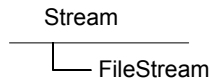


Файлов вход/изход

Абстрактен клас **Stream** (**System.IO**) – четене и запис на байтове.

Операции:

1. **Запис** – преобразуване на данните от структура данни в поток.
2. **Четене** – преобразуване на данните от поток в структура данни (масив от байтове).
3. **Търсене** – търсене и модифициране на текущата позиция в поток.



Клас **FileStream**

Чете и записва байтове; поддържа произволен достъп, синхронни и асинхронни операции.

```
public FileStream (string path, FileMode mode);
```

path – път за файла;

mode – режим:
FileMode.Append
FileMode.Create
FileMode.Open

```
public FileStream (string path, FileMode mode, FileAccess access);
```

access – достъп:
FileAccess.Read
FileAccess.Write
FileAccess.ReadWrite

Свойства

CanRead, CanWrite, CanSeek – върнатата стойност от тип **bool** показва дали потокът поддържа четене | запис | търсене.

Length – връща дължината на потока в байтове.

Position – връща/установява текущата позиция на потока.

Методи

```
public override void Close ();
```

Затваря файла и освобождава ресурсите, свързани с файловия поток.

```
public override void WriteByte (byte value);
```

Записва value в текущата позиция на файла.

```
public override void Write (byte[] array, int offset, int count);
```

Записва масива от байтове array в потока. Взима count на брой байтове от array, като започва от отместването в байтове offset в array.

```
public override int ReadByte ();
```

Чете един байт от файла и премества позицията за четене с един байт. Преобразува байта в int или връща -1 при достигане край на файла.

```
public override long Seek (long offset, SeekOrigin origin);
```

Позиционира потока в позиция offset спрямо позицията origin, определяща относителната позиция на търсене:

SeekOrigin.Begin – начало;

SeekOrigin.Current – текуща позиция;

SeekOrigin.End – край.

```
public override void SetLength (long value);
```

Установява value като дължина на потока.

Пример:

```
using System;
using System.IO;
class FileStreamApp
{ static void Main (string[] args)
{
    byte[] buf = new Byte[] {73,110,32,118,105,110,111};
    // Създаване и запис във файл
    FileStream s = new FileStream ("My.txt", FileMode.Create);
    s.Write (buf, 0, buf.Length);
    s.Close ();
    // Отваряне и четене от файл
    s = new FileStream ("My.txt", FileMode.Open);
    int i;
    string str = "";
    if (s.CanRead)
        for (i = 0; (i = s.ReadByte()) != -1; i++)
            str += (char);
    s.Close ();
    Console.WriteLine (str);
    // In vino
```

```
// Отваряне и добавяне на данни към файл
byte[] bufadd = new Byte[] {32, 118, 101, 114, 101, 105, 116, 97, 115, 33};
s = new FileStream("My.txt", FileMode.Append, FileAccess.Write);
s.Write(bufadd, 0, bufadd.Length);
s.Close();
// In vino veritas!

// Обновяване на данни
byte[] bufnew = new Byte[] {102};
s = new FileStream("My.txt", FileMode.Open);
Console.WriteLine ("Length: {0}, Position: {1}", s.Length, s.Position);
if (s.CanSeek)
{
    s.Seek(8, SeekOrigin.Begin);
    Console.WriteLine ("Position: {0}", s.Position);
    s.Write(bufnew, 0, bufnew.Length);
}
str = "";
s.Seek(0, SeekOrigin.Begin);
for (i=0; i = s.ReadByte()) != -1; i++) str += (char);
Console.WriteLine (str);
// Length: 16, Position: 0
// Position: 8
// In vino veritas!
```

```
// Установяване дължина на поток
str = "";
s.SetLength(s.Length - 9);
s.Seek(0, SeekOrigin.Begin);
for (i = 0; (i = s.ReadByte()) != -1; i++)
    str += (char);
Console.WriteLine (str);
s.Close();
// In vino
}
```

Абстрактни класове TextReader и TextWriter

Клас TextWriter – за записване на последователност от символи.

Клас TextReader – за четене на последователност символи

```

    TextReader
    |
    |--- StreamReader
    |--- StringReader
    |
    |--- StreamWriter
    |--- StringWriter
    
```

Класове StreamReader и StreamWriter

Клас StreamWriter – записва последователност от символи в поток **stream** с определено кодиране **encoding** (подразбира се UTF-8).

```
public StreamWriter (Stream stream);
public StreamWriter (Stream stream, Encoding encoding);
```

Клас StreamReader – чете символи от байтов поток **stream** с определено кодиране.

```
public StreamReader (Stream stream);
public StreamReader (Stream stream, Encoding encoding);
public override string ReadToEnd ();
```

Чете от текущата позиция на потока до неговия край.

Пример:

```
using System;
using System.IO;
class StreamReaderWriter
{
    static void Main (string[] args)
    {
        // Запис във файл
        FileStream s = new FileStream ("My.txt", FileMode.Create);
        StreamWriter w = new StreamWriter(s);
        w.WriteLine ("In vino veritas!");
        w.Close ();

        // Четене от файл
        s = new FileStream ("My.txt", FileMode.Open);
        StreamReader r = new StreamReader (s);
        string str;
        while ((str = r.ReadLine()) != null) Console.WriteLine (str);
        r.Close ();
    }
}
```

Път към файл:

- 1. Двойни обратно наклонени черти \\
FileStream s = new FileStream ("C:\\Temp\\My.txt", FileMode.Create);**
- 2. Наклонени черти (стил UNIX) /
FileStream s = new FileStream ("C:/Temp/My.txt", FileMode.Create);**
- 3. Знак @, който подтиска контролния символ
FileStream s = new FileStream (@ "C:\\Temp\\My.txt", FileMode.Create);**

Пример: Текстов файл съдържа информация за стоките в склад: номер, име, налично количество, цена.

1. Създава текстов файл, на който всеки ред съдържа информацията за една стока.
2. Разпечатва текстовия файл.
3. Реализира търсене на стока по зададен номер.

```
using System;
using System.IO;
using System.Text.RegularExpressions;
class InventoryItem // Клас СТОКА
{
    private int key; // номер
    public int Key
    {
        get { return key; }
        set { key=value; }
    }
    private string name; // име
    private int units; // налично количество
    private float price; // цена
}
```

```
public InventoryItem()
{
    try
    {
        Console.Write ("Номер:\t"); key = int.Parse (Console.ReadLine ());
        Console.Write ("Име:\t"); name = Console.ReadLine ();
        Console.Write ("Количество:\t");
        units = int.Parse (Console.ReadLine ());
        Console.Write ("Цена:\t");
        price = float.Parse (Console.ReadLine ());
    }
    catch (Exception e)
    {
        Console.WriteLine ("Данните се игнорират!");
        Console.WriteLine (e.Message);
        throw;
    }
}
public InventoryItem (int key, string name, int units, float price)
{
    this.key = key;
    this.name = name;
    this.units = units;
    this.price = price;
}
```

```
public InventoryItem (string s)
{
    try
    {
        // Разделителят е многократно празна позиция като [\n\r\t]
        Regex o = new Regex(@"[\s]+");
        string[] str = o.Split (s);
        key = int.Parse (str[0]);
        name = str[1];
        units = int.Parse (str[2]);
        price = float.Parse (str[3]);
    }
    catch (Exception e)
    {
        Console.WriteLine ("Данните се игнорират!");
        Console.WriteLine (e.Message);
        throw;
    }
}
public override string ToString ()
{
    return "" + key + "\t" + name + "\t" + units + "\t" + price;
}
```

```
class Test
{
    static void Main (string[] args)
    {
        string file = "inventory.txt";
        InventoryItem find = null;
        if (CreateInventory (file))
            Console.WriteLine ("Успешна операция Създаване на файл!");
        if (ReadInventory (file))
            Console.WriteLine ("Успешна операция Четене от файл!");
        if (SearchInventory (file, 100, out find))
            Console.WriteLine ("Стоката {0} е намерена!", find);
        else
            Console.WriteLine ("Неуспешна операция Търсене на стока!");
    }
}
// Създаване на текстов файл със стоки
public static bool CreateInventory (string fileName)
{
    FileStream fs;
    StreamWriter w;
    InventoryItem item;
```

```
try
{
    fs = new FileStream (fileName, FileMode.Create);
    w = new StreamWriter (fs);
    try
    {
        Console.WriteLine ("Брой стоки: ");
        int number = int.Parse(Console.ReadLine());
        for (int i = 1; i <= number; i++)
        {
            try
            {
                item = new InventoryItem ();
                w.WriteLine (item);
            }
            catch (Exception e)
            {
                Console.WriteLine ("Грешка при запис.");
                Console.WriteLine (e.Message);
            }
        }
    }
    catch (Exception e)
    {
        Console.WriteLine ("Данните се игнорират.");
        Console.WriteLine (e.Message);
    }
}
```

```

catch (Exception e)
{
    Console.WriteLine ("Файлът {0} не може да се създаде.", fileName);
    Console.WriteLine (e.Message);
    return false;
}
w.Close ();
Console.WriteLine ("Файлът {0} е създаден.", fileName);
return true;
}
// Четене от файл и печат на данните
public static bool ReadInventory (string fileName)
{
    FileStream fs;
    StreamReader r = null;
    InventoryItem item;
    string t;

```

```

try
{
    fs = new FileStream (fileName, FileMode.Open);
    r = new StreamReader (fs);
    try
    {
        while ((t = r.ReadLine()) != null)
        {
            item = new InventoryItem (t);
            Console.WriteLine (item);
        }
    }
    catch (Exception e)
    {
        Console.WriteLine ("Грешка при четене.");
        Console.WriteLine (e.Message);
        r.Close ();
        return false;
    }
}
catch (Exception e)
{
    Console.WriteLine ("Файлът {0} не може да се отвори.", fileName);
    Console.WriteLine (e.Message);
    r.Close ();
    return false;
}

```

```

r.Close ();
Console.WriteLine ("Файлът {0} е прочетен.", fileName);
return true;
}
// Търсене на стока със зададен номер
public static bool SearchInventory (string fileName, int keySearch,
out InventoryItem findItem)
{
    FileStream fs;
    StreamReader r = null;
    InventoryItem item;
    findItem = null;
    string t;

```

```

try
{
    fs = new FileStream (fileName, FileMode.Open);
    r = new StreamReader (fs);
    try
    {
        while ((t = r.ReadLine()) != null)
        {
            item = new InventoryItem (t);
            if (item.Key == keySearch)
            {
                findItem = new InventoryItem (t);
                r.Close ();
                return true;
            }
        }
    }
    catch (Exception e)
    {
        Console.WriteLine ("Грешка при четене.");
        Console.WriteLine (e.Message);
        r.Close ();
        return false;
    }
}

```

```

catch (Exception e)
{
    Console.WriteLine ("Файлът {0} не може да се отвори.", fileName);
    Console.WriteLine (e.Message);
    r.Close ();
    return false;
}
r.Close ();
Console.WriteLine ("Файлът {0} е прочетен.", fileName);
return false;
}
}

```

Класът `System.Text.RegularExpressions.Regex` представя неизменен правилен израз.
Инициализира обекта с определен правилен израз `pattern`.

`\s` съответства на празен символ като `[\n\r\t\f]`
[символи] съответства на един символ от списъка
+ едно или повече съответствия;
@ отменя разглеждането на `\` като управляващ символ.

Класове BinaryReader и BinaryWriter

Клас BinaryWriter – записва примитивни типове в двоичен код в поток и поддържа записания низ в определено кодиране (по подразбиране UTF-8 за кодиране на низовете).

```
public BinaryWriter (Stream output);
```

```
public virtual void Write(char[] chars);
```

Записва символен масив chars в потока output и премества позицията на потока според използваното кодиране.

Клас BinaryReader – чете примитивни типове данни като двоични стойности при определено кодиране (по подразбиране UTF-8 за кодиране на низове).

```
public BinaryReader(Stream input);
```

```
public virtual int Read();
```

Чете символи от потока input и премества позицията на потока според използваното кодиране.

Пример:

```
using System;
using System.IO;
class Binary
{
    static void Main (string[] args)
    {
        string s = "In vino veritas!";
        // Създаване и запис във файл
        FileStream fs = new FileStream ("My.txt", FileMode.Create);
        BinaryWriter w = new BinaryWriter (fs);
        w.Write (s);
        w.Close ();
        fs.Close ();
        // Отваряне и четене от файл
        fs = new FileStream ("My.txt", FileMode.Open);
        BinaryReader r = new BinaryReader (fs);
        for (int i = 0; (i = r.Read ()) != -1; i++) Console.WriteLine ("{0,-4}", i);
        Console.WriteLine ();
        r.Close ();
        fs.Close ();
        // 16 73 110 32 118 105 110 111 32 118 101 114 105 116 97 115 33
    }
}
```

Класове, свързани с файловата система

Клас FileSystemInfo – абстрактен клас с методи за обработка на файлове и директории.

Свойства:

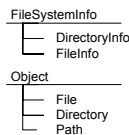
FullName – връща пълното име на директория или файл;

LastWriteTime – дава/установява времето, когато във файла или директорията е записвано;

Parent – дава директорията родител;

Attributes – дава/установява атрибутите на файловете:

FileAttributes.Archive, FileAttributes.Compressed, FileAttributes.Directory, FileAttributes.Encrypted, FileAttributes.Hidden, FileAttributes.Normal, FileAttributes.System и др.



Клас DirectoryInfo – съдържа методи за създаване, преместване и преглеждане съдържанието на директории и поддиректории.

```
public DirectoryInfo (string path);
```

```
public FileInfo[] GetFiles ();
```

Връща списък от файлове от текущата директория.

```
public DirectoryInfo[] GetDirectories ();
```

Връща поддиректориите на текущата директория.

```
public DirectoryInfo CreateSubdirectory (string path);
```

Създава поддиректория/поддиректории на определения path.

```
public override void Delete ();
```

Изтрива обекта, ако е празен.

Клас Directory – съдържа статични методи за създаване, преместване и преглеждане съдържанието на директории и поддиректории.

```
public static string GetCurrentDirectory ();
```

Връща текущата директория на приложението.

```
public static DirectoryInfo CreateDirectory (string path);
```

Създава всички директории и поддиректории, както са определени от path.

```
public static string[] GetLogicalDrives ();
```

Връща имената на логическите устройства на компютъра във формата драйв:\.

```
public static void Delete (string path);
```

Изтрива празна директория от определения path.

```
public static void Move (string sourceDirName, string destDirName);
```

Премества файл или директория и нейното съдържание на ново място.

```
public static bool Exists (string path);
```

Определя дали даден path съответства на съществуваща директория на диска.

```
public static string[] GetFiles (string path);
```

Връща имената на файловете с определена директория path.

Клас FileInfo – съдържа методи за създаване, копиране, изтриване, преместване и отваряне на файлове.

```
public FileInfo (string fileName);
```

Свойства:

- Name** – дава името на файла;
- Length** – дава размера на файла;
- Exists** – дава стойност, показваща дали файлът съществува.

Клас File – съдържа статични методи за създаване, копиране, изтриване, преместване и отваряне на файлове.

```
public static FileStream Create (string path);
```

Създава файл с определен path.

```
public static FileStream Open (string path, FileMode mode);
```

Отваря файлов поток с определен path , режим mode и достъп за четене/запис.

```
public static StreamWriter CreateText (string path);
```

Създава или отваря файл path за запис на текст, кодиран в код UTF-8.

```
public static StreamReader OpenText (string path);
```

Отваря съществуващ файл path за четене на текст, кодиран в код UTF-8.

```
public static StreamWriter AppendText (string path);
```

Отваря съществуващ файл path за добавяне на текст, кодиран в код UTF-8.

Пример:

```
using System;
using System.IO;
class DirInfoApp
{ static void Main (string[] args)
{
// Разпечатва имената на логическите устройства на компютъра
string[] drives = Directory.GetLogicalDrives ();
Console.WriteLine ("Логически устройства:");
foreach (string s in drives)
Console.WriteLine (s);
// Разпечатва името на текущата директория на приложението
DirectoryInfo dir = new DirectoryInfo(Directory.GetCurrentDirectory());
Console.WriteLine ("Текуща директория: {0}", dir.FullName);
// Console.WriteLine ("Текуща директория : {0}",
// Directory.GetCurrentDirectory());
// Разпечатва имената, дължината и времената на последния запис
// във файловете от текущата директория
foreach(FileInfo f in dir.GetFiles())
Console.WriteLine("{0,-14}{1,10},{2,20}",f.Name,f.Length,f.LastWriteTime);
}
}
```

```
// Установява директорията C:\Program Files
dir = new DirectoryInfo (@"C:\Program Files");
// Разпечатва имената и времената на последния запис в директория
Console.WriteLine("\n{0,-32}{1}", "Name", "LastWriteTime");
foreach (DirectoryInfo d in dir.GetDirectories ())
Console.WriteLine (" {0,-32}{1}", d.Name, d.LastWriteTime);
dir = new DirectoryInfo ("."); // Установява текущата директория
// Създава нова поддиректория на текущата
dir = new DirectoryInfo ("MyDirectory");
if (false == dir.Exists)
dir.Create ();
// Създава поддиректория на MyDirectory
DirectoryInfo dis = dir.CreateSubdirectory ("MySubDirectory");
// Установява и връща атрибути на поддиректорията
dis.Attributes |= FileAttributes.Hidden | FileAttributes.Archive;
// Разпечатва върнатите атрибути на поддиректорията
Console.WriteLine (" {0,-15}{1,-15}{2}", dis.Name, dis.Parent, dis.Attributes);
// MySubDirectory MyDirectory Hidden, Directory, Archive
dis.Delete (true); // Изтрива поддиректорията
dis.Parent.Delete (true); // Изтрива директорията-родител
}
```

Пример:

```
using System;
using System.IO;
class FileInfoApp
{
    static void Main (string[] args)
    {
        // Създаване и запис в текстов файл
        FileStream fs = File.Create ("My.txt");
        StreamWriter w = new StreamWriter (fs);
        w.WriteLine ("In vino veritas!");
        w.Close ();

        // Чете от текстов файл
        fs = File.Open ("My.txt", FileMode.Open);
        StreamReader r = new StreamReader (fs);
        string t;
        while ((t = r.ReadLine ()) != null)
            Console.WriteLine (t);
        r.Close ();
        fs.Close ();
    }
}
```

```
// Създаване и запис в текстов файл
w = File.CreateText ("My.txt");
w.WriteLine ("In vino veritas!");
w.Close ();

// Чете от текстов файл
r = File.OpenText ("My.txt");
while ((t = r.ReadLine()) != null)
    Console.WriteLine (t);
r.Close ();

// Добавяне на текст към съществуващ текстов файл
w = File.AppendText ("My.txt");
w.WriteLine ("Append text");
w.Close ();
}
```

Клас OpenFileDialog (System.Windows.Forms) – представя диалогов прозорец, който позволява да се отворят файлове.

```
public OpenFileDialog ();
```

Свойства:

InitialDirectory – дава/установява началната директория, която се изобразява във файловия диалогов прозорец.

FileName – дава/установява името на файла, избран в диалоговия прозорец.

Метод:

```
public DialogResult ShowDialog ();
```

Изпълнява диалоговия прозорец, като връща **DialogResult.OK**, ако потребителят е натиснал **OK** в диалоговия прозорец, в противен случай – **DialogResult.Cancel**.

Събитие

FileOk – вдига се, когато потребителят натисне бутон **Open** или **Save** на файловия диалогов прозорец.

```
public event CancelEventHandler FileOk;
```

```
public delegate void CancelEventHandler (object sender, CancelEventArgs e);
```

Манипулаторът на събитието FileOk представя метода, който се извиква автоматично при вдигане на събитието, където **sender** е източник на събитието, **e** съдържа данни за събитието.

Клас Path (System.IO) – извършва операции върху низове, които съдържат информация за пътя до файл или директория.

```
public static string GetDirectoryName (string path);
```

Връща информация за директорията за определен **path**.

```
public static string GetFileName (string path);
```

Връща името и разширението на файла за определен **path**.

Пример:

```
using System;
using System.IO;
using System.Windows.Forms;
class FileDialogApp
{
    private static OpenFileDialog ofd;
    static void Main (string[] args)
    {
        ofd = new OpenFileDialog ();
        // Определя началната директория две нива над текущата .bin\debug
        ofd.InitialDirectory = Path.GetDirectoryName(
            Path.GetDirectoryName(Directory.GetCurrentDirectory()));
        ofd.FileOk += // Добавя манипулатор на събитието FileOk
            new System.ComponentModel.CancelEventHandler (ofd_OK);
        ofd.ShowDialog();
    }
    public static void ofd_OK (object sender, // Код в манипулатора
        System.ComponentModel.CancelEventArgs e)
    {
        StreamReader r = new StreamReader (ofd.FileName);
        while((string s = r.ReadLine()) != null) Console.WriteLine (s);
        r.Close ();
    }
}
```

Класове за четене на Web страници

Клас Uri (System) – представя URI (Uniform Resource Identifier), който осигурява лесен достъп до отделни части на URI.

```
public Uri (string uriString);
```

Клас WebRequest – абстрактен клас, прави запитване за URI.

```
public static WebRequest Create (string requestUriString);
```

Инициализира обекта за определена URI схема.

```
public virtual WebResponse GetResponse ();
```

Връща отговор на запитване в Интернет.

Клас WebResponse – абстрактен клас, осигурява отговор от URI.

```
public virtual Stream GetResponseStream ();
```

Връща поток от данни от източник в Интернет.

Пример:

```
using System;
using System.IO;
using System.Net;
class WebPagesApp
{ [STAThread]
  static void Main (string[] args)
  {
    string s = "http://www.tu-sofia.bg";
    Uri uri = new Uri(s);
    WebRequest req = WebRequest.Create (uri);
    WebResponse resp = req.GetResponse ();
    Stream str = resp.GetResponseStream ();
    StreamReader r = new StreamReader (str);
    string t = r.ReadToEnd();
    int i = t.IndexOf("<<HEAD>");
    int j = t.IndexOf("</HEAD>");
    string u = t.Substring (i, j);
    Console.WriteLine (u);
  }
}
```

Сериализация

Сериализация е механизъм, чрез който потребителските обектите се конвертират в двоичен поток, който може да се запише във файл, за да се съхранят или прехвърлят от едно място на друго.

Сериализираният поток съдържа:

- стойностите на полетата;
- типова информация за потока от данни;
- метаданни за реконструкция на екземпляр.

Сериализацията се прилага за единичен обект и за граф от свързани обекти.

Сериализация на клас:

- атрибут **Serializable** пред дефиницията на класа или
- наследява интерфейса **ISerializable** и атрибут **Serializable** пред дефиницията на класа.

Атрибути Serializable и NonSerialized – членове на класа могат/не могат да бъдат сериализирани.

Използват се методите Serialize и Deserialize на класовете BinaryFormatter и SoapFormatter, които реализират интерфейса IFormatter.

1. Сериализация с BinaryFormatter (System.Runtime.Serialization.Formatters.Binary) – сериализира и десериализира обект или цял граф от свързани обекти в двоичен формат.

```
public BinaryFormatter ();
```

```
public virtual void Serialize (Stream serializationStream, object graph);
```

Сериализира обект или граф от обекти с определен връх (корен) graph в даден поток serializationStream.

```
public virtual object Deserialize (Stream serializationStream);
```

Десериализира даден поток serializationStream в граф от обекти.

Пример: Колекция съдържа информация за стоките в склад: номер, име, налично количество, цена. Сериализира колекцията чрез BinaryFormatter. Разпечатва двоичния файл след десериализация.

```
using System;
using System.IO;
using System.Runtime.Serialization.Formatters.Binary;
using System.Collections;

[Serializable]
class InventoryItem // Клас СТОКА
{ private int key; // номер
  public int Key
  { get { return key; }
    set { key=value; }
  }
  private string name; // име
  [NonSerialized]
  private int units; // количество
  private float price; // цена
}
```



```

public InventoryItem()
{
    try
    {
        Console.WriteLine ("Хомер:\t");
        key = int.Parse (Console.ReadLine ());
        Console.WriteLine ("Име:\t");
        name = Console.ReadLine ();
        Console.WriteLine ("Количество:\t");
        units = int.Parse (Console.ReadLine ());
        Console.WriteLine ("Цена:\t");
        price = float.Parse (Console.ReadLine ());
    }
    catch (Exception e)
    {
        Console.WriteLine ("Данните се игнорират!");
        Console.WriteLine (e.Message);
        throw;
    }
}

public override string ToString ()
{
    return ""+key+"\t"+name+"\t"+units+"\t"+price;
}
}

```

```

class Test
{
    static void Main (string[] args)
    {
        string file = "inventory.bin";
        if (CreateInventory (file))
            Console.WriteLine ("Успешна операция Сериализация!");
        if (ReadInventory (file))
            Console.WriteLine ("Успешна операция Десериализация!");
    }

    public static bool CreateInventory (string fileName)
    {
        Stream w;
        BinaryFormatter bf;
        InventoryItem item;
        ArrayList list;
    }
}

```

```

try
{
    w = File.Create (fileName);
    bf = new BinaryFormatter ();
    list = new ArrayList ();
    try
    {
        Console.WriteLine ("Брой: ");
        int number = int.Parse (Console.ReadLine ());
        for (int i = 1; i <= number; i++)
        {
            try
            {
                item = new InventoryItem ();
                list.Add (item);
            }
            catch (Exception e)
            {
                Console.WriteLine ("Грешка при запис");
                Console.WriteLine (e.Message);
            }
        }
    }
    catch (Exception e)
    {
        Console.WriteLine ("Данните се игнорират.");
        Console.WriteLine (e.Message);
    }
    bf.Serialize (w, list);
}
}

```

```

catch (Exception e)
{
    Console.WriteLine ("Файлът {0} не може да се сериализира.",
        fileName);
    Console.WriteLine (e.Message);
    return false;
}
w.Close ();
Console.WriteLine ("Файлът {0} е сериализиран.", fileName);
return true;
}
}

```

```

public static bool ReadInventory (string fileName)
{
    Stream r = null;
    BinaryFormatter bf;
    ArrayList list;
    try
    {
        r = File.OpenRead (fileName);
        bf = new BinaryFormatter ();
        list = new ArrayList ();
        try
        {
            list = (ArrayList)bf.Deserialize (r);
            foreach (InventoryItem item in list)
                Console.WriteLine (item);
        }
        catch (Exception e)
        {
            Console.WriteLine ("Грешка при десериализация.");
            Console.WriteLine (e.Message);
            r.Close ();
            return false;
        }
    }
}
}

```

```

catch (Exception e)
{
    Console.WriteLine ("Файлът {0} не може да се отвори.", fileName);
    Console.WriteLine (e.Message);
    r.Close ();
    return false;
}
r.Close ();
Console.WriteLine ("Файлът {0} е десериализиран.", fileName);
return true;
}
}

```

2. Сериализация със SoapFormatter (System.Runtime.Serialization.Formatters.Soap) – сериализира и десериализира обект или цял граф от свързани обекти във формат Soap. Полученият файл е форматиран във формат XML.

- public SoapFormatter ();
- **Добавя се** System.Runtime.Serialization.Formatter.Soap.dll
Project => **Add Reference ...** => **избира се System.Runtime.Serialization.Formatter.Soap.dll**
=> **Select** => **OK**
- **Добавя се**
using System.Runtime.Serialization.Formatters.Soap;
- **Замества се BinaryFormatter със SoapFormatter.**
- **Разширението на файла трябва да бъде .xml.**

```
using System;
using System.IO;
using System.Runtime.Serialization.Formatters.Soap;
using System.Collections;
[Serializable]
class InventoryItem
{
    private int key;
    public int Key
    {
        get { return key; }
        set { key=value; }
    }
    private string name;
    [NonSerialized]
    private int units;
    private float price;
}
```

```
public InventoryItem ()
{
    try
    {
        Console.WriteLine ("Homep:\t");
        key = int.Parse (Console.ReadLine ());
        Console.WriteLine ("Име:\t");
        name = Console.ReadLine ();
        Console.WriteLine ("Количество:\t");
        units = int.Parse(Console.ReadLine ());
        Console.WriteLine ("Цена:\t");
        price = float.Parse (Console.ReadLine ());
    }
    catch (Exception e)
    {
        Console.WriteLine ("Данните се игнорират!");
        Console.WriteLine (e.Message);
        throw;
    }
}
public override string ToString ()
{
    return ""+key+"\t"+name+"\t"+units+"\t"+price;
}
}
```

```
class Test
{
    static void Main (string[] args)
    {
        string file = "inventory.xml";
        if (CreateInventory (file))
            Console.WriteLine ("Успешна сериализация!");
        if (ReadInventory (file))
            Console.WriteLine ("Успешна десериализация!");
    }
    public static bool CreateInventory (string fileName)
    {
        Stream w;
        SoapFormatter bf;
        InventoryItem item;
        ArrayList list;
    }
}
```

```
try
{
    w = File.Create (fileName);
    bf = new SoapFormatter ();
    list = new ArrayList ();
    try
    {
        Console.WriteLine ("Брой: ");
        int number = int.Parse (Console.ReadLine ());
        for (int i = 1; i <= number; i++)
        {
            try
            {
                item = new InventoryItem ();
                list.Add (item);
            }
            catch (Exception e)
            {
                Console.WriteLine ("Грешка при запис в списъка.");
                Console.WriteLine (e.Message);
            }
        }
    }
    catch (Exception e)
    {
        Console.WriteLine ("Данните се игнорират.");
        Console.WriteLine (e.Message);
    }
    bf.Serialize (w, list);
}
}
```

```
catch (Exception e)
{
    Console.WriteLine ("Файлът {0} не може да се създаде.", fileName);
    Console.WriteLine (e.Message);
    return false;
}
w.Close ();
Console.WriteLine ("Файлът {0} е сериализиран.", fileName);
return true;
}
public static bool ReadInventory (string fileName)
{
    Stream r = null;
    SoapFormatter bf;
    ArrayList list;
}
```

```

try
{
    r = File.OpenRead(fileName);
    bf = new SoapFormatter ();
    list = new ArrayList ();
    try
    {
        list = (ArrayList)bf.Deserialize (r);
        foreach (InventoryItem item in list)
            Console.WriteLine (item);
    }
    catch (Exception e)
    {
        Console.WriteLine ("Грешка при четене.");
        Console.WriteLine (e.Message);
        r.Close ();
        return false;
    }
}
catch (Exception e)
{
    Console.WriteLine ("Файлът {0} не може да се отвори.", fileName);
    Console.WriteLine(e.Message);
    r.Close ();
    return false;
}
}

```

```

r.Close ();
Console.WriteLine ("Файлът {0} е десериализиран.", fileName);
return true;
}
}

```

3. Сериализация с XmlSerializer (System.Xml.Serialization) – сериализира и десериализира обекти в и от XML документи, като контролира кодирането на обектите в XML.

Полученият файл е форматиран във формат XML без допълнителните специфични характеристики на Soap.

XmlSerializer не използва атрибута Serializable, вместо атрибута NonSerialized използва XmlIgnore.

```

public XmlSerializer (Type type);
public void Serialize (Stream stream, object o);
public object Deserialize (Stream stream);

```

```

using System;
using System.IO;
using System.Xml.Serialization;
public class InventoryItem
{
    public int key;
    public string name;
    [XmlIgnore]
    public int units;
    public float price;
    public InventoryItem ()
    {
    }
}

```

```

public InventoryItem (bool f)
{
    if (f)
    {
        try
        {
            Console.Write ("Хомер:\t");
            key = int.Parse (Console.ReadLine ());
            Console.Write ("Име:\t");
            name = Console.ReadLine ();
            Console.Write ("Количество:\t");
            units = int.Parse (Console.ReadLine ());
            Console.Write ("Цена:\t");
            price = float.Parse (Console.ReadLine ());
        }
        catch (Exception e)
        {
            Console.WriteLine ("Данните се игнорират!");
            Console.WriteLine (e.Message);
            throw;
        }
    }
}
}

```

```

public override string ToString ()
{
    return ""+key+"\t"+name+"\t"+units+"\t"+price;
}
}
public class Inventory
{
    public InventoryItem[] inventory;
    public Inventory() {}
    public Inventory (int number)
    {
        inventory = new InventoryItem[number];
        for (int i = 0; i < number; i++)
        {
            try
            {
                inventory[i] = new InventoryItem (true);
            }
            catch (Exception e)
            {
                Console.WriteLine ("Грешка при конструиране на обект.");
                Console.WriteLine (e.Message);
            }
        }
    }
}
}

```

```
public override string ToString ()
{
    string result = "";
    foreach (InventoryItem item in inventory)
        result += item.ToString () + "\n";
    return result;
}
}
```

```
class Test
{
    static void Main(string[] args)
    {
        string file = "inventory.xml";
        if (CreateInventory (file))
            Console.WriteLine("Успешна операция Сериализация!");
        if (ReadInventory (file))
            Console.WriteLine ("Успешна операция Десериализация!");
    }
    public static bool CreateInventory (string fileName)
    {
        Stream w;
        XmlSerializer bf;
        Inventory i = null;
    }
}
```

```
try
{
    w = File.Create (fileName);
    bf = new XmlSerializer (typeof (Inventory));
    try
    { Console.WriteLine ("Брой: ");
      int number = int.Parse (Console.ReadLine ());
      i = new Inventory (number);
    }
    catch (Exception e)
    { Console.WriteLine ("Данните се игнорират.");
      Console.WriteLine (e.Message);
    }
    bf.Serialize (w, i);
}
catch (Exception e)
{
    Console.WriteLine("Файлът {0} не може да се сериализира.",
        fileName);
    Console.WriteLine (e.Message);
    return false;
}
}
```

```
w.Close ();
Console.WriteLine ("Файлът {0} е сериализиран.", fileName);
return true;
}
public static bool ReadInventory (string fileName)
{
    Stream r = null;
    XmlSerializer bf;
    try
    {
        r = File.OpenRead (fileName);
        bf = new XmlSerializer (typeof (Inventory));
        try
        {
            Inventory i = (Inventory)bf.Deserialize (r);
            Console.WriteLine (i);
        }
    }
}
```

```
catch (Exception e)
{
    Console.WriteLine ("Грешка при десериализация.");
    Console.WriteLine (e.Message);
    r.Close ();
    return false;
}
}
catch (Exception e)
{
    Console.WriteLine ("Файлът {0} не може да се десериализира",
        fileName);
    Console.WriteLine (e.Message);
    r.Close ();
    return false;
}
r.Close ();
Console.WriteLine ("Файлът {0} е десериализиран.", fileName);
return true;
}
}
```