Взаимодействие с файловой системой

1. Модель потоков в C#. Пространство System.IO.

2. Класс Stream.

3. Анализ байтовых классов потоков.

3.1 Использование класса FileStream для файловых операций.

4. Анализ символьных классов потоков.

4.1 Использование класса StreamWriter для файловых операций.

4.2 Использование класса StreamReader для файловых операций.

5 Анализ двоичных классов потоков.

5.1 Использование класса BinaryWriter для файловых операций.

5.2 Использование класса BinaryReader для файловых операций.

6. Использование классов Directory, DirectoryInfo, FileInfo для файловых операций

Поток — это абстракция производства или потребления информации. С физическим устройством поток связывает система ввода-вывода. Все потоки действуют одинаково - даже если они связаны с разными физическими устройствами.

Потоки ввода/вывода используются для передачи/чтения данных в (из) различные источники. Основными являются:

- ***файлы;***
- ***консоль;***
- ***области памяти;***
- ***сетевые соединения;***
- ***прочие абстракции, связанным с потоками.***

Потоки представляют **собой объекты соответствующих классов**. Пространство имен **System.IO** предоставляет большое число классов и методов для работы с потоками ввода/вывода.

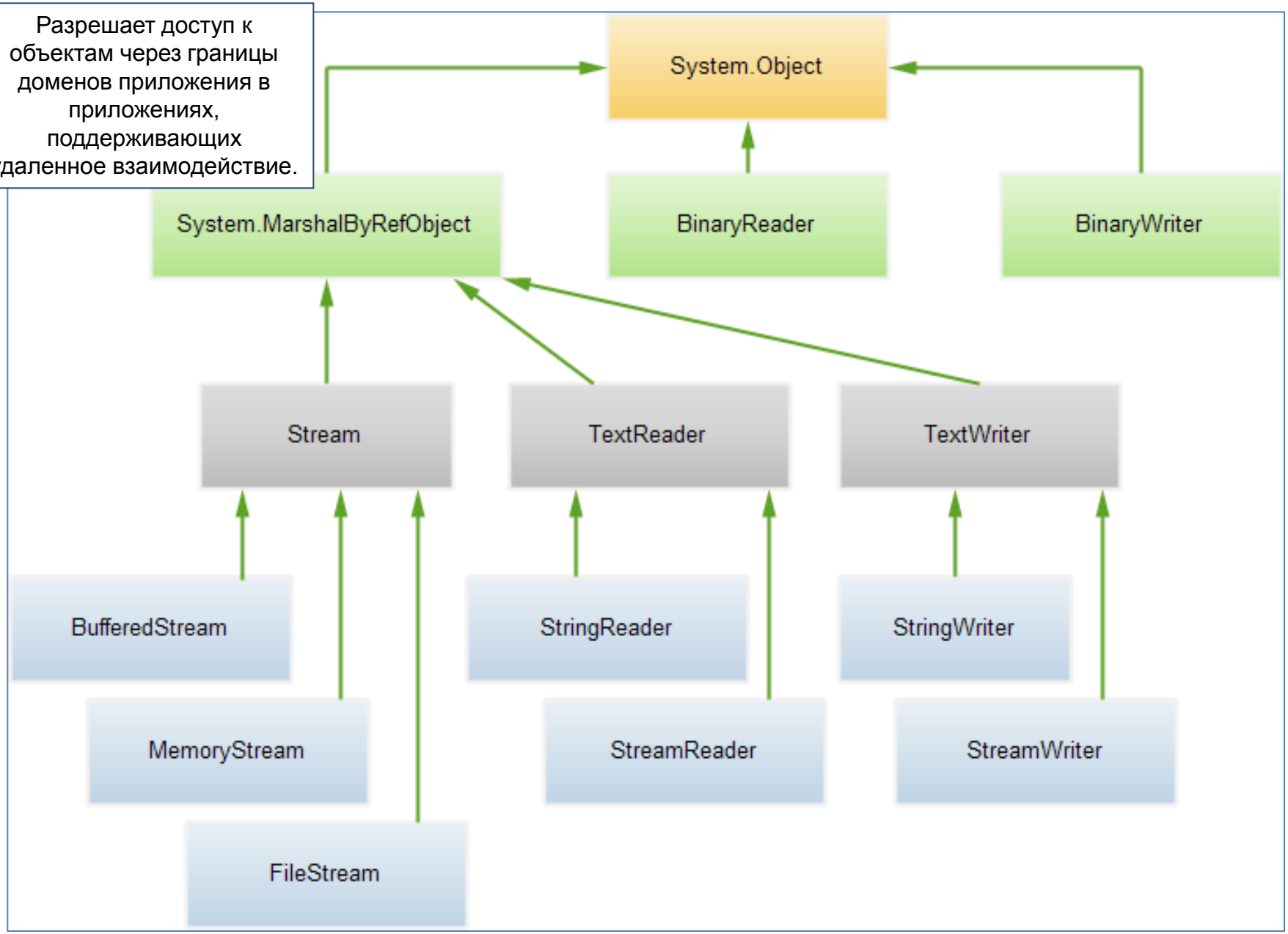
Основным для потоков является класс - **System.IO.Stream**.

- является байтовым потоком
- является базовым для всех остальных классов потоков.
- является абстрактным классом – создать объект класса Stream невозможно.

bool CanRead, **bool CanWrite,** **bool CanSeek**	Определяют, поддерживает ли текущий поток чтение, поиск и/или запись
long Length	Возвращает длину потока в байтах
long Position	Определяет текущую позицию в потоке
int ReadTimeout	Представляет продолжительность времени ожидания в операциях ввода. Доступно как для чтения, так и для записи.
int WriteTimeout	Представляет продолжительность времени ожидания в операциях вывода. Доступно как для чтения, так и для записи

Метод	Описание
void Close ()	Закрывает поток и освобождает ресурсы. Аналог Dispose().
void Flush ()	Обновляет лежащий в основе источник данных
int ReadByte ()	Возвращает целочисленное представление следующего байта, доступного для ввода из потока. При обнаружении конца файла возвращает значение -1
int Read (byte [] buffer, int offset, int count)	Делает попытку ввести count байтов в массив buffer, начиная с элемента buffer[offset]. Возвращает количество успешно введенных байтов
long Seek (long offset, SeekOrigin origin)	Устанавливает текущее положение в потоке по указанному смещению offset относительно заданного начала отсчета origin. Возвращает новое положение в потоке
void WriteByte (byte value)	Выводит один байт в поток вывода
void Write (byte []buffer, int offset, int count)	Выводит подмножество count байтов из массива buffer, начиная с элемента buffer[offset]. Возвращает количество выведенных байтов
SetLength()	Устанавливает длину текущего потока

Разрешает доступ к объектам через границы доменов приложения в приложениях, поддерживающих удаленное взаимодействие.



Иерархия классов в пространстве имен System.IO, которые имеют отношение к потокам

Классы байтовых потоков

Класс потока	Описание
BufferedStream	Заключает в оболочку байтовый поток и добавляет буферизацию. Буферизация, как правило, повышает производительность
FileStream	Байтовый поток, предназначенный для файлового ввода-вывода
MemoryStream	Байтовый поток, использующий память для хранения данных
UnmanagedMemoryStream	Байтовый поток, использующий неуправляемую память для хранения данных

Классы символьных потоков

- Для создания символьного потока достаточно заключить байтовый поток в один из классов символьных потоков.
- На вершине иерархии классов символьных потоков абстрактные классы **TextReader** и **TextWriter**. Класс TextReader организует ввод, а класс TextWriter - вывод.
- Методы, определенные в обоих этих классах, доступны для всех их подклассов. Они образуют минимальный набор функций ввода-вывода, которыми должны обладать все символьные потоки.

Класс потока	Описание
StreamReader	Предназначен для ввода символов из байтового потока. Этот класс является оболочкой для байтового потока ввода
StreamWriter	Предназначен для вывода символов в байтовый поток. Этот класс является оболочкой для байтового потока вывода
StringReader	Предназначен для ввода символов из символьной строки
StringWriter	Предназначен для вывода символов в символьную строку

Отличие: StringWriter для записи в строку и базируется на StringBuilder. StreamWriter для записи символов в поток в указанной кодировке.

Методы ввода, определенные в классе TextReader

Метод	Описание
int Peek()	Получает следующий символ из потока ввода, но не удаляет его. Возвращает значение -1, если ни один из символов не доступен
int Read()	Возвращает целочисленное представление следующего доступного символа из вызывающего потока ввода. При обнаружении конца потока возвращает значение -1
int Read(char[]buffer, int index, int count)	Делает попытку ввести количество count символов в массив buffer, начиная с элемента buffer [index], и возвращает количество успешно введенных символов
int ReadBlock(char[]buffer, int index, int count)	Делает попытку ввести количество count символов в массив buffer, начиная с элемента buffer [index], и возвращает количество успешно введенных символов
string ReadLineO	Вводит следующую текстовую строку и возвращает ее в виде объекта типа string. При попытке прочитать признак конца файла возвращает пустое значение
string ReadToEnd()	Вводит все символы, оставшиеся в потоке, и возвращает их в виде объекта типа string

Двоичные потоки

Для создания файла, содержащего данные типа int, double или short, а также для чтения и записи двоичных значений встроенных в C# типов данных служат классы потоков **BinaryReader** и **BinaryWriter**.

Внимание ! В двоичных потоках данные считываются и записываются во внутреннем двоичном формате, а не в удобочитаемой текстовой форме.

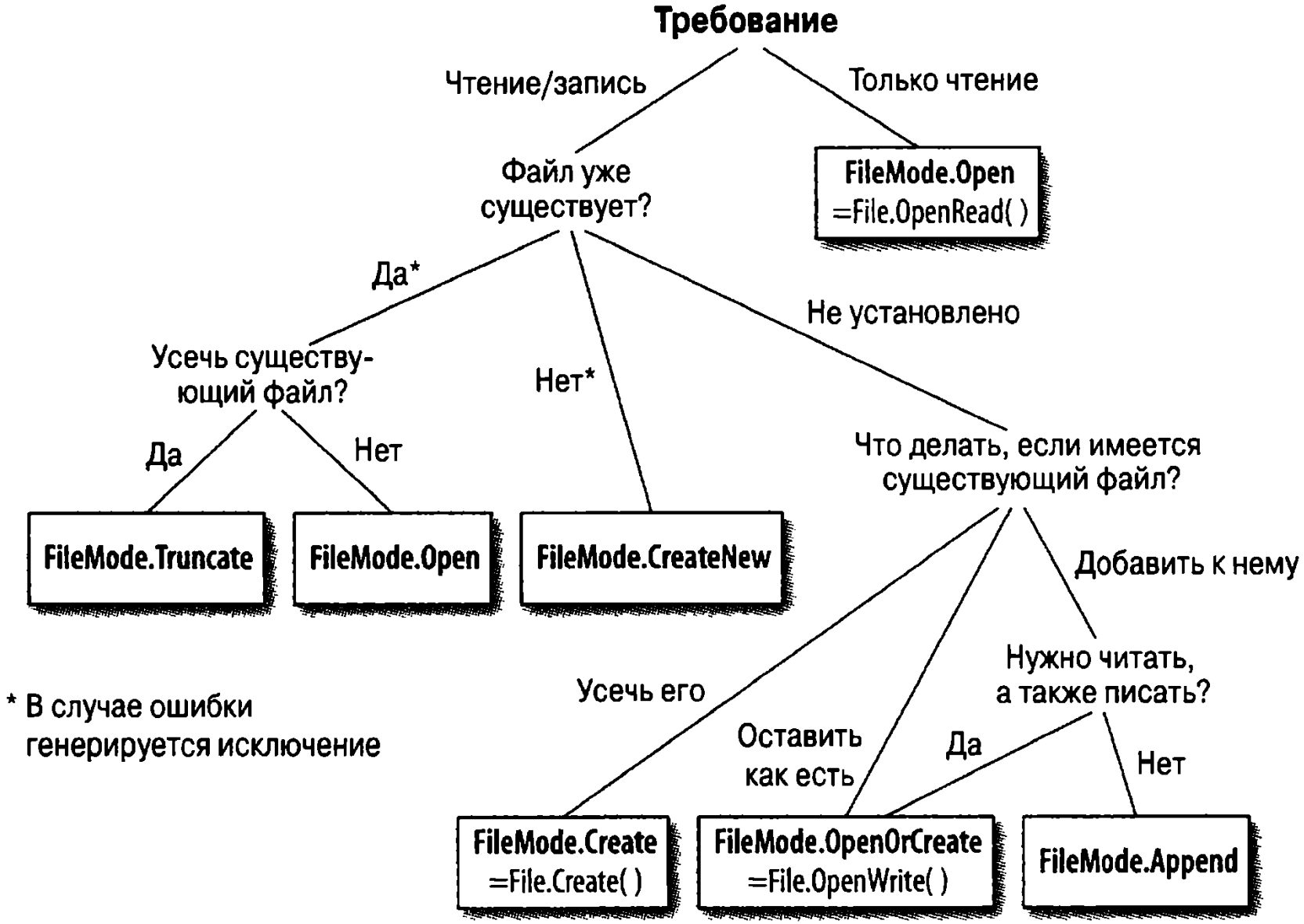
Класс потока	Описание
BinaryReader	используется для ввода двоичных данных
BinaryWriter	используется для вывода двоичных данных

Класс FileStream и байтовый ввод-вывод в файл

Для **создания байтового потока**, привязанного к **файлу**.

```
FileStream(string путь, FileMode режим)
```

Имя члена	Описание
CreateNew	Указывает, что будет создан новый файл. Если файл уже существует, создается исключение **IOException**.
Create	Указывает, что будет создан новый файл. Если файл уже существует, он будет переписан.
Open	Указывает, что будет открыт существующий файл. Если данный файл не существует, создается исключение **FileNotFoundException**.
OpenOrCreate	Указывает, что будет открыт файл, если он существует, в противном случае должен быть создан новый файл.
Truncate	Указывает, что будет открыт существующий файл. После открытия должно быть выполнено усечение файла таким образом, чтобы его размер стал равен нулю.. Попытка чтения из файла, открытого с помощью Truncate, вызывает исключение.
Append	Открывает файл, если он существует, и находит конец файла; либо создает новый файл. FileMode.Append можно использовать только вместе с FileAccess.Write.


Требование
Чтение/запись
Только чтение
Файл уже существует?
FileMode.Open =File.OpenRead( )
Да*
Не установлено
Усечь существующий файл?
Нет*
Что делать, если имеется существующий файл?
Да
Нет
FileMode.Truncate
FileMode.Open
FileMode.CreateNew
Добавить к нему
Усечь его
Оставить как есть
Нужно читать, а также писать?
Да
Нет
FileMode.Create =File.Create( )
FileMode.OpenOrCreate =File.OpenWrite( )
FileMode.Append
* В случае ошибки генерируется исключение

Ограничение доступа к файлу **только для чтения или же только для записи**, то в таком случае следует использовать такой конструктор.

```
FileStream(string путь, FileMode режим, FileAccess доступ)
```

Имя члена	Описание
Read	Доступ для чтения файла. Данные можно прочитать из файла. Для получения доступа для чтения и записи необходимо объединить с Write.
ReadWrite	Доступ для чтения и записи файла. Данные можно записать в файл и прочитать из файла.
Write	Доступ для записи в файл. Данные можно записать в файл. Для получения доступа для чтения и записи комбинируется с Read.

Перечисление **FileShare**, указывающее, какой уровень доступа должен быть выдан другим процессам, чтобы они могли просматривать файл до того, как вы завершите с ним работу (Read (по умолчанию)).

Имя члена	Описание
Delete	Разрешает последующее удаление файла.
None	Отклоняет совместное использование текущего файла. Любой запрос на открытие файла (данным процессом или другим процессом) не выполняется до тех пор, пока файл не будет закрыт.
Read	Разрешает последующее открытие файла для чтения. Если этот флаг не задан, любой запрос на открытие файла для чтения (данным процессом или другим процессом) не выполняется до тех пор, пока файл не будет закрыт. Однако, даже если этот флаг задан, для доступа к данному файлу могут потребоваться дополнительные разрешения.
ReadWrite	Разрешает последующее открытие файла для чтения или записи. Если этот флаг не задан, любой запрос на открытие файла для записи или чтения (данным процессом или другим процессом) не выполняется до тех пор, пока файл не будет закрыт. Однако, даже если этот флаг задан, для доступа к данному файлу могут потребоваться дополнительные разрешения.
Write	Разрешает последующее открытие файла для записи. Если этот флаг не задан, любой запрос на открытие файла для записи (данным процессом или другим процессом) не выполняется до тех пор, пока файл не будет закрыт. Однако, даже если этот флаг задан, для доступа к данному файлу могут потребоваться дополнительные разрешения.

Демонстрация работы с FileStream

```
FileStream fileWr = new FileStream(filePath, FileMode.Create, FileAccess.Write);

if (fileWr.CanRead) Console.WriteLine("Чтение возможно!");
    else Console.WriteLine("Чтение не возможно!");

if (fileWr.CanWrite) Console.WriteLine("Запись возможна!");
    else Console.WriteLine("Запись не возможна!");

byte[] bytesWr = Encoding.Unicode.GetBytes("Демонстрация работы с FileStream");

fileWr.Write(bytesWr, 0, bytesWr.Length);  // запись в файл

fileWr.Close();
// ------------------------------------------------------------------------------
FileStream fileR = new FileStream(filePath, FileMode.Open, FileAccess.ReadWrite);

if (fileR.CanRead) Console.WriteLine("Чтение возможно!");
   else Console.WriteLine("Чтение не возможно!");

if (fileR.CanWrite) Console.WriteLine("Запись возможна!");
   else Console.WriteLine("Запись не возможна!");

byte[]bytesR=new byte[fileR.Length];  // чтение из файла

fileR.Read(bytesR, 0, bytesR.Length);
Console.WriteLine("Информация из файла:");

Console.WriteLine(Encoding.Unicode.GetString(bytesR));

fileR.Close();
```

Демонстрация работы с FileStream

```
using (FileStream fs = new FileStream(filePath, FileMode.Create,
                                                FileAccess.Write))
{   // Получаем данные для записи в файл
    Console.WriteLine("Введите строку для записи в файл:");
    string writeText = Console.ReadLine();
    // Преобразуем строку в массив байт,
    // т.к. FileStream умеет работать только с байтами
    byte[] writeBytes = Encoding.UTF8.GetBytes(writeText);
    // Запишем данные в файл
    fs.Write(writeBytes, 0, writeBytes.Length);
}

using (FileStream fs = new FileStream(filePath, FileMode.Open,
                                                FileAccess.Read))
{   // Теперь прочитаем данные в другой массив байт
    byte[] readBytes = new byte[(int)fs.Length];
    fs.Read(readBytes, 0, readBytes.Length);
    // Преобразуем байты в строку
    string readText = Encoding.UTF8.GetString(readBytes);
    // Выведем ее на консоль
    Console.WriteLine("Данные, прочитанные из файла: {0}", readText);
}
```

Демонстрация работы с FileStream

```
using (FileStream fs =
      new FileStream(filePath, FileMode.Create, FileAccess.ReadWrite))
{   // Получаем данные для записи в файл
    string writeText = Console.ReadLine();
    // Преобразуем строку в массив байт
    byte[] writeBytes = Encoding.UTF8.GetBytes(writeText);
    // Запишем данные в файл
    fs.Write(writeBytes, 0, writeBytes.Length);

    // Сохраним данные из буфера на диск
    fs.Flush();
    // установить курсор  на начало файла
    fs.Seek(0, SeekOrigin.Begin);

    byte[] readBytes = new byte[(int)fs.Length];
    fs.Read(readBytes, 0, readBytes.Length);

    // Преобразуем байты в строку
    string readText = Encoding.UTF8.GetString(readBytes);

    // Выведем ее на консоль
    Console.WriteLine("Данные, прочитанные из файла: {0}", readText);
}
```

Классы StreamReader и StreamWriter

- Классы используют, лежащий в основе **байт-ориентированный поток**.
- Для преобразования между символами и байтами используется класс **Encoding** из пространства имен System.Text (по умолчанию - UTF-8)
- Класс StreamWriter является производным от класса TextWriter, а класс StreamReader - производным от класса TextReader.

```
StreamWriter(Stream поток)
StreamWriter(Stream поток, Encoding.ASCII)
StreamWriter(string путь)
StreamWriter(string путь, bool append)
```

```
StreamReader(Stream поток)
StreamReader(Stream поток, Encoding.ASCII)
StreamReader(string путь)
```

Демонстрация работы с StreamReader

```
string filePath = Console.ReadLine();
// Создаем объект StreamWriter для записи строк в
// различных кодировках в файл.
StreamWriter sw = new StreamWriter(filePath);

// Получаем данные для записи в файл
Console.WriteLine("Введите строку для записи в файл:");
string writeText = Console.ReadLine();

// Запишем данные в поток
sw.Write(writeText);

// Сохраним данные из буфера на диск и закроем поток
sw.Close();

// Теперь прочитаем данные из файла для этого используем StreamReader.
StreamReader sr = new StreamReader(filePath);

// Прочитаем данные
string readText = sr.ReadToEnd();

// Закроем поток
sr.Close();
// Выведем их на консоль
Console.WriteLine("Данные, прочитанные из файла: {0}", readText);
```

Демонстрация работы с StreamWriter

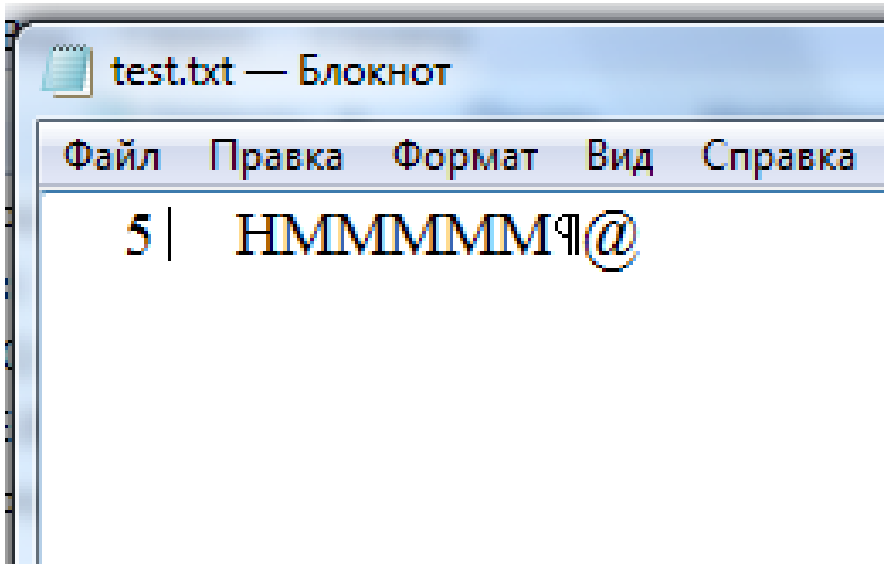
```
string filePath = Console.ReadLine();
// Создаем объект StreamWriter для записи строк
// в различных кодировках в файл.
StreamWriter sw = new StreamWriter(filePath);
// Данные для записи в файл
// Запишем данные  в поток построчно!
sw.WriteLine("Принципы ООП:");
sw.WriteLine("1. Инкапсуляция");
sw.WriteLine("2. Наследование");
sw.WriteLine("3. Полиморфизм");
// Сохраним данные из буфера на диск и закроем поток
sw.Close();

// Теперь прочитаем данные из файла для этого
//  используем StreamReader.
StreamReader sr = new StreamReader(filePath);
string str;
// Выведем данные на консоль (по строчно)
for (int i = 0;  (str=sr.ReadLine()) != null; i++)
{
    Console.WriteLine(str);
}
```

Классы BinaryReader и BinaryWriter

выполняют чтение и запись простых типов данных:

- bool,
- byte,
- char,
- decimal,
- float,
- double,
- short,
- int,
- long,
- sbyte,
- ushort,
- Uint,
- ulong,
- string
- массивов примитивных типов данных.



Методы, определенные в классе BinaryWriter

Метод	Описание
`void Write(sbyte value)`	Записывает значение типа `sbyte` со знаком
`void Write(byte value)`	Записывает значение типа `byte` без знака
`void Write(byte[] buffer)`	Записывает массив значений типа `byte`
`void Write(short value)`	Записывает целочисленное значение типа `short` (короткое целое)
`void Write(ushort value)`	Записывает целочисленное значение типа `ushort` (короткое целое без знака)
`void Write(int value)`	Записывает целочисленное значение типа `int`
`void Write(uint value)`	Записывает целочисленное значение типа `uint` (целое без знака)
`void Write(long value)`	Записывает целочисленное значение типа `long` (длинное целое)
`void Write(ulong value)`	Записывает целочисленное значение типа `ulong` (длинное целое без знака)
`void Write(float value)`	Записывает значение типа `float` (с плавающей точкой одинарной точности)
`void Write(double value)`	Записывает значение типа `double` (с плавающей точкой двойной точности)
`void Write(decimal value)`	Записывает значение типа `decimal` (с двумя десятичными разрядами после запятой)
`void Write(char ch)`	Записывает символ
`void Write(char[] buffer)`	Записывает массив символов
`void Write(string value)`	Записывает строковое значение типа `string`, представленное во внутреннем формате с указанием длины строки

Методы, определенные в классе BinaryReader

Метод	Описание
`bool ReadBoolean()`	Считывает значение логического типа `bool`
`byte ReadByte()`	Считывает значение типа `byte`
`sbyte ReadSByte()`	Считывает значение типа `sbyte`
`byte[] ReadBytes(int count)`	Считывает количество `count` байтов и возвращает их в виде массива
`char ReadChar()`	Считывает значение типа `char`
`char[] ReadChars(int count)`	Считывает количество `count` символов и возвращает их в виде массива
`decimal ReadDecimal()`	Считывает значение типа `decimal`
`double ReadDouble()`	Считывает значение типа `double`
`float ReadSingle()`	Считывает значение типа `float`
`short ReadInt16()`	Считывает значение типа `short`
`int ReadInt32()`	Считывает значение типа `int`
`long ReadInt64()`	Считывает значение типа `long`
`ushort ReadUInt16()`	Считывает значение типа `ushort`
`uint ReadUInt32()`	Считывает значение типа `uint`
`ulong ReadUInt64()`	Считывает значение типа `ulong`
`string ReadString()`	Считывает значение типа `string`, представленное во внутреннем двоичном формате с указанием длины строки. Этот метод следует использовать для считывания строки, которая была записана средствами класса `BinaryWriter`

```
  string filePath = Console.ReadLine();
  using (FileStream fs =  new FileStream(filePath, FileMode.Create,
                              FileAccess.ReadWrite, FileShare.ReadWrite))
   {
// Создаем объект BinaryWriter для  записи простых типов данных
   BinaryWriter bw = new BinaryWriter(fs);
// Получаем данные для записи в файл
   string writeText = Console.ReadLine();
   int writeNum = -1;
   Console.WriteLine("Введите целое число для записи в файл:");
   Int32.TryParse(Console.ReadLine(), out writeNum);
// Запишем данные в поток
   bw.Write(writeText);
   bw.Write(writeNum);
// Сохраним данные из буфера на диск
   bw.Flush();
// Установим курсор на начало файла
   bw.Seek(0, SeekOrigin.Begin);
   BinaryReader br = new BinaryReader(fs);
// Читать нужно в том же порядке, в котором записывались в поток
   Console.WriteLine("Строка, прочитанная из файла:");
   Console.WriteLine(br.ReadString());
   Console.WriteLine("Целое число, прочитанное из файла:");
   Console.WriteLine(br.ReadInt32());
}
```

Ключевые члены пространства имен System.I0

Классы	Описание
Directory **DirectoryInfo**	**Эти классы используются для манипуляций структурой каталогов машины.** Тип **Directory** открывает функциональность с использованием **статических членов**. Тип **DirectoryInfo** обеспечивает аналогичную функциональность через **действительную объектную ссылку**
DriveInfo	Этот класс предоставляет детальную информацию относительно **дисковых устройств, используемых на заданной машине**
File **FileInfo**	**Эти классы служат для манипулирования множеством файлов на машине.** Тип **File** открывает функциональность через **статические члены**. Тип **FileInfo** обеспечивает аналогичную функциональность через **действительную объектную ссылку**
Path	Этот класс выполняет операции над типами System.String, содержащими информацию **о пути к файлу или каталогу** в независимой от платформы манере
FileSystemWatcher	Этот класс позволяет отслеживать **модификации внешних файлов в определенном каталоге**

Абстрактный базовый класс FileSystemInfo

DirectoryInfo и FileInfo наследуют значительную часть своего поведенияя от абстрактного базового класса **FileSystemInfo**.

Свойство	Описание
`Attributes`	Получает или устанавливает ассоциированные с текущим файлом атрибуты, которые представлены перечислением `FileAttributes` (например, доступный только для чтения, зашифрованный, скрытый или сжатый)
`CreationTime`	Получает или устанавливает время создания текущего файла или каталога
`Exists`	Может использоваться для определения, существует ли данный файл или каталог
`Extension`	Извлекает расширение файла
`FullName`	Получает полный путь к файлу или каталогу
`LastAccessTime`	Получает или устанавливает время последнего доступа к текущему файлу или каталогу
`LastWriteTime`	Получает или устанавливает время последней записи в текущий файл или каталог
`Name`	Получает имя текущего файла или каталога

Класс DirectoryInfo

Класс **DirectoryInfo** содержит набор членов, используемых для создания, перемещения, удаления и перечисления каталогов и подкаталогов.

Член	Описание
`Create()` `CreateSubdirectory()`	Создает каталог (или набор подкаталогов) по заданному путевому имени
`Delete()`	Удаляет каталог и все его содержимое
`GetDirectories()`	Возвращает массив объектов `DirectoryInfo`, представляющих все подкаталоги в текущем каталоге
`GetFiles()`	Извлекает массив объектов `FileInfo`, представляющий набор файлов в заданном каталоге
`MoveTo()`	Перемещает каталог со всем содержимым по новому пути
`Parent`	Извлекает родительский каталог данного каталога
`Root`	Получает корневую часть пути

Класс DirectoryInfo

```
****** Directory Info ******
FullName: C:\MyCode\Testing
Name: Testing
Parent: MyCode
Creation: 10.06.2015 22:30:35
Attributes: Directory
Root: C:\
******************************
```

```
// Привязаться к текущему рабочему каталогу.
DirectoryInfo dir1 = new DirectoryInfo(".");
// Привязаться к C:\Windows, используя дословную строку.
DirectoryInfo dir2 = new DirectoryInfo(@"C:\Windows");
// Привязаться к несуществующему каталогу, затем создать его.
DirectoryInfo dir3 = new DirectoryInfo(@"C:\MyCode\Testing");
dir3.Create ();
// Вывести информацию о каталоге.
DirectoryInfo dir = new DirectoryInfo(@"C:\MyCode\Testing");
Console.WriteLine("***** Directory Info *****");
Console.WriteLine("FullName: {0}",          dir.FullName);
Console.WriteLine("Name: {0}",              dir.Name);
Console.WriteLine("Parent: {0}",            dir.Parent);
Console.WriteLine("Creation: {0}",          dir.CreationTime);
Console.WriteLine("Attributes: {0}",        dir.Attributes);
Console.WriteLine("Root: {0}",              dir.Root);
Console.WriteLine("******************************\n");
```

Класс DirectoryInfo

```
Found 179 *.jpg files

**********************************

File name: img13.jpg
File size: 1231580
Creation: 14.07.2009 1:57:24
Attributes: Archive
**********************************
```

```
DirectoryInfo dir = new DirectoryInfo(@"C:\Windows\Web\Wallpaper");
// Получить все файлы с расширением *.jpg.
FileInfo[] imageFiles = dir.GetFiles("*.jpg",
                                     SearchOption.AllDirectories);

Console.WriteLine("Found {0} *.jpg files\n", imageFiles.Length);
// Вывести информацию о каждом файле.
foreach (FileInfo f in imageFiles)
{  Console.WriteLine("**********************\n") ;
   Console.WriteLine("File name: {0}",      f.Name);
   Console.WriteLine("File size: {0}",      f.Length);
   Console.WriteLine("Creation: {0}",       f.CreationTime);
   Console.WriteLine("Attributes: {0}",     f.Attributes);
   Console.WriteLine("**********************\n") ;
}
```

Класс DirectoryInfo

```
DirectoryInfo dir = new DirectoryInfo(@"C:\");
// Создать \MyFolder в каталоге приложения.
dir.CreateSubdirectory("MyFolder");
// Получить возвращенный объект DirectoryInfo.
DirectoryInfo myDataFolder =
                              dir.CreateSubdirectory(@"MyFolder2\Data");
// Выводит путь к ..\MyFolder2\Data.
Console.WriteLine("New Folder is: {0}", myDataFolder);
```

```
New Folder is: C:\MyFolder2\Data
```

Класс Directory

Directory повторяют функциональность, предоставляемую членами класса DirectoryInfo.

Метод	Описание	Пример кода
CreateDirectory	Позволяет создавать все еще не существующие каталоги, указанные в пути.	`string dirPath = @"C:\NewFolder\SubFolder";` `Directory.CreateDirectory(dirPath);`

DeleteDirectory	Позволяет удалить один или несколько каталогов файловой системы.	`string dirPath = @"C:\Users\Student\" +`
		`"MyDirectory";`
		`bool deleteSubFolders = true;`
		`Directory.Delete(dirPath,deleteSubFolders);`

GetDirectories	Позволяет получить все подкаталоги по указанному пути.	`string dirPath = "...";`
		`string[] dirs =`
		`Directory.GetDirectories(dirPath);`

GetFiles	Позволяет получить все файлы по указанному пути.	`string dirPath = "...";`
		`string[] files =`
		`Directory.GetFiles(dirPath);`

Exists	Позволяет определить, существует ли каталог по указанному пути.	`string dirPath = "...";` `bool dirExists = Directory.Exists(dirPath);`

Move	Позволяет переместить каталог. Нельзя использовать метод для перемещения каталогов с разных носителей.	`string sourcePath = "...";`
		`string destPath = "...";`
		`Directory.Move(sourcePath, destPath);`

Класс DriveInfo

Класс DriveInfo позволяет получить информацию о подключенных устройствах

```
Name: A:\
Type: Removable
Root: A:\

Name: C:\
Type: Fixed
Root: C:\
Total space: 52732235776
Free space: 1216589824
Available space: 1216589824
Format: NTFS
Label:
```

```
DriveInfo[] myDrives = DriveInfo.GetDrives();
// Вывести сведения об устройствах.
foreach (DriveInfo d in myDrives)
  {
   Console.WriteLine("Name: {0}", d.Name);
   Console.WriteLine("Type: {0}", d.DriveType);
   Console.WriteLine("Root: {0}", d.RootDirectory);
// Проверить, смонтировано ли устройство
if (d.IsReady)
 {
  Console.WriteLine("Total space: {0}",      d.TotalSize);
  Console.WriteLine("Free space: {0}",       d.TotalFreeSpace);
  Console.WriteLine("Available space: {0}",  d.AvailableFreeSpace);
  Console.WriteLine("Format: {0}",           d.DriveFormat);
  Console.WriteLine("Label: {0}",            d.VolumeLabel);
}}
```

Класс FileInfo

Класс **FileInfo** позволяет получать подробные сведения о существующих файлах на жестком диске (например, время создания, размер и атрибуты) и помогает создавать, копировать, перемещать и удалять файлы.

Классы	Описание
AppendText()	Создает объект StreamWriter и добавляет текст в файл
Create ()	Создает новый файл и возвращает объект FileStream (описанный ниже) для взаимодействия с вновь созданным файлом
CreateText ()	Создает объект StreamWriter, записывающий новый текстовый файл
Directory	Получает экземпляр родительского каталога
MoveTo ()	Перемещает указанный файл в новое местоположение, предоставляя возможность указания нового имени для файла
Open()	Открывает файл с различными привилегиями чтения/записи и совместного доступа
OpenText()	Создает объект StreamReader и читает из существующего текстового файла

Основные методы класса FileInfo

CreationTime (свойство)	Позволяет получить или установить время создания для конкретного файла.	`string filePath = "...";`
		`FileInfo file = new FileInfo(filePath);`
		`file.CreationTime = DateTime.Now;`
		`...`
		`DateTime time = file.CreationTime;`

CopyTo (метод)	Позволяет копировать файл в новое место в файловой системе.	`string filePath = "...";`
		`FileInfo file = new FileInfo(filePath);`
		`string destPath = "...";`
		`file.CopyTo(destPath);`

Delete (свойство)	Позволяет удалить файл.	`string filePath = "...";`
		`FileInfo file = new FileInfo(filePath);`
		`file.Delete();`

DirectoryName (свойство)	Позволяет получить путь к каталогу файла.	`string filePath = "...";`
		`FileInfo file = new FileInfo(filePath);`
		`string dirPath = file.DirectoryName;`

Exists (свойство)	Позволяет определить, существует ли заданный файл.	`string filePath = "...";`
		`FileInfo file = new FileInfo(filePath);`
		`bool exists = file.Exists;`

Extension (свойство)	Позволяет получить расширение файла.	`string filePath = "...";`
		`FileInfo file = new FileInfo(filePath);`
		`string ext = file.Extension;`

Length (свойство)	Позволяет получить длину файла в байтах.	`string filePath = "...";`
		`FileInfo file = new FileInfo(filePath);`
		`long length = file.Length;`

Name (свойство)	Позволяет получить имя файла.	`string filePath = "...";`
		`FileInfo file = new FileInfo(filePath);`
		`string name = file.Name;`

Open (метод)	Позволяет открыть файл файловой системы Windows. Метод возвращает объект FileStream, который позволяет взаимодействовать с файлом с помощью потоковой модели.	`string filePath = "...";`
		`FileInfo file = new FileInfo(filePath);`
		`FileStream stream =`
		`file.Open(FileMode.OpenOrCreate);`

Примеры использования класса FileInfo

```
// Определение контекста using для файлового ввода-вывода.
FileInfo file2 = new FileInfo(@"C:\Test.dat");
using (FileStream fs2 = file2.Create())
{
   // Использовать объект FileStream...
}
```

```
// Создать новый файл через FileInfo.Open().
FileInfo file3 = new FileInfo(@"C:\Test2.dat");
using(FileStream fs3 = file3.Open(FileMode.OpenOrCreate,
                                  FileAccess.ReadWrite, FileShare.None))
{
   // Использовать объект FileStream...
}
```

```
// Получить объект StreamReader.
FileInfo file6 = new FileInfo(@"C:\boot.ini");
using(StreamReader sreader = file6.OpenText())
{
   // Использовать объект StreamReader...
}
```

Класс File

Тип File предоставляет функциональность, почти идентичную типу FileInfo, с помощью нескольких статических методов.

Метод	Описание	Пример кода

AppendAllText	Позволяет открыть существующий файл, добавить текст в этот файл, а затем закрыть файл, выполнив все действия в одну операцию.	`string filePath = "...";`
		`string fileContents = "...";`
		`File.AppendAllText(filePath, fileContents);`

Copy	Позволяет скопировать существующий файл в новое место.	`string sourceFile = "...";`
		`string destFile = "...";`
		`bool overwrite = false;`
		`File.Copy(sourceFile,destFile, overwrite);`

Create	Позволяет создать новый файл файловой системы Windows. Метод возвращает объект FileStream, который позволяет взаимодействовать с файлом с помощью потоковой модели.	`string filePath = "...";`
		`int bufferSize = 128;`
		`FileStream file =`
		`File.Create(filePath,bufferSize, FileOptions.None);`

Delete	Позволяет удалить файл из файловой системы Windows.	`string filePath = "...";` `File.Delete(filePath);`
Exists	Позволяет определить, существует ли указанный файл.	`string filePath = "...";` `bool exists = File.Exists(filePath);`

Класс File. Дополнительные члены.

Метод	Описание
`ReadAllBytes()`	Открывает указанный файл, возвращает двоичные данные в виде массива байт и затем закрывает файл
`ReadAllLines()`	Открывает указанный файл, возвращает символьные данные в виде массива строк, затем закрывает файл
`ReadAllText()`	Открывает указанный файл, возвращает символьные данные в виде `System.String()`, затем закрывает файл
`WriteAllBytes()`	Открывает указанный файл, записывает в него массив байтов и закрывает файл
`WriteAllLines()`	Открывает указанный файл, записывает в него массив строк и закрывает файл
`WriteAllText()`	Открывает указанный файл, записывает в него данные из указанной строки и закрывает файл

Класс File. Примеры.

```
string[] myTasks = {  "Выучить C#", "Выучить ADO.NET",
                      "Выучить WPF", "Выучить ASP.NET"};
// Записать все данные в файл на диске C: .
File.WriteAllLines(@"tasks.txt", myTasks);

// Прочитать все данные и вывести на консоль.
foreach (string task in File.ReadAllLines(@"tasks.txt"))
  {
     Console.WriteLine("Выполнено: {0}", task);
  }
```

```
 // Получить объект FileStream череэ File.Create().
using(FileStream fs = File.Create(@"C:\Test.dat"))                {}
// Получить объект FileStream с правами только для чтения.
using(FileStream readOnlyStream = File.OpenRead(@"Test3.dat"))    {}
// Получить объект FileStream с правами только для записи
using(FileStream writeOnlyStream = File.OpenWrite(@"Test4.dat")) {}
// Получить объект StreamReader.
using(StreamReader sreader = File.OpenText(@"C:\boot.ini"))       {}
```

Класс Path

предоставляет статические методы, которые упрощают выполнение операций с путевыми именами

```
string dir = @"c:\mydir";
string file = "myfile.txt";
string path = @"c:\mydir\myfile.txt";

Console.WriteLine(Path.GetPathRoot (path));      // главный каталог
Console.WriteLine(Path.GetDirectoryName (path)); // каталог
Console.WriteLine(Path.GetFileName (path));      // имя файла
Console.WriteLine(Path.GetFullPath (file));      // полный путь
Console.WriteLine(Path.Combine(dir, file));      // соединить путь и имя
// Работа с расширениями
Console.WriteLine(Path.HasExtension (file));                  // True
Console.WriteLine(Path.GetExtension (file));                  // .txt
Console.WriteLine(Path.GetFileNameWithoutExtension (file)); // myfile
Console.WriteLine(Path.ChangeExtension (file, ".log"));    // myfile.log
//Временные файлы
Console.WriteLine(Path.GetTempPath());      // каталог временных файлов
Console.WriteLine(Path.GetRandomFileName());  // случайное имя файла
```

Работа со специальными каталогами

```
Environment.GetFolderPath(Environment.SpecialFolder.History);
Environment.GetFolderPath(Environment.SpecialFolder.ApplicationData);
Environment.GetFolderPath(Environment.SpecialFolder.CommonPictures);
Environment.GetFolderPath(Environment.SpecialFolder.ProgramFilesX86);
```

AdminTools	CommonVideos	Personal
ApplicationData	Cookies	PrinterShortcuts
CDBurning	Desktop	ProgramFiles
CommonAdminTools	DesktopDirectory	ProgramFilesX86
CommonApplicationData	Favorites	Programs
CommonDesktopDirectory	Fonts	Recent
CommonDocuments	History	Resources
CommonMusic	InternetCache	SendTo
CommonOemLinks	LocalApplicationData	StartMenu
CommonPictures	LocalizedResources	Startup
CommonProgramFiles	MyComputer	System
CommonProgramFilesX86	MyDocuments	SystemX86
CommonPrograms	MyMusic	Templates
CommonStartMenu	MyPictures	UserProfile
CommonStartup	MyVideos	Windows
CommonTemplates	NetworkShortcuts	