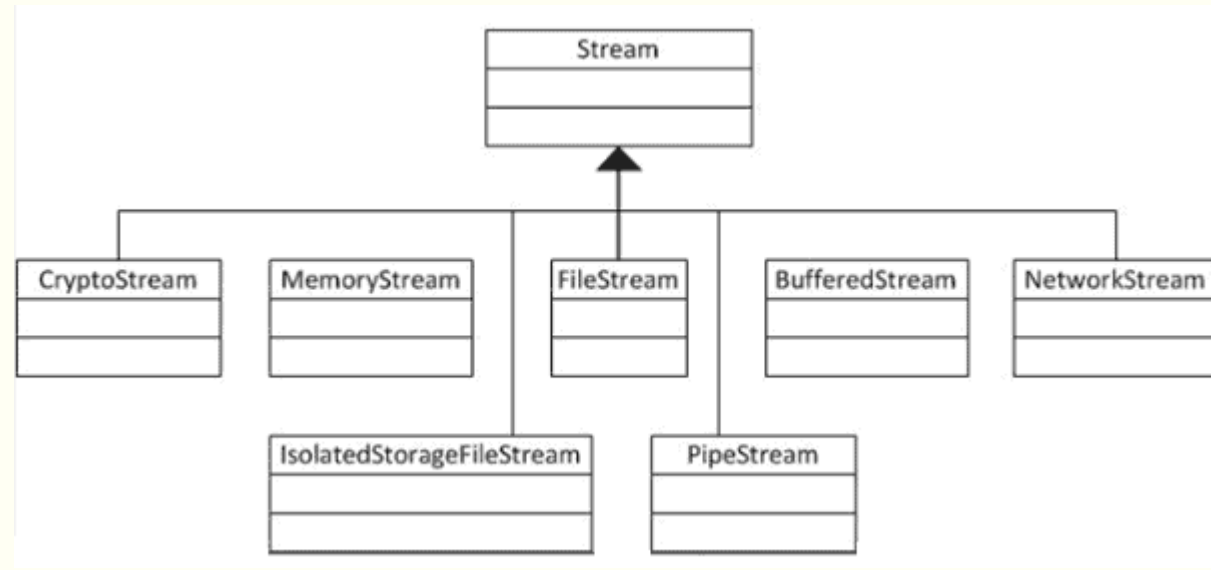




ПОТОКИ ВВОДУ-ВИВОДУ

Питання 10.2

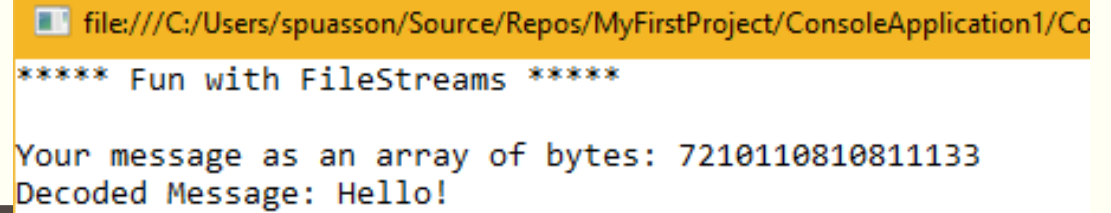
Потоки вводу-виводу. Абстрактний клас Stream



- 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

```
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();
```



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

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

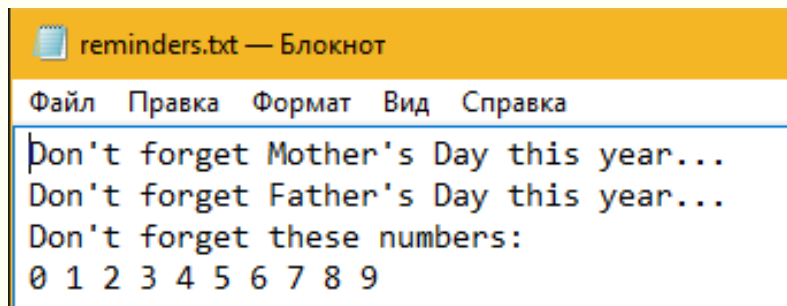
- Клас FileStream надає реалізацію абстрактного члена Stream в манері, доречній для потокової роботи з файлами.
 - Це елементарний потік, який може записувати чи зчитувати тільки 1 байт або масив байтів.
 - Проте взаємодіяти з членами типу FileStream доводиться нечасто.
 - Ймовірніше використання різноманітних 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`.
 - Когда 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

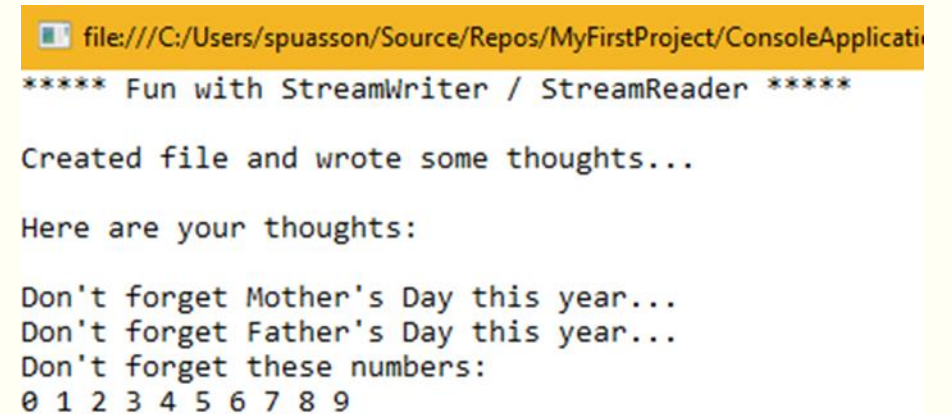
- Тип `StreamReader` успадкований від абстрактного класу `TextReader`.

- 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 символьних 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
