1. Какие классы содержатся в System.IO?

Пространство имен System.IO включает классы для работы с вводом/выводом данных, файловыми потоками, чтением и записью файлов и данных, а также взаимодействием с файловой системой. Некоторые основные классы:

- File, FileInfo – для работы с файлами (чтение, запись, удаление, получение информации).

- Directory, DirectoryInfo – для работы с директориями.

- StreamReader, StreamWriter – для чтения и записи текстовых данных.

- BinaryReader, BinaryWriter – для работы с бинарными данными.

- Path – для работы с путями файлов и директорий.

- FileStream, MemoryStream – для работы с потоками данных.

- DriveInfo – информация о дисках.

- GZipStream – для сжатия и распаковки данных.

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

Оба класса предназначены для работы с директориями, но разница в их использовании:

- Directory – статический класс, предоставляющий методы для создания, удаления, перемещения и получения информации о директориях. Его методы не требуют создания объекта класса.

- DirectoryInfo – класс, который требует создания экземпляра для работы с конкретной директорией. Предоставляет более эффективные операции для многократного обращения к одной и той же директории.

// Directory

Directory.CreateDirectory("NewFolder");

// DirectoryInfo

DirectoryInfo dirInfo = new DirectoryInfo("NewFolder");

dirInfo.Create();

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

Оба класса используются для работы с файлами, но они различаются так же, как и Directory/DirectoryInfo:

- File – статический класс, предоставляющий методы для создания, копирования, перемещения, удаления файлов и проверки их существования.

- FileInfo – экземплярный класс, позволяющий получать информацию о конкретном файле и выполнять операции с ним.

Методы:

- File.Exists, File.Create, File.Delete, File.Copy, File.Move, File.ReadAllText, File.WriteAllText, File.AppendAllText.

- В FileInfo: CopyTo, Delete, MoveTo, Open, AppendText, Create, OpenRead, OpenWrite.

4. Для чего используются классы StreamReader и StreamWriter?

- StreamReader – класс для чтения текстовых данных из потока (например, файла). Используется для работы с текстовыми файлами.

- StreamWriter – класс для записи текстовых данных в поток (например, файл). Также используется для работы с текстовыми файлами.

// Запись в файл

using (StreamWriter writer = new StreamWriter("file.txt"))

{ writer.WriteLine("Пример текста");}

// Чтение из файла

using (StreamReader reader = new StreamReader("file.txt"))

{ string content = reader.ReadToEnd();

Console.WriteLine(content);}

5. Для чего используются классы BinaryWriter и BinaryReader?

Эти классы используются для записи и чтения бинарных данных (не текстовых) в потоки, например, файлы.

- BinaryWriter – записывает данные в бинарном формате.

- BinaryReader – читает бинарные данные.

// Запись бинарных данных

using (BinaryWriter writer = new BinaryWriter(File.Open("file.bin", FileMode.Create)))

{ writer.Write(1234); // Запись числа}

// Чтение бинарных данных

using (BinaryReader reader = new BinaryReader(File.Open("file.bin", FileMode.Open)))

{ int number = reader.ReadInt32();

Console.WriteLine(number);}

6. Как можно сжимать и восстанавливать файлы?

Для сжатия и восстановления файлов можно использовать классы из пространства имен System.IO.Compression, такие как GZipStream и DeflateStream.

Пример сжатия и восстановления с помощью GZipStream:

// Сжатие файла

using (FileStream originalFileStream = new FileStream("file.txt", FileMode.OpenOrCreate))

using (FileStream compressedFileStream = File.Create("file.gz"))

using (GZipStream compressionStream = new GZipStream(compressedFileStream, CompressionMode.Compress))

{ originalFileStream.CopyTo(compressionStream);}

// Восстановление файла

using (FileStream compressedFileStream = new FileStream("file.gz", FileMode.Open))

using (FileStream decompressedFileStream = File.Create("restored.txt"))

using (GZipStream decompressionStream = new GZipStream(compressedFileStream, CompressionMode.Decompress))

{ decompressedFileStream.CopyTo(decompressionStream);}

7. Алгоритм сжатия GZip

GZip использует комбинацию алгоритма сжатия Deflate и алгоритма сжатия Huffman. Основные шаги алгоритма:

1. Применяется алгоритм Deflate для удаления избыточных данных с использованием методов, таких как поиск повторяющихся последовательностей и замена их короткими ссылками.

2. Используется кодирование Хаффмана для дальнейшего сокращения частотных данных, сжимающихся по принципу "чем чаще элемент встречается, тем короче его код".

8. Для чего служит класс Path?

Класс Path предоставляет методы для работы с путями к файлам и директориям. Он не взаимодействует с файловой системой, а работает только с текстовыми строками, представляющими путь.

Пример методов:

- Path.Combine – объединяет несколько частей пути.

- Path.GetExtension – возвращает расширение файла.

- Path.GetFileName – возвращает имя файла из полного пути.

- Path.GetDirectoryName – возвращает путь к директории.

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

- Текстовый файл содержит данные в виде читаемых символов (например, .txt, .csv).

- Бинарный файл содержит данные в бинарном формате, который не предназначен для чтения человеком (например, .exe, .jpg).

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

Произвольный доступ (random access) позволяет работать с произвольными частями файла, читать или записывать данные не последовательно, а с любой позиции.

Для этого часто используется класс FileStream.

using (FileStream fs = new FileStream("file.bin", FileMode.Open))

{ fs.Seek(5, SeekOrigin.Begin); // Переход к 5 байту от начала файла

int data = fs.ReadByte(); // Чтение байта с этой позиции}

11. Как применяется конструкция using при работе с файловыми потоками? Для чего ее используют?

Конструкция using используется для автоматического освобождения ресурсов (закрытия файлов, освобождения памяти и т.д.), как только работа с объектом завершена. Она часто применяется при работе с файловыми потоками, чтобы гарантировать, что поток будет закрыт даже в случае исключений.

using (StreamWriter writer = new StreamWriter("file.txt"))

{ writer.WriteLine("Пример текста");} // После выхода из блока using, ресурс writer будет освобожден автоматически