1. \*\*Классы, содержащиеся в `System.IO`:\*\*

В пространстве имен `System.IO` содержится множество классов для работы с файловой системой и потоками. К основным классам относятся:

- `File`, `FileInfo` — для работы с файлами.

- `Directory`, `DirectoryInfo` — для работы с каталогами.

- `Stream`, `FileStream`, `MemoryStream` — для работы с потоками данных.

- `StreamReader`, `StreamWriter` — для чтения и записи текстовых файлов.

- `BinaryReader`, `BinaryWriter` — для чтения и записи бинарных файлов.

- `Path` — для работы с путями к файлам и папкам.

- `DriveInfo`, `FileSystemWatcher` и другие для работы с дисками и отслеживания изменений в файловой системе.

2. \*\*Directory и DirectoryInfo:\*\*

- \*\*`Directory`\*\* — это статический класс, который предоставляет методы для создания, перемещения и удаления каталогов.

- \*\*`DirectoryInfo`\*\* — экземплярный класс, который предоставляет доступ к свойствам и методам для работы с конкретными каталогами.

- \*\*Отличие\*\*: `Directory` используется без создания экземпляра и содержит только статические методы. `DirectoryInfo` требует создания объекта и предоставляет больше возможностей для получения информации о каталоге.

3. \*\*File и FileInfo:\*\*

- \*\*`File`\*\* — статический класс для создания, копирования, удаления и перемещения файлов.

- \*\*`FileInfo`\*\* — класс, который предоставляет свойства и методы для работы с конкретным файлом.

- \*\*Методы\*\*:

- `File.Create()`, `File.Delete()`, `File.Exists()` и др.

- У `FileInfo` — `Create()`, `Delete()`, `CopyTo()`, `MoveTo()` и свойства типа `Length`, `Directory`.

4. \*\*StreamReader и StreamWriter:\*\*

- Используются для чтения и записи текстовых данных в файлы.

- \*\*`StreamReader`\*\* — чтение текстовых данных построчно или целиком.

- \*\*`StreamWriter`\*\* — запись текста в файл.

- Эти классы упрощают работу с текстовыми потоками и обеспечивают удобное управление кодировкой.

5. \*\*BinaryWriter и BinaryReader:\*\*

- Эти классы используются для записи и чтения данных в бинарном формате.

- \*\*`BinaryWriter`\*\* — запись данных примитивных типов (например, `int`, `float`, `string`) в поток в бинарной форме.

- \*\*`BinaryReader`\*\* — чтение данных примитивных типов из бинарного потока.

- Оба класса полезны для работы с файлами, где требуется работа с нестандартным форматом данных.

6. \*\*Сжатие и восстановление файлов:\*\*

- Для сжатия и восстановления файлов в .NET можно использовать классы из `System.IO.Compression`, такие как `GZipStream` и `ZipArchive`.

- Пример использования `GZipStream` позволяет сжимать данные в поток или восстанавливать их.

7. \*\*Алгоритм сжатия GZip:\*\*

- GZip использует алгоритм сжатия DEFLATE, который комбинирует LZ77 и метод кодирования Хаффмана.

- Он работает, находя повторяющиеся последовательности байтов и заменяя их на более короткие ссылки, снижая общий объем данных.

- `GZipStream` в .NET предоставляет интерфейс для работы с этим алгоритмом.

8. \*\*Класс Path:\*\*

- Класс `Path` используется для работы с именами файлов и путями.

- Он предоставляет методы для работы с различными частями пути, такими как `GetExtension()`, `GetFileName()`, `Combine()`, и т. д.

- Этот класс помогает избежать ошибок при манипуляции строками, представляющими пути.

9. \*\*Тестовые и бинарные файлы:\*\*

- \*\*Тестовый файл\*\* содержит данные в текстовом формате (например, `.txt`), которые легко читаются человеком.

- \*\*Бинарный файл\*\* содержит данные в машинном формате (например, `.bin`), которые могут включать структуры данных и не читаются человеком.

- \*\*Чтение и запись\*\*: Для текстовых файлов используются `StreamReader` и `StreamWriter`, а для бинарных — `BinaryReader` и `BinaryWriter`.

10. \*\*Произвольный доступ к файлу:\*\*

- Произвольный доступ позволяет читать или записывать данные в произвольное место в файле без последовательного обхода.

- Пример: использование класса `FileStream` с установкой позиции с помощью `Seek()`, чтобы перейти к определенной части файла и записать/прочитать данные.

11. \*\*Конструкция `using` при работе с файловыми потоками:\*\*

- Используется для автоматического управления ресурсами.

- Пример:

```csharp

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

{

string content = reader.ReadToEnd();

} // Здесь `reader` автоматически закроется и освободит ресурсы.

```

- Конструкция гарантирует вызов метода `Dispose()`, что освобождает занятые ресурсы и предотвращает утечки памяти.