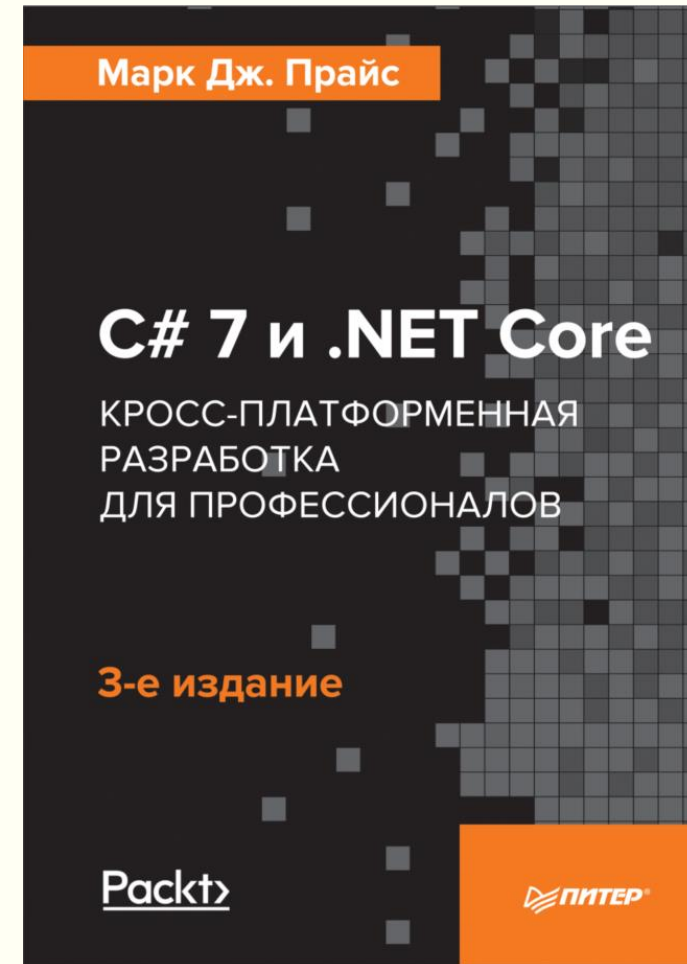
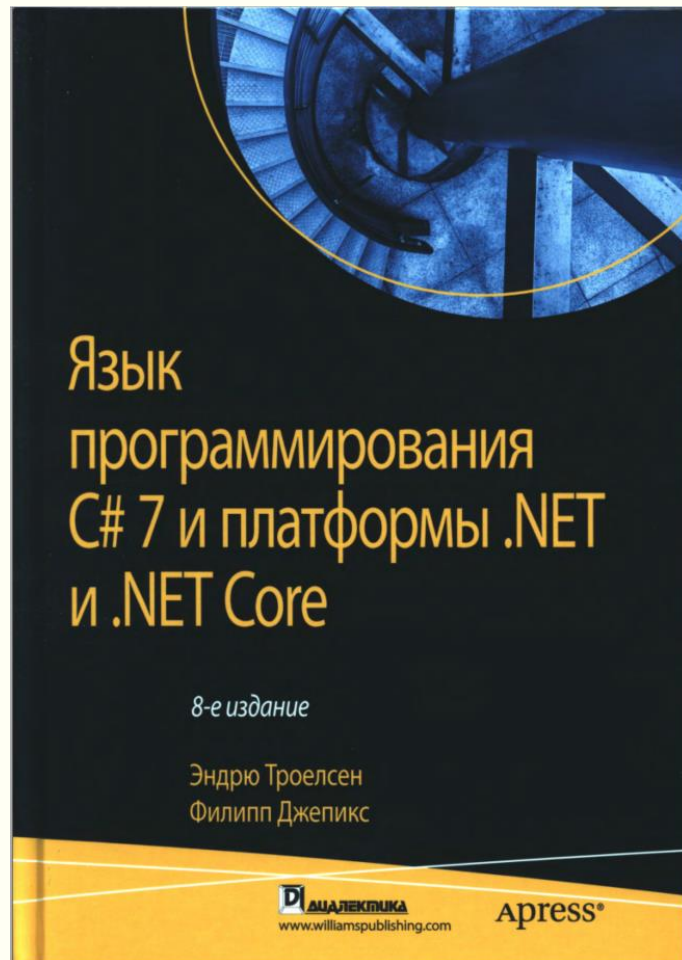




РОБОТА З ФАЙЛАМИ ТА ПОТОКАМИ ДАНИХ

Питання 1.3.

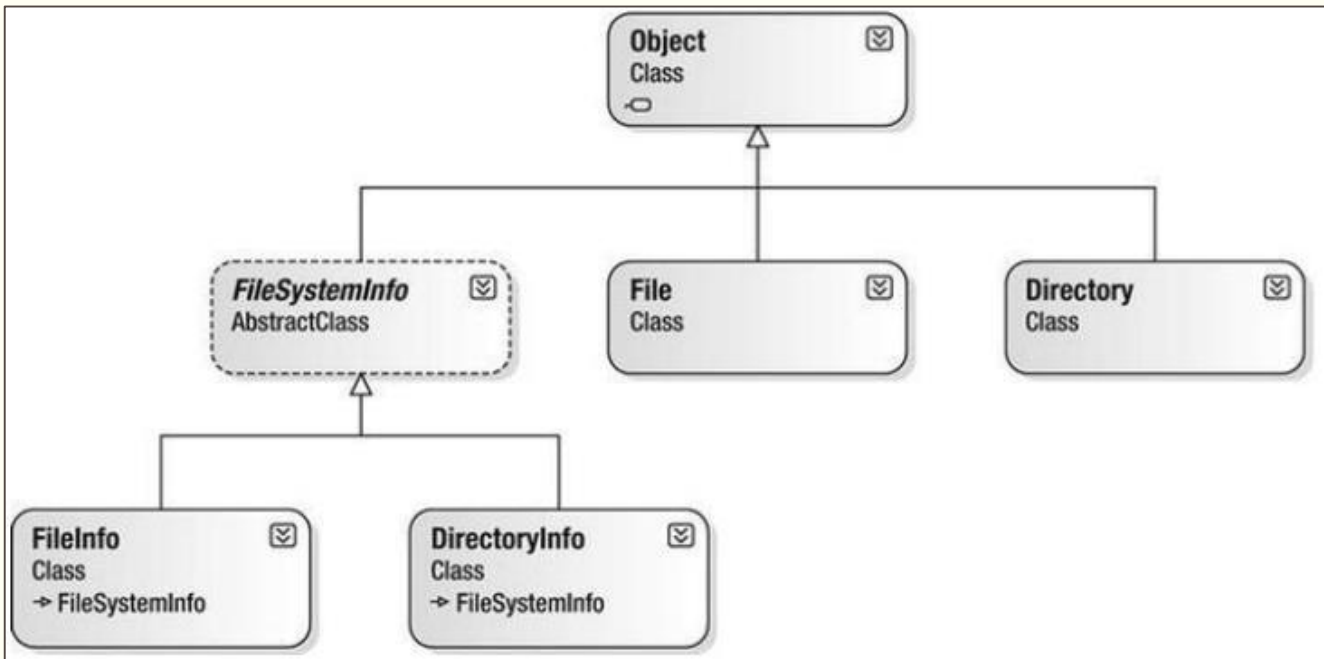
Література



Простір System.io

- Область бібліотек базових класів, присвячена службам файлового вводу-виводу та вводу-виводу з пам'яті.
 - У System.IO визначено набір класів, інтерфейсів, перелічень, структур та делегатів, більшість з яких знаходяться в mscorlib.dll.
 - На додачу, в збірці System.dll визначені додаткові члени System.IO.
 - У всіх проектах Visual Studio автоматично встановлюються посилання на обидві збірки.
- Багато типів з простору імен System.IO концентруються на програмній маніпуляції фізичними каталогами та файлами.
 - Додаткові типи надають підтримку читання/запису даних у рядкові буфери та області пам'яті.
- Члени простору System.io (.NET Framework 4.8)
- Члени простору System.io (.NET 5.0 Preview 7)

Класи Directory (DirectoryInfo) та File (FileInfo)



- Класи **Directory** та **File** пропонують операції створення, видалення, копіювання та переміщення з використанням різних статичних членів.
 - Класи **FileInfo** й **DirectoryInfo** забезпечують схожу функціональність у вигляді методів рівня екземпляра (тому повинні розміщуватись у пам'яті за допомогою ключового слова **new**).
- Зазвичай **FileInfo** й **DirectoryInfo** доречніші для отримання повних деталей щодо файлу чи каталогу (наприклад, часу створення, можливості читання/запису тощо), оскільки їх члени повертають строго типізовані об'єкти.
 - Члени класів **Directory** та **File**, як правило, повертають прості рядкові значення.
 - Проте в багатьох випадках роботу можна виконати, використовуючи **File/FileInfo** або **Directory/DirectoryInfo**.

Використання класу DirectoryInfo

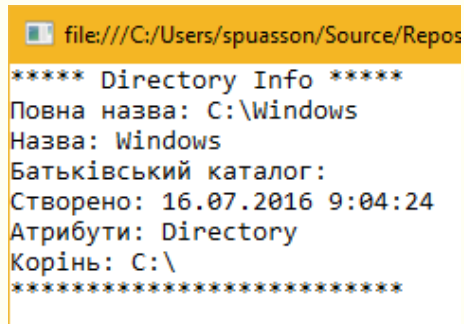
- Класи DirectoryInfo та FileInfo успадковують значну частину своєї поведінки від абстрактного базового класу FileSystemInfo.
 - Тип DirectoryInfo містить набір членів для створення, переміщення, видалення та перелічення каталогів і підкаталогів.
 - На додачу до успадкованої від FileSystemInfo функціональності, клас DirectoryInfo пропонує наступні ключові члени

| Член | Описание |
|------------------------------------|---|
| Create () CreateSubdirectory () | Создает каталог (или набор подкаталогов) по заданному путевому имени |
| Delete () | Удаляет каталог и все его содержимое |
| GetDirectories () | Возвращает массив объектов DirectoryInfo, представляющих все подкаталоги в текущем каталоге |
| GetFiles () | Извлекает массив объектов FileInfo, представляющий набор файлов в заданном каталоге |
| MoveTo () | Перемещает каталог со всем содержимым по новому пути |
| Parent | Извлекает родительский каталог данного каталога |
| Root | Получает корневую часть пути |

Використання класу DirectoryInfo. Статичні атрибути

```
class Program {
    static void Main(string[] args) {
        ShowWindowsDirectoryInfo();
        Console.ReadLine();
    }

    static void ShowWindowsDirectoryInfo() {
        // Dump directory information.
        DirectoryInfo dir = new DirectoryInfo(@"C:\Windows");
        Console.WriteLine("***** Directory Info *****");
        Console.WriteLine("Повна назва: {0}", dir.FullName);
        Console.WriteLine("Назва: {0}", dir.Name);
        Console.WriteLine("Батьківський каталог: {0}", dir.Parent);
        Console.WriteLine("Створено: {0}", dir.CreationTime);
        Console.WriteLine("Атрибути: {0}", dir.Attributes);
        Console.WriteLine("Корінь: {0}", dir.Root);
        Console.WriteLine("*****\n");
    }
}
```

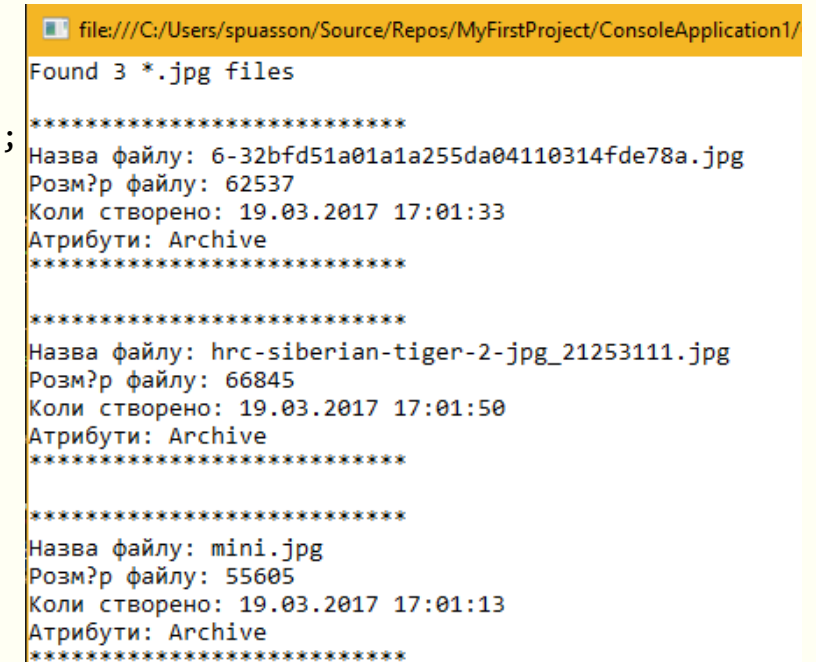


```
file:///C:/Users/spuasson/Source/Repos
***** Directory Info *****
Повна назва: C:\Windows
Назва: Windows
Батьківський каталог:
Створено: 16.07.2016 9:04:24
Атрибути: Directory
Корінь: C:\
*****
```

- Робота з типом DirectoryInfo починається із вказівки деякого шляху в якості параметру конструктора.
 - Шлях до поточного каталогу = ".".
 - DirectoryInfo dir1 = new DirectoryInfo(".");
 - DirectoryInfo dir2 = new DirectoryInfo(@"C:\Windows");
- При спробі взаємодії з неіснуючим каталогом генерується виняток System.IO.DirectoryNotFoundException.
 - Щоб вказати ще не створений каталог, доведеться викликати метод Create():
 - DirectoryInfo dir3 = new DirectoryInfo(@"C:\MyCode\Testing"); dir3.Create();

Використання класу DirectoryInfo. Перелічення файлів

```
DirectoryInfo dir = new DirectoryInfo(@"C:\Wallpaper");
// Отримати всі файли з розширенням *.jpg.
FileInfo[] imageFiles = dir.GetFiles("*.jpg", SearchOption.AllDirectories);
// Скільки було знайдено?
Console.WriteLine("Found {0} *.jpg files\n", imageFiles.Length);
// Тепер виведемо інформацію про кожний файл.
foreach (FileInfo f in imageFiles)
{
    Console.WriteLine("*****");
    Console.WriteLine("Назва файлу: {0}", f.Name);
    Console.WriteLine("Розмір файлу: {0}", f.Length);
    Console.WriteLine("Коли створено: {0}", f.CreationTime);
    Console.WriteLine("Атрибути: {0}", f.Attributes);
    Console.WriteLine("*****\n");
}
```



file:///C:/Users/spuasson/Source/Repos/MyFirstProject/ConsoleApplication1/

Found 3 *.jpg files

Назва файлу: 6-32bfd51a01a1a255da04110314fde78a.jpg
Розм?р файлу: 62537
Коли створено: 19.03.2017 17:01:33
Атрибути: Archive

Назва файлу: hrc-siberian-tiger-2-jpg_21253111.jpg
Розм?р файлу: 66845
Коли створено: 19.03.2017 17:01:50
Атрибути: Archive

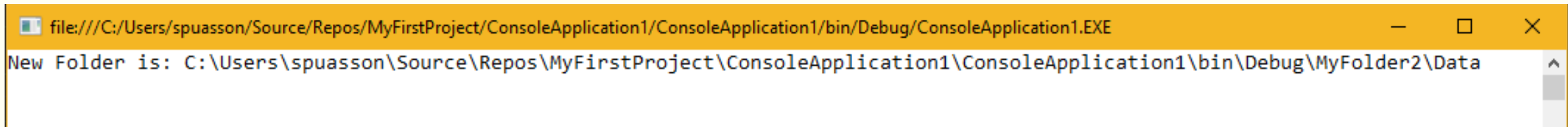
Назва файлу: mini.jpg
Розм?р файлу: 55605
Коли створено: 19.03.2017 17:01:13
Атрибути: Archive

- Метод `GetFiles()` повертає масив об'єктів типу `FileInfo`, кожний з яких відображає детальну інформацію щодо конкретного файлу.
 - Зверніть увагу на опцію пошуку `SearchOption.AllDirectories` у виклику `GetFiles()`, яка забезпечує перегляд усіх підкаталогів кореня.

Використання класу DirectoryInfo. Створення підкаталогів

```
static void ModifyAppDirectory() {  
    DirectoryInfo dir = new DirectoryInfo(".");  
    // Створення \MyFolder  
    dir.CreateSubdirectory("MyFolder");  
    // Перехоплення об'єкту DirectoryInfo, що повертається  
    DirectoryInfo myDataFolder =  
        dir.CreateSubdirectory(@"MyFolder2\Data");  
    // Друкує шлях ..\MyFolder2\Data.  
    Console.WriteLine("New Folder is: {0}", myDataFolder);  
}
```

- Використовується метод `DirectoryInfo.CreateSubdirectory()`.
 - Хоч отримувати значення від методу `CreateSubdirectory()` не обов'язково, майте на увазі, що при успішному виконанні повертається об'єкт `DirectoryInfo`, що представляє створений елемент.

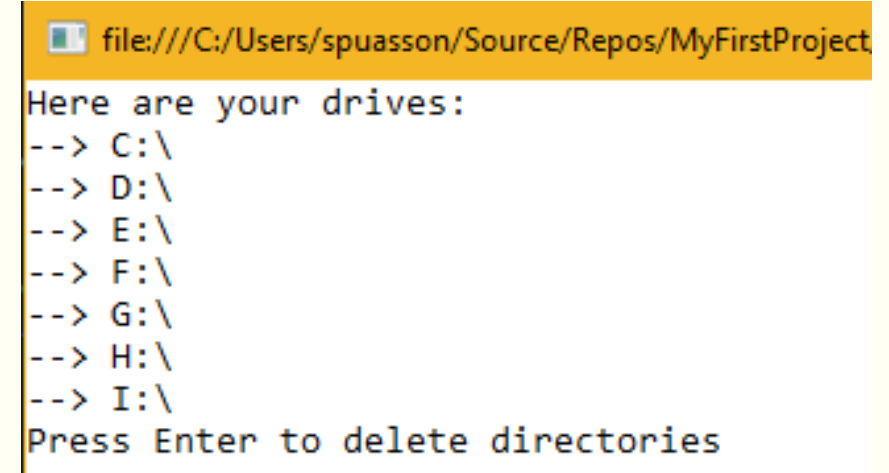


The screenshot shows a Windows command prompt window with a yellow title bar. The title bar text is "file:///C:/Users/spuasson/Source/Repos/MyFirstProject/ConsoleApplication1/ConsoleApplication1/bin/Debug/ConsoleApplication1.EXE". The command prompt area has a white background and shows the output "New Folder is: C:\Users\spuasson\Source\Repos\MyFirstProject\ConsoleApplication1\ConsoleApplication1\bin\Debug\MyFolder2\Data". The text is in a monospaced font, with the path parts colored in blue and red. There is a small upward arrow icon on the right side of the command prompt area.

Робота з типом Directory

```
static void FunWithDirectoryType() {  
    // Вивести список усіх дискових пристроїв комп'ютера.  
    string[] drives = Directory.GetLogicalDrives();  
    Console.WriteLine("Here are your drives:");  
    foreach (string s in drives)  
        Console.WriteLine("--> {0} ", s);  
  
    // Видалити те, що було створено.  
    Console.WriteLine("Press Enter to delete directories");  
    Console.ReadLine();  
    try {  
        Directory.Delete(@"C:\MyFolder");  
        // Другий параметр задає,  
        // чи хочете видаляти підкаталоги.  
        Directory.Delete(@"C:\MyFolder2", true);  
    } catch (IOException e) {  
        Console.WriteLine(e.Message);  
    }  
}
```

- члени Directory зазвичай повертають рядкові дані замість строго типізованих об'єктів FileInfo/DirectoryInfo.



```
file:///C:/Users/spuasson/Source/Repos/MyFirstProject  
Here are your drives:  
--> C:\  
--> D:\  
--> E:\  
--> F:\  
--> G:\  
--> H:\  
--> I:\  
Press Enter to delete directories
```

Робота з типом DriveInfo

```
Console.WriteLine("***** Fun with DriveInfo *****\n");
// Get info regarding all drives.
DriveInfo[] myDrives = DriveInfo.GetDrives();
// Now print drive stats.
foreach (DriveInfo d in myDrives) {
    Console.WriteLine("Name: {0}", d.Name);
    Console.WriteLine("Type: {0}", d.DriveType);
    // Check to see whether the drive is mounted.
    if (d.IsReady) {
        Console.WriteLine("Free space: {0}", d.TotalFreeSpace);
        Console.WriteLine("Format: {0}", d.DriveFormat);
        Console.WriteLine("Label: {0}", d.VolumeLabel);
    }
    Console.WriteLine();
}
```

- Аналогічно до `Directory.GetLogicalDrives()`, статичний метод `DriveInfo.GetDrives()` дозволяє отримати назви пристроїв на машині.
 - На відміну від `Directory.GetLogicalDrives()`, клас `DriveInfo` надає багато додаткових деталей (тип пристрою, доступний вільний простір, мітка тому).

```
file:///C:/Users/spuasson/Source/Repos/MyFirst
***** Fun with DriveInfo *****

Name: C:\
Type: Fixed
Free space: 4381319168
Format: NTFS
Label:

Name: D:\
Type: Fixed
Free space: 4193439744
Format: NTFS
Label: Новый том

Name: E:\
Type: Fixed
Free space: 10515169280
Format: NTFS
Label: Learning

Name: F:\
Type: Fixed
Free space: 965853184
Format: NTFS
Label:

Name: G:\
Type: Fixed
Free space: 3739521024
Format: NTFS
Label: PhD

Name: H:\
Type: Fixed
Free space: 4997697536
Format: NTFS
Label: Библиотека
```

Робота з класом FileInfo

- Клас FileInfo дозволяє отримувати детальні відомості про існуючі файли на диску (час створення, розмір, атрибути та ін.) та допомагає створювати, копіювати, переміщати та видаляти файли.
 - На додачу до набору функціональності, успадкованої від FileSystemInfo, клас FileInfo має ряд унікальних членів:

| Член | Опис |
|--------------|--|
| AppendText() | Створює об'єкт StreamWriter (розглянемо в наступному питанні) та додає текст у файл |
| CopyTo() | Копіює існуючий файл у новий файл |
| Create() | Створює новий файл та повертає об'єкт FileStream (розглянемо в наступному питанні) для взаємодії зі створеним файлом |
| CreateText() | Створює об'єкт StreamWriter, який записує новий текстовий файл |

Метод FileInfo.Create()

- Один із способів створення дескриптора файлу передбачає застосування методу FileInfo.Create().
 - Повернений ним об'єкт FileStream надає синхронну та асинхронну операції запису/зчитування для файлу всім користувачам.
 - Зверніть увагу, що після завершення роботи з поточним об'єктом FileStream необхідно закрити його дескриптор для вивільнення внутрішніх некерованих ресурсів потоку.
 - Враховуючи, що FileStream реалізує інтерфейс IDisposable, можна застосувати контекст using та дозволити компілятору згенерувати логіку завершення

```
// Створює новий файл на диску C.  
FileInfo f = new FileInfo(@"C:\Test.dat");  
FileStream fs = f.Create();  
// Використовує FileStream...  
// Закриває file stream.  
fs.Close();
```

```
FileInfo f = new FileInfo(@"C:\Test.dat");  
using (FileStream fs = f.Create())  
{  
    // Використовуємо об'єкт FileStream...  
}
```

Метод FileInfo.Open()

- Дозволяє відкривати існуючі файли, а також створювати нові файли з набагато вищою точністю, ніж FileInfo.Create() завдяки своїм параметрам, що описують внутрішню структуру цих файлів.
 - Виклик методу повертає об'єкт типу FileStream та в цій версії потребує 3 параметри.

```
// Створення нового файлу за допомогою FileInfo.Open().
FileInfo f2 = new FileInfo(@"C:\Test2.dat");
using (FileStream fs2 = f2.Open(FileMode.OpenOrCreate, FileAccess.ReadWrite, FileShare.None))
{
    // Використовуємо об'єкт типу FileStream...
}
```

- Перший параметр - загальний тип запиту вводу-виводу (створити новий файл, відкрити існуючий, дописати в файл тощо).

```
public enum FileMode
{
    CreateNew,
    Create,
    Open,
    OpenOrCreate,
    Truncate,
    Append
}
```

```
public enum FileAccess
{
    Read,
    Write,
    ReadWrite
}
```

```
public enum FileShare
{
    Delete,
    Inheritable,
    None,
    Read,
    ReadWrite,
    Write
}
```

Методи FileInfo.OpenRead() та FileInfo.OpenWrite()

- У класі FileInfo також передбачені методи OpenRead() та OpenWrite().
 - Повертають відповідним чином сконфігурований об'єкт типу FileStream без потреби у вказуванні різних значень перелічення.

```
// Отримуємо об'єкт FileStream з дозволами лише на зчитування.  
FileInfo f3 = new FileInfo(@"C:\Test3.dat");  
using (FileStream readOnlyStream = f3.OpenRead()) {  
    // Використовуємо об'єкт типу FileStream...  
}
```

```
// Отримуємо об'єкт FileStream з дозволами лише на запис.  
FileInfo f4 = new FileInfo(@"C:\Test4.dat");  
using (FileStream writeOnlyStream = f4.OpenWrite())  
{  
    // Використовуємо об'єкт типу FileStream...  
}
```

Методи FileInfo.OpenText(), FileInfo.CreateText() і FileInfo.AppendText()

- На відміну від попередніх методів, OpenText() повертає екземпляр типу StreamReader, а не FileStream.

- Оскільки файл boot.ini вже є на диску, отримуємо доступ до його вмісту так:

```
// Отримуємо об'єкт типу StreamReader.  
FileInfo f5 = new FileInfo(@"C:\boot.ini");  
using (StreamReader sreader = f5.OpenText())  
{  
    // Використовуємо StreamReader-об'єкт...  
}
```

- FileInfo.CreateText() і FileInfo.AppendText() повертають об'єкт StreamWriter:

```
FileInfo f6 = new FileInfo(@"C:\Test6.txt");  
using (StreamWriter swriter = f6.CreateText())  
{  
    // Використовуємо StreamWriter-об'єкт...  
}  
FileInfo f7 = new FileInfo(@"C:\FinalTest.txt");  
using (StreamWriter swriterAppend = f7.AppendText())  
{  
    // Використовуємо StreamWriter-об'єкт...  
}
```


Робота з типом File

```
// Отримати об'єкт FileStream через File.Create().  
using(FileStream fs = File.Create(@"C:\Test.dat")) {...}
```

```
// Отримати об'єкт FileStream через File.Open().  
using(FileStream fs2 = File.Open(@"C:\Test2.dat", FileMode.OpenOrCreate,  
FileAccess.ReadWrite, FileShare.None)) {...}
```

```
// Отримати об'єкт FileStream з правами тільки для зчитування.  
using(FileStream readOnlyStream = File.OpenRead(@"Test3.dat11)) {...}
```

```
// Отримати об'єкт FileStream з правами тільки для запису.  
using(FileStream writeOnlyStream = File.OpenWrite(@"Test4.dat")) {...}
```

```
// Отримати об'єкт StreamReader.  
using(StreamReader sreader = File.OpenText(@"C:\boot.ini11)) {...}
```

```
// Отримати кілька об'єктів StreamWriter.  
using(StreamWriter swriter = File.CreateText(@"C:\Test6.txt")) {...}  
using(StreamWriter swriterAppend = File.AppendText(@"C:\FinalTest.txt")) {...}
```

- Тип File надає функціональність, майже ідентичну до FileInfo, за допомогою кількох статичних методів.

- Подібно до FileInfo, тип File підтримує методи AppendText(), Create(), CreateText(), Open(), OpenRead(), OpenWrite() та OpenText().
- У багатьох випадках типи File і FileInfo взаємозамінні.

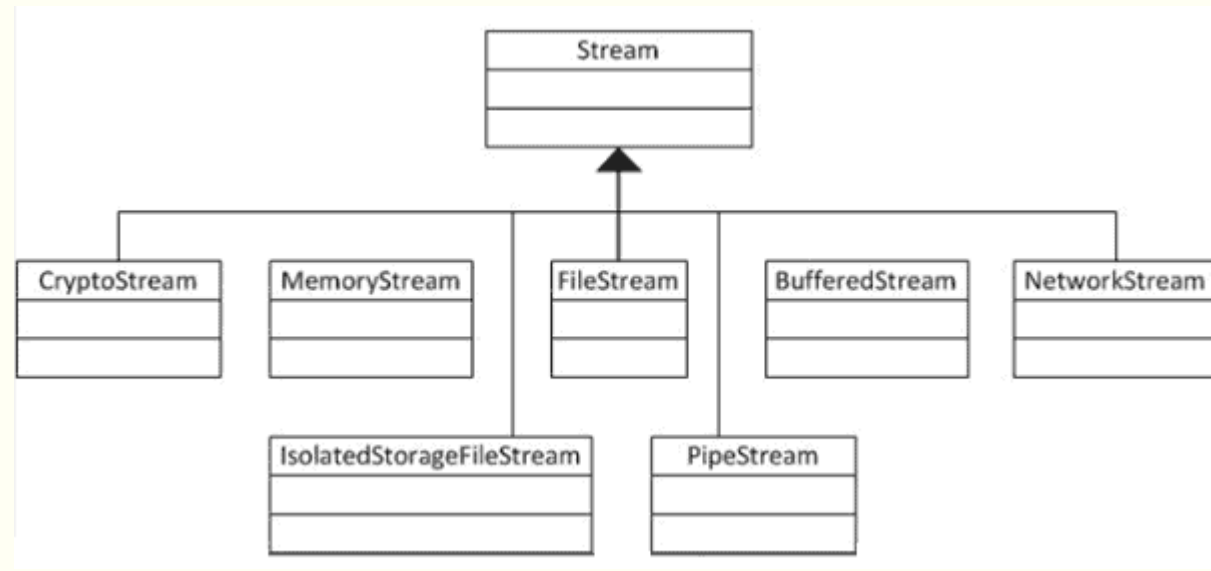
Додаткові члени File.

- Консольна програма зберігає рядкові дані в новому файлі на диску C: (і читає їх в пам'ять) з мінімальними зусиллями.
 - Коли необхідно швидко отримати файловий дескриптор, тип File дозволить зекономити на об'ємі коду.
 - Проте перевага попереднього створення об'єкта FileInfo пов'язана з можливістю дослідження файлу за допомогою членів абстрактного базового класу FileSystemInfo.

| Метод | Описание |
|------------------|--|
| ReadAllBytes () | Открывает указанный файл, возвращает двоичные данные в виде массива байт и затем закрывает файл |
| ReadAllLines () | Открывает указанный файл, возвращает символьные данные в виде массива строк, затем закрывает файл |
| ReadAllText () | Открывает указанный файл, возвращает символьные данные в виде System.String (), затем закрывает файл |
| WriteAllBytes () | Открывает указанный файл, записывает в него массив байтов и закрывает файл |
| WriteAllLines () | Открывает указанный файл, записывает в него массив строк и закрывает файл |
| WriteAllText () | Открывает указанный файл, записывает в него данные из указанной строки и закрывает файл |

```
Console.WriteLine("***** Simple I/O with the File Type *****\n");
string[] myTasks = {"Fix bathroom sink", "Call Dave",
                    "Call Mom and Dad", "Play Xbox One"};
// Записати всі дані в файл на диску C: .
File.WriteAllLines(@"C:\tasks.txt", myTasks);
// Прочитати всі дані та вивести на консоль.
foreach (string task in File.ReadAllLines(@"C:\tasks.txt"))
{
    Console.WriteLine("TODO: {0}", task);
}
Console.ReadLine();
```

Потоки вводу-виводу. Абстрактний клас Stream



- У галузі керування вводом-виводом **потік** (**stream**) представляє порцію даних, яка «протікає» від джерела (source) до цілі (target).
 - Потоки надають загальний спосіб взаємодії з послідовністю байтів, незалежно від того, якого роду пристрій (файл, мережа, з'єднання, принтер і т.п.) зберігає чи відображає ці байти.
- В абстрактному класі System.IO.Stream визначено набір членів, які забезпечують підтримку синхронної та асинхронної взаємодії зі сховищем (файлом, областю пам'яті тощо).
 - Нащадки класу Stream представляють дані як низькорівневі потоки байтів, а безпосередня робота з низькорівневими потоками може виявитись досить загадковою.
 - Деякі типи, успадковані від Stream, підтримують пошук, що означає можливість отримання й зміни поточної позиції в потоці.

Робота з класом FileStream

```
file:///C:/Users/spuasson/Source/Repos/MyFirstProject/ConsoleApplication1/Co
***** Fun with FileStreams *****

Your message as an array of bytes: 7210110810811133
Decoded Message: Hello!
```

```
Console.WriteLine("***** Fun with FileStreams *****\n");
// Получить объект FileStream.
using (FileStream fStream = File.Open(@"C:\myMessage.dat", FileMode.Create))
{
    // Закодировать строку в виде массива байт.
    string msg = "Hello!";
    byte[] msgAsByteArray = Encoding.Default.GetBytes(msg);
    // Записать byte[] в файл.
    fStream.Write(msgAsByteArray, 0, msgAsByteArray.Length);
    // Сбросить внутреннюю позицию потока.
    fStream.Position = 0;
    // Прочитать типы из файла и вывести на консоль.
    Console.Write("Your message as an array of bytes: ");
    byte[] bytesFromFile = new byte[msgAsByteArray.Length];
    for (int i = 0; i < msgAsByteArray.Length; i++) {
        bytesFromFile[i] = (byte)fStream.ReadByte();
        Console.Write(bytesFromFile[i]);
    }
    // Вывести декодированные сообщения.
    Console.Write("\nDecoded Message: ");
    Console.WriteLine(Encoding.Default.GetString(bytesFromFile));
}
Console.ReadLine();
```

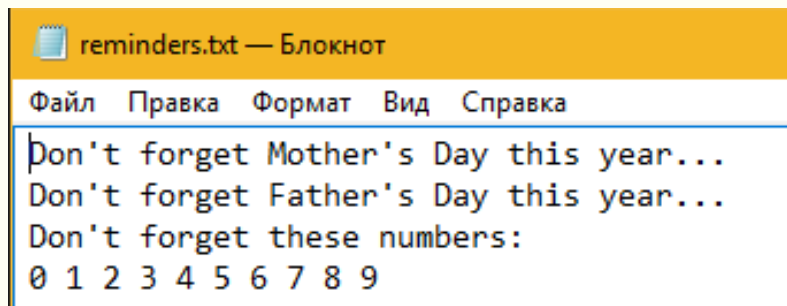
- Класс FileStream надає реалізацію абстрактного члена Stream в манері, доречній для потокової роботи з файлами.
 - Це елементарний потік, який може записувати чи зчитувати тільки 1 байт або масив байтів.
 - Проте взаємодіяти з членами типу FileStream доводиться нечасто.
 - Ймовірніше використання різноманітних оболонок потоків, які полегшують роботу з текстовими даними або типами .NET.
 - Для ілюстрації поекспериментуємо з можливостями синхронних зчитування / запису типу FileStream.

Робота з класами `StreamWriter` і `StreamReader`

- Класи `StreamWriter` та `StreamReader` зручні, коли потрібно читати або записувати символні дані.
 - Обидва типи працюють за умовчанням з символами `Unicode`; проте це можна змінити наданням правильно сконфігурованого посилання на об'єкт `System.Text.Encoding`.
 - Для простоти припустимо, що стандартне кодування `Unicode` нас влаштовує.
- Клас `StreamReader`, як і `StringReader`, успадкований від абстрактного класу `TextReader`.
 - Базовий клас пропонує дуже обмежений набір функціональності своїм нащадкам, зокрема — можливість читати й “заглядати” (`peek`) в символний потік.
- Клас `StreamWriter` (а також `StringWriter`) породжений від абстрактного базового класу `TextWriter`, в якому визначені члени, які дозволяють породженим типам записувати текстові дані в заданий символний потік.
 - Клас `StreamWriter` надає відповідну реалізацію методів `Write()`, `Close()` і `Flush()`, а також визначає додаткову властивість `AutoFlush`.
 - Коли ця властивість має значення `true`, вона змушує `StreamWriter` виштовхувати дані при кожній операції запису.
 - Встановивши `AutoFlush` в `false`, можна отримати вищу продуктивність, проте при цьому потрібно завжди викликати `Close()` після роботи з `StreamWriter`.

Запис у текстовий файл

```
Console.WriteLine("***** Fun with StreamWriter / StreamReader *****\n");
// Получить StreamWriter и записать строковые данные.
using (StreamWriter writer = File.CreateText("reminders.txt"))
{
    writer.WriteLine("Don't forget Mother's Day this year...");
    writer.WriteLine("Don't forget Father's Day this year...");
    writer.WriteLine("Don't forget these numbers:");
    for (int i = 0; i < 10; i++)
        writer.Write(i + " ");
    // Вставить новую строку.
    writer.WriteLine();
}
Console.WriteLine("Created file and wrote some thoughts...");
Console.ReadLine();
```



■ У прикладі створюється новий файл reminders.txt за допомогою методу File.CreateText().

- Використовуючи отриманий об'єкт StreamWriter, у новий файл будуть додані деякі текстові дані.

Читання з текстового файлу

- Тип `StreamReader` успадкований від абстрактного класу `TextReader`.

- Основні члени:

- Продовжимо попередній приклад:

```
// Прочитать данные из файла.  
Console.WriteLine("Here are your thoughts:\n");  
using (StreamReader sr = File.OpenText("reminders.txt"))  
{  
    string input = null;  
    while ((input = sr.ReadLine()) != null)  
    {  
        Console.WriteLine(input);  
    }  
}  
Console.ReadLine();
```

| Член | Описание |
|--------------------------|--|
| <code>Peek()</code> | Возвращает следующий доступный символ, не изменяя текущей позиции средства чтения. Значение <code>-1</code> указывает на достижение конца потока |
| <code>Read()</code> | Читает данные из входного потока |
| <code>ReadBlock()</code> | Читает указанное максимальное количество символов из текущего потока и записывает данные в буфер, начиная с заданного индекса |
| <code>ReadLine()</code> | Читает строку символов из текущего потока и возвращает данные в виде строки (строка <code>null</code> указывает на признак конца файла) |
| <code>ReadToEnd()</code> | Читает все символы от текущей позиции до конца потока и возвращает их в виде одной строки |

```
file:///C:/Users/spuasson/Source/Repos/MyFirstProject/ConsoleApplicati  
***** Fun with StreamWriter / StreamReader *****  
  
Created file and wrote some thoughts...  
  
Here are your thoughts:  
  
Don't forget Mother's Day this year...  
Don't forget Father's Day this year...  
Don't forget these numbers:  
0 1 2 3 4 5 6 7 8 9
```


Робота з класами StringWriter та StringReader

```
Console.WriteLine("***** Fun with StreamWriter / StreamReader *****\n");
// Получить StreamWriter и записать строковые данные.
using (StreamWriter writer = new StreamWriter("reminders.txt"))
{ ... }
// Прочитать данные из файла.
using (StreamReader sr = new StreamReader("reminders.txt"))
{ ... }
```

```
Console.WriteLine("***** Fun with StringWriter / StringReader *****\n");
// Создать StringWriter и записать символьные данные в память.
using (StringWriter strWriter = new StringWriter())
{
    strWriter.WriteLine("Don't forget Mother's Day this year...");
    // Получить копию содержимого (хранящегося в строке)
    // и вывести на консоль.
    Console.WriteLine("Contents of StringWriter:\n{0}", strWriter);
}
Console.ReadLine();
```

- Один результат можна отримати різними способами, використовуючи типи з простору імен System.IO.
 - Наприклад, було показано, що за допомогою методу CreateText() можна отримати об'єкт StreamWriter з типом File або FileInfo.
 - Альтернатива: пряме створення об'єктів StreamWriter і StreamReader.
- Класи StringWriter і StringReader можна використовувати для трактовки текстової інформації як потоку символів з пам'яті.
 - Корисно в випадках, коли потрібно додати символьну інформацію до основного буферу.
 - Для ілюстрації в прикладі блок рядкових даних записується в об'єкт StringWriter замість файлу на локальному диску.

Робота з класами StringWriter та StringReader

```
using (StringWriter strWriter = new StringWriter())
{
    strWriter.WriteLine("Don't forget Mother's Day this year...");
    Console.WriteLine("Contents of StringWriter:\n{0}", strWriter);
    // Получить внутренний StringBuilder.
    StringBuilder sb = strWriter.GetStringBuilder();
    sb.Insert(0, "Hey!! ");
    Console.WriteLine("-> {0}", sb.ToString());
    sb.Remove(0, "Hey!! ".Length);
    Console.WriteLine("-> {0}", sb.ToString());
}

using (StringWriter strWriter = new StringWriter()) {
    strWriter.WriteLine("Don't forget Mother's Day this year...");
    Console.WriteLine("Contents of StringWriter:\n{0}", strWriter);
    // Читать данные из StringWriter.
    using (StringReader strReader = new
        StringReader(strWriter.ToString())) {
        string input = null;
        while ((input = strReader.ReadLine()) != null)
        {
            Console.WriteLine(input);
        }
    }
}
```

- Оскільки і StringWriter, і StreamWriter породжені від одного базового класу, логіка запису в деякій мірі схожа.
 - Проте враховуючи природу StringWriter, клас дозволяє застосовувати метод GetStringBuilder() для витягування об'єкта System.Text.StringBuilder
- За потреби зчитування з потоку рядкових даних використовуйте відповідний тип StringReader, який функціонує ідентично до класу StreamReader.
 - Насправді, в класі StringReader лише заміщаються успадковані члени для зчитування з блоку символічних даних, а не з файлу

Робота з класами BinaryWriter і BinaryReader

```
Console.WriteLine("***** Fun with Binary Writers / Readers  
****\n");  
// Open a binary writer for a file.  
FileInfo f = new FileInfo("BinFile.dat");  
using (BinaryWriter bw = new BinaryWriter(f.OpenWrite()))  
{  
    // Print out the type of BaseStream.  
    // (System.IO.FileStream in this case).  
    Console.WriteLine("Base stream is: {0}", bw.BaseStream);  
    // Create some data to save in the file.  
    double aDouble = 1234.67;  
    int anInt = 34567;  
    string aString = "A, B, C";  
    // Write the data.  
    bw.Write(aDouble);  
    bw.Write(anInt);  
    bw.Write(aString);  
}  
Console.WriteLine("Done!");  
Console.ReadLine();
```

- BinaryWriter і BinaryReader – прямі нащадки System.Object.
 - Дозволяють зчитувати й записувати дискретні типи даних в потоки в компактному двійковому форматі.
 - У класі BinaryWriter визначено перевантажений метод Write() для поміщення типів даних у потік в основі.
 - На додачу, надаються додаткові члени для отримання або встановлення об'єктів, успадкованих від Stream-типів, а також підтримується довільний доступ до даних.
- У прикладі об'єкти даних різних типів записуються в файл *.dat
 - Об'єкт FileStream, повернений методом FileInfo.OpenWrite(), передається конструктору типу BinaryWriter.
 - Конструктор BinaryWriter приймає будь-який тип, успадкований від Stream, тому за потреби записати двійкові дані в пам'ять використовуйте об'єкт MemoryStream.

Робота з класами BinaryWriter і BinaryReader

```
static void Main(string[] args)
{
    ...
    FileInfo f = new FileInfo("BinFile.dat");
    ...
    // Читать двоичные данные из потока.
    using(BinaryReader br = new BinaryReader(f.OpenRead()))
    {
        Console.WriteLine(br.ReadDouble());
        Console.WriteLine(br.ReadInt32());
        Console.WriteLine(br.ReadString());
    }
    Console.ReadLine();
}
```

- Для зчитування даних з файлу BinFile.dat в класі BinaryReader пропонується ряд опцій.
 - У приклади викликаються різні члени, які виконують зчитування, для витягування кожного фрагменту даних з файлового потоку.

Програмне відстеження файлів (клас FileSystemWatcher)

```
public enum NotifyFilters
{
    Attributes, CreationTime,
    DirectoryName, FileName,
    LastAccess, LastWrite,
    Security, Size,
}
```

- Цей тип корисний за потреби програмно відстежувати стан файлів у системі.
 - Зокрема, можна організувати моніторинг файлів щодо будь-яких дій, вказаних у переліченні System.IO.NotifyFilters.
- 1) Необхідно встановити властивість ***Path***, щоб вона вказувала назву (і розташування) каталогу, який містить файли для відстеження, а також властивість ***Filter***, яка визначає розширення цих файлів.
 - Нині можна вибрати обробку подій Changed, Created і Deleted — всі працюють разом з делегатом FileSystemEventHandler.

```
// Делегат FileSystemEventHandler повинен вказувати
// на метод, який відповідає сигнатурі
void MyNotificationHandler(object source, FileSystemEventArgs e)
```

Програмне відстеження файлів (клас FileSystemWatcher)

```
Console.WriteLine("***** The Amazing File Watcher App *****\n");
// Establish the path to the directory to watch.
FileSystemWatcher watcher = new FileSystemWatcher();
try
{
    watcher.Path = @"C:\MyFolder";
} catch (ArgumentException ex) {
    Console.WriteLine(ex.Message);
    return;
}
// Set up the things to be on the lookout for.
watcher.NotifyFilter = NotifyFilters.LastAccess | NotifyFilters.LastWrite
    | NotifyFilters.FileName | NotifyFilters.DirectoryName;
// Only watch text files.
watcher.Filter = "*.txt";
// Add event handlers.
watcher.Changed += new FileSystemEventHandler(OnChanged);
watcher.Created += new FileSystemEventHandler(OnChanged);
watcher.Deleted += new FileSystemEventHandler(OnChanged);
watcher.Renamed += new RenamedEventHandler(OnRenamed);
// Begin watching the directory.
watcher.EnableRaisingEvents = true;
// Wait for the user to quit the program.
Console.WriteLine(@"Press 'q' to quit app.");
while (Console.Read() != 'q') ;
```

- Подія Renamed може оброблятись делегатом типу RenamedEventHandler:

```
void MyNotificationHandler(object source, RenamedEventArgs e)
```

- Для ілюстрації процесу моніторингу файлів, припустимо, що на диску C: створено новий каталог MyFolder, що містить різні файли *.txt.
- У прикладі виконується моніторинг файлів *.txt всередині каталога MyFolder та вивід на консоль повідомлення при створенні, видаленні, модифікації або перейменування файлів

```
static void OnChanged(object source, FileSystemEventArgs e)
{
    // Показать, что сделано, если файл изменен, создан или удален.
    Console.WriteLine("File: {0} {1}!", e.FullPath, e.ChangeType);
}

static void OnRenamed(object source, RenamedEventArgs e)
{
    // Показать, что файл был переименован.
    Console.WriteLine("File: {0} renamed to {1}", e.OldFullPath, e.FullPath);
}
```



ДЯКУЮ ЗА УВАГУ!

Наступне питання: Серіалізація об'єктів