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

Пространство имен \*\*`System.IO`\*\* предоставляет классы для работы с файлами, папками и потоками данных. Основные классы:

- \*\*`File`\*\*: Статические методы для работы с файлами.

- \*\*`FileInfo`\*\*: Методы для работы с файлами через экземпляры.

- \*\*`Directory`\*\*: Статические методы для работы с директориями.

- \*\*`DirectoryInfo`\*\*: Методы для работы с директориями через экземпляры.

- \*\*`Stream`\*\*: Базовый класс для потоков.

- \*\*`FileStream`\*\*: Поток для работы с файлами.

- \*\*`MemoryStream`\*\*: Поток для работы с данными в памяти.

- \*\*`BufferedStream`\*\*: Буферизированный поток.

- \*\*`StreamReader`\*\*: Чтение текстовых данных.

- \*\*`StreamWriter`\*\*: Запись текстовых данных.

- \*\*`BinaryReader`\*\* и \*\*`BinaryWriter`\*\*: Чтение и запись бинарных данных.

- \*\*`Path`\*\*: Утилиты для работы с путями файлов и директорий.

- \*\*`DriveInfo`\*\*: Информация о дисках.

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

**Directory**:

 Directory предоставляет только статические методы.

 DirectoryInfo предоставляет методы и свойства для получения информации о самой директории.

 Directory.CreateDirectory()

 Directory.Delete()

 Directory.Exists()

 Directory.GetFiles()

Console.WriteLine($"имя директории: {dirInfo.Name}");

Console.WriteLine($"дата создания: {dirInfo.CreationTime}");

Console.WriteLine($"количество файлов: {dirInfo.GetFiles().Length}");

Console.WriteLine($"количество поддиректорий: {dirInfo.GetDirectories().Length}"); /

* Это статический класс.
* Используется для выполнения операций с каталогами, таких как создание, удаление, перемещение или получение информации о каталогах.

*DirectoryInfo*

Содержит аналогичные методы, но работает через экземпляры.

- Удобен для хранения информации о конкретной директории.

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

File предоставляет статические методы для работы с файлами.

FileInfo предоставляет методы и свойства, позволяющие управлять конкретным файлом и получать его информацию.

#### \*\*`File`\*\*:

- Статический класс для работы с файлами.

- Основные методы:

- \*\*`Create()`\*\*: Создать файл.

- \*\*`Delete()`\*\*: Удалить файл.

- \*\*`Copy()`\*\*: Скопировать файл.

- \*\*`Move()`\*\*: Переместить файл.

- \*\*`ReadAllText()`\*\*, \*\*`WriteAllText()`\*\*: Читать/писать весь текст.

- \*\*`AppendText()`\*\*: Добавить текст в файл.

#### \*\*`FileInfo`\*\*:

- Класс для работы с файлами через экземпляры.

- Основные методы:

- \*\*`Create()`\*\*, \*\*`Delete()`\*\*, \*\*`CopyTo()`\*\*, \*\*`MoveTo()`\*\*: Аналогичные методам `File`.

- \*\*`Length`\*\*: Получить размер файла.

- \*\*`Exists`\*\*: Проверить наличие файла.

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

`StreamReader`\*\*: Используется для чтения текстовых данных из потока.

- Методы:

- \*\*`ReadLine()`\*\*: Прочитать строку.

- \*\*`ReadToEnd()`\*\*: Прочитать весь текст.

- \*\*`Close()`\*\*: Закрыть поток.

- \*\*`StreamWriter`\*\*: Используется для записи текстовых данных в поток.

- Методы:

- \*\*`Write()`\*\*, \*\*`WriteLine()`\*\*: Записать строку.

- \*\*`Close()`\*\*: Закрыть поток.

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

\*`BinaryWriter`\*\*:

- Для записи данных в бинарном формате.

- Методы:

- \*\*`Write()`\*\*: Записывает значения (int, double, string и т.д.).

- \*\*`Close()`\*\*: Закрывает поток.

- \*\*`BinaryReader`\*\*:

- Для чтения данных в бинарном формате.

- Методы:

- \*\*`ReadInt32()`\*\*, \*\*`ReadDouble()`\*\*, \*\*`ReadString()`\*\*: Чтение данных разных типов.

- \*\*`Close()`\*\*: Закрывает поток.

1. Как можно сжимать и восстанавливать файлы?

Для сжатия и восстановления файлов в .NET используются классы из

ZipFile.CreateFromDirectory(filesDir, archivePath);

string extractPath = Path.Combine(sourceDir, "SMDExtract");

ZipFile.ExtractToDirectory(archivePath, extractPath);\*\*`System.IO.Compression`\*\*, например:

- \*\*`GZipStream`\*\*: Для сжатия и распаковки данных.

Основные классы для работы с сжатием файлов — это GZipStream

1. Расскажите алгоритм сжатия *GZip*.

\*\*GZip\*\* — это формат сжатия, основанный на алгоритме DEFLATE.

Шаги алгоритма:

1. \*\*Разделение данных\*\*: Разбивка данных на блоки.

2. \*\*Сжатие блоков\*\*: Используется комбинация методов LZ77 и кодирования Хаффмана.

3. \*\*Добавление метаданных\*\*: Содержит CRC32 (контрольная сумма) и размер исходных данных.

1. Для чего служит класс *Path*?

Класс **Path** предоставляет методы для работы с путями файлов и каталогов. Он помогает избежать ошибок, связанных с неправильными путями.

\*\*`GetDirectoryName()`\*\*: Возвращает путь к каталогу.

- \*\*`GetFileName()`\*\*: Возвращает имя файла.

- \*\*`GetExtension()`\*\*: Возвращает расширение файла.

- \*\*`Combine()`\*\*: Соединяет пути.

- \*\*`ChangeExtension()`\*\*: Изменяет расширение.

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

- \*\*Текстовый файл\*\*: Хранит данные в виде текста (ASCII, UTF-8).

- Чтение/запись: `StreamReader`, `StreamWriter`.

- \*\*Бинарный файл\*\*: Хранит данные в двоичном формате.

- Чтение/запись: `BinaryReader`, `BinaryWriter`.

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

Произвольный доступ к файлу означает возможность читать и записывать данные в произвольные позиции файла, не следуя последовательности.

**Класс FileStream** поддерживает произвольный доступ:

* Чтение/запись данных с использованием свойства Position.

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

Конструкция **using** используется для автоматического управления ресурсами. Она гарантирует, что объект будет корректно освобожден после завершения работы, даже если произойдет исключение.

\*\*Конструкция `using`\*\* автоматически освобождает ресурсы (вызов `Dispose()`) после завершения работы.

Метод не записывает текст напрямую, а создаёт поток, через который вы можете писать данные в файл.

using System;

using System.Collections.Generic;

using System.IO;

using System.IO.Compression;

using System.Linq;

namespace laba12

{

public class SMDLog

{

private static string logFile = "smdlogfile.txt"; //содержащее путь и имя файла, куда будут записываться логи.Поскольку logFile используется всеми методами класса и не меняется для разных экземпляров, оно сделано статическим.

public static void WriteLog(string action, string details) //метод записывает новую строку в лог-файл. описание действия, которое произошло, дополнительные детали события.

{

try

{

string logEntry = $"{DateTime.Now}: {action} - {details}"; //добавляет текущую дату и время.

//action и details включают описание действия и его детали.

File.AppendAllText(logFile, logEntry + Environment.NewLine);//File.AppendAllText находится в пространстве имён System.IO,Добавляет указанную строку к файлу, создавая файл, если он еще не существует.Environment. NewLine — это независимый от платформы и независимый от языка способ представления разрыва строки.

}

catch (Exception ex)

{

Console.WriteLine($"ошибка записи в log: {ex.Message}");

}

}

public static void ReadLog() //предназначен для чтения содержимого лог-файла (logFile) и вывода его строк в консоль.

{ //Метод принадлежит классу, а не объекту, поэтому его можно вызывать напрямую через имя класса, например:

//SMDLog.ReadLog();

try

{

string[] logs = File.ReadAllLines(logFile);//Читает все строки из файла, указанного в переменной logFile. Возвращает массив строк (string[]), где каждая строка файла становится отдельным элементом массива.

foreach (var log in logs) // Цикл перебирает каждую строку из массива logs

Console.WriteLine(log);

}

catch (Exception ex)

{

Console.WriteLine($"ошибка чтения log: {ex.Message}");

}

}

public static void SearchLog(string keyword) //т метод предназначен для поиска записей в лог-файле

{

try

{

var logs = File.ReadAllLines(logFile) // Чтение всех строк из файла

.Where(log => log.Contains(keyword)); //Он проверяет каждую строку в массиве из файла и выбирает только те строки, которые содержат ключевое слово.условие, определённое с помощью лямбда-выражения

foreach (var log in logs) //Цикл перебирает каждую строку из отфильтрованной коллекции logs.

Console.WriteLine(log); //Выводит каждую строку, содержащую ключевое слово, в консоль.

}

catch (Exception ex)

{

Console.WriteLine($"ошибка поиска в log: {ex.Message}");

}

}

}

public class SMDDiskInfo //класс SMDDiskInfo, который содержит метод для работы с информацией о дисках.

{

public static void GetDiskInfo()

{

foreach (var drive in DriveInfo.GetDrives())//возвращает список всех доступных логических дисков на компьютере. DriveInfo используется для получения информации о логических дисках, подключённых к компьютеру.

{

if (drive.IsReady) //возвращает true, если диск доступен.

{

Console.WriteLine($"имя диска: {drive.Name}");

Console.WriteLine($"тип диска: {drive.DriveType}");

Console.WriteLine($"общий размер: {drive.TotalSize} байт");

Console.WriteLine($"свободное место: {drive.TotalFreeSpace} байт");

Console.WriteLine($"метка тома: {drive.VolumeLabel}");

Console.WriteLine(new string('-', 40));

}

}

}

}

public class SMDFileInfo

{

public static void GetFileInfo(string filePath) //используется для получения информации о файле, такой как его имя, размер, расширение, дата создания и дата последнего изменения.

{

try

{

var fileInfo = new FileInfo(filePath); //Класс FileInfo из пространства имён System.IO предоставляет методы и свойства для работы с файлами.Этот объект будет использоваться для доступа к метаданным файла, таким как его размер, имя, расширение и другие свойства.

if (!fileInfo.Exists) //это свойство, которое проверяет, существует ли файл по указанному пути.

{

Console.WriteLine("файл не существует");

return;

}

Console.WriteLine($"полный путь: {fileInfo.FullName}");

Console.WriteLine($"размер: {fileInfo.Length} байт");

Console.WriteLine($"расширение: {fileInfo.Extension}");

Console.WriteLine($"имя: {fileInfo.Name}");

Console.WriteLine($"дата создания: {fileInfo.CreationTime}"); //возвращает дату и время, когда файл был создан.

Console.WriteLine($"последнее изменение: {fileInfo.LastWriteTime}");

}

catch (Exception ex)

{

Console.WriteLine($"ошибка получения информации о файле: {ex.Message}");

}

}

}

public class SMDDirInfo

{

public static void GetDirInfo(string dirPath) //Метод предназначен для получения и вывода информации о директории (папке) на основе переданного пути.

{

try

{

var dirInfo = new DirectoryInfo(dirPath); //это класс из пространства имён System.IO, который предоставляет информацию о директории и позволяет работать с ней.

if (!dirInfo.Exists) //то свойство, которое возвращает true, если указанная директория существует, и false, если она не существует.

{

Console.WriteLine("директория не существует");

return;

}

Console.WriteLine($"имя директории: {dirInfo.Name}"); //возвращает имя директории (без пути).

Console.WriteLine($"дата создания: {dirInfo.CreationTime}");// возвращает дату и время создания директории.

Console.WriteLine($"количество файлов: {dirInfo.GetFiles().Length}"); //GetFiles() — это метод, который возвращает массив файлов, находящихся в директории. .Length — это свойство, которое возвращает количество файлов в массиве.

Console.WriteLine($"количество поддиректорий: {dirInfo.GetDirectories().Length}"); //

Console.WriteLine("родительские директории: ");

var parent = dirInfo.Parent; //это свойство, которое возвращает объект DirectoryInfo для родительской директории текущей

while (parent != null)

{

Console.WriteLine($"- {parent.Name}");

parent = parent.Parent;

}

}

catch (Exception ex)

{

Console.WriteLine($"ошибка получения информации о директории: {ex.Message}");

}

}

}

public class SMDFileManager

{

public static void ManageFiles(string sourceDir, string fileExtension)//sourceDir — путь к исходной директории, с которой будут работать файлы.fileExtension — расширение файлов, с которыми необходимо работать

{

try

{

string inspectDir = Path.Combine(sourceDir, "SMDInspect"); //этот метод соединяет путь sourceDir и строку "SMDInspect", чтобы создать полный путь к новой папке. Папка будет называться SMDInspect и располагаться внутри sourceDir.

Directory.CreateDirectory(inspectDir); // создает директорию по указанному пути inspectDir

string logFile = Path.Combine(inspectDir, "smddirinfo.txt"); // комбинирует путь inspectDir и имя файла "smddirinfo.txt", создавая полный путь к файлу, где будет записана информация о директории.

File.WriteAllText(logFile, $"информация о директории: {sourceDir}");

// Копируем и переименовываем файл

string copyFile = Path.Combine(inspectDir, "smddirinfo\_copy.txt");

File.Copy(logFile, copyFile); // Копируем файл

Console.WriteLine($"Создана копия файла: {copyFile}");

// Удаляем оригинальный файл

File.Delete(logFile);

Console.WriteLine($"Оригинальный файл {logFile} был удален.");

string filesDir = Path.Combine(sourceDir, "SMDFiles"); //создаёт путь для новой папки SMDFiles внутри директории sourceDir.

Directory.CreateDirectory(filesDir);

foreach (var file in Directory.GetFiles(sourceDir, $"\*{fileExtension}")) //метод GetFiles возвращает список файлов в указанной директории sourceDir, соответствующих фильтру \*{fileExtension}.

{

string destFile = Path.Combine(filesDir, Path.GetFileName(file));//Эта строка создаёт путь для нового файла destFile, куда будет скопирован исходный файл file

File.Copy(file, destFile);//File.Copy используется для копирования файла из исходного пути в целевой путь.

}

string archivePath = Path.Combine(inspectDir, "SMDFiles.zip");//В этой строке создается полный путь к архиву SMDFiles.zip

ZipFile.CreateFromDirectory(filesDir, archivePath);//Этот метод создает архив .zip из содержимого директории.filesDir — это путь к директории, которую нужно заархивировать.archivePath — это путь, по которому будет сохранен архив

string extractPath = Path.Combine(sourceDir, "SMDExtract");//Теперь создается путь для папки, в которую будет извлечен архив.

ZipFile.ExtractToDirectory(archivePath, extractPath);//Извлекает все файлы из указанного zip-архива в указанную директорию..

Console.WriteLine("операции с файлами выполнены успешно");

}

catch (Exception ex)

{

Console.WriteLine($"ошибка управления файлами: {ex.Message}");

}

}

}

public class Program

{

static void Main(string[] args)

{

SMDLog.WriteLog("программа запущена", "начало выполнения метода Main");

SMDLog.ReadLog();

Console.WriteLine("---------------------------------------------------------------------------------------------------");

SMDDiskInfo.GetDiskInfo();

SMDFileInfo.GetFileInfo("C:\\Users\\User\\Documents\\2курс1сем\\лабыООП\\lab12\\lab12\\a.txt");

Console.WriteLine("---------------------------------------------------------------------------------------------------");

SMDDirInfo.GetDirInfo("C:\\Users");

Console.WriteLine("---------------------------------------------------------------------------------------------------");

SMDFileManager.ManageFiles("C:\\TestDir", ".txt");

SMDLog.WriteLog("программа завершена", "выполнение метода Main завершено");

}

}