталогу
```

```
DirectoryInfo dir1 = new DirectoryInfo(".");
```

```
// Привязаться к C:\Windows
```

```
DirectoryInfo dir2 = new DirectoryInfo(@"C:\Windows");
```

@

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

@ "C:\Program Files"

# 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

# Создание каталога Create

пример

```
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: {0}", dirInfo.Name);
```

```
Console.WriteLine("Полное название каталога: {0}", dirInfo.FullName);
```

```
Console.WriteLine("Время создания каталога: {0}", dirInfo.CreationTime);
```

```
Console.WriteLine("Корневой каталог: {0}", dirInfo.Root);
```

пример

# 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);
}
catch (Exception ex)
{
    Console.WriteLine(ex.Message);
}
```

пример

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 переместить 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.
- Все методы класса File статические, методы класса FileInfo работают только через объектную ссылку.

# Некоторые полезные методы и свойства класса **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

Класс **File** реализует похожую функциональность с помощью статических методов:

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

# Методы класса File

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");
}
```

пример

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 ");

    //допишет текст в конец файла
}
```

пример

# Методы класса File

Метод `ReadAllText()` читает данные из файла в строку:

```
static void Main(string[] args)
{
    string content =
        File.ReadAllText(path);
    Console.WriteLine(content);
}
```

пример

Метод `ReadAllLines ()` читает данные из файла в строковый массив :

```
static void Main(string[] args)
{
    string[] lines =
        File.ReadAllLines(path);
    foreach (var x in lines)
    {
        Console.WriteLine(x);
    };
}
```

пример

# Получение информации о файле

```
string path = @"C:\apache\hta.txt";
FileInfo fileInf = new FileInfo(path);
if (fileInf.Exists)
{
    Console.WriteLine("Имя файла: {0}", fileInf.Name);
    Console.WriteLine("Время создания: {0}", fileInf.CreationTime);
    Console.WriteLine("Размер: {0}", fileInf.Length);
}
```

пример

# Удаление файла

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

пример

# Удаление файла

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

пример

# Перемещение файла

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

пример

# Копирование файла

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

пример

# Методы CopyTo и Copy

- Метод CopyTo класса FileInfo принимает два параметра: путь, по которому файл будет копироваться, и булево 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.
- Метод Copy класса File принимает три параметра: путь к исходному файлу, путь, по которому файл будет копироваться, и булево значение, указывающее, будет ли файл перезаписываться.