ЖИТОМИРСЬКА ПОЛІТЕХНІКА
ДЕРЖАВНИЙ УНІВЕРСИТЕТ

C#. Робота з файлами

Лекція 13-14

План

1. Робота з потоками
2. Основні класи простіру імен System.IO
3. Основні методи класу StreamReader
4. Основні методи класу StreamWriter
5. Робота з файлами та каталогами
6. Серіалізація та десеріалізація в C#

Робота з потоками

В .NET Framework та C# існує потужна система для виконання операцій введення-виведення (I/O), яка дозволяє програмам взаємодіяти з різними джерелами даних, включаючи консоль, файлову систему, мережу тощо.

System.IO містить класи для роботи з потоками (*streams*), файлами, каталогами та іншими операціями, пов'язаними з файловою системою. Це основний простір імен для роботи з файлами.

System містить базові класи, включаючи *Console* для введення та виведення в консоль.

Робота з потоками

У C# введення - виведення здійснюється за допомогою концепції потоків (streams). Потік (Stream) у C# є абстракцією, яка уніфікує спосіб роботи з різними джерелами та приймачами даних. Замість того, щоб оперувати безпосередньо з файлами, мережевими з'єднаннями, пам'яттю або іншими джерелами/приймачами, програмісти C# працюють з потоками.

- Потоки приховують деталі конкретної реалізації введення-виведення.
- Дані в потоці зазвичай обробляються послідовно, байт за байтом (або блоками байтів).
- Потоки можуть представляти різні джерела даних (файли, мережеві сокети, пам'ять, стандартний ввід/вивід тощо) та різні приймачі даних.
- Основні операції, які підтримують потоки, – це читання даних з потоку та запис даних у потік.

Основні класи простіру імен System.IO

Основні класи для роботи з потоками в C# знаходяться в просторі імен System.IO:

Stream - абстрактний базовий клас для всіх потоків, визначає основні операції: читання (Read), запис (Write), позиціонування (Seek), очищення буфера (Flush) та закриття (Close).

FileStream – довільний (прямий) доступ до файлу, який предствлений як потік байтів.

MemoryStream довільний доступ до потоку байтів у оперативній памяті. Корисний для тимчасового зберігання та обробки даних.

NetworkStream представляє потік для роботи з мережевими з'єднаннями (TCP/IP). Дозволяє передавати дані через мережу.

Основні класи простіру імен System.IO

BufferedStream – тимчасове зберігання потоку байтів, що може підвищити продуктивність операцій вводу-виводу, особливо при частих невеликих операціях.

Console.In та Console.Out - статичні властивості класу Console, які представляють стандартний потік введення та виведення (клавіатура - консоль).

StringReader та StringWriter - класи, які дозволяють розглядати рядки як потоки для читання та запису відповідно.

BinaryReader та BinaryWriter - для читання та запису двійкових даних у потоки.

StreamReader та StreamWriter – класи для читання та запису текстових даних у потоки з підтримкою різних кодувань. Довільний доступ не підтримується

Робота з потоками

Щоб створити байтовий потік, пов'язаний з файлом, створюється об'єкт класу FileStream. При цьому в класі визначено декілька конструкторів. Найчастіше використовується конструктор, який відкриває потік для читання і/або запису:
FileStream(string filename, FileMode mode).

Інша версія конструктора дозволяє обмежити доступ тільки читанням або тільки записом:
FileStream (string filename, FileMode mode, FileAccess how)

Робота з потоками

У .NET Framework класи StreamReader та StreamWriter надають зручні засоби для читання та запису текстових даних у файли. Вони забезпечують буферизований потік, що підвищує продуктивність операцій введення-виведення.

Основні методи класу StreamReader

StreamReader(string path)	Конструктор, який створює новий екземпляр StreamReader для вказаного шляху до файлу.
StreamReader(string path, Encoding encoding)	Конструктор, який дозволяє вказати кодування файлу (наприклад, Encoding.UTF8, Encoding.Unicode).
ReadLine()	Зчитує один рядок з потоку та повертає його у вигляді рядка. Повертає null, якщо досягнуто кінець потоку.
ReadToEnd()	Зчитує весь вміст потоку, починаючи з поточної позиції до кінця потоку, та повертає його у вигляді одного рядка.
Read()	Зчитує наступний символ з потоку та повертає його як ціле число (його Unicode-значення) або -1, якщо досягнуто кінець потоку.
Read(char[] buffer, int index, int count)	Зчитує вказану кількість символів з поточного потоку та записує їх у буфер, починаючи з вказаного індексу. Повертає кількість зчитаних символів.
Peek()	Повертає наступний доступний символ, не переміщуючи поточну позицію в потоці. Повертає -1, якщо досягнуто кінець потоку.

Приклад

```
string filePath = "my_new_file.txt";

try
{
    using(FileStream fileStream = new FileStream(filePath, FileMode.Open, FileAccess.Read))
    using (StreamReader reader = new StreamReader(fileStream))
    {
        string line;
        while ((line = reader.ReadLine()) != null)
        {
            Console.WriteLine(line);
        }
    }
}
catch (IOException e)
{
    Console.WriteLine($"Помилка при читанні файлу: {e.Message}");
}
```

Приклад

```
using(FileStream fileStream = new FileStream(filePath, FileMode.Open, FileAccess.Read))
    using (StreamReader reader = new StreamReader(fileStream))
    {
        string line;
        while ((line = reader.ReadLine()) != null)
        {
            Console.WriteLine(line);
        }
    }
```

Клас надає базовий потік для файлів, дозволяючи читати та записувати байти у файл.

Клас, який використовується для читання символів з потоку. Він полегшує читання текстових даних, оскільки автоматично обробляє кодування символів.

Значення FileMode

FileMode.	Опис	Дія, якщо файл існує	Дія, якщо файл не існує	Дозволені операції
CreateNew	Створює новий файл.	IOException	Створює новий файл.	Читання та запис
Create	Створює новий файл. Якщо файл вже існує, він буде перезаписаний (вміст буде втрачено).	Перезаписує існуючий файл (вміст втрачається).	Створює новий файл.	Читання та запис
Open	Відкриває існуючий файл.	Відкриває існуючий файл.	FileNotFoundException.	Читання та запис
OpenOrCreate	Відкриває файл, якщо він існує. Якщо файл не існує, створює новий файл.	Відкриває існуючий файл.	Створює новий файл.	Читання та запис
Truncate	Відкриває існуючий файл і обрізає його вміст до нульової довжини.	Обрізає вміст існуючого файлу до 0 довжини.	FileNotFoundException.	Читання та запис
Append	Відкриває файл, якщо він існує, і встановлює позицію запису в кінець файлу. Якщо файл не існує, створює новий файл.	Відкриває існуючий файл та встановлює позицію в кінець.	Створює новий файл.	Лише запис (спроби змінити позицію ігноруються)

using

У C# конструкція using використовується для гарантованого звільнення некерованих ресурсів після їх використання, навіть якщо в блоці коду виникне виняток. У прикладі using застосовується для об'єктів FileStream та StreamReader, які працюють з файлами та є прикладами таких некерованих ресурсів.

Оператор using використовується для керування ресурсами, що реалізують інтерфейс Idisposable.

Оператор using гарантує, що метод Dispose() об'єкта буде викликано після завершення блоку using (нормально чи через виняток), і будуть звільнені зайняті ресурси (наприклад, закриває файл). Це дуже важливо для запобігання витоку ресурсів і проблем з доступом до файлів.

Обробка винятків

Винятки, пов'язані з роботою з файлами та каталогами:

IOException - базовий клас для всіх винятків, пов'язаних з помилками вводу/виводу:

FileNotFoundException - виникає, коли програма намагається отримати доступ до файлу, який не існує за вказаним шляхом;
DirectoryNotFoundException - виникає, коли програма намагається отримати доступ до каталогу, який не існує за вказаним шляхом;

Обробка виняткових ситуацій

try
{
Це блок обробки винятків. Блок try містить код, який може спричинити помилку (виняток) під час виконання. Якщо така помилка трапляється, виконання блоку try негайно припиняється, і керування передається блоку catch.
}

catch (IOException e)
{
Цей блок виконується лише в тому випадку, якщо в блоці try виник виняток типу IOException.
IOException - це клас винятків у C#, який представляє помилки, що виникають під час операцій введення-виведення (наприклад, при роботі з файлами, мережею тощо).
e змінна, яка представляє об'єкт винятку IOException, містить інформацію про помилку, включаючи повідомлення про помилку.

}

Основні методи класу StreamWriter

StreamWriter(string path)	Створює новий екземпляр StreamWriter для вказаного шляху до файлу. Якщо файл існує, його вміст буде перезаписано.
StreamWriter(string path, bool append)	Дозволяє вказати, чи потрібно додавати дані до існуючого файлу (append = true) чи перезаписувати його (append = false).
StreamWriter(string path, bool append, Encoding encoding)	Конструктор, який дозволяє вказати кодування файлу.

Основні методи StreamWriter

Write(char value)	Записує символ у потік.
Write(string value)	Записує рядок у потік.
Write(int value)	Записує ціле число у потік.
WriteLine(string value)	Записує рядок у потік та додає символ кінця рядка.
WriteLine()	Записує символ кінця рядка у потік
Flush()	Очищає всі буфери для поточного записувача та змушує всі буферизовані дані записатися у базовий потік
Close()	Закриває об'єкт StreamWriter та базовий потік. Записує всі буферизовані дані у потік перед закриттям. Звільняє всі пов'язані системні ресурси.
Dispose()	Звільняє некеровані ресурси, які використовуються об'єктом
StreamWriter	Рекомендується використовувати блок using для автоматичного виклику Dispose().

Приклад

```
string filePath = "output.txt";
string textToAppend = "Ми, дзвіночки лісові,\r\nПіднялися у траві.\r\nДінь-дінь-дінь! — в блакитний світ\r\nПосилаємо привіт.\r\nВолодимир Івасюк";

try
{
    using (StreamWriter writer = new StreamWriter(filePath, true, Encoding.UTF8))
    {
        writer.WriteLine(textToAppend);
        Console.WriteLine($"Дані додано до файлу '{filePath}'.");
    }
}

catch (IOException e)
{
    Console.WriteLine($"Виникла помилка при записі у файл: {e.Message}");
}
```

Робота з файлами та каталогами

Для забезпечення функціональності взаємодії з файловою системою, мова програмування C# надає ряд відповідних класів. Вони забезпечують можливість доступу до каталогів, маніпулювання файлами (включаючи відкриття, читання, запис, створення та переміщення) та виконання інших операцій, пов'язаних з файловою системою.

Основні класи для файлів та каталогів

Class Name	Usage
File	статичний клас, який надає різні функції, такі як копіювання, створення, переміщення, видалення, відкриття для читання або/запису, шифрування або розшифрування, перевірка існування файлу, додавання рядків або тексту до вмісту файлу, отримання часу останнього доступу тощо.
FileInfo	надає ті самі функції, що й статичний клас File, але більше контролю над тим, як виконувати операції читання/запису файлу, написавши код вручну для читання або запису байтів із файлу.
Directory	статичний клас, який забезпечує функціональність для створення, переміщення, видалення та доступу до підкаталогів.
DirectoryInfo	надає екземпляри методів для створення, переміщення, видалення та доступу до підкаталогів.
Path	статичний клас, який забезпечує такі функції, як отримання розширення файлу, зміна розширення файлу, отримання абсолютного фізичного шляху та інші функції, пов'язані зі шляхом.

Найбільш часто використовувані методи класу File

Exists(string path)	Визначає, чи існує вказаний файл. Повертає true або false.
Create(string path)	Створює файл за вказаним шляхом. Якщо файл вже існує, він буде перезаписаний. Повертає об'єкт FileStream для роботи з файлом.
Delete(string path)	Видаляє вказаний файл. Якщо файл не існує, виняток не генерується.
Copy(string sourceFileName, string destFileName)	Копіює вказаний файл до іншого місця. Дозволяє вказати, чи перезаписувати існуючий файл з таким самим ім'ям.
Move(string sourceFileName, string destFileName)	Переміщує вказаний файл до нового місця.
ReadAllText(string path)	Відкриває текстовий файл, зчитує всі рядки в один рядок і потім закриває файл.
WriteAllText(string path, string contents)	Створює новий файл, записує вказаний рядок у файл, а потім закриває файл. Якщо файл вже існує, його вміст буде перезаписано.
ReadAllLines(string path)	Відкриває текстовий файл, зчитує всі рядки у масив рядків, а потім закриває файл.

Найбільш часто використовувані методи класу File

WriteAllLines(string path, string[] contents)	Створює новий файл, записує масив рядків у файл, а потім закриває файл. Якщо файл вже існує, його вміст буде перезаписано.
AppendAllText(string path, string contents)	Додає вказаний рядок до кінця файлу. Якщо файл не існує, він буде створений.
AppendAllLines(string path, IEnumerable<string> contents)	Додає колекцію рядків до кінця файлу. Якщо файл не існує, він буде створений.
GetAttributes(string path)	Отримує атрибути вказаного файлу або каталогу.
SetAttributes(string path, FileAttributes attributes)	Встановлює атрибути для вказаного файлу або каталогу.
GetCreationTime(string path)	Отримує дату й час створення вказаного файлу або каталогу.
GetLastAccessTime(string path)	Отримує дату й час останнього доступу до вказаного файлу або каталогу.
Open(string path, FileMode mode)	Відкриває файл за вказаним шляхом з вказаним режимом доступу
OpenRead(string path)	Відкриває існуючий файл для читання.
OpenWrite(string path)	Відкриває існуючий файл для запису.Якщо файл не існує, він буде створений.

Запис тексту у файл

File.WriteAllText() має два основних перевантаження:

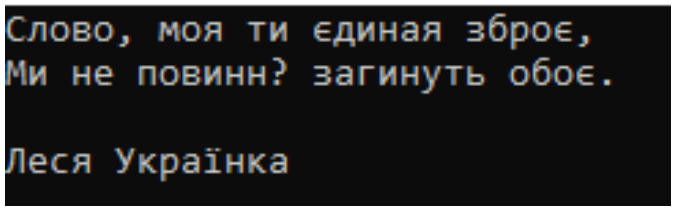
File.WriteAllText(string path, string contents) це перевантаження приймає шлях до файлу та вміст рядка для запису. За замовчуванням використовується кодування UTF-8.

File.WriteAllText(string path, string contents, Encoding encoding) це перевантаження дозволяє вказати кодування, яке буде використовуватися під час запису файлу (наприклад, UTF-8, UTF-16, ASCII).

Читання тексту з файлу

```
string filePath = "my_new_file.txt";
try
{
    string[] lines = System.IO.File.ReadAllLines(filePath);
    if (lines != null && lines.Length > 0)
        foreach (string line in lines)
            Console.WriteLine(line);
    else
        Console.WriteLine($"Файл '{filePath}' порожній або не існує.");
}

catch (FileNotFoundException){
    Console.WriteLine($"Файл '{filePath}' не знайдено.");
}
catch (IOException e){
    Console.WriteLine($"Виникла помилка при читанні файлу: {e.Message}");
}
catch (Exception e){
    Console.WriteLine($"Виникла непередбачена помилка: {e.Message}");
}
```



Запис тексту у файл

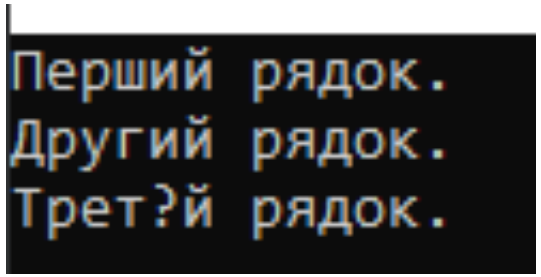
File.WriteAllLines(string path, IEnumerable<string> contents) - це статичний метод класу File використовується для запису колекції рядків у файл. Кожен рядок у колекції буде записано у файл як окремий рядок.

Якщо файл за вказаним шляхом вже існує, його вміст буде повністю перезаписано.

За замовчуванням використовує кодування UTF-8. Існує також перевантаження, що дозволяє вказати кодування - File.WriteAllLines(string path, IEnumerable<string> contents, Encoding encoding).

Запис тексту у файл

Запис рядків у файл (перезапис файлу):




```
try
{
    string[] linesToWrite = { "Перший рядок.", "Другий рядок.", "Третій рядок." };
    File.WriteAllLines("output.txt", linesToWrite);

}

catch (IOException ex)
{
    Console.WriteLine($"Помилка при записі у файл: {ex.Message}");
}
```

Запис та читання

```
try
{
    File.WriteAllText("output.txt", "Добридень, люди! Вбитий на снігу,\r\nя спам'ятався в зоряній пустелі." +
        "\r\nІ все, що має на землі вагу,\r\nосипалось, як мертві імортелі." +
        "\r\nТам, на землі, щось падало, цвіло,\r\nбуло рожеве, синє і зелене.");

    string[] lines = File.ReadAllLines("output.txt");
    Console.WriteLine("Вміст файлу (по рядках):");
    foreach (string line in lines)
        Console.WriteLine(line);

}
catch (IOException ex)
{
    Console.WriteLine($"Помилка при записі у файл: {ex.Message}");
}
```

```
Вм?ст файлу (по рядках):
Добридень, люди! Вбитий на сн?гу,
я спам'ятався в зорян?й пустел?.
? все, що має на земл? вагу,
осипалось, як мертв? ?мортел?.
Там, на земл?, щось падало, цв?ло,
було рожеве, синє ? зелене.
```

Запис та додавання тексту у файл

File.AppendAllText(string path, string contents) - це статичний метод класу File використовується для додавання вказаного рядка в кінець існуючого файлу.

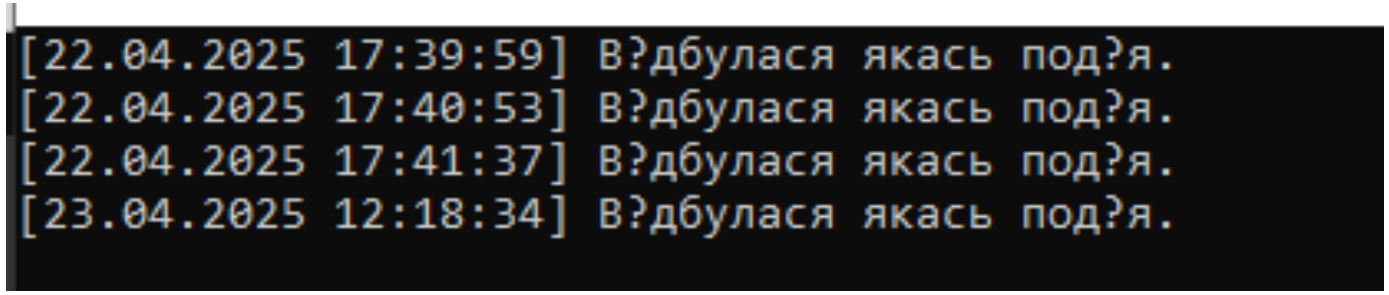
Якщо файл за вказаним шляхом існує, вказаний рядок буде додано в кінець файлу.
Якщо файл за вказаним шляхом не існує, метод створить новий файл і запише в нього вказаний рядок.
За замовчуванням використовує кодування UTF-8. Існує також перевантаження, що дозволяє вказати кодування - File.AppendAllText(string path, string contents, Encoding encoding).

Запис тексту у файл

Додавання тексту до файлу

```
File.AppendAllText("log.txt", $"[{DateTime.Now}] Відбулася якась подія.\n");
```

```
string[] lin = File.ReadAllLines("log.txt");
foreach (string line in lin)
    Console.WriteLine(line);
```



Клас FileInfo

Клас FileInfo у C# надає властивості та методи для виконання операцій з файлами.

На відміну від статичного класу File, методи FileInfo є методами екземпляра, тобто для їх використання спочатку потрібно створити об'єкт класу FileInfo.

Клас FileInfo (File)

AppendText()	Створює об’єкт типу StreamWriter і додає текст у файл
CopyTo()	Копіює файл
Create()	Створює новий файл і повертає об’єкт FileStream для взаємодії зі створеним файлом
CreateText()	Створює об’єкт типу StreamWriter з метою запису текстової інформації
Delete()	Видаляє файл
Directory	Отримує батьківський каталог у вигляді об’єкта DirectoryInfo
DirectoryName	Отримує повний шлях до батьківського каталогу

Клас FileInfo (File)

Length	Отримує розмір поточного файла або каталога
MoveTo()	Переміщує вказаний файл з можливістю зміни назви файла
Name	Отримує назву файла
Open()	Відкриває файл у заданому режимі (повертає об'єкт FileStream)
OpenRead()	Створює доступний тільки для читання об'єкт FileStream
OpenText()	Створює об'єкт StreamReader для читання текстової інформації з файла
OpenWrite()	Створює доступний тільки для запису об'єкт FileStream

Приклад

```
string filePath = "my_document.txt";

FileInfo fileInfo = new FileInfo(filePath);

try
 {
    using (FileStream fs = fileInfo.Create())
    {
       Console.WriteLine($"Файл створено: {fileInfo.FullName}");

       string content = "Я піду в далекі гори\r\nНа широкі полонини,\r\nI попрошу вітру зворів,\r\nАби він не спав до днини.";
       byte[] info = new System.Text.UTF8Encoding(true).GetBytes(content);
       fs.Write(info, 0, info.Length);
    }
 }
 catch (IOException e)
 {
    Console.WriteLine($"Помилка при створенні файлу: {e.Message}");
 }
```

```
Файл створено: C:\Users\Gala\source\repos\Lec_13\task_01\bin\Debug\my_document.txt
```

Приклад

```
// Перевірка, чи існує файл

if (fileInfo.Exists)
{
    Console.WriteLine($"Файл існує: {fileInfo.FullName}");
    Console.WriteLine($"Розмір файлу: {fileInfo.Length} байт");
    Console.WriteLine($"Дата створення: {fileInfo.CreationTime}");
    Console.WriteLine($"Дата останнього доступу: {fileInfo.LastAccessTime}");
    Console.WriteLine($"Дата останнього запису: {fileInfo.LastWriteTime}");
    Console.WriteLine($"Атрибути файлу: {fileInfo.Attributes}");
    Console.WriteLine($"Ім'я файлу: {fileInfo.Name}");
    Console.WriteLine($"Розширення файлу: {fileInfo.Extension}");
    Console.WriteLine($"Каталог файлу: {fileInfo.DirectoryName}");
    Console.WriteLine($"Об'єкт каталогу: {fileInfo.Directory}");
```

Приклад

Результат:

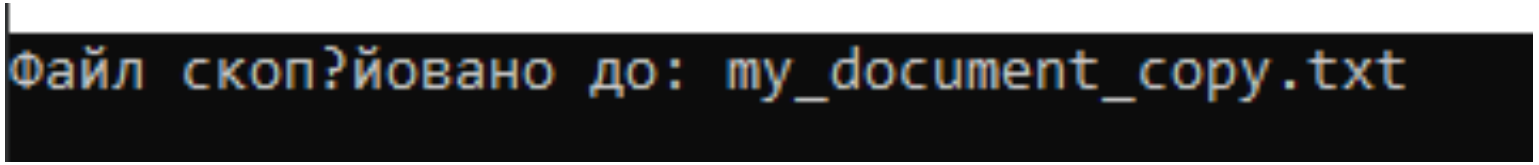
```
Файл ?снує: C:\Users\Gala\source\repos\Lec_13\task_01\bin\Debug\my_document.txt
Розм?р файлу: 163 байт
Дата створення: 23.04.2025 12:31:14
Дата останнього доступу: 23.04.2025 12:31:14
Дата останнього запису: 23.04.2025 12:31:14
Атрибути файлу: Archive, NotContentIndexed
?м'я файлу: my_document.txt
Розширення файлу: .txt
Каталог файлу: C:\Users\Gala\source\repos\Lec_13\task_01\bin\Debug
Об'єкт каталогу: C:\Users\Gala\source\repos\Lec_13\task_01\bin\Debug
```

Приклад

```
    // Копіювання файлу
string copyPath = "my_document_copy.txt";

try
{
    fileInfo.CopyTo(copyPath, true);
                        // true - дозволяє перезаписувати, якщо файл існує
    Console.WriteLine($"Файл скопійовано до: {copyPath}");
}

catch (IOException e)
{
    Console.WriteLine($"Помилка при копіюванні файлу: {e.Message}");
}
```



Приклад

```
    string newPath = "шлях до файлу\\my_document2.txt/my_document.txt";
try
{
    // Перевірка, чи існує каталог для переміщення
    Directory.CreateDirectory(Path.GetDirectoryName(newPath));
    fileInfo.MoveTo(newPath);
    Console.WriteLine($"Файл переміщено до: {newPath}");

    // Оновлення об'єкта fileInfo після переміщення
    fileInfo = new FileInfo(newPath);
    Console.WriteLine($"Новий повний шлях файлу: {fileInfo.FullName}");
}
catch (IOException ex)
{
    Console.WriteLine($"Помилка при переміщенні файлу: {ex.Message}");
}
```

```
Файл перем?щено до: C:\Users\Gala\source\repos\Lec_13\my_document2.txt/my_document.txt
Новий повний шлях файлу: C:\Users\Gala\source\repos\Lec_13\my_document2.txt\my_document.txt
```

Приклад

```
 // Видалення файлу
try
{
    fileInfo.Delete();
    Console.WriteLine($"Файл видалено: {fileInfo.FullName}");
}
catch (IOException e)
{
    Console.WriteLine($"Помилка при видаленні файлу: {e.Message}");
}
```

Серіалізація та десеріалізація в C#

Серіалізація - це процес перетворення об'єкта в потік байтів для збереження його в пам'яті, файлі, базі даних або передачі через мережу.

Десеріалізація - це зворотний процес, який відновлює об'єкт з потоку байтів.Ці механізми є ключовими для зберігання стану об'єктів, обміну даними між різними частинами програми або між різними додатками.

Серіалізація та десеріалізація в C#

C# надає кілька способів для серіалізації та десеріалізації об'єктів:

1. Бінарна серіалізація (BinaryFormatter). Серіалізує об'єкт у двійковому форматі, включає в себе інформацію про тип об'єкта, його структуру та дані. Ефективна за розміром та швидкістю, але менш сумісна між різними версіями .NET Framework або іншими платформами.
2. XML серіалізація (XmlSerializer). Серіалізує об'єкт у форматі XML. Не потребує атрибута [Serializable], але вимагає наявності публічного конструктора без параметрів та публічних властивостей/полів для серіалізації.
3. JSON серіалізація (System.Text.Json). Серіалізує об'єкт у форматі JSON (JavaScript Object Notation).

Робота з JSON-файли

У C# робота з JSON-файлами є дуже поширеною задачею, особливо при розробці веб-сервісів, конфігураційних файлів та обміні даними.

.NET надає кілька способів для серіалізації (перетворення об'єктів C# в JSON) та десеріалізації (перетворення JSON в об'єкти C#).

Найпопулярнішим і рекомендованим способом є використання бібліотеки System.Text.Json, яка є вбудованою в .NET Core 3.0+ та .NET 5+.

Приклад 1

```
public class PersonJson{
    public string Name { get; set; }
    public int Age { get; set; }
}

public class JsonSerializerExample{
    public static void Serialize(string filePath, PersonJson person)
    {
        string jsonString = JsonSerializer.Serialize(person);
        File.WriteAllText(filePath, jsonString);
        Console.WriteLine("Об'єкт серіалізовано в JSON.");
    }

    public static PersonJson Deserialize(string filePath){
        string jsonString = File.ReadAllText(filePath);
        PersonJson person = JsonSerializer.Deserialize<PersonJson>(jsonString);
        Console.WriteLine("Об'єкт десеріалізовано з JSON.");
        return person;
    }
}
```

Перетворення об'єкту person у JSON-формат

перетворення JSON-рядка в об'єкт типу Person

Приклад 1

```
Console.OutputEncoding = System.Text.Encoding.UTF8;

PersonJson personToSerialize = new PersonJson { Name = "Дональд", Age = 45 };
    string filePath = "person.json";

// Викликаємо статичний метод Serialize, вказуючи ім'я класу
JsonSerializerExample.Serialize(filePath, personToSerialize);

// Викликаємо статичний метод Deserialize, вказуючи ім'я класу
PersonJson deserializedPerson = JsonSerializerExample.Deserialize(filePath);
Console.WriteLine($"Ім'я: {deserializedPerson.Name}, Вік: {deserializedPerson.Age}"); ;
```

```
Об'єкт серіалізовано в JSON.
Об'єкт десеріалізовано з JSON.
Ім'я: Дональд, Вік: 45
```

Приклад 2

```
public class Person
{
    public string Name { get; set; }
    public int Age { get; set; }
    public List<string> Hobbies { get; set; }
}
public static void SerializeToJson()
{
    Person person = new Person
    {
        Name = "Арнольд Шварцнеггер",
        Age = 77,
        Hobbies = new List<string> { "Автомобілі", "Лижи", "Качалка" }
    };

    string jsonString = JsonSerializer.Serialize(person);

    File.WriteAllText("person.json", jsonString);
}
```

Перетворення об'єкту person у JSON-формат

Приклад 2

```
public static void DeserializeFromJson()
{
    string jsonStringFromFile = File.ReadAllText("person.json");

Person person = JsonSerializer.Deserialize<Person>(jsonStringFromFile);

    if (person != null)
    {
        Console.OutputEncoding = System.Text.Encoding.UTF8;
        Console.WriteLine($"Name: {person.Name}");
        Console.WriteLine($"Age: {person.Age}");
        Console.WriteLine($"Hobbies: {string.Join(", ", person.Hobbies)}");
    }
    else
    {
        Console.WriteLine("Помилка десеріалізації.");
    }
}
```

Читання всього вмісту файлу person.json у рядок jsonStringFromFile.

перетворення JSON-рядка jsonStringFromFile назад в об'єкт типу Person

```
Name: Арнольд Шварцнеггер
Age: 77
Hobbies: Автомобілі, Лижи, Качалка
```

Приклад 3

```
public class PersonXml{
    public string Name { get; set; }
    public int Age { get; set; }
}
public class XmlSerializerExample{
    public static void Serialize(string filePath, PersonXml person){
        XmlSerializer serializer = new XmlSerializer(typeof(PersonXml));
        using (FileStream fileStream = new FileStream(filePath, FileMode.Create)){
            serializer.Serialize(fileStream, person);
            Console.WriteLine("Об'єкт серіалізовано в XML.");
        }
    }
    public static PersonXml Deserialize(string filePath){
        XmlSerializer serializer = new XmlSerializer(typeof(PersonXml));
        using (FileStream fileStream = new FileStream(filePath, FileMode.Open)){
            PersonXml person = (PersonXml)serializer.Deserialize(fileStream);
            Console.WriteLine("Об'єкт десеріалізовано з XML.");
            return person;
        }
    }
}
```

Приклад 3

```
static void Main(string[] args)
{
    PersonXml personToSerialize = new PersonXml { Name = "Дональд", Age = 45 };
    string filePath = "person.xml";

    XmlSerializerExample.Serialize(filePath, personToSerialize);

    PersonXml deserializedPerson = XmlSerializerExample.Deserialize(filePath);
    Console.WriteLine($"Ім'я: {deserializedPerson.Name}, Вік: {deserializedPerson.Age}");
}
```