Кам’янець-Подільський національний університет імені Івана Огієнка

КАФЕДРА КОМП’ЮТЕРНИХ НАУК

Навчальна дисципліна «Об’єктно-орієнтоване програмування»

ЛАБОРАТОРНА РОБОТА #0108

Тема:

Робота з файлами

Варіант №3

**Виконала**:  
студентка 1-го курсу  
групи KNms1-B24  
Некрасова Ю.Д.

**Прийняв**:  
Слободянюк О.В.

Кам’янець-Подільський – 2025

1. **Короткі теоретичні відомості**

**Робота з потоками і файловою системою**

Більшість задач у програмуванні так чи інакше пов’язані з роботою з файлами та каталогами. Нам може знадобитися прочитати текст із файлу або отримати запис, видалити файл чи весь каталог, не кажучи вже про більш комплексні завдання, як, наприклад, створення текстового редактора та інших подібних завдань.

Фреймворк .NET надає великі можливості для управління та маніпуляцій файлами та каталогами, які є більшими частинами сосредоточени в просторі імені System.IO . Класи, розташовані в цьому просторі (такі як Stream, StreamWriter, FileStream тощо), дозволяють керувати файлами вводом-виводом.

**Работа з дисками**

Для представлення диску в просторі імен : **System.IO** є клас **DriveInfo** .

Цей клас має статичний метод GetDrives, який повертає імена всіх логічних дисків комп'ютера. Також він пропонує ряд корисних властивостей:

* AvailableFreeSpace : вказує на обсяг доступного вільного місця на диску в байтах
* DriveFormat : отримує ім'я файлової системи
* DriveType : представляє тип диска
* IsReady : готовий диск (наприклад, DVD-диск може бути не вставлений в дисковод)
* Ім'я : отримує ім'я диска
* TotalFreeSpace : отримує загальний обсяг вільного місця на диску в байтах
* TotalSize : загальний розмір диска в байтах
* VolumeLabel : отримує або встановлює метку тома

Приклад коду, який дозволяє отримати імена і властивості всіх дисків на комп'ютері:

using System;

using System.IO;

namespace HelloApp

{

class Program

{

static void Main(string[] args)

{

DriveInfo[] drives = DriveInfo.GetDrives();

foreach (DriveInfo drive in drives)

{

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

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

if (drive.IsReady)

{

Console.WriteLine($"Об’єм диску: {drive.TotalSize}");

Console.WriteLine($"Вільне місце: {drive.TotalFreeSpace}");

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

}

Console.WriteLine();

}

}

}

}

Зображення, що містить текст, електроніка, знімок екрана, монітор

Вміст, створений ШІ, може бути неправильним.

**Работа с каталогами.**

Для роботи з каталогами у просторі імен System.IO призначені відразу два класи: **Directory** та **DirectoryInfo**.

**Клас Directory**

Клас Directory надає низку статичних методів для керування каталогами. Деякі з цих методів:

* CreateDirectory(path) : створює каталог по вказаному шляху path
* Delete(path) : видаляє каталог вказаним шляхом path
* Exists(path) : визначає, чи існує каталог вказаним шляхом path. Якщо існує, повертається true, якщо не існує, тоfalse
* GetDirectories(path) : отримує список каталогів у каталозі path
* GetFiles(path) : отримує список файлів у каталозі path
* Move(sourceDirName, destDirName) : переміщує каталог
* GetParent(path) : отримання батьківського каталогу

**Клас DirectoryInfo**

Цей клас надає функціональність для створення, видалення, переміщення та інших операцій з каталогами. Багато в чому він нагадує Directory. Деякі з його властивостей та методів:

* Create() : створює каталог
* CreateSubdirectory(path) : створює підкаталог по вказаному шляху path
* Delete() : видаляє каталог
* Властивість Exists : визначає, чи існує каталог
* GetDirectories() : отримує список каталогів
* GetFiles() : отримує список файлів
* MoveTo(destDirName) : переміщує каталог
* Властивість Parent : отримання батьківського каталогу
* Властивість Root : отримання кореневого каталогу

Подивимося на прикладах застосування цих класів

**Отримання списку файлів та підкаталогів**

string dirName = "C:\\flutter\_sdk";

if (Directory.Exists(dirName))

{

Console.WriteLine("Пiдкаталоги:");

string[] dirs = Directory.GetDirectories(dirName);

foreach (string s in dirs)

{

Console.WriteLine(s);

}

Console.WriteLine();

Console.WriteLine("Файли:");

string[] files = Directory.GetFiles(dirName);

foreach (string s in files)

{

Console.WriteLine(s);

}

}

Зображення, що містить текст, знімок екрана, монітор

Вміст, створений ШІ, може бути неправильним.

Зверніть увагу на використання слешів в іменах файлів. Або ми використовуємо подвійний слеш: "C:\\flutter\_sdk" або одинарний, але тоді перед усім шляхом ставимо знак @: @"C:\Program Files"

**Створення каталогу.**

string path = @"D:\tmp";

string subpath = @"program\avalon";

DirectoryInfo dirInfo = new DirectoryInfo(path);

if (!dirInfo.Exists)

{

dirInfo.Create();

}

dirInfo.CreateSubdirectory(subpath);

Зображення, що містить знімок екрана, текст, ряд, Шрифт

Вміст, створений ШІ, може бути неправильним.

Спочатку перевіряємо, а чи немає такої директорії, тому що якщо вона існує, то її створити буде не можна, і програма викине помилку. У результаті в нас вийде наступний шлях: "D:\tmp\program\avalon"

**Отримання інформації про каталог**

string dirName = "C:\\Program Files";

DirectoryInfo dirInfo = new DirectoryInfo(dirName);

Console.WriteLine($"Назва каталогу: {dirInfo.Name}");

Console.WriteLine($"Повна назва каталогу: {dirInfo.FullName}");

Console.WriteLine($"Час створення каталогу: {dirInfo.CreationTime}");

Console.WriteLine($"Кореневий каталог: {dirInfo.Root}");

Зображення, що містить текст, знімок екрана, Шрифт, число

Вміст, створений ШІ, може бути неправильним.

**Видалення каталогу**

Якщо ми просто застосуємо метод Delete до непустої папки, в якій є файли або підкаталоги, то додаток нам викине помилку. Тому нам треба передати до методу Delete додатковий параметр булевого типу, який вкаже, що папку треба видаляти з усім вмістом:

string dirName = @"C:\SomeFolder";

try

{

DirectoryInfo dirInfo = new DirectoryInfo(dirName);

dirInfo.Delete(true);

Console.WriteLine("Каталог удален");

}

catch (Exception ex)

{

Console.WriteLine(ex.Message);

}

Або так:

string dirName = @"C:\SomeFolder";

Directory.Delete(dirName, true);

**Переміщення каталогу**

string oldPath = @"C:\SomeFolder";

string newPath = @"C:\SomeDir";

DirectoryInfo dirInfo = new DirectoryInfo(oldPath);

if (dirInfo.Exists && Directory.Exists(newPath) == false)

{

dirInfo.MoveTo(newPath);

}

При переміщенні слід враховувати, що новий каталог, до якого ми хочемо перемістити весь вміст старого каталогу, не повинен існувати.

**Робота із файлами. Класи File та FileInfo.**

Подібно до пари Directory/DirectoryInfoдля роботи з файлами призначена пара класів **File** і **FileInfo** . З їхньою допомогою ми можемо створювати, видаляти, переміщувати файли, отримувати їх властивості та багато іншого.

Деякі корисні методи та властивості класу FileInfo:

* CopyTo(path) : копіює файл у нове місце за вказаним шляхом path
* Create() : створює файл
* Delete() : видаляє файл
* MoveTo(destFileName) : переміщує файл у нове місце
* Властивість Directory : отримує батьківський каталог у вигляді об'єкта DirectoryInfo
* Властивість DirectoryName : отримує повний шлях до батьківського каталогу
* Властивість Exists : вказує, чи існує файл
* Властивість Length : отримує розмір файлу
* Властивість Extension : отримує розширення файлу
* Властивість Name : отримує ім'я файлу
* Властивість FullName : отримує повне ім'я файлу

Клас File реалізує схожу функціональність за допомогою статичних методів:

* Copy() : копіює файл у нове місце
* Create() : створює файл
* Delete() : видаляє файл
* Move : переміщує файл у нове місце
* Exists(file) : визначає, чи існує файл

**Отримання інформації про файл**

string path = @"C:\apache\hta.txt";

FileInfo fileInf = new FileInfo(path);

if (fileInf.Exists)

{

Console.WriteLine("Имя файла: {0}", fileInf.Name);

Console.WriteLine("Время создания: {0}", fileInf.CreationTime);

Console.WriteLine("Размер: {0}", fileInf.Length);

}

**Видалення файлу**

string path = @"C:\apache\hta.txt";

FileInfo fileInf = new FileInfo(path);

if (fileInf.Exists)

{

fileInf.Delete();

// альтернатива за допомогою класу File

// File.Delete(path);

}

**Переміщення файлу**

string path = @"C:\apache\hta.txt";

string newPath = @"C:\SomeDir\hta.txt";

FileInfo fileInf = new FileInfo(path);

if (fileInf.Exists)

{

fileInf.MoveTo(newPath);

// альтернатива за допомогою класу File

// File.Move(path, newPath);

}

**Копіювання файлу**

string path = @"C:\apache\hta.txt";

string newPath = @"C:\SomeDir\hta.txt";

FileInfo fileInf = new FileInfo(path);

if (fileInf.Exists)

{

fileInf.CopyTo(newPath, true);

// альтернатива за допомогою класу File

// File.Copy(path, newPath, true);

}

Метод CopyTo класу FileInfo приймає два параметри: шлях, яким файл буде копіюватися, і булеве значення, яке вказує, чи потрібно при копіюванні перезаписувати файл (якщо true, як у випадку вище, файл при копіюванні перезаписується). Якщо ж як останній параметр передати значення false, то якщо такий файл вже існує, програма видасть помилку.

Метод Copy класу File приймає три параметри: шлях до вихідного файлу, шлях, яким файл буде копіюватися, і булеве значення, що вказує, чи буде файл перезаписуватися.

**FileStream. Читання та запис файлу**

Клас **FileStream** представляє можливості для зчитування з файлу та запису у файл. Він дозволяє працювати як із текстовими файлами, так і з бінарними.

**Створення FileStream**

Для створення об'єкта FileStream можна використовувати як конструктори цього класу, і статичні методи класу File. Конструктор FileStream має безліч перевантажених версій, з яких відзначу лише одну, найпростішу і використовується:

FileStream(string filename, FileMode mode)

Тут у конструктор передається два параметри: шлях до файлу та перерахування **FileMode**. Цей перелік вказує на режим доступу до файлу і може приймати такі значення:

* Append : якщо файл існує, текст додається в кінець файл. Якщо файлу немає, він створюється. Файл відкривається лише для запису.
* Create : створюється новий файл. Якщо такий файл вже існує, він перезаписується
* CreateNew : створюється новий файл. Якщо такий файл вже існує, він додаток викидає помилку
* Open : відкриває файл. Якщо файл не існує, викидається виняток
* OpenOrCreate : якщо файл існує, він відкривається, якщо ні – створюється новий
* Truncate : якщо файл існує, він перезаписується. Файл відкривається лише для запису.

Інший спосіб створення об'єкта FileStream представляють статичні методи класу File:

FileStream File.Open(string file, FileMode mode);

FileStream File.OpenRead(string file);

FileStream File.OpenWrite(string file);

Перший метод відкриває файл з урахуванням об'єкта FileMode та повертає файловий потік FileStream. Цей метод також має кілька перевантажених версій. Другий метод відкриває потік для читання, а третій відкриває потік для запису.

**Властивості та методи FileStream**

Розглянемо найважливіші його властивості та методи класу FileStream:

* Властивість Length : повертає довжину потоку в байтах
* Властивість Position : повертає поточну позицію в потоці
* void CopyTo(Stream destination): копіює дані з поточного потоку до потоку destination
* Task CopyToAsync(Stream destination): асинхронна версія методу CopyToAsync
* int Read(byte[] array, int offset, int count): зчитує дані з файлу масив байтів і повертає кількість успішно лічених байтів. Приймає три параметри:
* array - масив байтів, куди будуть поміщені дані, що зчитуються з файлу
* offset представляє зміщення в байтах у масиві array, в який лічені байти будуть поміщені
* count- Максимальна кількість байтів, призначених для читання. Якщо у файлі знаходиться менша кількість байтів, то всі вони будуть прочитані.
* Task<int> ReadAsync(byte[] array, int offset, int count): асинхронна версія методу Read
* long Seek(long offset, SeekOrigin origin): встановлює позицію в потоці зі зміщенням кількість байт, зазначених у параметрі offset.
* void Write(byte[] array, int offset, int count): записує у файл дані з масиву байтів Приймає три параметри:
* array - масив байтів, звідки дані записуватимуться у файл
* offset - Зміщення в байтах у масиві array, звідки починається запис байтів у потік
* count - максимальна кількість байтів, призначених для запису
* ValueTask WriteAsync(byte[] array, int offset, int count): асинхронна версія методу Write

**Читання та запис файлів**

FileStream представляє доступ до файлів на рівні байтів, тому, наприклад, якщо вам треба рахувати або записати один або кілька рядків у текстовий файл, то масив байтів треба перетворити на рядки, використовуючи спеціальні методи. Тому для роботи з текстовими файлами використовуються інші класи.

У той же час при роботі з різними бінарними файлами, що мають певну структуру, FileStream може бути дуже корисний для отримання певних порцій інформації та її обробки.

Подивимося на прикладі зчитування-запису до текстового файлу:

using System;

using System.IO;

namespace HelloApp

{

class Program

{

static void Main(string[] args)

{

// створюємо каталог для файлу

string path = @"C:\SomeDir2";

DirectoryInfo dirInfo = new DirectoryInfo(path);

if (!dirInfo.Exists)

{

dirInfo.Create();

}

Console.WriteLine("Введіть рядок для запису в файл:");

string text = Console.ReadLine();

// запис у файл

using (FileStream fstream = new FileStream($"{path}\note.txt", FileMode.OpenOrCreate))

{

// перетворимо рядок в байти

byte[] array = System.Text.Encoding.Default.GetBytes(text);

// запис масиву байтів в файл

fstream.Write(array, 0, array.Length);

Console.WriteLine("Текст записаний в файл");

}

// читання з файлу

using (FileStream fstream = File.OpenRead($"{path}\note.txt"))

{

// перетворимо рядок в байти

byte[] array = new byte[fstream.Length];

// зчитуємо дані

fstream.Read(array, 0, array.Length);

// декодуємо байти в рядок

string textFromFile = System.Text.Encoding.Default.GetString(array);

Console.WriteLine($"Текст з файлу: {textFromFile}");

}

Console.ReadLine();

}

}

}

**Довільний доступ до файлів**

Нерідко бінарні файли становлять певну структуру. І, знаючи цю структуру, ми можемо взяти з файлу потрібну порцію інформації або навпаки записати в певному місці файлу певний набір байтів. Наприклад, у wav-файлах безпосередньо звукові дані починаються з 44 байти, а до 44 байти йдуть різні метадані - кількість каналів аудіо, частота дискретизації тощо.

За допомогою методу **Seek**() ми можемо керувати положенням курсору потоку, починаючи з якого здійснюється зчитування або запис файлу. Цей метод приймає два параметри: offset (зміщення) та позиція у файлі. Позиція у файлі описується трьома значеннями:

* SeekOrigin.Begin : початок файлу
* SeekOrigin.End : кінець файлу
* SeekOrigin.Current : поточна позиція у файлі

Курсор потоку, з якого починається читання або запис, зміщується вперед на значення offset щодо позиції, зазначеної як другий параметр. Зміщення може бути від’ємним, тоді курсор зсувається назад, якщо додатнім – то вперед.

Розглянемо приклад:

using System.IO;

using System.Text;

class Program

{

static void Main(string[] args)

{

string text = "hello world";

// запис у файл

using (FileStream fstream = new FileStream(@"D:\note.dat", FileMode.OpenOrCreate))

{

// перетворимо рядок в байти

byte[] input = Encoding.Default.GetBytes(text);

// запис масиву байтів в файл

fstream.Write(input, 0, input.Length);

Console.WriteLine("Текст записаний у файл");

// перемещаем покажчик в кінець файлу, до кінця файлу - п'ять байт

fstream.Seek(-5, SeekOrigin.End); // мінус 5 символів з кінця потоку

// зчитуємо чотири символи з поточної позиції

byte[] output = new byte[4];

fstream.Read(output, 0, output.Length);

// декодуємо байти в рядок

string textFromFile = Encoding.Default.GetString(output);

Console.WriteLine($"Текст з файлу: {textFromFile}"); // worl

// замінимо в файлі слово world на слово house

string replaceText = "house";

fstream.Seek(-5, SeekOrigin.End); // мінус 5 символів з кінця потоку

input = Encoding.Default.GetBytes(replaceText);

fstream.Write(input, 0, input.Length);

// зчитуємо весь файл

// повертаємо покажчик в початок файлу

fstream.Seek(0, SeekOrigin.Begin);

output = new byte[fstream.Length];

fstream.Read(output, 0, output.Length);

// декодуємо байти в рядок

textFromFile = Encoding.Default.GetString(output);

Console.WriteLine($"Текст з файлу: {textFromFile}"); // hello house

}

Console.Read();

}

}

Зображення, що містить текст, знімок екрана, Шрифт, число

Вміст, створений ШІ, може бути неправильним.

Виклик fstream.Seek(-5, SeekOrigin.End)переміщує курсор потоку на кінець файлів назад на п'ять символів:

Зображення, що містить текст, Шрифт, знімок екрана, білий

Вміст, створений ШІ, може бути неправильним.

Переміщуємось з кінця на 5 символів назад.

Тобто, після запису в новий файл рядка "hello world" курсор стоятиме на позиції символу "w".

Після цього зчитуємо чотири байти, починаючи з символу "w". У цьому кодуванні 1 символ представлятиме 1 байт. Тому читання 4 байтів буде еквівалентне читання чотирьох сивол: "worl".

Потім знову ж таки переміщуємося в кінець файлу, не доходячи до кінця п'ять символів (тобто знову ж таки з позиції символу "w"), і здійснюємо запис рядка "house". Таким чином, рядок "house" замінює рядок "world".

**Закриття потоку**

У прикладах вище для закриття потоку застосовується конструкція **using** . Після того, як всі оператори і вирази в блоці, які використовуються, відпрацюють, об'єкт FileStream знищується. Однак ми можемо вибрати інший спосіб:

FileStream fstream = null;

try

{

fstream = new FileStream(@"D:\note3.dat", FileMode.OpenOrCreate);

// операции с потоком

}

catch(Exception ex)

{

}

finally

{

if (fstream != null)

fstream.Close();

}

Якщо ми не використовуємо конструкцію using, то нам треба явно викликати метод Close():fstream.Close()

**Читання та запис текстових файлів. StreamReader та StreamWriter**

Клас FileStream не дуже зручно використовувати для роботи з текстовими файлами. До того ж для цього у просторі System.IO визначені спеціальні класи: **StreamReader** та **StreamWriter** .

**Запис у файл і StreamWriter**

Для запису в текстовий файл використовується клас **StreamWriter** . Деякі його конструктори, які можуть застосовуватися для створення об'єкта StreamWriter:

* StreamWriter(string path): через параметр path передається шлях до файлу, який буде пов'язаний із потоком
* StreamWriter(string path, bool append): параметр append вказує, чи потрібно додавати в кінець файлу дані або перезаписувати файл. Якщо дорівнює true, нові дані додаються в кінець файлу. Якщо одно false, то файл перезаписується наново
* StreamWriter(string path, bool append, System.Text.Encoding encoding): параметр encoding вказує на кодування, яке буде застосовуватись під час запису

Свою функціональність StreamWriter реалізує через такі методи:

* int Close(): закриває записуваний файл та звільняє всі ресурси
* void Flush(): записує в файл дані, що залишилися в буфері, і очищає буфер.
* Task FlushAsync(): асинхронна версія методу Flush
* void Write(string value): записує файл найпростіших типів, як int, double, char, string і т.д. Відповідно, має ряд перевантажених версій для запису даних елементарних типів, наприклад, Write(char value), Write(int value), Write(double value)і т.д.
* Task WriteAsync(string value): асинхронна версія методу Write
* void WriteLine(string value): також записує дані, тільки після запису додає у файл символ закінчення рядка
* Task WriteLineAsync(string value): асинхронна версія методу WriteLine

Розглянемо запис до файлу на прикладі:

using System;

using System.IO;

namespace HelloApp

{

class Program

{

static void Main(string[] args)

{

string writePath = @"C:\SomeDir\hta.txt";

string text = "Привіт світ!\nПока світ...";

try

{

using (StreamWriter sw = new StreamWriter(writePath, false, System.Text.Encoding.Default))

{

sw.WriteLine(text);

}

using (StreamWriter sw = new StreamWriter(writePath, true, System.Text.Encoding.Default))

{

sw.WriteLine("Дозапис");

sw.Write(4.5);

}

Console.WriteLine("Запис виконано");

}

catch (Exception e)

{

Console.WriteLine(e.Message);

}

}

}

}

У цьому випадку двічі створюємо об'єкт StreamWriter. У першому випадку, якщо файл існує, то він буде перезаписаний. Якщо не існує, його буде створено. І в неї буде записаний текст із змінної text. У другому випадку файл відкривається для дозапису і будуть записані атомарні дані - рядок і число. В обох випадках буде використовуватися кодування за промовчанням.

Після завершення програми у папці C://SomeDir ми зможемо знайти файл hta.txt, який матиме наступні рядки:

Привіт світ!

Поки що світ...

Дозапис

4,5

Оскільки операції з файлами можуть займати тривалий час, то рекомендується використовувати асинхронний запис. Використовуємо асинхронні версії методів:

using System;

using System.IO;

using System.Threading.Tasks;

namespace HelloApp

{

class Program

{

static async Task Main(string[] args)

{

string writePath = @"C:\SomeDir\hta2.txt";

string text = "Привіт світ!\nПока світ...";

try

{

using (StreamWriter sw = new StreamWriter(writePath, false, System.Text.Encoding.Default))

{

await sw.WriteLineAsync(text);

}

using (StreamWriter sw = new StreamWriter(writePath, true, System.Text.Encoding.Default))

{

await sw.WriteLineAsync("Дозапис");

await sw.WriteAsync("4,5");

}

Console.WriteLine("Запис виконано");

}

catch (Exception e)

{

Console.WriteLine(e.Message);

}

}

}

}

**Читання з файлу та StreamReader**

Клас StreamReader дозволяє нам легко читати весь текст або окремі рядки з текстового файлу.

Деякі з конструкторів класу StreamReader:

* StreamReader(string path): через параметр path передається шлях до зчитуваного файлу
* StreamReader(string path, System.Text.Encoding encoding): параметр encoding задає кодування для читання файлу

Серед методів StreamReader можна виділити такі:

* void Close(): закриває файл, що зчитується, і звільняє всі ресурси.
* int Peek(): повертає наступний символ, якщо символів більше немає, то повертає -1
* int Read(): зчитує та повертає наступний символ у чисельному поданні. Має перевантажену версію: Read(char[] array, int index, int count), де array- масив, куди зчитуються символи, index- індекс в масиві array, починаючи з якого записуються символи, що зчитуються, і count - максимальна кількість зчитуваних символів
* Task<int> ReadAsync(): асинхронна версія методу Read
* string ReadLine(): зчитує один рядок у файлі
* string ReadLineAsync(): асинхронна версія методу ReadLine
* string ReadToEnd(): зчитує весь текст із файлу
* string ReadToEndAsync(): асинхронна версія методу ReadToEnd

Спочатку прочитаємо текст повністю із раніше записаного файлу:

using System;

using System.IO;

using System.Threading.Tasks;

namespace HelloApp

{

class Program

{

static async Task Main(string[] args)

{

string path = @"C:\SomeDir\hta.txt";

try

{

using (StreamReader sr = new StreamReader(path))

{

Console.WriteLine(sr.ReadToEnd());

}

// асинхронне читання

using (StreamReader sr = new StreamReader(path))

{

Console.WriteLine(await sr.ReadToEndAsync());

}

}

catch (Exception e)

{

Console.WriteLine(e.Message);

}

}

}

}

Прочитаємо текст із файлу порядково:

string path = @"C:\SomeDir\hta.txt";

using (StreamReader sr = new StreamReader(path, System.Text.Encoding.Default))

{

string line;

while ((line = sr.ReadLine()) != null)

{

Console.WriteLine(line);

}

}

// асинхронне читання

using (StreamReader sr = new StreamReader(path, System.Text.Encoding.Default))

{

string line;

while ((line = await sr.ReadLineAsync()) != null)

{

Console.WriteLine(line);

}

}

В даному випадку зчитуємо рядково через цикл while: while ((line = sr.ReadLine()) != null) - спочатку присвоюємо змінної line результат функції sr.ReadLine(), а потім перевіряємо, чи вона не дорівнює null. Коли об'єкт sr дійде до кінця файлу і більше рядків не залишиться, метод sr.ReadLine() повертатиме null.

**Бінарні файли. BinaryWriter і BinaryReader**

Для роботи з бінарними файлами призначена пара класів **BinaryWriter** і **BinaryReader** . Ці класи дозволяють читати і записувати дані в двойному форматі.

**Основні методи класу BinaryWriter**

* Close() : закриває потік і освобождає ресурси
* Flush() : очищає буфер, дописуючи його з нього залишилися дані у файлі
* Seek() : встановлює позицію в потокі
* Write() : записує дані в поток

**Основні методи класу BinaryReader**

* Close() : закриває потік і освобождає ресурси
* ReadBoolean() : обчислює значення bool і переміщує вказівник на один байт
* ReadByte() : обчислює один байт і переміщує вказівник на один байт
* ReadChar() : обчислює значення char, то є один символ і переміщує вказівник на лише байтів, скільки займає символ у поточній кодировці
* ReadDecimal() : обчислює значення десяткового і переміщує вказівник на 16 байт
* ReadDouble() : обчислює значення double і переміщує показник на 8 байт
* ReadInt16() : обчислює значення short і переміщує показник на 2 байта
* ReadInt32() : обчислює значення int і переміщує показник на 4 байта
* ReadInt64() : обчислює значення long і переміщує вказівник на 8 байт
* ReadSingle() : обчислює значення float і переміщує показник на 4 байта
* ReadString() : обчислює значення рядка. Каждая строка передварюється значенням довжини строки, яке являє собою 7-бітне ціле число

З читанням бінарних даних все просто: відповідний метод обчислює дані певного типу і переміщує вказівник на розмір цього типу в байтах, наприклад, значення типу int займає 4 байта, тому BinaryReader вважає 4 байта і перемістить вказувати на ці 4 байти.

Розглянемо реальну задачу застосування цих класів. Спробуємо з їх допомогою записувати і виводити в масив структур з файлу:

using System;

using System.IO;

using System.Threading.Tasks;

namespace HelloApp

{

struct State

{

public string name;

public string capital;

public int area;

public double people;

public State(string n, string c, int a, double p)

{

name = n;

capital = c;

people = p;

area = a;

}

}

class Program

{

static void Main(string[] args)

{

State[] states = new State[2];

states[0] = new State("Німеччина", "Берлін", 357168, 80.8);

states[1] = new State("Франція", "Париж", 640679, 64.7);

string path = @"D:\tmp\states.dat";

try

{

// створюємо об'єкт BinaryWriter

using (BinaryWriter writer = new BinaryWriter(File.Open(path, FileMode.OpenOrCreate)))

{

// записуємо в файл значення кожного поля структури

foreach (State s in states)

{

writer.Write(s.name);

writer.Write(s.capital);

writer.Write(s.area);

writer.Write(s.people);

}

}

// створюємо об'єкт BinaryReader

using (BinaryReader reader = new BinaryReader(File.Open(path, FileMode.Open)))

{

// поки не досягнуто кінець файлу

// зчитуємо кожне значення з файлу

while (reader.PeekChar() > -1)

{

string name = reader.ReadString();

string capital = reader.ReadString();

int area = reader.ReadInt32();

double population = reader.ReadDouble();

Console.WriteLine("Країна: {0} столиця: {1} площа {2} кв. км чисельність населення: {3} млн. чол.", name, capital, area, population);

}

}

}

catch (Exception e)

{

Console.WriteLine(e.Message);

}

Console.ReadLine();

}

}

}

Так, у нас є структура держави з некоторым набором полей. У основній програмі створюємо масивну структуру і записуємо за допомогою BinaryWriter. Цей клас у якості параметра в конструкторі приймає об'єкт Stream, який створюється викликом File.Open(path, FileMode.OpenOrCreate).

Потім в циклі пробігаємо по масовій структурі і запишемо кожну структуру поля в потоці. В тому порядку, в якому ці значення поля записуються, в тому порядку вони і будуть розміщатися у файлі.

Потім обчислюємо із записаного файлу. Конструктор класу BinaryReader також у параметрах якості приймає об'єкт потоку, тільки в цьому випадку встановлюємо в якості режиму FileMode.Open:new BinaryReader(File.Open(path, FileMode.Open))

В циклі whileобчислюємо дані. Щоб дізнатися закінчення потоку, викликаємо метод PeekChar(). Цей метод обчислює наступний символ і повертає його числове зображення. Якщо символ відсутня, то метод повертає -1, що буде означати, що ми досягли кінця файлу.

У циклі послідовно обчислюємо значення поля структури в тому же порядку, в якому вони записалися.

1. **Умова завдання 1**

Реалізувати наступні операції з файловою системою:

1) На диску D: створіть каталог OOP\_lab08.

2) Cтворіть у ньому наступні каталоги (назви англійською мовою):

* Номер\_вашої\_групи (підставте свою групу);
* Ваше\_прізвище (підставте своє прізвище);
* Sources;
* Reports;
* Texts.

3) Скопіюйте каталоги Texts, Sources та Reports до каталогу Ваше\_прізвище.

4) Перемістіть каталог Ваше\_прізвище до каталогу Номер\_вашої\_групи.

5) У каталозі Texts створіть текстовий файл dirinfo.txt у який помістіть усю інформацію про даний каталог (див. приклад вище).

1. **Послідовність виконання роботи**

* **Створення директорії OOP\_lab08 на диску C (оскільки диску D немає):**

У програмі перевіряється наявність каталогу C:\OOP\_lab08. Якщо він відсутній — створюється автоматично.

* **Створення у каталозі OOP\_lab08 підкаталогів:**

Створено п’ять підкаталогів з назвами:

* Reports
* Sources
* Texts
* <Nekrasova>
* <KNms1-B24>

**• Копіювання вмісту каталогів Texts, Sources, Reports до каталогу < Nekrasova >**

Використано метод CopyDirectoryUsingFileInfo, який копіює всі файли та підкаталоги рекурсивно.

Файли копіюються через FileInfo.CopyTo(...), як у прикладі.

* **Переміщення каталогу < Nekrasova > до каталогу < KNms1-B24>**

Якщо така директорія вже існує — видаляється, після чого виконується переміщення через Directory.Move(...).

* **Створення текстового файлу dirinfo.txt у каталозі Texts**  
  У цей файл записується інформація про директорію Texts:
* Повний шлях
* Час створення
* Час останнього доступу
* Атрибути

Запис виконується за допомогою StreamWriter.

**• Додано обробку виключень**

Весь код розміщений у блоці try-catch для безпечного виконання з виведенням повідомлень про помилки.

**• Додано очікування натискання клавіші перед завершенням**

Використано Console.ReadKey() — щоб користувач бачив результат перед закриттям вікна консолі.

1. **Код програми**

using System;

using System.IO;

class Program

{

static void Main()

{

string group = "KNms1-B24";

string lastName = "Nekrasova";

string basePath = @"C:\OOP\_lab08";

try

{

DirectoryInfo mainDir = new DirectoryInfo(basePath); //Створення основної директорії

if (!mainDir.Exists)

{

mainDir.Create();

Console.WriteLine("Main directory created.");

}

string[] subDirs = { "Sources", "Reports", "Texts", lastName, group }; //Створення підкаталогів

foreach (string dir in subDirs)

{

string subDirPath = Path.Combine(basePath, dir);

if (!Directory.Exists(subDirPath))

Directory.CreateDirectory(subDirPath);

}

Console.WriteLine("Initial subdirectories created.");

CopyDirectoryUsingFileInfo(Path.Combine(basePath, "Sources"), Path.Combine(basePath, lastName, "Sources")); // Копіювання вмісту каталогів у каталог з прізвищем

CopyDirectoryUsingFileInfo(Path.Combine(basePath, "Reports"), Path.Combine(basePath, lastName, "Reports"));

CopyDirectoryUsingFileInfo(Path.Combine(basePath, "Texts"), Path.Combine(basePath, lastName, "Texts"));

Console.WriteLine("Subdirectories copied into personal folder.");

string sourcePath = Path.Combine(basePath, lastName); //Переміщення каталогу прізвища до каталогу групи

string destinationPath = Path.Combine(basePath, group, lastName);

if (Directory.Exists(destinationPath))

Directory.Delete(destinationPath, true);

Directory.Move(sourcePath, destinationPath);

Console.WriteLine("Personal folder moved to group folder.");

string textsPath = Path.Combine(basePath, "Texts"); // Створення файлу dirinfo.txt у каталозі Texts

string infoFilePath = Path.Combine(textsPath, "dirinfo.txt");

DirectoryInfo textsDir = new DirectoryInfo(textsPath);

using (StreamWriter writer = new StreamWriter(infoFilePath))

{

writer.WriteLine("Directory Info:");

writer.WriteLine($"Full Name: {textsDir.FullName}");

writer.WriteLine($"Creation Time: {textsDir.CreationTime}");

writer.WriteLine($"Last Access Time: {textsDir.LastAccessTime}");

writer.WriteLine($"Attributes: {textsDir.Attributes}");

}

Console.WriteLine("dirinfo.txt created.");

}

catch (Exception ex)

{

Console.WriteLine($"Error: {ex.Message}");

}

Console.WriteLine("Press any key to exit...");

Console.ReadKey();

}

static void CopyDirectoryUsingFileInfo(string sourceDir, string targetDir) // Метод копіювання каталогів через FileInfo.CopyTo

{

DirectoryInfo source = new DirectoryInfo(sourceDir);

DirectoryInfo target = new DirectoryInfo(targetDir);

if (!source.Exists)

return;

if (!target.Exists)

target.Create();

foreach (FileInfo file in source.GetFiles()) // Копіювання файлів за допомогою FileInfo.CopyTo

{

string newPath = Path.Combine(target.FullName, file.Name);

FileInfo fileInf = new FileInfo(file.FullName);

if (fileInf.Exists)

{

fileInf.CopyTo(newPath, true);

}

}

foreach (DirectoryInfo subDir in source.GetDirectories()) // Рекурсивне копіювання підкаталогів

{

string newSubDir = Path.Combine(target.FullName, subDir.Name);

CopyDirectoryUsingFileInfo(subDir.FullName, newSubDir);

}

}

}

1. **Приклад виконання програми**

**Зображення, що містить текст, знімок екрана, програмне забезпечення, Мультимедійне програмне забезпечення

Вміст, створений ШІ, може бути неправильним.**

**Зображення, що містить текст, знімок екрана, програмне забезпечення, комп’ютер

Вміст, створений ШІ, може бути неправильним.**

1. **Умови завдання 2 (Варіант 3)**

Дано файл f, компоненти якого є цілими числами. Знайти:

1) кількість парних чисел серед компонент;

2) кількість подвоєних парних чисел серед компонент;

3) кількість квадратів непарних чисел серед компонент.

1. **Послідовність виконання роботи**

* string fileName вказує шлях до файлу f.
* DisplayNumbersInFile(fileName) − метод, який виведе на екран всі числа, що є у файлі.
* CreateFile(fileName) − метод для запису чисел у файл.
* ReadNumbersFromFile(fileName) – зчитування чисел з файлу і збереження їх у масиві numbers.
* Обчислення:
  + Кількість парних чисел (evenCount).
  + Кількість чисел, кратних 4 (doubledEvenCount).
  + Кількість квадратів непарних чисел (oddSquareCount).
* Виведення результатів обчислень на екран.

1. **Код програми**

using System;

using System.IO;

using System.Linq;

namespace LAB8App

{

class Program

{

static void Main(string[] args)

{

string fileName = @"C:\Users\notha\Desktop\ООП\8\f.txt"; // Шлях до файлу f

DisplayNumbersInFile(fileName); // Вивід всіх чисел з файлу на екран

CreateFile(fileName); // Запис чисел у файл

int[] numbers = ReadNumbersFromFile(fileName); // Зчитування чисел з файлу

int evenCount = numbers.Count(x => x % 2 == 0); // Кількість парних чисел

int doubledEvenCount = numbers.Count(x => x % 4 == 0); // Кількість подвоєних парних чисел (кратних 4)

int oddSquareCount = numbers.Count(x => // Кількість квадратів непарних чисел

{

if (x > 0)

{

int root = (int)Math.Sqrt(x);

return root \* root == x && root % 2 == 1;

}

return false;

});

Console.WriteLine("Count of even numbers: " + evenCount); // Виведення результатів

Console.WriteLine("Count of doubled even numbers (multiples of 4): " + doubledEvenCount);

Console.WriteLine("Count of squares of odd numbers: " + oddSquareCount);

Console.ReadLine();

}

static void DisplayNumbersInFile(string fileName) // Виведення всіх чисел з файлу на екран

{

int[] numbers = ReadNumbersFromFile(fileName);

Console.WriteLine("Numbers in the file:");

foreach (var number in numbers)

{

Console.Write(number + " ");

}

Console.WriteLine();

}

static void CreateFile(string fileName) // Створення файлу з числами (запис чисел у файл)

{

int[] numbers = { 1, 2, 4, 5, 8, 9, 16, 25, 3, 12, 49, 64, 7, 11 };

using (FileStream fstream = new FileStream(fileName, FileMode.OpenOrCreate))

{

byte[] array = System.Text.Encoding.Default.GetBytes(string.Join(" ", numbers));

fstream.Write(array, 0, array.Length);

}

Console.WriteLine("Numbers written to file.");

}

static int[] ReadNumbersFromFile(string fileName) // Зчитування чисел з файлу

{

string content;

using (FileStream fstream = File.OpenRead(fileName))

{

byte[] array = new byte[fstream.Length];

fstream.Read(array, 0, array.Length);

content = System.Text.Encoding.Default.GetString(array);

}

return content.Split(new[] { ' ', '\n', '\r' }, StringSplitOptions.RemoveEmptyEntries) // Перетворення рядку на масив чисел

.Select(int.Parse)

.ToArray();

}

}

}

1. **Приклад виконання програми**

**Зображення, що містить текст, знімок екрана, програмне забезпечення, Мультимедійне програмне забезпечення

Вміст, створений ШІ, може бути неправильним.**