Operatii de intrare-iesire

Sistemul de intrare-iesire din C# este constituit ca o ierarhie de clase. Intrucat limbajul C# utilizeaza clasele arhitecturii .NET, discutia despre intrari-iesiri in C# este deci o discutie despre sistemul de intrare-iesire al arhitecturii .NET in general.

Ne propunem sa trecem in revista o serie de clase utilizate pentru crearea si gestionarea fisierelor, citirea din si scrirerea in fisiere, precum si ideile principale in care limbajul C# implementeaza intrarile si iesirile.

Conceptul de stream. Programele C# efectueaza operatii de intrare-iesire prin intermediul stream-urilor. Un stream este o entitate abstracta care fie produce, fie consuma informatii. Sistemul de intrare-iesire din C# asociaza stream-urile cu dispozitivele fizice (de regula discul magnetic) utilizate efectiv. Toate stream-urile se comporta la fel, chiar daca dispozitivele fizice nu sunt aceleasi. De regula, utilizam aceleasi metode pentru a scrie atat in consola cat si intr-un fisier de pe disc sau intr-o retea.

Putem clasifica streamurile in streamuri de iesire (Output stream) si streamuri de intrare (Input stream). Streamurile Output sunt utilizate atunci cand datele sunt scrise intr-un dispozitiv extern, care poate fi: in fisier de pe discul fizic, o imprimanta, o retea sau un alt program. Streamurile Input sunt folosite pentru a citi date. Un stream Input poate proveni din orice sursa: tastatura, fisier de pe disc, retea, etc.

In cele ce urmeaza ne vom axa pe studiul operatiilor de intrare/iesire care utilizeaza fisiere. Conceptele aplicate pentru citirea sau scrierea fisierelor se aplica majoritatii dispozitive fizice. Astfel, acestea pot fi aplicate diverselor situatii care pot aparea in practica.

Clase pentru operatii de intrare-iesire. Spatiul de nume System.IO contine majoritatea claselor pentru lucrul cu fisiere. Vom analiza in cele ce urmeaza urmatoarele clase:

File	O clasa statica care expune numeroase metode statice pentru copierea, steregerea, crearea fisierelor.
Directory	O clasa statica care expune numeroase metode statice pentru copierea, steregerea, crearea directoarelor.
Path	O clasa care realizeaza operatii asupra stringurilor care reprezinta numele sau calea catre fisiere sau directoare.
FileInfo	Un fisier este reprezentat prin intermediul unui obiect de tipul acestei clase. Obiectul beneficiaza de o serie de metode pentru manipularea fisierului respectiv.
DirectoryInfo	Similar clasei FileInfo, insa obiectul reprezinta un director.
FileSystemInfo	Serveste drept clasa de baza pentru FileInfo si DirectoryInfo. Utilizant conceptul de polimorfism, creaza facilitatea de a lucra simultan cu fisiere si directoare.
FileStream	Un strem de tipul acestei clase permite scrierea intr-un sau citirea dintr-un fisier.
Stream	O clasa abstracta care sta la baza implementarii conceptului de stream in C#. Este clasa de baza pentru toate celelalte clase care reprezinta stream-uri, inclusiv FileStream.

StreamReader	Citeste caractere dintr-un stream. Un stream de acest tip poate fi creat prin impachetarea unui stream de tipul FileStream.
StreamWriter	Scrie caractere intr-un stream. La fel ca in cazul clasei de mai sus, stream de acest tip poate fi creat prin impachetarea unui stream de tipul FileStream.
FileSystemWatcher	Se utilizeaza pentru a monitoriza fisiere si directoare si expune evenimente pe care aplicatia utilizatorului le poate intercepta atunci cand au loc modificari in aceste locatii.

Vom aborda problema serializarii obiectelor utilizand spatiul de nume System.Runtime.Serialization si spatiile de nume pe care le contine. Mai precis, vom analiza clasa BinaryFormatter din spatiul de nume System.Runtime.Serialization.Formatters.Binary, care permite serializarea obiectelor intr-un stream ca date binare si deserializarea lor.

Clasele File si Directory. Aceste doua clase expun numeroase metode statice pentru manipularea fisierelor si directoarelor. Aceste metode fac posibile o serie de operatii asupra fisierelor si directoarelor precum si crearea unor streamuri FileStream. Toate metodele sunt statice, asadar acestea sunt apelate fara a crea instante ale clasei File sau Directory.

Cateva dintre cele mai utile metode ale clasei File sunt:

Copy()	Copiaza un fisier dintr-o locatie sursa intr-o locatie destinatie;
Create()	Creaza un fisier in directorul specificat (sau utilizand calea specificata);
Delete()	Sterge un fisier;
Open()	Returneaza un stream FileStream corespunzator stringului (care reprezinta calea) specificat;
Move()	Muta un fisier intr-o noua locatie. Fisierului i se poate specifica un alt nume in noua locatie.

Cateva metode utile ale clasei Directory sunt:

CreateDirectory()	Creaza un director utilizand calea specificata;
Delete( )	Sterge directorul specificat si toate fisierele continute de acesta;
GetDirectories()	Returneaza un tablou de tip string care contine numele directoarelor aflate in directorul specificat;
GetFiles()	Returneaza un tablou de tip string care contine numele fisierelor aflate in directorul specificat;
GetFileSystemEntries()	Returneaza un tablou de tip string care contine numele fisierelor si directoarelor aflate in directorul specificat;
Move()	Muta un director intr-o noua locatie. Directorului i se poate specifica un alt nume in noua locatie.

Exemplu: Programul returneaza fisierele sau/si directoarele continute pe unitatea D

```
using System;
using System.IO;
class Program
{
   public static void Main()
   {
      string str;
      char c;
      string[] tablou;
      do{
         Console.WriteLine(@"Type:
   f) for files;
   d) for directories
   e) for files and directories");
      str = Console.ReadLine();
      }
      while((str!="f") ^ (str!="d") ^(str!="e"));
      c=char.Parse(str);
      switch(c)
         {
            case 'f':
               tablou=Directory.GetFiles(@"D:\");
               foreach (string s in tablou)
                  Console.WriteLine(s);
               break;
            case 'd':
               tablou=Directory.GetDirectories(@"D:\");
               foreach (string s in tablou)
                  Console.WriteLine(s);
               break;
            case 'e':
              tablou=
      Directory.GetFileSystemEntries(@"D:\");
               foreach (string s in tablou)
                  Console.WriteLine(s);
               break;
         }
      Console.ReadKey();
      }
}
```

Clasa FileInfo. Contrar clasei File, clasa FileInfo nu este statica si nu contine metode statice. Asadar, clasa este utila doar cand este instantiata. Un obiect de tipul FileInfo reprezinta un fisier de pe disc sau dintr-o retea si poate fi creat doar prin furnizarea caii catre acel fisier. Spre exemplu, daca pe unitatea C: \ se afla fisierul Log.tex atunci instructiunea FileInfo unFisier=new FileInfo(@"C:\Log.tex"); creaza un obiect FileInfo care reprezinta acest fisier.

Multe dintre metodele expuse de FileInfo sunt similare celor din clasa File, insa deoarece clasa File este statica, la apelarea fiecarei metode este necesara specificarea unui paramentru de tip string care sa specifice locatia fisierului. Spre exemplu: instructiunile de mai jos realizeaza acelasi lucru:

a)
```
FileInfo unFisier=new FileInfo(@"C:\Log.tex");
if (unFisier.Exists)
Console.WriteLine("Fisierul exista");
```

b)
```
if(File.Exists("C:\\Log.tex"))
 Console.WriteLine("Fisierul exista");
```

In majoritatea cazurilor, nu are importanta care tehnica este utilizata pentru manipularea fisierelor, insa se poate utiliza urmatorul criteriu pentru a decide care este mai potrivita: -are sens sa utilizam metodele clasei File daca apelam o metoda o singura data. Un singur apel se executa rapid intrucat nu se trece in prealabil prin procesul de instantiere a unui obiect; -daca aplicatia realizeaza mai multe operatii asupra unui fisier atunci are sens instantierea unui obiect si utilizarea metodelor acestuia. Se salveaza timp intrucat obiectul refera deja fisierul corect, in timp ce clasa statica trebuie sa il gaseasca de fiecare data.

Clasa FileInfo expune de asemenea si proprietati care pot fi utilizate pentru manipularea fisierului referit. Multe dintre aceste proprietati sunt mostenite de la clasa FileSystemInfo, si deci se aplica atat clasei FileInfo cat si clasei DirectoryInfo. Proprietatile clasei FileSystemInfo sunt:

Attributes	Citeste sau scrie atributele fisierului sau directorului curent utilizand enumerarea FileAttributes;
CreationTimeUtc CreationTime	Citeste si scrie data si ora crearii fisierului sau directorului curent, in timp universal (UTC-coordinated universal time) sau nu;
Extension	Intoarce extensia fisierului, accesibila doar la citire;
Exists	Motoda abstracta acesibila doar la citire. Este implementata atat de FileInfo cat si de DirectoryInfo si determina daca fisierul sau directorul exista;
FullName	Proprietate virtuala accesibila doar la citire. Intoarce calea pana la fiser sau director;
LastAccessTimeUtc LastAccessTime	Citeste si scrie data si ora cand fisierul (directorul) a fost accesat (in UTC si non-UTC);
LastWriteTimeUtc	Citeste si scrie data si ora cand s-a scris in fisier (sau director)
LastWriteTime	(in UTC si non-UTC);
Name	Proprietate abstracta accesibila doar la citire. Intoarce calea pana la fiser sau director. Este implementata atat de FileInfo cat si de DirectoryInfo.

Proprietatile specifice clasei FileInfo sunt:

Directory	Intoarce un obiect de tipul DirectoryInfo care reprezinta directorul care contine fisierul curent. Proprietate accesibila la citire;
DirectoryName	Intoarce un string care reprezinta calea pana la directorul care contine fisierul curent. Proprietate accesibila la citire;
IsReadOnly	Determina daca fisierul este sau nu accesibil doar la citire;
Length	Intoarce o valoare de tip long care reprezinta dimensiunea fisierului in octeti. Proprietate accesibila la citire;

Un obiect FileInfo nu reprezinta un stream. Pentru a scrie intr-un sau citi dintr-un fisier trebuie creat un obiect de tipul Stream. Clasa FileInfo poate fi utila in acest sens intrucat expune cateva metode care returneaza obiecte Stream (in fapt obiecte de tipul unor clase derivate din clasa Stream). Vom prezenta aceste metode dupa descrierea streamurilor.

Clasa DirectoryInfo. Clasa DirectoryInfo se utilizeaza ca si clasa FileInfo. Cand este instantiata, obiectul creat reprezinta un director. Multe dintre metode sale sunt metode duplicat ale clasei Directory. Criteriul de alegere dintre metodele File ori FileInfo se aplica la fel si in acest caz.

Majoritatea proprietatilor sunt mostenite de la clasa FileSystemInfo. Doua dintre proprietati sunt specifice:

Parent	Intoarce un obiect DirectoryInfo reprezentant directorul care contine directorul curent. Proprietate acesibila la citire;

Root	Intoarce un obiect DirectoryInfo reprezentand unitatea directorului curect. Spre exemplu C:\ Proprietate accesibila la citire.

Exercitiu: rescrieti exemplul anterior folosind metodele si proprietatile claselor FileInfo si DirectoryInfo.

Clasa FileStream. Un obiect FileStream reprezinta un stream care indica spre un anumit fisier de pe disc sau dintr-o retea. Desi, clasa expune mai multe metode pentru citirea sau scrierea octetilor dintr-un sau intr-un fisier, de cele mai multe ori vom utiliza un obiect StreamReader sau StreamWriter pentru a realiza aceste operatii. Aceasta deoarece clasa FileStream opereaza cu octeti (bytes) sau tablouri de octeti, in timp ce clasele StreamReader si StreamWriter opereaza cu date caracter. Este mai convenabil lucrul cu date caracter, insa anumite operatii, spre exemplu accesul unor date aflate undeva la mijlocul fisierului, pot fi realizate doar de un obiect FileStream.

Un obiect FileStream poate fi creat in mai multe moduri. Constructorul are mai multe versiuni supraincarcate, cea mai simpla forma fiind:

```
FileStream unfisier = new FileStream(numeFisier, FileMode.<Member>);
```

unde enumerarea FileMode are mai multi membrii care specifica modul cum fisierul este deschis sau creat. O alta versiune, des intrebuintata este:

```
FileStream unfisier = new FileStream(numeFisier, FileMode.<Member>, FileAccess.<Member>);
```

unde enumerarea FileAccess specifica scopul streamului. Membrii acestei enumerari sunt: Read (deschide fisierul pentru citire), Write (deschide fisierul pentru scriere), ReadWrite (deschide fisierul pentru citire si scriere).

Daca se incearca realizarea unei alte operatii decat cea specificata de enumerarea FileAccess atunci va fi lansata o exceptie.

In prima versiune a constructorului, cea care nu contine parametrul FileAccess, este utilizata valoarea default FileAccess.ReadWrite.

Enumerarea FileMode, impreuna cu o descrire a comportarii membrilor acesteia atat in cazul in care fisierul exista sau nu, este prezentata in tabelul:

Membru	Fisierul exista	Fisierul nu exista
Append	Fisierul este deschis cu stream-ul pozitionat la sfarsitul fisierului. Poate fi utilizat numai in conjunctie FileAccess.Write.	Este creat un fisier. Poate fi utilizat in conjunctie cu cu FileAccess.Write.
Create	Fisierul este distrus si un nou fisier este creat in locul acestuia.	Este creat un nou fisier.
CreateNew	Este lansata o exceptie.	Este creat un nou fisier.
Open	Fisierul este deschis cu stream-ul pozitionat la inceputul fisierului.	Este lansata o exceptie.
OpenOrCreate	Fisierul este deschis cu stream-ul pozitionat la inceputul fisierului.	Este creat un nou fisier.
Truncate	Fisierul este deschis si continutul este sters. Stream-ul este pozitionat la inceputul fisierului. Este retinuta data de creare a fisierului original.	Este lansata o exceptie.

Exemplu: (Utilizarea clasei FileStream. Programul copiaza un fisier)

```
using System;
using System.IO;
class Scrie_Octeti
{
    public static void Main()
   {
   if (File.Exists(@"D:\Curs6.ppt"))
   { try
       {
        int i;
        FileStream fin, fout;
        fin = new FileStream(@"D:\Curs6.ppt",
                              FileMode.Open);
        fout = new FileStream(@"D:\Curs6duplicat.ppt“
                              , FileMode.OpenOrCreate);
do {
        i = fin.ReadByte();
        if (i != -1)
          { fout.WriteByte((byte)i); }
      }
 while (i != -1);
          fin.Close();
          fout.Close();
          }
  catch (IOException exc)
    {
     Console.WriteLine(exc.Message + "Nu poate
     deschide sau accesa unul dintre fisiere");
            return;
          }
       }
    }
}
```

Ambele clase File si FileInfo expun metodele OpenRead() si OpenWrite() care permit crearea de obiecte FileStream. Prima metoda deschide un fisier doar pentru citire, iar cea de-a doua metoda doar pentru scriere. Aceste metode ofera de fapt shortcut-uri incat nu este necesara precizarea parametrilor necesari constructorului clasei FileStream. De exemplu, codul de mai jos deschide fisierul "Data.txt" pentru citire:

```
FileStream unFisier= File.OpenRead("Data.txt");
```

Un rezultat similar se obtine astfel:

```
FileInfo unFisierInfo=new FileInfo("Data.txt");
FileStream unFisier=unFisierInfo.OpenRead();
```

Pozitia in fisier. Clasa FileStream mentine un pointer intern care indica o locatie in fisier unde se va produce urmatoarea operatie de citire sau de scriere. In cele mai multe cazuri, cand un fisier este deschis, pointerul indica inceputul fisierului, insa aceasta pozitie poate fi modificata. Acest fapt permite unei aplicatii sa citeasca sau sa scrie oriunde in fisier.

Metoda care implementeaza aceasta functionalitate este metoda Seek(), care are doi parametrii. Primul parametru, un parametru de tip long, specifica cu cati octeti (bytes) trebuie deplasat pointerul, iar al doilea parametru este enumerarea SeekOrigin care contine trei valori: Begin, Current si End.

Spre exemplu, comanda unfisier.Seek(3, SeekOrigin.Begin) muta pointerul 3 octeti inainte fata de pozitia initiala, sau unfisier.Seek(-5, SeekOrigin.End) muta pointerul 5 pozitii inapoi fata de pozitia de sfarsit a fisierului.

Clasele StreamReader si StreamWriter acceseaza fisierele secvential si nu permit manipularea pointerului in acest fel.

Citirea datelor. Citirea datelor utilizand clasa FileStream nu se face la fel de usor ca in cazul citirii cu ajutorul clasei StreamReader. Aceasta deoarece clasa FileStream lucreaza cu octeti in forma bruta, neprelucrati. Acest fapt permite clasei FileStream o mare flexibilitate in citirea oricaror fisiere, precum imagini, muzica, video, etc. Costul acestei flexibilitati este acela ca nu se poate utiliza un obiect FileStrem pentru a citi date in mod direct si a le converti intr-un string asa cum se poate face cu StreamReader. Cu toate acestea, exista o serie de clase care convertesc tablourile de octeti in tablouri de caractere si viceversa.

Metoda ReadByte() a fost utilizata in exemplul anterior.

O alta metoda a clasei FileStream este Read() care citeste date dintr-un fisier si ii scrie intr-un tablou de tip byte. Metoda intoarce un int care reprezinta numarul de octeti cititi din stream si are trei parametrii. Primul paramentru reprezinta un tablou de tip byte in care vor fi scrise datele, al doilea parametru de tip int specifica pozitia elementului din tablou unde va incepe scrierea (de regula acesta este 0 daca se incepe scrierea la primul element din tablou), iar al treilea parametru indica numarul de octeti cititi din fisier.

Urmatoarea aplicatie citeste 200 de octeti din fisierul Program.cs, incepand de la pozitia 113. Octetii sunt decodificati si convertiti in caractere, utilizand clase din spatiul de nume System.Text;

Scrierea datelor. Procesul de scrire a datelor este similar, trebuie creat un tablou de tip byte. In acest sens, cream un tablou de caractere, apoi sa il codificam intr-un tablou byte si in final il scriem in fisier. Vezi exemplul scrierea datelor intr-un fisier.

Exemplu: Citirea datelor dintr-un fisier

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.IO;

namespace CitesteFisier
{
   class Program
   {
      static void Main(string[] args)
      {
         byte[] byData = new byte[200];
         char[] charData = new Char[200];
         try
         {
            FileStream aFile = new
     FileStream("../../Program.cs", FileMode.Open);
            aFile.Seek(113, SeekOrigin.Begin);
            aFile.Read(byData, 0, 200);
         }
catch (IOException e)
    {
    Console.WriteLine("A fost lansata o
              exceptie IO!");
    Console.WriteLine(e.ToString());
    Console.ReadKey();
            return;
      }
   Decoder d = Encoding.UTF8.GetDecoder();
   d.GetChars(byData, 0, byData.Length,
              charData, 0);
    Console.WriteLine(charData);
    Console.ReadKey();
      }
   }
}
```

Exemplu: Scrierea datelor intr-un fisier

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.IO;

namespace ScrieInFisier
{
    class Program
    {
        static void Main(string[] args)
        {
            byte[] byData;
            char[] charData;
            try
            {
                FileStream unFisier = new FileStream("Temp.txt",
        FileMode.Create);
                charData = "Un string pe care il scriem in
        fisier.".ToCharArray();
                byData = new byte[charData.Length];
                Encoder e = Encoding.UTF8.GetEncoder();
                e.GetBytes(charData, 0, charData.Length, byData,
        0, true);
// Mutam pointerul la inceputul fisierului.
                unFisier.Seek(0, SeekOrigin.Begin);
                unFisier.Write(byData, 0,
        byData.Length);
            }
            catch (IOException ex)
            {
                Console.WriteLine("S-a produs o
        exceptie IO!");
                Console.WriteLine(ex.ToString());
                Console.ReadKey();
                return;
            }
        }
    }
}
```

Clasele StreamReader si StreamWriter. A lucra cu tablouri de octeti nu este intodeauna placut. O cale mai simpla, odata ce este creat un stream de tipul FileStream, este de a-l impacheta intr-un stream de caractere.

Pentru a impacheta un stream octet intr-un stream caracter trebuie sa utilizati subclasele StreamReader si StreamWriter ale claselor abstracte: TextReader si TextWriter. Odata ce stream-ul este impachetat se pot utiliza metodele acestor doua clase (StreamReader si StreamWriter) pentru a manipula fisierul. Daca nu este necesar schimbarea pozitiei pointerului in interiorul fisierului atunci aceste clase fac lucrul mult mai usor.

Clasa StreamReader contine mai multe metode, dintre care cele mai importante sunt Read(), ReadLine(). Clasa StringWriter contine, (pe langa alti membri) metodele Write() si WriteLine(). Acestea de utilizeaza la fel ca in cazul metodelor similare puse la dispozitie de clasa Console.

Exista mai multe moduri de a crea un obiect StreamReader (sau StreamWriter). Daca deja exista un stream FileStream atunci pentru a crea un StreamWriter se proceaza astfel:

```
FileStream unFisier= new FileStream("Log.tex", FileMode.CreateNew);
StreamWriter sw=new StreamWriter(unFisier);
```

Un obiect StreamWriter poate fi creat direct, utilizand un fisier:

```
StreamWriter sw=new StreamWriter("Log.txt", true);
```

Acest constructor are ca parametrii calea pana la fisier si un parametru bolean care daca este true atunci fisierul este deschis, iar datele continute de acesta sunt retinute. Daca fisierul nu exista atunci se creaza un nou fisier. Daca parametrul este false atunci daca fisierul exista acesta este deschis si continutul sters, iar daca nu exista atunci se creaza un fisier.

Exemplu (Utilizarea clasei StreamWriter)

```
using System;     using System.IO;
class Scrie_Caracter_Octeti {
   public static void Main()    {
      try  {
         string str;
         FileStream fout;
         fout = new FileStream("fisier.txt", FileMode.Create);
         StreamWriter fstr_out = new StreamWriter(fout);
         Console.WriteLine("Introduceti textul fisierului. (Daca doriti sa terminati tastati: stop ");
         do
         {
            str = Console.ReadLine();
            if (str != "stop")
            {
               str = str + "\r\n";
               try
               { fstr_out.Write(str); }
               catch (IOException exc)
               {  Console.WriteLine(exc.Message + "Eroare la scrierea in fisier");   return;    }
            }
         }
         while (str != "stop");
         fstr_out.Close();
      }
      catch (IOException exc)
      {
         Console.WriteLine(exc.Message + "Nu poate deschide sau accesa fisierul");
         return;
      }
   }
}
```

Exemplu (Utilizarea Clasei StreamReader)

```
using System;
using System.IO;
class Citeste_Octeti_Caracter
{
   public static void Main()
   {
      string str;
      try
      {
         FileStream fin = new FileStream(“fisier.txt", FileMode.Open);

         StreamReader fstr_in = new StreamReader(fin);
         while ((str = fstr_in.ReadLine()) != null)
         {
            Console.WriteLine(str);
         }
         fstr_in.Close();
      }
      catch (IOException exc)
      {
         Console.WriteLine(exc.Message + "Fisierul nu exista sau nu poate fi accesat");
         return;
      }
   }
}
```

Clasa Stream. Clasa Stream, din spatiul de nume System.IO, sta la baza implementarii conceptului de stream in C#. Clasa Stream reprezinta un stream octet si este clasa de baza pentru toate celelalte clase care reprezinta stream-uri. Aceasta clasa este abstracta, deci nu puteti crea instante ale acestei clase. Ea defineste mai multe metode atat pentru citirea cat si pentru scrierea datelor care sunt implementate de clasele derivate. Nu toate stream-urile implementeaza ambele categorii de date. Unele stream-uri pe care le creati sunt accesibile la scriere, altele la citire.. Dintre metodele clasei stream amintim: void Close(), void Flush(), int ReadByte(), long Seek(), void WriteByte(), etc. intre proprietati amintim: bool CanRead, bool CanSeek, bool CanWrite, etc.

Serializarea obiectelor. Aplicatiile pe care le cream necesita de multe ori stocarea datelor pe hard disc. In exemplele anterioare fisierele au fost construite octet cu octet (sau caracter cu caracter). De multe ori, aceasta cale nu este cea mai convenabila. Uneori este mai bine ca datele sa fie stocate in forma in care se gasesc, adica ca obiecte.

Arhitectura .NET ofera acea infrastructura pentru serializarea obiectelor in spatiile de nume System.Runtime.Serialization si System.Runtime.Serialization.Formatters, prin clasele specifice care sunt puse la dispozitie. Sunt posibile doua implemetari:

System.Runtime.Serialization.Formatters.Binary Acest spatiu de nume contine clasa BinaryFormatter care este capabila sa serializeze obiecte in date binare si viceversa.

System.Runtime.Serialization.Formatters.Soap Acest spatiu de nume contine clasa SoapFormatter care este capabila sa serializeze obiecte in format SOAP pentru date XML, si viceversa.

In cele ce urmeaza vom aborda serializarea obiectelor utilizand BinaryFormatter. In fapt, ambele clase BinaryFormatter si SoapFormatter implementeaza interfata IFormatter care furnizeaza metodele:

void Serialize(Stream stream, object source)	Serializeaza source in stream.
object Deserialize(Stream stream)	Deserializeaza datele din stream si returneaza obiectul rezultat.

Serializarea utilizand BinaryFormatter se poate face simplu, spre exemplu:

```
IFormatter s = new BinaryFormatter();
s.Serialize(streamulMeu, obiectulMeu);
```

Deserializarea este de asemenea simpla:

```
IFormatter s = new BinaryFormatter();
TipObiect noulMeuObiect= s.Deserialize(streamulMeu) as TipObiect;
```

Aceste secvente de cod sunt valide in majoritatea circumstantelor.

Urmatorul exemplu arata cum se procedeaza in practica.

a) Executati programul. Veti observa ca apare o eroare la serializare.

b) Stergeti comentariul din fata atributului [Serializable]. Rulati din nou programul si observati diferenta.

c) De asemenea observati ca stringul Observatii nu a fost serializat intrucat atributul sau este [Nonserialized]

```
using System;   using System.Collections.Generic;
using System.Collections;   using System.Linq;
using System.Text;    using System.IO;
using System.Runtime.Serialization;
using System.Runtime.Serialization.Formatters.Binary;

namespace Magazin
{
   //[Serializable]
   public class Produs
   {  public long Id;  public string Nume;   public double Pret;
  [NonSerialized]  string Observatii;
      public Produs(long id, string nume, double pret, string observatii)
      {
         Id = id;
         Nume = nume;
         Pret = pret;
         Observatii = observatii;
      }
      public override string ToString()
      {
         return string.Format("{0}: {1} ({2:#.##} lei) {3}", Id, Nume, Pret,  Observatii);
      }
   }
```

```
class Program
   {
      static void Main(string[] args)
      {
         try
         {
            // Cream produse.
            ArrayList produse = new ArrayList();
            produse.Add(new Produs(1, "minge", 100.0, "de calitate"));
            produse.Add(new Produs(2, "rochie", 500.0, "cam scumpa"));
            produse.Add(new Produs(4, "joc lego", 50.0, "pentru copii"));
            Console.WriteLine("Scriem produsele:");
            foreach (Produs produs in produse)
            {
               Console.WriteLine(produs);
            }
            Console.WriteLine();
            // cream serializatorul.
            IFormatter serializator = new BinaryFormatter();
            // Serializam produsele.
            FileStream salvam =   new FileStream("Produse.bin", FileMode.Create, FileAccess.Write);
            serializator.Serialize(salvam, produse);
            salvam.Close();
```

```
// deserializam produsele.
            FileStream incarcamFisier =   new FileStream("Produse.bin", FileMode.Open, FileAccess.Read);
            ArrayList produseSalvate = serializator.Deserialize(incarcamFisier) as ArrayList;
            incarcamFisier.Close();

            Console.WriteLine("Produse incarcate:");
            foreach (Produs produs in produseSalvate)
            {
                Console.WriteLine(produs);
            }
        }
        catch (SerializationException e)
        {
            Console.WriteLine("O execptie s-a produs la serializare!");
            Console.WriteLine(e.Message);
        }
        catch (IOException e)
        {
            Console.WriteLine("S-a produs o exceptie IO!");
            Console.WriteLine(e.ToString());
        }
        Console.ReadKey();
    }
  }
}
```

Monitorizarea fisierelor si directoarelor. De multe ori o aplicatie trebuie sa faca mai mult decat sa citeasca dintr-un fisier sau sa scrie intr-un fisier. Spre exemplu, poate fi important de stiut cand fisierele si directoarele sunt modificate.

Clasa care ne permite sa monitorizam fisierele si directoarele este FileSystemWatcher. Aceasta expune cateva evenimente pe care aplicatiile pot sa le intercepteze.

Procedura de utilizare a clasei FileSystemWatcher este urmatoarea: In primul rand trebuiesc setate o serie de proprietati care specifica unde si ce se monitorizeaza si cand trebuie lansat evenimentul pe care aplicatia urmeaza sa-l trateze. Apoi trebuie precizate adresele metodelor de tratare a evenimentelor astfel incat acestea sa fie apelate cand evenimentele sunt lansate. In final, aplicatia este pornita si se asteapta producerea evenimentelor.

Proprietatile care trebuie setate sunt:

Path	Se seteaza locatia fisierului sau directorului de monitorizat.
NotifyFilter	O combinatie de valori ale enumerarii NotifyFilters care specifica ce se urmareste in fisierele monitorizate. Acestea reprezinta proprietati ale fisierelor sau directoarelor. Daca vreo proprietate specificata se modifica atunci este lansat un eveniment. Posibilele valori ale enumerarii sunt: Attributes, CreationTime, DirectoryName, FileName, LastAccess, LastWrite, Security si Size. Acestea pot fi combinate utilizand operatorul OR.
Filter	Un filtru care specifica care fisiere se monitorizeaza (spre exemplu *.txt)

Odata setate aceste proprietati se pot trata urmatoarele evenimente: Changed, Created, Deleted si Renamed. Fiecare eveniment este lansat odata ce un fisier sau un director care satisface proprietatile: Path, NotifyFilter si Filter este modificat.

Dupa setarea evenimentelor trebuie setata proprietatea EnableRasingEvents cu true pentru a incepe monitorizarea.

Exemplu: FileSystemWatcher. La executie, includeti un parametru de comanda reprezentand directorul de monitorizat

```
using System;
using System.IO;

public class Watcher
{
   public static void Main()
   {
      Run();
   }

   public static void Run()
   {
      string[] args = System.Environment.GetCommandLineArgs();

      // If a directory is not specified, exit program.
      if (args.Length != 2)
      {
         // Display the proper way to call the program.
         Console.WriteLine("Usage: Watcher.exe (directory)");
         return;
      }
```

```
// Create a new FileSystemWatcher and set its properties.
        FileSystemWatcher watcher = new FileSystemWatcher();
        watcher.Path = args[1];
        /* Watch for changes in LastAccess and LastWrite times, and
           the renaming of files or directories. */
        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.
        watcher.EnableRaisingEvents = true;

        // Wait for the user to quit the program.
        Console.WriteLine("Press \'q\' to quit the sample.");
        while (Console.Read() != 'q') ;
    }

    // Define the event handlers.
    private static void OnChanged(object source, FileSystemEventArgs e)
    {
        // Specify what is done when a file is changed, created, or deleted.
        Console.WriteLine("File: " + e.FullPath + " " + e.ChangeType);
    }

    private static void OnRenamed(object source, RenamedEventArgs e)
    {
        // Specify what is done when a file is renamed.
        Console.WriteLine("File: {0} renamed to {1}", e.OldFullPath, e.FullPath);
    }
}
```