Доступ до файлової системи. Серіалізація

Клакович Л.М.

Моделі обміну даними

1. Простір імен System.IO
2. Класи, що надають потоки Stream
3. Класи System.IO для роботи з символьними даними
4. Serialization

System.IO

- Простір імен **System.IO** містить набір типів для виконання операцій з файлами і іншими об’єктами введення-виведення.
- Всі типи **System.IO** знаходяться в бібліотеці **mscorlib.dll.**
- Більшість класів призначені для роботи з каталогами і файлами на диску, а також з буфером в оперативній пам’яті чи областями оперативної пам’яті

Основні класи простору System.IO

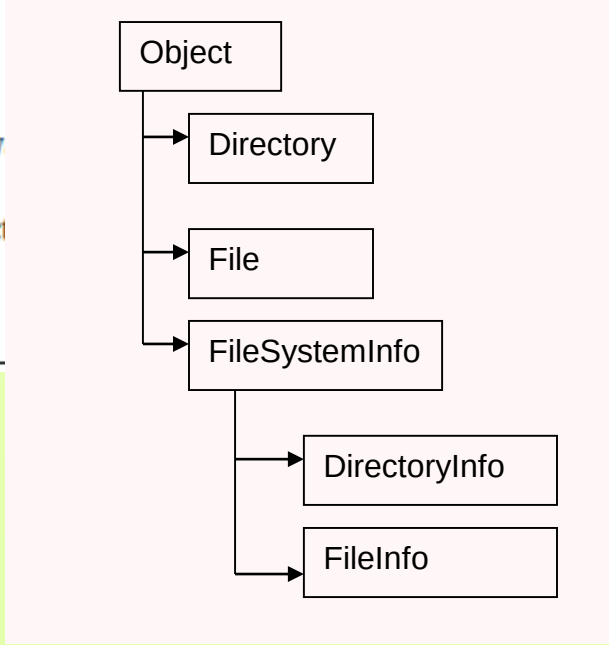
Class	Description
`Directory`	Надає статичні члени для створення, переміщення, видалення, отримання інформації про каталоги, підкаталоги в файловій системі.
`DirectoryInfo`	призначений для створення, переміщення, видалення, отримання інформації про каталоги, підкаталоги в файловій системі.
`File`	Надає статичні методи для створення, копіювання, видалення, переміщення і відкривання файлів
`FileInfo`	дозволяє отримати інформацію про файл, а також здійснювати різні операції, наприклад по створенню і видаленню
`FileStream`	Надає потік (Stream) до файлу, із синхронними та асинхронними операціями читання-запису
`BinaryReader, BinaryWriter`	Надають функціональність для зчитування\запису бінарних даних.

Основні класи простору System.IO

IOException	Винятки, які генеруються підчас операцій I/O.
MemoryStream	Надає потік для запису в пам’ять.
Stream	Базовий клас, що описує потік.
TextReader, TextWriter	Представляє зчитувач(записувач), що може читати (записувати) послідовність символів
StreamReader, StreamWriter,	Їх реалізація для потоків
StringReader, StringWriter	Їх реалізація для рядків

Абстрактний базовий клас FileSystemInfo

Property	Meaning in Life
`Attributes`	Gets or sets the attributes associated with the current file that are represented by the `FileAttributes` enumeration (e.g., is the file or directory read-only, encrypted, hidden, or compressed?).
`CreationTime`	Gets or sets the time of creation for the current file or directory.
`Exists`	You can use this to determine whether a given file or directory exists.
`Extension`	Retrieves a file's extension.
`FullName`	Gets the full path of the directory or file.
`LastAccessTime`	Gets or sets the time the current file or directory w
`LastWriteTime`	Gets or sets the time when the current file or direc
`Name`	Obtains the name of the current file or directory.



Клас DirectoryInfo

Member	Meaning in Life
`Create()` `CreateSubdirectory()`	Create a directory (or set of subdirectories) when given a path name.
`Delete()`	Deletes a directory and all its contents.
`GetDirectories()`	Returns an array of `DirectoryInfo` objects that represent all subdirectories in the current directory.
`GetFiles()`	Retrieves an array of `FileInfo` objects that represent a set of files in the given directory.
`MoveTo()`	Moves a directory and its contents to a new path.
`Parent`	Retrieves the parent directory of this directory.
`Root`	Gets the root portion of a path.

Приклад використання класу DirectoryInfo

```
class Program
    {
        static void Main()
        {
            // Get info.
            DirectoryInfo info = new DirectoryInfo(@"C:\Windows\");

            // Write name.
            Console.WriteLine(info.Name);

            // Write file count.
            FileInfo[] array = info.GetFiles();
            Console.WriteLine(array.Length);
        }
    }
```

Клас FileInfo

Member	Meaning in Life
`AppendText()`	Creates a `StreamWriter` object (described later) that appends text to a file.
`CopyTo()`	Copies an existing file to a new file.
`Create()`	Creates a new file and returns a `FileStream` object (described later) to interact with the newly created file.
`CreateText()`	Creates a `StreamWriter` object that writes a new text file.
`Delete()`	Deletes the file to which a `FileInfo` instance is bound.
`Directory`	Gets an instance of the parent directory.
`DirectoryName`	Gets the full path to the parent directory.
`Length`	Gets the size of the current file.
`MoveTo()`	Moves a specified file to a new location, providing the option to specify a new file name.
`Name`	Gets the name of the file.

Приклад використання класу File

```
static void Main()
    {
        string file = File.ReadAllText("C:\\file.txt");
        Console.WriteLine(file);
    }
```

```
string[] lines = File.ReadAllLines("file.txt");
foreach (string line in lines)
{
   if (line.Length > 80)
    {
      // Do something with the line.
    }
}
```

Приклад використання класу File

```
// Write a string array to a file.
string[] stringArray = new string[]{"cat","dog","arrow"};

File.WriteAllLines("file.txt", stringArray);
```

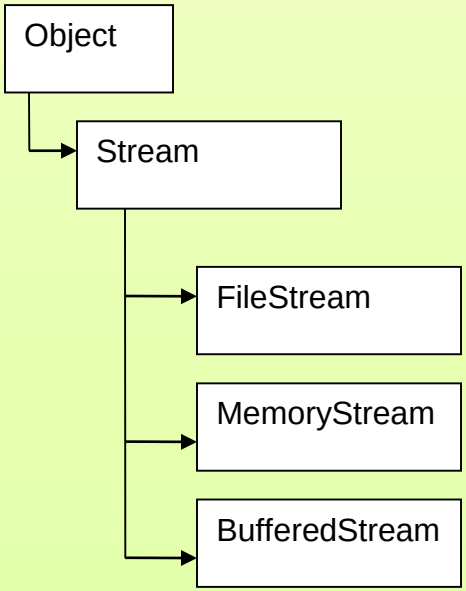
```
File.WriteAllText("C:\\perls.txt","Dot Net Perls");
```

```
File.AppendAllText("C:\\perls.txt", "first part\n");
File.AppendAllText("C:\\perls.txt", "second part\n");
```

class Stream

stream (потік) – сутність, що використовується для роботи з блоками даних.

• **`Stream –`** абстрактний клас для всіх потоків.
• потік – це абстракція послідовності байтів, таких як файл, пристрій введення/виведення, область в оперативній пам’яті
• **клас Stream** забезпечує як синхронну так і асинхронну взаємодію з середовищем зберігання даних (файлом на диску чи областю в оперативній пам’яті).
• класи, похідні від **Stream** призначені для роботи з блоками двійкових даних, підтримують пошук в потоці даних.



Основні операції з потоками

- **Читання з потоку** – переміщення даних з потоку в структуру даних, таку як масив бітів (**Read**(), **ReadByte**()).
- **Запис в потік** – переміщення даних зі структури даних в потік (**Write**(), **WriteByte**()).
- **Пошук** – запит і модифікація поточної позиції в потоці (**Seek**()).

Основні властивості класу Stream

Name	Description
`CanWrite`	Повертає величину, яка вказує чи даний потік може бути використаний на запис.
`Length`	Довжина потоку в байтах
`Position`	Повертає або встановлює позицію в потоці.
`ReadTimeout`	Отримання чи встановлення Timeout для зчитування з потоку.
`WriteTimeout`	Отримання чи встановлення Timeout для запмсу в потік.

Основні методи класу Stream

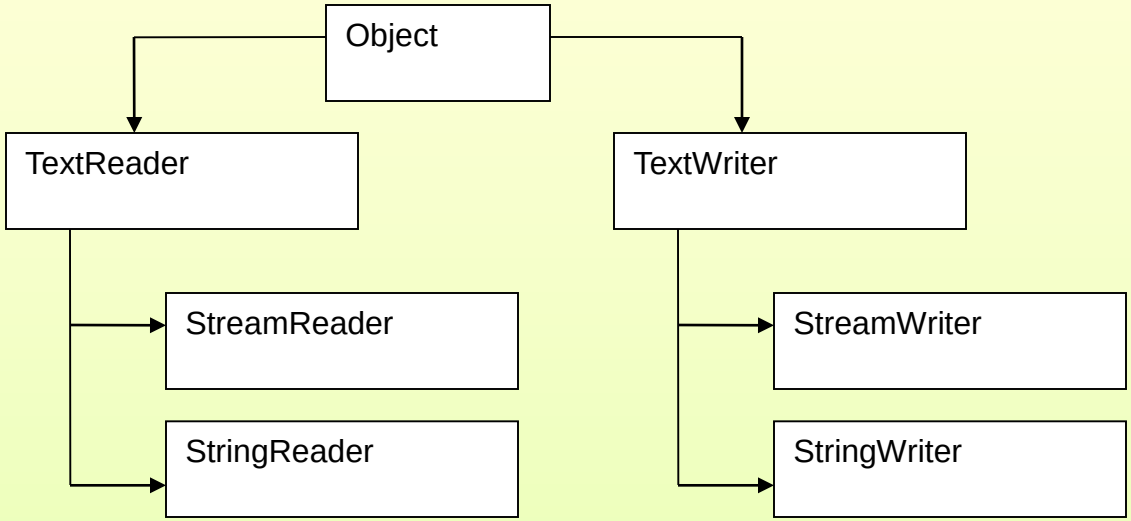
Name	Description
`Read`	Після перевизначення в похідних класах надає зчитування даних в різних форматах з потоку.
`Write`	Після перевизначення в похідних класах надає запис даних в потік
`Seek`	Після перевизначення в похідних класах встановлює позицію в потоці.
`ReadAsync, WriteAsync, CopyToAsync, FlushAsync.`	Асинхронні методи, які не блокують головного потоку.
`Close`	Закриває потік і звільняє всі ресурси, пов’язані з цим потоком.
`Flush`	Очищає всі буфери для даного потоку і записує всі дані буфера у вибраний пристрій.

Клас FileStream

- Клас **FileStream** забезпечує реалізацію абстрактних членів класу **Stream** для роботи з файлами на диску.
- дозволяє відкрити існуючі файли і створити нові, використовуються значення з перелічуваних типів **FileMode, FileAccess, FileShare.**

```
FileStream mystream = new FileStream("test.dat",
                                     FileMode.OpenOrCreate,
                                     FileAccess.ReadWrite);
mystream.WriteByte((byte)6);
mystream.Position=0;
Console.Write(mystream.ReadByte());
mystream.Close();
```

Класи System.IO для роботи з символьними даними.



- Абстрактні класи **TextReader** та **TextWriter** забезпечують похідним класам набір можливостей по зчитуванню та запису символьних даних. За замовчуванням ці типи працюють з кодуванням Unicode.
- Класи **StreamReader** та **StreamWriter** дають змогу зчитувати та записувати символьні дані в потік.
- Класи **StringWriter і StringReader** дозволяють звертатися до текстової інформації як до потоку в оперативній пам’яті.

Приклади використання StreamReader, StreamWriter

```
using System.IO;
class Program
{
    static void Main()
    {
        // Read every line in the file.
        StreamReader reader = new StreamReader("file.txt");
        string line;
        while ((line = reader.ReadLine()) != null)
        {
           // Do something with the line.
           string[] parts = line.Split(',');
           //…
        }
         reader.Close();
        }
    }
}
```

Приклади використання StreamReader, StreamWriter

```
static void Main()
    {
       using (StreamWriter writer =
           new StreamWriter("important.txt"))
       {
           writer.Write("Word ");
           writer.WriteLine("word 2");
           writer.WriteLine("Line");
       }
    }
```

```
//додавання до існуючого файлу
StreamWriter writer =
     new StreamWriter("C:\\log.txt", true))
```

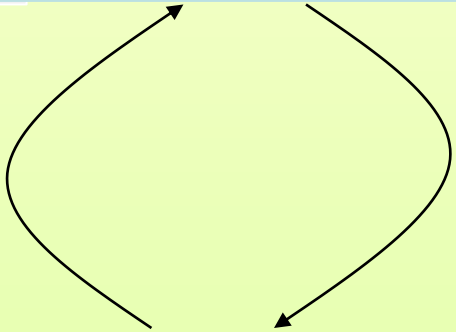
Серіалізація

Serialization (**Серіалізація) –** процес перетворення стану об'єкта (чи набору взаємопов'язаних об'єктів) в спеціальне представлення (наприклад, в форматі XML чи двійковому форматі), яке може бути розміщено в потік і далі відновлено з нього.
Deserialization (Десеріалізація) – зворотній процес відновлення об'єкта в його оригінальному стані з потоку байтів.

```
[Serializable]
public class Person
{
    …
}
```

```
Person st1 = new Person();
st1.FirstName = "Iryna";
st1.LastName = "Koval";
st1.BirthDate = new DateTime(1981, 8, 17);
```

Deserialization



Serialization

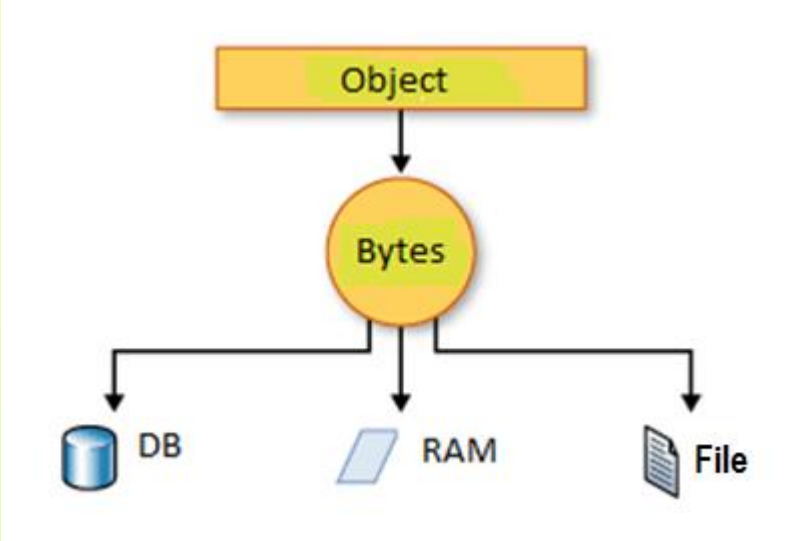
```
AC ED 00 05 73 72 00 0A 53 65 72 69 61 6C 54 65
73 74 A0 0C 34 00 FE B1 DD F9 02 00 02 42 00 05
63 6F 75 6E 74 42 00 07 76 65 72 73 69 6F 6E 78
70 00 64
```

Серіалізація

- **Serialization/Deserialization** найчастіше використовується для транспортування об’єктів (віддалена взаємодія) або зберігання об’єктів (в файлі чи базі даних).

- Серіалізація містить дані про стан об’єкта та його зв’язки з іншими класами, дані про збірку і ін.

- Набір взаємопов’язаних об’єктів, серіалізованих в потік, називається ***графом об’єктів***. Графи дозволяють фіксувати відношення між об’єктами



Серіалізація

- .NET Framework містить класи (в просторах ***System.Runtime.Serialization*** та ***System.Xml.Serialization***) які надають такі види серіалізації:
  - **binary,**
  - **XML,**
  - **JSON**,
  - **Власна перевизначена серіалізація (IFormatter** і **ISerializable)**
- В бібліотеці класів .NET є три окремі механізми серіалізації:
  - **XmlSerializer** (використовується для Web Services)
  - **SoapFormatter/BinaryFormatter** (для зберігання даних)
    - **BinaryFormatter** серіалізує об'єктний граф в компактному потоці двійкового формату
    - **SoapFormatter** представляє граф як повідомлення протоколу **SOAP** (Simple Object Access Protocol – простий протокол доступу до об'єктів) в форматі **XML**.
  - ***DataContractSerializer***

XMLSerializer

- Для серіалізації об'єктів:
- Створи об'єкт і заповни його даними
- Створи об’єкт **XmlSerializer,** задавши тип об'єкта серіалізації.
- Виклич метод **Serialize** для генерації потоку XML і запису туди public властивостей та полів об'єкта.

```
Person st1 = new Person();
// Insert code to set properties and fields of the object.

XmlSerializer xmlser = new XmlSerializer(typeof(Person));
Stream serialStream = new FileStream("person.xml", FileMode.Create);

xmlser.Serialize(serialStream, st1);
```

```
// Displays
//<?xml version=”1.0” encoding=”utf-16”?>
//<Person xmlns:xsi=”http://www.w3.org/2001/XMLSchema-instance”
// xmlns:xsd=”http://www.w3.org/2001/XMLSchema”>
// <FirstName>John</FirstName>
// <LastName>Doe</LastName>
//</Person>
```

XMLSerializer

- Для десеріалізації об’єктів:
  - Утвори **XmlSerializer** задавши тип об’єкта десеріалізації.
  - Виклич метод **Deserialize** для отримання об’єкта. Після цього потрібно об’єкт привести до потрібного типу

```
XmlSerializer xmlser = new XmlSerializer(typeof(Person));

serialStream = new FileStream("person.xml", FileMode.Open);

Person st2 = xmlser.Deserialize(serialStream) as Person;

Console.WriteLine(st2);
```

XMLSerializer

- З допомогою атрибутів можна налаштувати серіалізацію :
  - ***XmlIgnore -*** can be used to make sure that an element is not serialized
  - ***XmlAttribute -*** you can map a member to an attribute on its parent node.
  - ***XmlElement –*** *by default*
  - ***XmlArray -*** *is* used when serializing collections.
  - ***XmlArrayItem -*** is used when serializing collections.

Серіалізація складних та похідних типів

```
[Serializable]
public class Person
{
    public string FirstName { get; set; }
    public string LastName { get; set; }
    public int Age { get; set; }
}
```

```
[Serializable]
public class Order
{
     [XmlAttribute]
     public int ID { get; set; }

    [XmlIgnore]
     public bool IsDirty { get; set; }

     [XmlArray("Lines")]
     [XmlArrayItem("OrderLine")]
     public List<OrderLine> OrderLines { get; set; }
}
```

```
[Serializable]
public class VIPOrder : Order
{
    public string Description { get; set;}
}
[Serializable]
public class OrderLine
{
    [XmlAttribute]
     public int ID { get; set; }

     [XmlAttribute]
     public int Amount { get; set; }

      [XmlElement("OrderedProduct")]
      public Product Product { get; set; }
}
```

```
[Serializable]
public class Product
{
    [XmlAttribute]
    public int ID { get; set; }

    public decimal Price { get; set; }
    public string Description { get; set; }}
```

Серіалізація складних та похідних типів

```
private static Order CreateOrder()
{
   Product p1 = new Product { ID = 1, Description = "p2", Price = 9 };
   Product p2 = new Product { ID = 2, Description = "p3", Price = 6 };
   Order order = new VIPOrder { ID = 4, Description = "Order for John Doe. Use the nice giftwrap",
                                OrderLines = new List<OrderLine> {
                                             new OrderLine { ID = 5, Amount = 1, Product = p1},
                                             new OrderLine { ID = 6 ,Amount = 10, Product = p2},
                                                          }
                              };
    return order;
}

XmlSerializer serializer = new XmlSerializer(typeof(Order), new Type[] { typeof(VIPOrder) });
string xml;
using (StringWriter stringWriter = new StringWriter())
{
    Order order = CreateOrder();
    serializer.Serialize(stringWriter, order);
     xml = stringWriter.ToString();
}
using (StringReader stringReader = new StringReader(xml))
   { Order o = (Order)serializer.Deserialize(stringReader);
      // Use the order}
```

BinaryFormatter

- визначений в просторі імен **System.Runtime.Serialization.Formatters.Binary**
- серіалізує об’єктний граф в компактному потоці двійкового формату
- Для маркування членів, що не повинні серіалізуватися – атрибут [NonSerialized]
- Конструктор не викликається при десеріалізації

BinaryFormatter

```
[Serializable]
public class MyObject {
  public int n1 = 0;
  public int n2 = 0;
  public String str = null;
}
```

```
MyObject obj = new MyObject();
obj.n1 = 1;
obj.n2 = 24;
obj.str = "Some String";
IFormatter formatter = new BinaryFormatter();
Stream stream = new FileStream("MyFile.bin", FileMode.Create,
FileAccess.Write, FileShare.None);
formatter.Serialize(stream, obj);
stream.Close();
```

```
IFormatter formatter = new BinaryFormatter();
Stream stream = new FileStream("MyFile.bin", FileMode.Open,
FileAccess.Read, FileShare.Read);
MyObject obj = (MyObject) formatter.Deserialize(stream);
stream.Close();

Console.WriteLine("n1: {0}", obj.n1);
Console.WriteLine("n2: {0}", obj.n2);
Console.WriteLine("str: {0}", obj.str);
```

SoapFormatter

```
<SOAP-ENV:Envelope
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:SOAP- ENC="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:SOAP- ENV="http://schemas.xmlsoap.org/soap/envelope/"
  SOAP-ENV:encodingStyle=
  "http://schemas.microsoft.com/soap/encoding/clr/1.0"
  "http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:a1="http://schemas.microsoft.com/clr/assem/ToFile">

  <SOAP-ENV:Body>
    <a1:MyObject id="ref-1">
      <n1>1</n1>
      <n2>24</n2>
      <str id="ref-3">Some String</str>
    </a1:MyObject>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

XmlSerializer, SoapFormatter чи BinaryFormatter

- ***XmlSerializer***
  - **обмеження** (для класу необхідність конструктора без параметрів, тільки public read/write властивості та поля можуть бути серіалізовані, вимагається додавання атрибуту [Serializable] до класу).
  - **переваги** (добра підтримка для налаштування XML документів)

- ***SoapFormatter і BinaryFormatter***
  - **обмеження** (вимагається додавання атрибуту [Serializable] до класу)
  - **переваги** (можуть серіалізувати приватні поля)
  - **BinaryFormatter** найбільш ефективний спосіб при здійсненні серіалізації-десеріалізації на платформі .NET.

Власна серіалізація. ISerializable

- Власна серіалізація визначає які об’єкти будуть серіалізуватися і як.
- Такий клас повинен мати атрибут **SerializableAttribute** та імплементувати інтерфейс **ISerializable**.
- **ISerializable**: метод **Formatter** викликає метод **GetObjectData()** під час серіалізації і заповнює **SerializationInfo** даними об’єкта
- Для власної десеріалізації необхідний конструктор

ISerializable. Приклад

```
[Serializable]
public class Person : ISerializable
{
    private int _id;
    public string FirstName;
    public string LastName;
    public void SetId(int id)
    {
      _id = id;
    }
    public Person() { }
    public Person(SerializationInfo info, StreamingContext context)
    {
            FirstName = info.GetString("custom field 1");
            LastName = info.GetString("custom field 2");
     }

    public void GetObjectData(SerializationInfo info, StreamingContext context)
   {
            info.AddValue("custom field 1", FirstName);
            info.AddValue("custom field 2", LastName);
    }
}
```