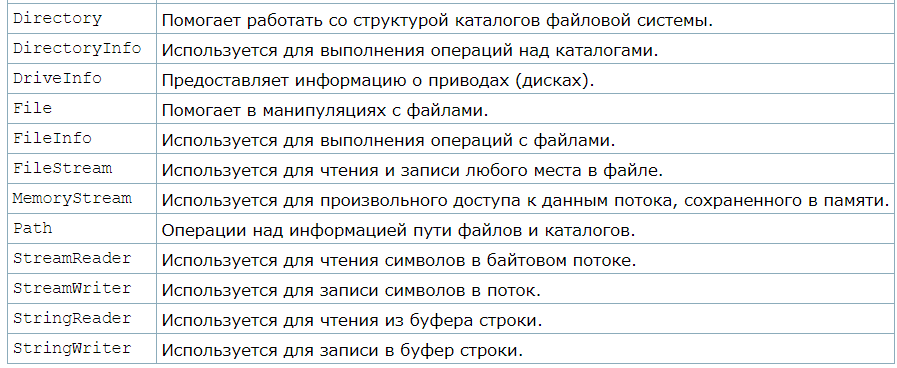
1. Какие классы содержаться с *System*.IO?



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

DirectoryInfo - это класс, представляющий каталог. Он предоставляет методы для создания, перемещения и перечисления каталогов и подкаталогов. Этот класс не позволяет иметь подклассы.

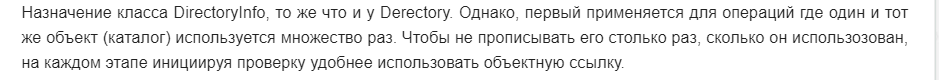
Разница между 2 классами Directory и DirectoryInfo заключается в том, что Directory является утилитарным классом, все его методы являются статическими, а DirectoryInfo представляет конкретный каталог.

Поскольку **DirectoryInfo** – ссылочный тип(класс), прежде чем воспользоваться его членами нужно создать его объект с помощью конструктора

*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.



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

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.

**FileInfo**

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

Для создания объекта FileInfo применяется конструктор, который получает в качестве параметра путь к файлу.

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

Для работы с текстовыми файлами применяются два класса StreamReader и StreamWriter. По названию классов видно, что первый используется для работы с файлами в режиме чтения, второй для работы в режиме записи. Они интуитивно понятны, так как все их методы явно отражают действия.

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

Для чтения и записи двоичных значений типов данных, которые встроены в C#, служат классы потоков "BinaryReader" и "BinaryWriter". Когда вы используете данные потоки, помните, что данные считываются и записываются во внутреннем двоичном формате, а не в удобной для чтения текстовой форме.

6. Как можно сжимать и восстанавливать файлы?

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

7. Расскажите алгоритм сжатия *GZip*.

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)

Для управления сжатием/восстанавлением данных GZipStream предоставляет ряд методов. Основые из них:

* 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.

8. Для чего служит класс *Path*?

Выполняет операции для экземпляров класса String, содержащих сведения о пути к файлу или каталогу. Эти операции выполняются межплатформенным способом.

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

Текстовые – в текстовом виде, бинарные – в бинарном.

==в.5 && в.4

+Класс **FileStream** представляет возможности по считыванию из файла и записи в файл. Он позволяет работать как с текстовыми файлами, так и с бинарными.

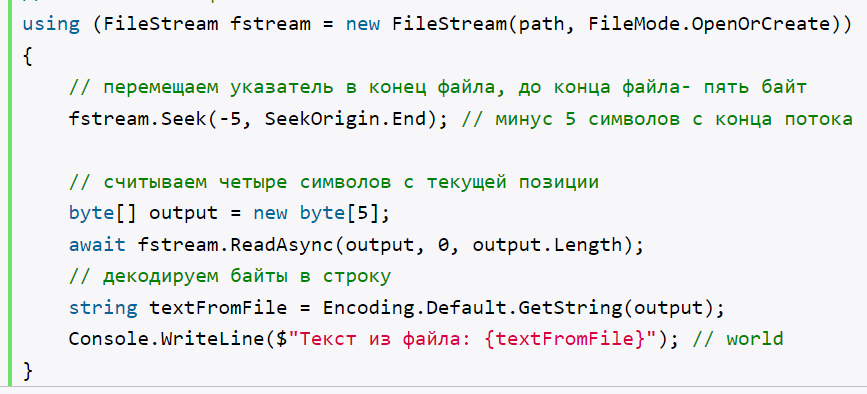
10. Что такое произвольный доступ к файлу? Приведите пример.

Нередко бинарные файлы представляют определенную структуру. И, зная эту структуру, мы можем взять из файла нужную порцию информации или наоброт записать в определенном месте файла определенный набор байтов. Например, в wav-файлах непосредственно звуковые данные начинаются с 44 байта, а до 44 байта идут различные метаданные - количество каналов аудио, частота дискретизации и т.д.

С помощью метода **Seek()** мы 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

Курсор потока, с которого начинается чтение или запись, смещается вперед на значение offset относительно позиции, указанной в качестве второго параметра. Смещение может быть отрицательным, тогда курсор сдвигается назад, если положительное - то вперед.



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

Оператор C# - using определяет границу объекта, за пределами которой объект автоматически уничтожается. Оператор using в C# завершается, когда заканчивается блок операторов using или когда выполнение выходит из блока операторов using косвенно, например, возникает исключение.

Оператор using позволяет указать несколько ресурсов в одном операторе.

По большому счёту, работает аналогично операции try-catch.