Лекція 7

ФАЙЛОВИЙ ВВІД-ВИВІД І СЕРІАЛІЗАЦІЯ ОБ’ЄКТІВ

Лекція 7. Файловий ввід-вивід і серіалізація об'єктів

План

1. Пространство імен System.IO.
2. Маніпуляції файлами і каталогами.
3. Потокові класи.
4. Серіалізація об’єктів.

1. Пространство імен System.IO

Платформа .NET Framework підтримує потужну бібліотеку класів для реалізації вводу-виводу та різних **маніпуляцій файлами та каталогами**. Вони описані в просторі імен під назвою System.IO.

Таблиця 1. Основні класи простору імені System.IO

Клас	Опис
BinaryReader, BinaryWriter	Реалізація вводу-виводу атомарних типів даних (числа, рядки і т.п.) у бінарному вигляді
BufferedStream	Тимчасове сховище для потоку байтів
Directory, DirectoryInfo	Маніпуляція зі структурою каталогів
DriveInfo	Надає детальну інформацію щодо дискових пристроїв, які використовуються на заданому комп'ютері
File, FileInfo	Реалізують різні маніпуляції з файлами на комп'ютері
FileStream	Забезпечує довільний доступ до файлу з даними, представленими у вигляді потоку байт
FileSystemWatcher	Дозволяє відстежувати модифікації зовнішніх файлів у заданому каталозі

1. Пространство імені System.IO

Таблиця 1 (продовження)

Клас	Опис
MemoryStream	Забезпечує вільний доступ до даних, що зберігаються в пам'яті
Path	Реалізує різні маніпуляції зі строками, що містять інформацію про шлях до файлу або каталогу в незалежному від платформи способі
StreamWriter, StreamReader	Використовуються для вводу-виводу текстової інформації з файлу (не підтримують довільний доступ до файлу)
StringWriter, StringReader	Використовуються для вводу-виводу текстової інформації зі строкового буфера

2. Маніпуляції файлами і каталогами

Класи **DirectoryInfo** і **FileInfo** призначені для маніпулювання **каталогами** та **файлами**. Вони є похідними від абстрактного базового класу **FileSystemInfo**, основні члени якого наведені в табл. 2.

Таблиця 2. Основні властивості та методи класу FileSystemInfo

Поле	Опис
Attributes	Атрибут файлу (прихований, зашифрований, доступний тільки для читання, ...)
CreationTime	Час створення поточного файлу або каталогу
Exists	Містить ознаку, чи існує цей файл або каталог
Extension	Розширення файлу
FullName	Повний шлях до файлу
LastAccessTime	Час останнього доступу до поточного файлу або каталогу
LastWriteTime	Час останнього запису в поточний файл або каталог
Name	Ім'я поточного файлу або каталогу
Delete()	Абстрактний метод видалення каталога або файлу
Refresh()	Служить для оновлення інформації про файл або каталог

2. Маніпуляції файлами і каталогами

Клас **DirectoryInfo** містить наступні основні властивості та методи (табл. 3).

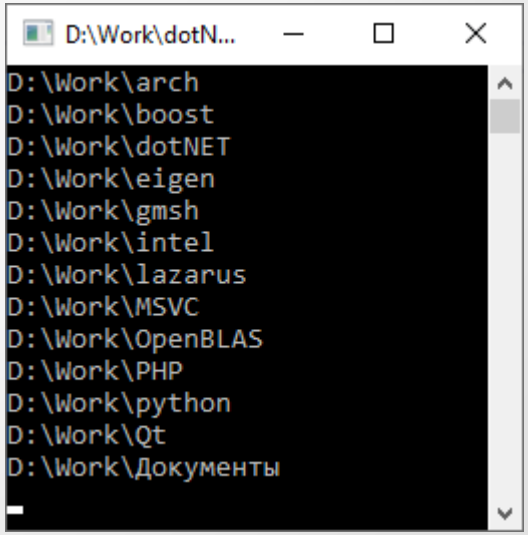
Таблиця 3. Основні члени класу DirectoryInfo

Поле	Опис
Parent	Містить батьківський каталог для поточного каталога
Root	Містить кореневую частину шляху
Create(), CreateSubdirectory()	Створює каталог (або набір підкаталогів) за заданим ім’ям
Delete()	Вилучає каталог і весь його вміст
GetDirectories()	Створює масив об'єктів DirectoryInfo, що представляють всі підкаталоги у поточному каталозі
GetFiles()	Створює масив об'єктів FileInfo, що представляє набір файлів у заданому каталозі
MoveTo()	Переміщує каталог зі всім вмістом в нову локацію

2. Маніпуляції файлами і каталогами

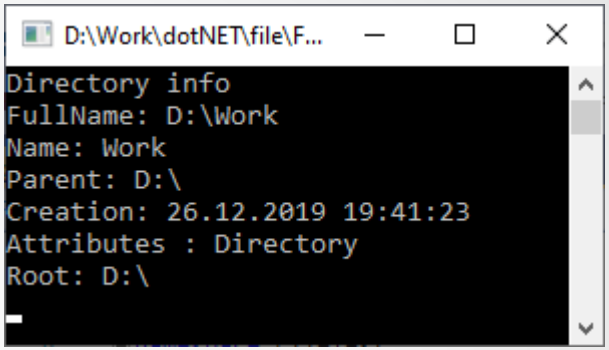
Наприклад, вивід на екран списку підкаталогів заданого каталогу можна реалізувати таким чином:

```
using System;
using System.IO;

namespace FileTest
{
    class Program
    {
        static void Main(string[] args)
        {
            // Підключення до заданого каталогу.
            DirectoryInfo dir = new DirectoryInfo("D:/Work");

            foreach (var cur in dir.GetDirectories())
                Console.WriteLine(cur.FullName);
            Console.ReadKey();
        }
    }
}
```



2. Маніпуляції файлами і каталогами

Вивод інформації про вказаний каталог можна виконати так:




```
using System;
using System.IO;
namespace FileTest
{
    class Program
    {
        static void Main(string[] args)
        {
            DirectoryInfo dir = new DirectoryInfo("D:/Work"); // Підключення до каталогу
            Console.WriteLine("Directory info");
            Console.WriteLine("FullName: {0}", dir.FullName); // Повне ім'я
            Console.WriteLine("Name: {0}", dir.Name); // Ім'я каталогу
            Console.WriteLine("Parent: {0}", dir.Parent); // Батьківський каталог
            Console.WriteLine("Creation: {0}", dir.CreationTime); // Дата/час створення
            Console.WriteLine("Attributes : {0}", dir.Attributes); // Атрибути
            Console.WriteLine("Root: {0}", dir.Root.Name); // Кореневий каталог
            Console.ReadKey();
        }
    }
}
```

2. Маніпуляції файлами і каталогами

Клас **FileInfo** реалізує різні операції з файлами та містить наступні основні поля (табл. 4.)

Таблиця 4. Основні члени класу FileInfo

Поле	Опис
Directory	Містить екземпляр батьківського каталогу
DirectoryName	Повний шлях до батьківського каталогу
Length	Розмір файлу
Name	Ім'я файлу
AppendText()	Створює об'єкт StreamWriter і додає текст у файл
CopyTo()	Копіює поточний файл у новий
Create()	Створює новий файл і повертає об'єкт FileStream
CreateText()	Створює об'єкт StreamWriter, записуючи новий текстовий файл
Delete()	Вилучає поточний файл

2. Маніпуляції файлами і каталогами

Таблиця 4 (продовження)

Поле	Опис
MoveTo()	Переміщує поточний файл у нове місце
Open()	Відкриває файл з різними привілеями читання/запису і спільного доступу
OpenRead()	Створює доступний тільки для читання об'єкт FileStream
OpenText()	Створює доступний для читання/запису об'єкт StreamReader
OpenWrite()	Створює доступний тільки для запису об'єкт FileStream

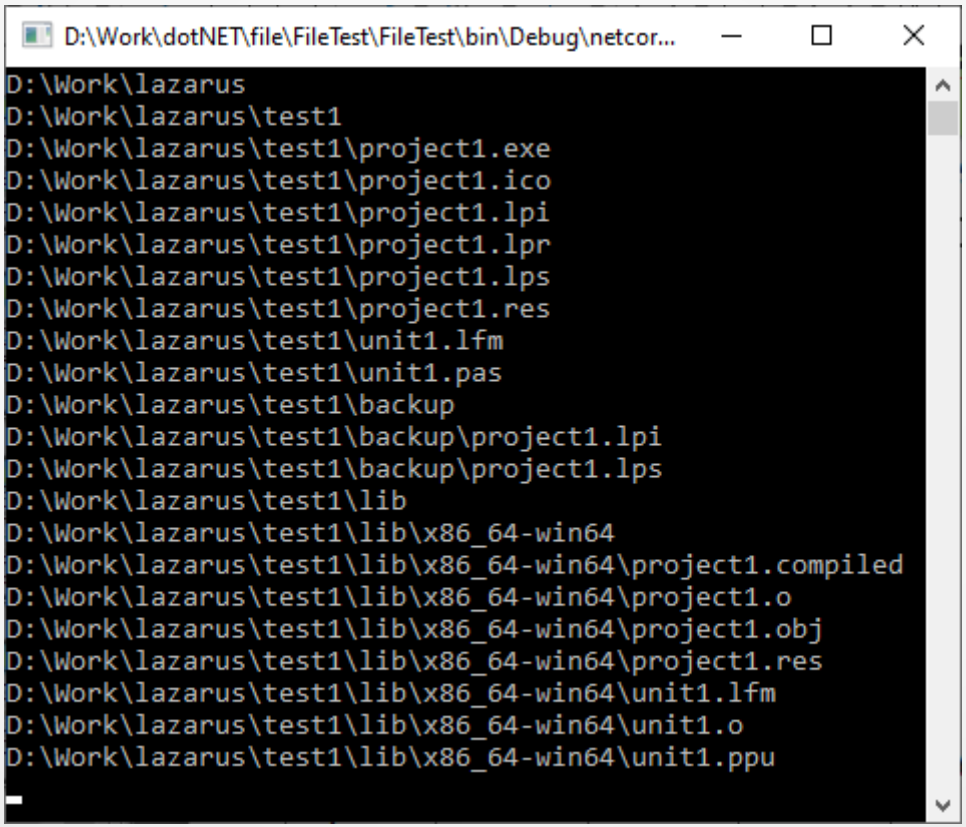
2. Маніпуляції файлами і каталогами

Наприклад, за допомогою класів **DirectoryInfo** і **FileInfo** написати програму, виведення на екран вмісту заданого каталогу можна так:

```
using System;
using System.IO;
namespace FileTest
{
  class Program
  {
    static void Main(string[] args)
    {
      printDirectory(new DirectoryInfo("D:/Work/lazarus"));
    }
    static void printDirectory(DirectoryInfo di)
    {
      Console.WriteLine(di.FullName); // Виведення назви поточного каталогу
      foreach (var it in di.GetFiles()) // Виведення списку файлів у каталозі
        Console.WriteLine(it.FullName);
      foreach (var it in di.GetDirectories()) // Рекурсивна обробка вкладених каталогів
        printDirectory(it);
    }
  }
}
```

2. Маніпуляції файлами і каталогами

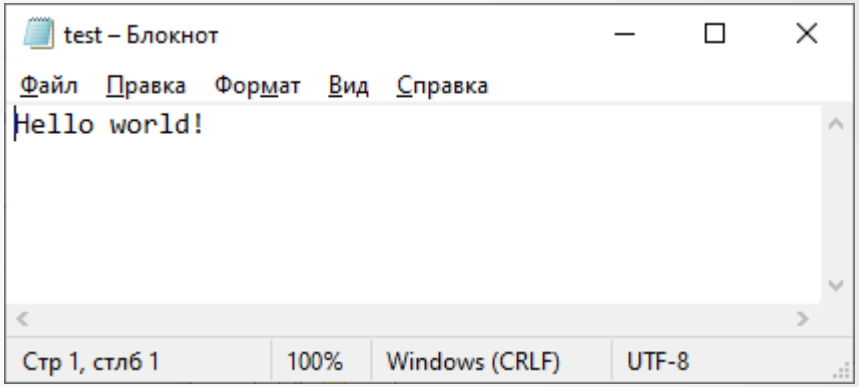
Результат роботи програми виведення вмісту заданого каталогу на екран



2. Маніпуляції файлами і каталогами

Для створення текстового файлу та запису в нього інформації можна використовувати клас FileInfo таким чином:

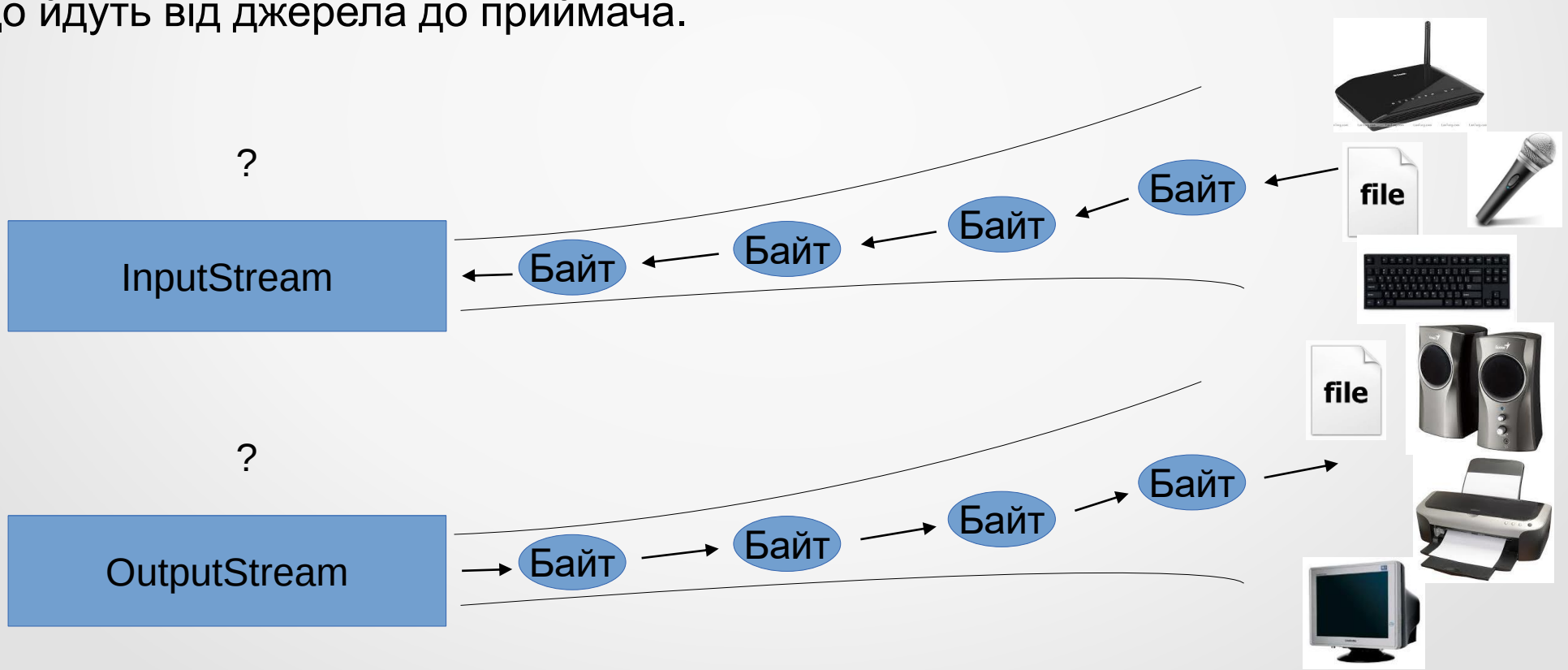
```
using System;
using System.IO;

namespace FileTest
{
    class Program
    {
        static void Main(string[] args)
        {
            FileInfo fi = new FileInfo("D:/test.txt");
            StreamWriter sw = fi.CreateText();

            sw.WriteLine("Hello world!");
            sw.Close();
            Console.ReadKey();
        }
    }
}
```



3. Потокові класи

У сучасних мовах програмування для роботи з файлами застосовуються **потокові класи** (**потоки**).

Потік (**stream**) – це абстракція, що описує неперервну послідовність байтів, що йдуть від джерела до приймача.



3. Потокові класи

В платформі .NET Framework потоки вводу-виводу описуються абстрактним класом **Stream** (табл. 5).

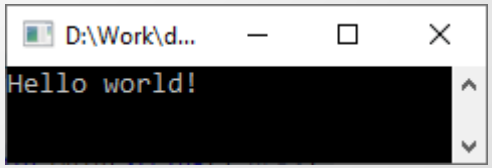
Таблиця 5. Основні властивості і методи класу Stream

Член	Описание
CanRead, CanWrite, CanSeek	Визначає, чи підтримує поточний потік читання, запуск та пошук
Length	Довжина потоку в байтах
Position	Містить поточну позицію в потоці
Close()	Закриває поточний потік
Flush()	Скидає буфер у джерело даних
Read(), ReadByte()	Читання послідовності байт (байту) з потоку
Seek()	Встановлює позицію в поточному потоці
SetLength()	Встановлює довжину поточного потоку
Write(), WriteByte()	Записує послідовність байтів (або одиночний байт) у поточному потоці

3. Потокові класи

Для роботи з файлами, як і з потоками, використовується клас **FileStream**. Прочитати інформацію з наявного текстового файлу можна так:

```
using System;
using System.IO;
namespace FileTest
{
    class Program
    {
        static void Main(string[] args)
        {
            FileStream fs = new FileInfo("D:/test.txt").OpenRead();
            byte[] buffer = new byte[fs.Length];

            fs.Read(buffer, 0, buffer.Length);
            foreach (var it in buffer)
                Console.Write((char)(it));
            fs.Close();
            Console.ReadKey();
        }
    }
}
```



3. Потокові класи

Для потокового читання-запису символьних даних використовуються абстрактні класи **TextWriter** і **TextReader**.

Основні властивості та методи класу **TextWriter** наведені в табл. 6.

Таблиця 6. Основні поля абстрактного класу TextWriter

Поле	Опис
NewLine	Визначає константу для маркування нового рядка (за замовчуванням у Windows використовується послідовність ""\r\n")
Close()	Закриває потік і звільняє всі пов'язані з ним ресурси
Flush()	Скидає буфер в джерело даних
Write()	Записує дані в текстовий потік без додавання константи нової строки
WriteLine()	Записує дані в текстовий потік з додаванням константи нової строки

3. Потокові класи

Основні властивості та методи класу **TextReader** наведені в табл. 7.

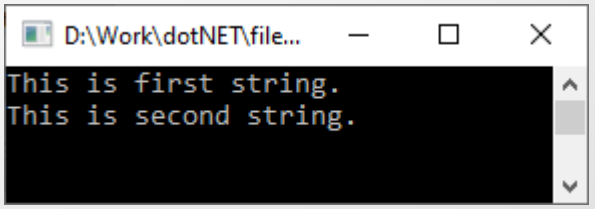
Таблиця 7. Основні члени абстрактного класу TextReader

Поле	Описание
Peek()	Читає черговий символ без зміни поточної позиції в потоці. Значення -1 вказує на досягнення кінця потоку
Read()	Читає дані з вхідного потоку
ReadBlock()	Читає задану максимальну кількість символів з поточного потоку і записує дані в буфер, починаючи з заданого індексу
ReadLine()	Читає рядок символів із поточного потоку та повертає дані у вигляді строки (null вказує на досягнення кінця файлу)
ReadToEnd()	Читає всі символи від поточної позиції до кінця потоку і повертає їх у вигляді одного рядка

3. Потокові класи

Для роботи з символьними потоками застосовуються класи **StreamWriter** і **StreamReader** .

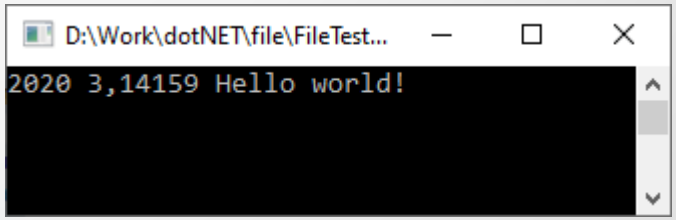
```
using System;
using System.IO;

namespace FileTest
{
    class Program
    {
        static void Main(string[] args)
        {
            StreamWriter sw = new StreamWriter("D:/test.txt");
            sw.WriteLine("This is first string.");
            sw.WriteLine("This is second string.");
            sw.Close();
            StreamReader sr = new StreamReader("D:/test.txt");
            while (sr.Peek() != -1)
                Console.WriteLine(sr.ReadLine());
            sr.Close();
        }
    }
}
```



3. Потокові класи

Для роботи з текстовими даними, як і з потоками, використовуються класи **StringWriter** і **StringReader** .

```
using System;
using System.IO;
class Program
{
    static void Main(string[] args)
    {
        string buffer = "2020\n3,14159\nHello world!";
        StringReader sr = new StringReader(buffer);
        int year;
        double pi;
        string Hello;
        year = Convert.ToInt32(sr.ReadLine());
        pi = Convert.ToDouble(sr.ReadLine());
        Hello = sr.ReadLine();
        sr.Close();
        Console.WriteLine("{0} {1} {2}", year, pi, Hello);
        Console.ReadKey();
    }
}
```



3. Потокові класи

Для роботи з бінарними даними використовуються потокові класи **BinaryWriter** і **BinaryReader** (табл. 7, 8).

Таблиця 7. Основні члени класу BinaryWriter

Поле	Опис
BaseStream	Потік, що виводиться
Close()	Закриває потік
Flush()	Скидає буфер у джерело даних
Seek()	Встановлює позицію в потоці
Write()	Записує значення в поточний потік

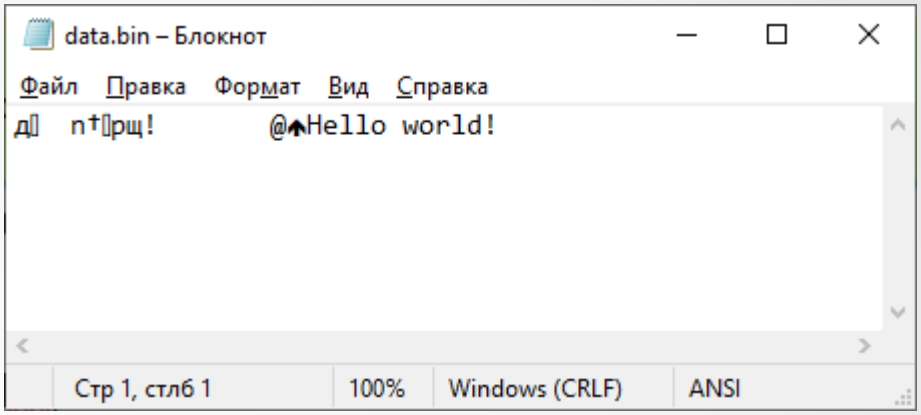
3. Потокові класи

Таблиця 8. Основні члени класу BinaryReader

Поле	Опис
BaseStream	Потік, з якого читаються дані
Close()	Закриває потік
PeekChar()	Повертає наступний доступний символ без зміни поточної позиції потоку
Read()	Читає заданий набір байт або символів
ReadBoolean(), ReadByte(), ReadInt32(), ...	Читає із потоку об'єкти різних типів

3. Потокові класи

Приклад виведення даних у двійковий файл:

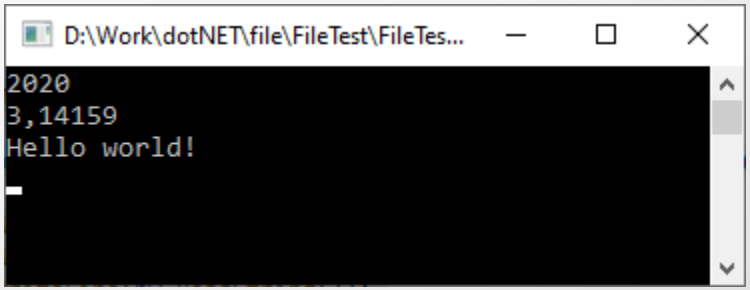



```
using System;
using System.IO;

class Program
{
    static void Main(string[] args)
    {
        int year = 2020;
        double pi = 3.14159;
        string Hello = "Hello world!";
        BinaryWriter bw = new BinaryWriter(new FileInfo("D:/data.bin").OpenWrite());

        bw.Write(year);
        bw.Write(pi);
        bw.Write(Hello);
        bw.Close();
        Console.ReadKey();
    }
}
```

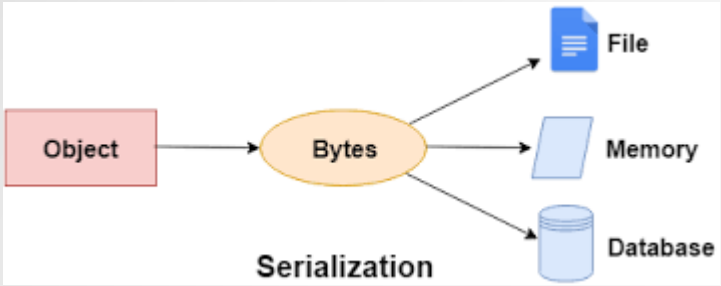
3. Потокові класи

Приклад читання даних із двійкового файлу:

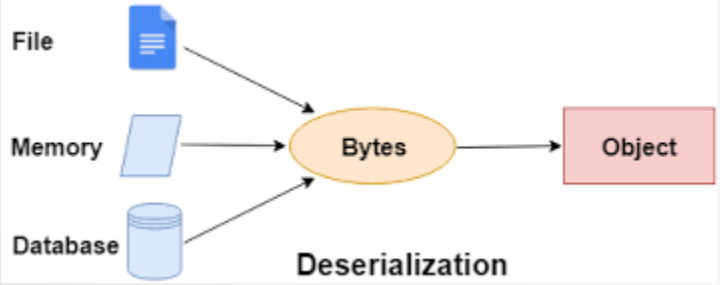
```
using System;
using System.IO;

class Program
{
    static void Main(string[] args)
    {
        BinaryReader bw = new BinaryReader(new FileInfo("D:/data.bin").OpenRead());

        Console.WriteLine(bw.ReadInt32());
        Console.WriteLine(bw.ReadDouble());
        Console.WriteLine(bw.ReadString());
        bw.Close();
        Console.ReadKey();
    }
}
```



4. Серіалізація об'єктів



На практиці часто виникає **задача збереження стану** іншого об'єкта. Наприклад, багато програм зберігають розмір свого вікна, його положення та інші індивідуальні налаштування користувача при закритті.



Після запуску така програма зчитує свій стан і відновлює вихідні налаштування користувача.

Серіалізацією (**Serialization**) називається процес збереження стану об'єкта (в потоках, пам'яті, базі даних і т. п.).

Десеріалізація (**Deserialization**) – це процес відновлення вихідного стану (реконструкція) об'єкта з метою наступного використання.

4. Серіалізація об'єктів

Платформа .NET Framework підтримує автоматичну серіалізацію-десеріалізацію. Для цього клас, який потребує серіалізації, повинен бути помічений спеціальним атрибутом **[Serializable]** .

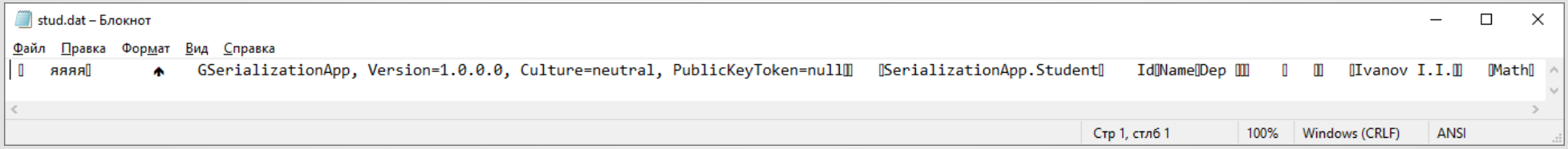
Наприклад, клас, що містить базову інформацію про студента, може бути описаний таким чином:

```
[Serializable]
class Student
{
    public int Id;                          // Id студента
    public string Name;                     // ФИО
    public string Dep;                      // Факультет
    public Student(int id, string name, string dep)
    {
        Id = id;
        Name = name;
        Dep = dep;
    }
}
```

4. Серіалізація об'єктів

Виконати автоматичну **серіалізацію** об’єкта типу Student можна за допомогою класу **BinaryFormatter** наступним чином (описано в просторі імені **System.Runtime.Serialization.Formatters.Binary**):

```
static void Main(string[] args)
{
   // Створення об'єкта, що підлягає серіалізації
   Student ivanov = new Student(1, "Ivanov I.I.", "Math");
   // Створення об'єкта для двійкової серіалізації
   BinaryFormatter formatter = new BinaryFormatter();
   // Створення потоку, в який буде виведено інформацію про об'єкт
   Stream fStream = new FileStream("stud.dat", FileMode.Create, FileAccess.Write);
   // Зберегти об'єкт у локальному файлі
   formatter.Serialize(fStream, ivanov);
   Console.ReadKey();
}
```



4. Серіалізація об'єктів

Виконати **десеріалізацію** об'єкта можна, наприклад, таким чином:

```
static void Main(string[] args)
{
	Student stud;
	BinaryFormatter formatter = new BinaryFormatter();
	Stream fStream = new FileStream("stud.dat", FileMode.Open, FileAccess.Read);

	// Зчитати об'єкт з локального файлу
	stud = (Student)formatter.Deserialize(fStream);

	// Вивести його значення
	Console.WriteLine("Id: {0}", stud.Id);
	Console.WriteLine("Name: {0}", stud.Name);
	Console.WriteLine("Dep: {0}", stud.Dep);

	Console.ReadKey();
}
```