1. **Какие классы содержатся в System.IO?** В **System.IO** содержатся классы, предназначенные для работы с вводом/выводом данных, файлами и директориями. Некоторые из них:
   * **File**: для работы с файлами.
   * **Directory**: для работы с директориями.
   * **Path**: для работы с путями к файлам и директориям.
   * **FileInfo** и **DirectoryInfo**: предоставляют дополнительную информацию о файлах и директориях.
   * **StreamReader** и **StreamWriter**: для чтения и записи символьных данных из/в файл.
   * **BinaryReader** и **BinaryWriter**: для чтения и записи бинарных данных из/в файл.
2. **Для чего используются классы Directory и DirectoryInfo? В чем отличие?** Класс **Directory** предоставляет статические методы для работы с директориями, в то время как **DirectoryInfo** представляет конкретную директорию и предоставляет экземплярные методы для работы с этой директорией. Например, **Directory.CreateDirectory(path)** создает директорию, а **new DirectoryInfo(path)** создает объект **DirectoryInfo** для указанной директории, позволяя вам выполнять операции над этой директорией.
3. **Для чего используются классы File и FileInfo? Какие методы они содержат?** **File** и **FileInfo** предназначены для работы с файлами. **File** содержит статические методы для операций над файлами, в то время как **FileInfo** предоставляет объект, представляющий файл, и содержит экземплярные методы. Например, **File.Exists(path)** проверяет существование файла, а **FileInfo.Length** возвращает размер файла.
4. **Для чего используются классы StreamReader и StreamWriter?** **StreamReader** используется для чтения символьных данных из потока, а **StreamWriter** - для записи символьных данных в поток. Они обертывают другие потоки (**FileStream**, **MemoryStream**, и т.д.) и предоставляют удобные методы для работы с текстовыми данными.
5. **Для чего используются классы BinaryWriter и BinaryReader?** **BinaryWriter** и **BinaryReader** предназначены для записи и чтения бинарных данных соответственно. Они обеспечивают методы для записи и чтения примитивных типов данных в бинарном формате.
6. **Как можно сжимать и восстанавливать файлы?** Для сжатия и восстановления файлов можно использовать классы **GZipStream** вместе с классами потоков (**FileStream**, **MemoryStream**). **GZipStream** предоставляет средства для сжатия и распаковки данных.
7. **Расскажите алгоритм сжатия GZip.** GZip использует алгоритм сжатия DEFLATE, который включает в себя два основных этапа: сжатие (путем удаления повторяющихся блоков данных) и кодирование (с использованием кодов Хаффмана). GZip также добавляет заголовок и контрольную сумму.
8. **Для чего служит класс Path?** **Path** предоставляет методы для работы с путями к файлам и директориям. Он позволяет объединять и разбирать пути, получать расширения файлов и другие операции, связанные с манипуляциями путями.
9. **Что такое текстовый и бинарный файл. Как можно выполнить чтение и запись в эти файлы?** Текстовый файл содержит читаемые символы, в то время как бинарный файл содержит данные в бинарном формате. Для чтения и записи в текстовый файл используются классы **StreamReader** и **StreamWriter**, а для бинарного файла - **BinaryReader** и **BinaryWriter**.
10. **Что такое произвольный доступ к файлу? Приведите пример.** Произвольный доступ (или случайный доступ) к файлу позволяет читать и записывать данные в произвольные места в файле. Это достигается через использование класса **FileStream**. Например:

csharpCopy code

using (FileStream fs = new FileStream("example.bin", FileMode.Open, FileAccess.ReadWrite)) { // Перейти к 100-му байту в файле fs.Seek(100, SeekOrigin.Begin); // Прочитать 10 байт byte[] buffer = new byte[10]; fs.Read(buffer, 0, 10); // Записать данные обратно // ... }

1. **Как применяется конструкция using (не директива) при работе с файловыми потоками? Для чего ее используют.** Конструкция **using** применяется для гарантированного освобождения ресурсов после завершения блока кода. Например, при работе с файловыми потоками, это может использоваться для автоматического закрытия файлового потока после завершения операций с файлом, что важно для правильного управления ресурсами и предотвращения утечек памяти. Пример:

csharpCopy code

using (FileStream fs = new FileStream("example.txt", FileMode.Open)) { // Работа с файловым потоком // ... } // После завершения блока using, файловый поток будет автоматически закрыт.

Начало формы