Streams

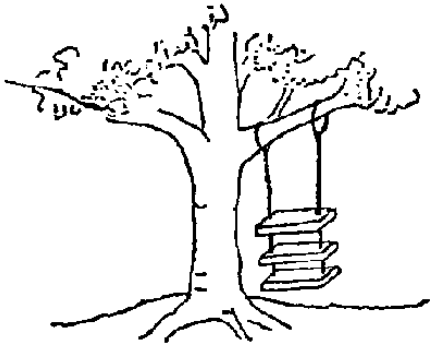
Streams

FileStream

StreamReader –Writer

BinaryReader –Writer

BinaryFormatter

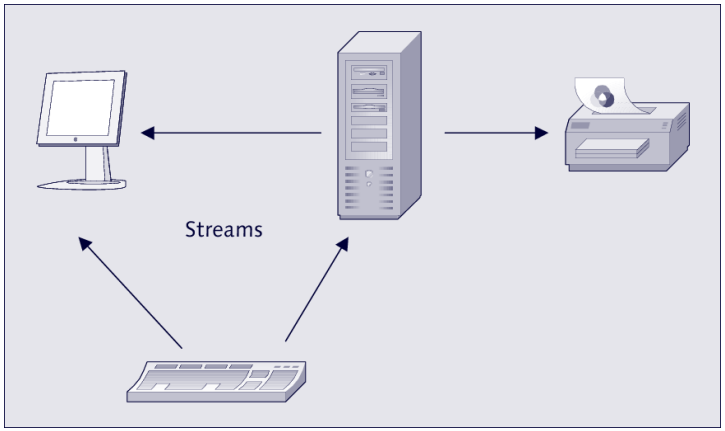


Dateien & Streams

- Daten von beliebiger Datenquelle holen
- Daten an ein beliebiges Ziel schicken

Mögliche Quelle oder Ziel eines Datenstroms:

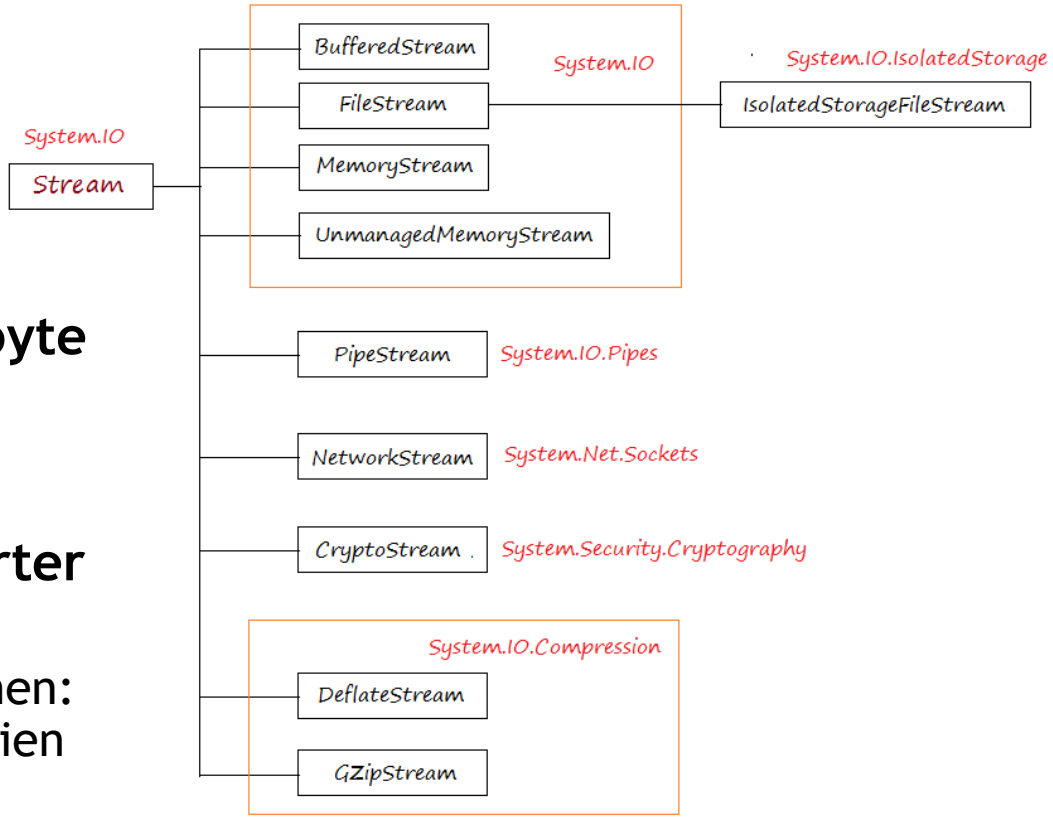
- Dateien
- Benutzeroberfläche
- Netzwerkverbindungen
- Speicherblöcke
- Drucker
- andere Peripheriegeräte



Stream

- ist eine Klasse zur Beschreibung einer Zeile von aufeinanderfolgender **byte**

- **byteweises Lesen und Schreiben unformatierter Daten**
  - allgemeine Operationen: zB Kopieren von Dateien
  - weniger gut für textuelle Ein- und Ausgabeoperationen geeignet

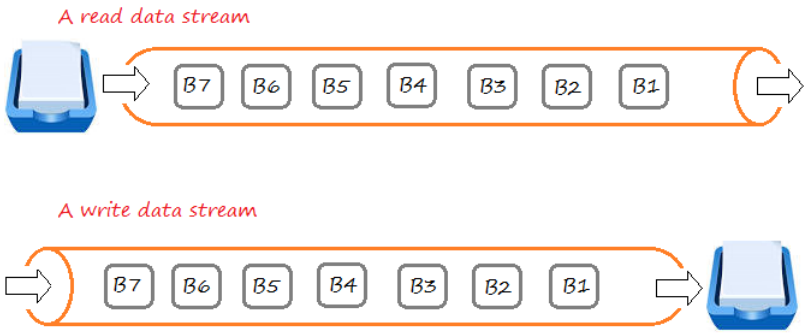


Streams (Datenfluss)

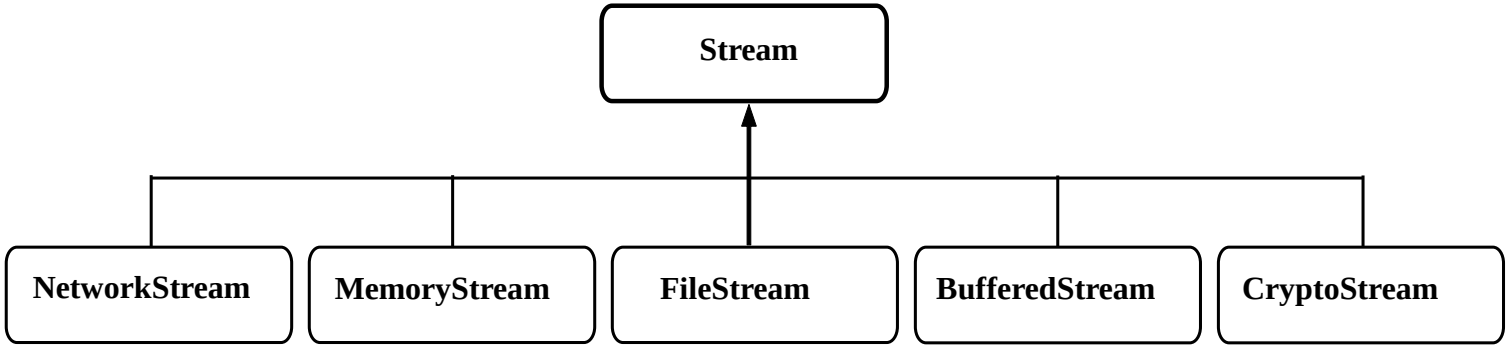
- Stream hat Anfangs- und Endpunkt:
  - eine Quelle an der der Datenstrom entspringt
  - ein Ziel das den Datenstrom empfängt
- Methoden Console.WriteLine und Console.ReadLine arbeiten mit Datenströmen
- **Stream**
  - ist nicht dauerhaft:
    - wird geöffnet und liest oder schreibt Daten
    - nach dem Schließen sind die Daten verloren
      außer sie werden dauerhaft in eine Datei gespeichert

Namespaces der Ein- bzw. Ausgabe

- namespace **System.IO**
  - beinhaltet die elementarsten Klassen für die Ein- und Ausgabe
- Fehlerfall: **IOException**
  - Ausnahmen im Zusammenhang mit E/A-Operationen werden auf eine gemeinsame Basis (**IOException**) zurückgeführt



Stream ist eine Basisklasse



Die Klasse	Die Bezeichnung
BufferedStream	Eine Utility-Klasse, die einen anderen Stream zur Leistungsverbesserung einpacken (wrap)
FileStream	**Zum Lesen/Aufschreiben der Daten in die File benutzen**
MemoryStream	Stream zum Umgang mit Daten in der Speicherung
NetworkStream	Stream sendet die Daten basierend auf Sockets
CryptoStream	Stream zum Lesen/Aufschreiben der kodierten Daten
…	

Properties der Klasse Stream

Name	Beschreibung
CanRead	Ruft beim Überschreiben in einer abgeleiteten Klasse einen Wert ab, der angibt, ob der aktuelle Stream Lesevorgänge unterstützt.
CanSeek	Ruft beim Überschreiben in einer abgeleiteten Klasse einen Wert ab, der angibt, ob der aktuelle Stream Suchvorgänge unterstützt.
CanWrite	Ruft beim Überschreiben in einer abgeleiteten Klasse einen Wert ab, der angibt, ob der aktuelle Stream Schreibvorgänge unterstützt.
Length	**Ruft beim Überschreiben in einer abgeleiteten Klasse die Länge des Streams in Bytes ab.**
Position	Ruft beim Überschreiben in einer abgeleiteten Klasse die Position im aktuellen Stream ab oder legt diese fest.

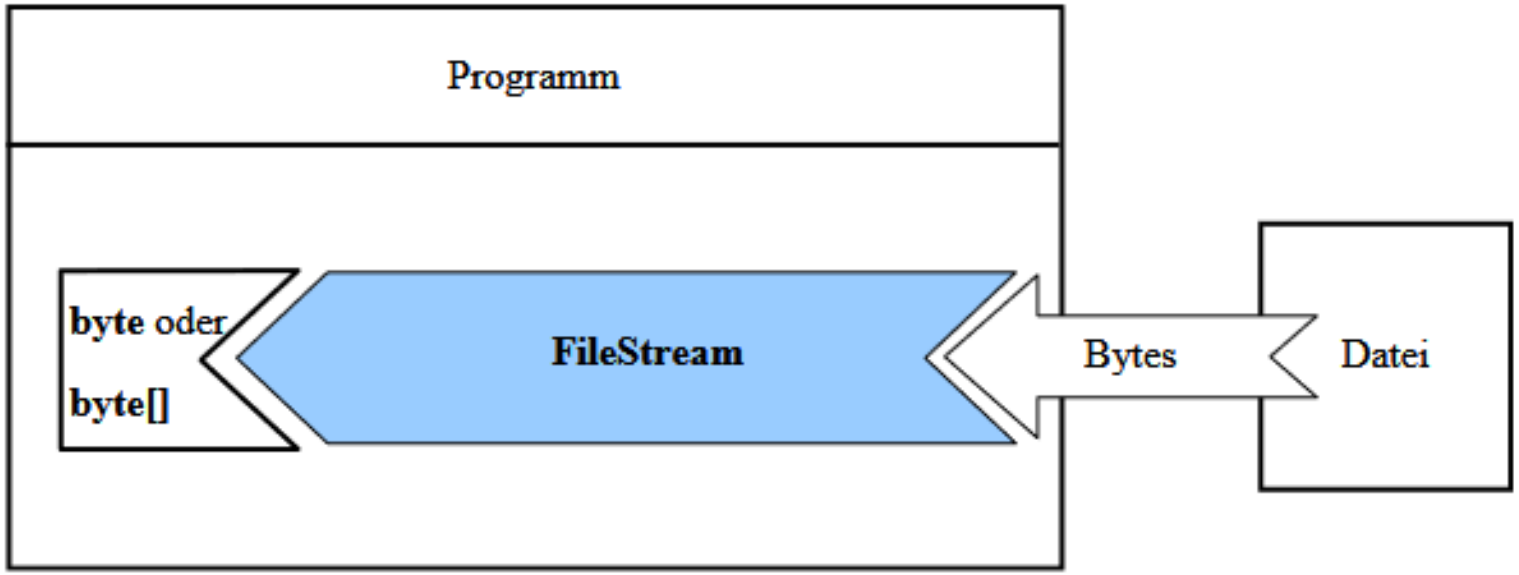
Methoden der Klasse Stream

Methode	Beschreibung
Close	**Schließt den aktuellen Stream und gibt alle dem aktuellen Stream zugeordneten Ressourcen frei.**
Read	**Liest eine Folge von Bytes aus dem aktuellen Stream und setzt den Datenzeiger im Stream um die Anzahl der gelesenen Bytes weiter.**
ReadByte	Liest ein Byte aus dem Stream und erhöht die Position im Stream um ein Byte. Der Rückgabewert ist –1, wenn das Ende des Streams erreicht ist.
Seek	**Legt die Position im aktuellen Stream fest.**
Write	**Schreibt eine Folge von Bytes in den aktuellen Stream und erhöht den Datenzeiger im Stream um die Anzahl der geschriebenen Bytes.**
WriteByte	Schreibt ein Byte an die aktuelle Position im Stream und setzt den Datenzeiger um eine Position im Stream weiter.

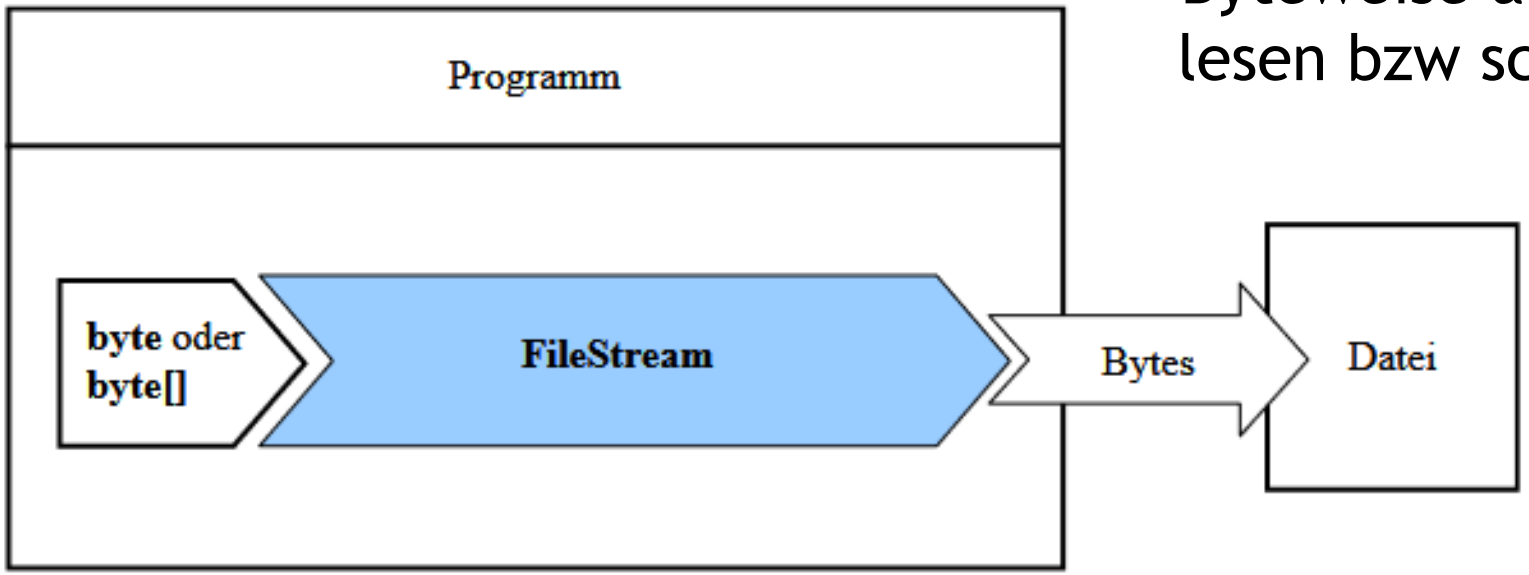
```
...
fs.Write(arr, 0, arr.Length);
...
fs.Seek(0, SeekOrigin.Begin);
fs.Read(arrRead, 0, 10);
...
```

FileStream

Universellste Klasse zum Lesen und Schreiben in Dateien



Byteweise aus Datei lesen bzw schreiben



Datei öffnen

- Öffnen einer Datei
  - erledigt das Betriebssystem

```
public static FileStream Open(
    string path, FileMode mode,
    FileAccess access, FileShare share);
```

  - Parameter path
    - wird Pfadangabe als Zeichenfolge mitgeteilt
    - besteht aus dem Pfad und dem Dateinamen
  - mode-Parameter
    - vom Typ FileMode steuert das Verhalten

Beispiel: Öffnen einer Datei

- Datei öffnet mit File.Open

```
FileStream stream = File.Open(@"C:\MyTestfile.txt",
FileMode.OpenOrCreate, FileAccess.ReadWrite,
FileShare.None);
```

  - Öffnet Datei MyTestfile.txt im Stammverzeichnis C:\
    - falls es diese gibt
    - wenn nicht, wird sie neu erzeugt
  - Lesen oder Ändern der Datei möglich
  - weitere Zugriffe auf die Datei sind unterbunden

Konstanten der Enumeration »FileMode«

Modus	Beschreibung
Append	Die Datei wird geöffnet oder neu erzeugt.
Create	Ist die Datei noch nicht vorhanden, wird sie angelegt, anderenfalls wird sie überschrieben.
CreateNew	Eine neue Datei wird angelegt
Open	**Eine vorhandene Datei wird geöffnet**
OpenOrCreate	Es wird eine neue Datei erzeugt oder eine vorhandene geöffnet, jedoch im Unterschied zu Create nicht automatisch überschrieben
Truncate	Es wird eine vorhandene Datei geöffnet und entleert

oder eine IOException geworfen im Fehlerfall

Konstanten der Enumeration »FileAccess«

- mode-Parameter
  - beschreibt Verhalten des Betriebssystems beim Öffnen einer Datei
- FileAccess
  - Bestimmt Schreib- oder Lesezugriff

FileAccess-Konstante	Beschreibung
Read	Datei wird für den Lesezugriff geöffnet.
Write	Datei wird für den Schreibzugriff geöffnet.
ReadWrite	Datei wird für den Lese- und Schreibzugriff geöffnet.

Konstanten der Enumeration »FileShare«

- share-Parameter
  - beschreibt das Verhalten der Datei, nach dem Öffnen
  - welche weiteren Zugriffe dürfen auf die Datei erfolgen

FileShare-Konstante	Beschreibung
None	Alle weiteren Versuche, diese Datei zu öffnen, werden konsequent abgelehnt.
Read	Diese Datei darf von anderen Anwendungen oder Threads nur zum Lesen geöffnet werden.
Write	Diese Datei darf von anderen Anwendungen oder Threads nur zum Editieren geöffnet werden.
ReadWrite	Diese Datei darf von anderen Anwendungen oder Threads sowohl zum Lesen als auch zum Editieren geöffnet werden.

FileStream - Write

Schreiben ein eine Datei mit:

```
public void Write(byte[] array, int offset, int count);
fs.Write(arr, 0, arr.Length);
```

Parameter	Beschreibung
array	Byte-Array, in das die übergebenen Daten gelesen werden
offset	Indexposition im Array, an der die Leseoperation beginnen soll
count	Anzahl der zu schreibenden Bytes

Filestream - Seek

Positionszeiger an beliebiger Stelle im Stream setzen

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

origin = SeekOrigin.Begin

Member	Beschreibung
Begin	Gibt den Anfang eines Streams an.
Current	Gibt die aktuelle Position innerhalb eines Streams an.
End	Gibt das Ende eines Streams an.

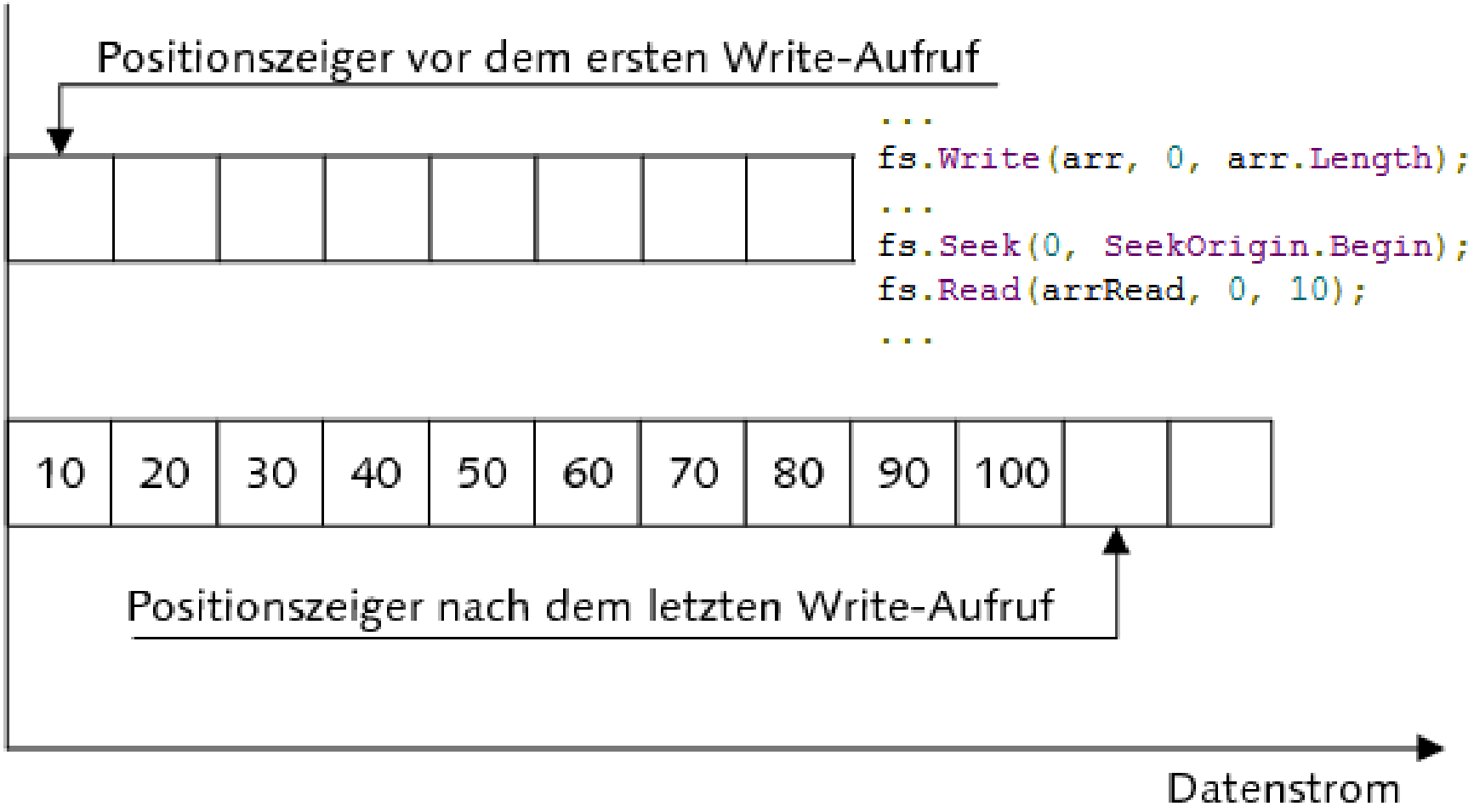
FileStream - Read

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

```
byte[] arrRead = new byte[10];
fs.Read(arrRead, 0, 10);
for (int I = 0; i < arr.Length; i++)
   Console.WriteLine(arrRead[i]);
fs.Close();
```

Parameter	Beschreibung
array	Ein Byte-Array, in das die übergebenen Daten geschrieben werden
offset	Die Indexposition im Array, an der die Leseoperation beginnen soll
count	Die Anzahl der zu lesenden Bytes

Arbeiten mit Positionszeiger



Lesen & Schreiben mit FileStream

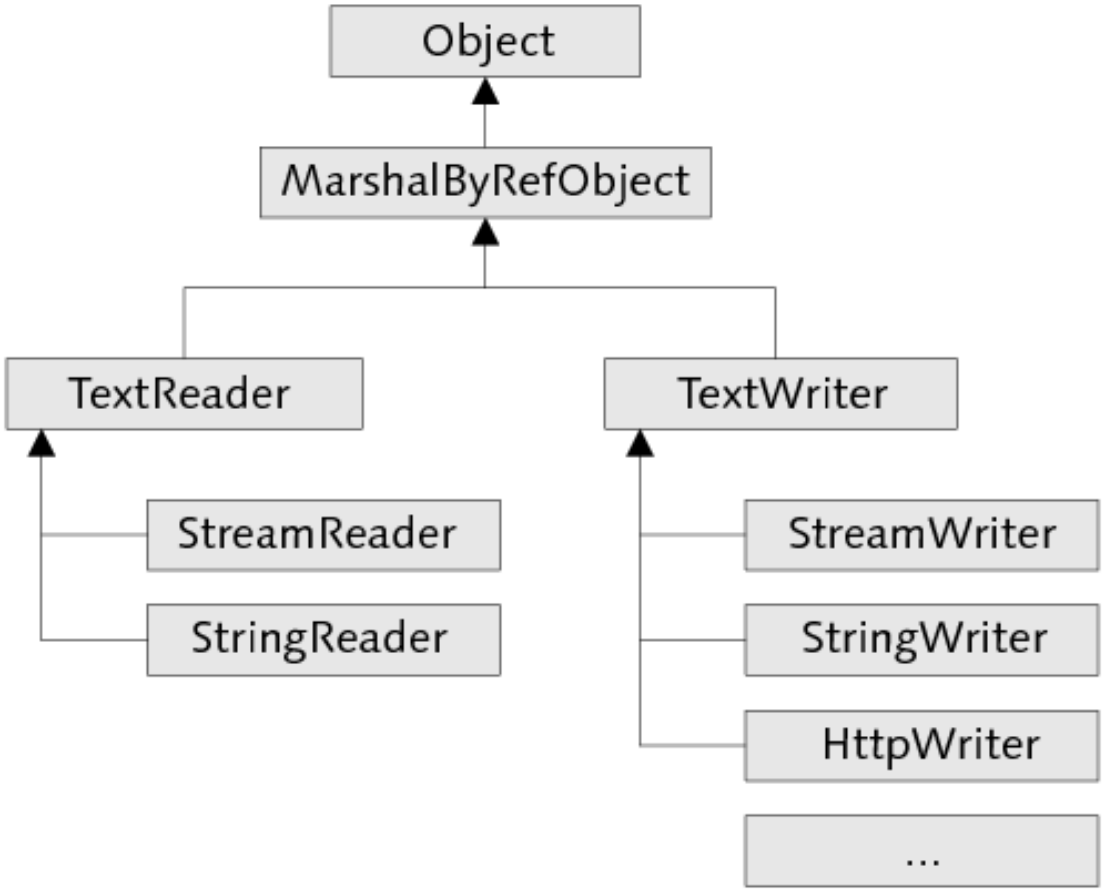
```
class Program {
  static void Main(string[] args) {
    byte[] arr = {10, 20, 30, 40, 50, 60, 70, 80, 90, 100};
    string path = "D:\\Testfile.txt";
    // FileStream öffnen
    FileStream fs = 
    // in den Stream schreiben
    fs.
    byte[] arrRead = new byte[10];
    // Positionszeiger auf den Anfang des Streams setzen
    fs.
    // Stream lesen
    fs.
    for (int i = 0; i < arr.Length; i++)
      Console.WriteLine(arrRead[i]);
    Console.ReadLine();
    // FileStream schließen
    fs.Close();
  }
}
```

Methoden der abstrakten Klasse »Stream«

Methode	Beschreibung
Close	**Schließt den aktuellen Stream und gibt alle dem aktuellen Stream zugeordneten Ressourcen frei.**
Read	**Liest eine Folge von Bytes aus dem aktuellen Stream und setzt den Datenzeiger im Stream um die Anzahl der gelesenen Bytes weiter.**
ReadByte	Liest ein Byte aus dem Stream und erhöht die Position im Stream um ein Byte. Der Rückgabewert ist –1, wenn das Ende des Streams erreicht ist.
Seek	**Legt die Position im aktuellen Stream fest.**
Write	**Schreibt eine Folge von Bytes in den aktuellen Stream und erhöht den Datenzeiger im Stream um die Anzahl der geschriebenen Bytes.**
WriteByte	Schreibt ein Byte an die aktuelle Position im Stream und setzt den Datenzeiger um eine Position im Stream weiter.

Stream Writer

Schreiben
von Text
in eine Datei



StreamWriter instantiieren:

erster Parameter string -> Pfad zur Datei

erster Parameter Stream -> anderer Stream

- Konstruktoren der Klasse StreamWriter:
  - **public StreamWriter(Stream);**
  - **public StreamWriter(string);**
  - public StreamWriter(Stream, Encoding);
  - public StreamWriter(string, bool);
  - public StreamWriter(Stream, Encoding, int);
  - public StreamWriter(string, bool, Encoding);
  - public StreamWriter(string, bool, Encoding, int);

StreamWriter:

```
public StreamWriter(string, bool);
public StreamWriter(string, bool, Encoding, int);
```

- string: Pfadangabe zu der Datei
- bool: True / False
  - True - zu schreibenden Daten - an das Ende der Datei gehängt vorausgesetzt, es existiert bereits eine Datei gleichen Namens in dem Verzeichnis
  - False - eine existierende Datei überschrieben
- Encoding kann explizit angegeben werden
  - Standardmäßig wird als Encoding UTF-8 verwendet
- int: (letzter Parameter)
  - Größe des Puffers beeinflussen / setzen

Konstanten der Klasse Encoding

```
StreamWriter sw = new StreamWriter(@"C:path.txt",
                                   false, Encoding.Unicode);
Console.WriteLine("Format: {0}", sw.Encoding);
```

Name	Beschreibung
ASCII	Ruft eine Codierung für den ASCII-Zeichensatz (7-Bit) ab.
Default	Ruft eine Codierung für die aktuelle ANSI-Codepage des Betriebssystems ab.
Unicode	Ruft eine Codierung für das UTF-16-Format ab.
UTF32	Ruft eine Codierung für das UTF-32-Format ab.
UTF7	Ruft eine Codierung für das UTF-7-Format ab.
UTF8	Ruft eine Codierung für das UTF-8-Format ab.

https://msdn.microsoft.com/de-de/library/system.text.encoding(v=vs.110).aspx

Schreiben in den Datenstrom

- Mit Write() oder WriteLine()

```
StreamWriter sw = new StreamWriter(@"D:\NewFile.txt");
sw.Write("Visual C#");
sw.WriteLine("macht Spaß!");
sw.Close();
```

Close - StreamWriter & Streams

- **StreamWriter** müssen aufgrund ihres lokalen Puffers unbedingt geschlossen werden
  - das durch einen **Close()** - Aufruf
  - oder einen äquivalenten **Dispose()** – Aufruf
  - sichergestellt wird
  - mit **using**-Block findet dies automatisiert statt

Beispiel

```
const string sDateiname = "test.txt";
StreamWriter oWrite = null;
StreamReader oRead = null;

try
{
    // Daten in Datei schreiben
    oWrite = new StreamWriter(sDateiname, false);
    oWrite.WriteLine("Hallo-Welt!");
    oWrite.WriteLine();
    oWrite.Write("C#-Buch V2.0 5.5");
    oWrite.Close();
}
catch (Exception ex)
{
    Console.WriteLine(ex.ToString());
    Console.ReadKey();
    return;
}
finally
{
    // Daten schreiben und Stream schließen
    if (oWrite != null)
        oWrite.Close();
}
```

Das Close vom SW und SR kann im finally Block positioniert werden.

```
try
{
    // Datei auslesen und auf der Konsole ausgeben
    oRead = new StreamReader(sDateiname);
    while (oRead.Peek() != -1)
        Console.WriteLine(oRead.ReadLine());
}
catch (Exception ex)
{
    Console.WriteLine(ex.ToString());
}
finally
{
    // Stream schließen
    if (oRead != null)
        oRead.Close();
}

Console.ReadKey();
```

Klasse <<StreamWriter>>

Eigenschaften	Beschreibung
AutoFlush	Löscht immer den Puffer nach jedem Aufruf von Write oder WriteLine **-> Textdatei ist damit immer aktuell (Puffer geleert)**
BaseStream	Liefert eine Referenz auf den Base-Stream zurück
Encoding	Liefert das aktuelle Encoding-Schema zurück

Methode	Beschreibung
Close	Schließt das aktuelle Objekt sowie alle eingebetteten Streams
Flush	Schreibt die gepufferten Daten in den Stream und löscht danach den Inhalt des Puffers
Write	Schreibt in den Stream, ohne einen Zeilenumbruch anzuhängen
WriteLine	Schreibt in den Stream und schließt mit einem Zeilenumbruch ab

Klasse StreamReader

StreamReader zum Lesen aus Datenströmen nutzen

Gegenstück zum StreamWriter

StreamReader - Beispiel

Wie lautet die Ausgabe?

Visual C#
macht Spaß.
Richtig??

```
class Program {
  static void Main(string[] args) {
    // Datei erzeugen und mit Text füllen
    StreamWriter sw = new StreamWriter(@"D:\MyTest.kkl");
    sw.WriteLine("Visual C#");
    sw.WriteLine("macht Spaß.");
    sw.Write("Richtig??");
    sw.Close();
    // Datei an der Konsole einlesen
    StreamReader sr = new StreamReader(@"D:\MyTest.kkl");
    while(sr.Peek() != -1)
      Console.WriteLine(sr.ReadLine());
    sr.Close();
    Console.ReadLine();
  }
}
```

```
Console.WriteLine(sr.ReadToEnd());
```

StreamReader - Methoden

Peek

- Liest ein Zeichen aus dem Strom
  - liefert den int-Wert zurück,
    der das Zeichen repräsentiert,
  - verarbeitet das Zeichen aber nicht

Read / ReadLine / ReadToEnd

  - liefert den int-Wert zurück,
    der das Zeichen repräsentiert. (End of File -1)
  - liest eine Zeile aus dem Datenstrom
  - liest bis zum Ende des Datenstroms

Read mit int als Return-Typ

- Read liefert den ASCII-Code als int retour
- Ist kein Zeichen mehr verfügbar, ist der Rückgabewert -1

```
using (StreamWriter sw = new StreamWriter(path))
{
    sw.WriteLine("This");
    sw.WriteLine("is some text");
    sw.WriteLine("to test");
    sw.WriteLine("Reading");
}

using (StreamReader sr = new StreamReader(path))
{
    while (sr.Peek() >= 0)
    {
        Console.Write((char)sr.Read());  //int (ASCII) -> cast to char
    }
}
```

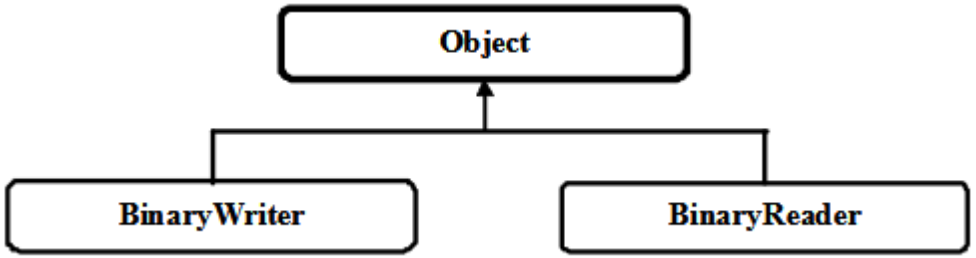
Using-Block

für StreamReader & -Writer

Using-Block

Using sorgt für das Schließen der Streams automatisch

```
static void Main(string[] args)
{
    // Get the directories currently on the C drive.
    DirectoryInfo[] cDirs = new DirectoryInfo(@"c:\").GetDirectories();

    // Write each directory name to a file.
    using (StreamWriter sw = new StreamWriter("CDriveDirs.txt"))
    {
        foreach (DirectoryInfo dir in cDirs)
        {
            sw.WriteLine(dir.Name);
        }
    }

    // Read and show each line from the file.
    string line = "";
    using (StreamReader sr = new StreamReader("CDriveDirs.txt"))
    {
        while ((line = sr.ReadLine()) != null)
        {
            Console.WriteLine(line);
        }
    }
}
```



BinaryReader & BinaryWriter

Informationsgehalt binärer Dateien kann nur dann korrekt ausgewertet werden, wenn der Typ, den die Daten repräsentieren, bekannt ist.

BinaryReader & BinaryWriter

- stammen nicht von der Klasse Stream ab
  - verwenden aber für die Verbindung mit einer Datenquelle ein Stream-Objekt
  - dieses Stream-Objekt wird im Konstruktor angegeben

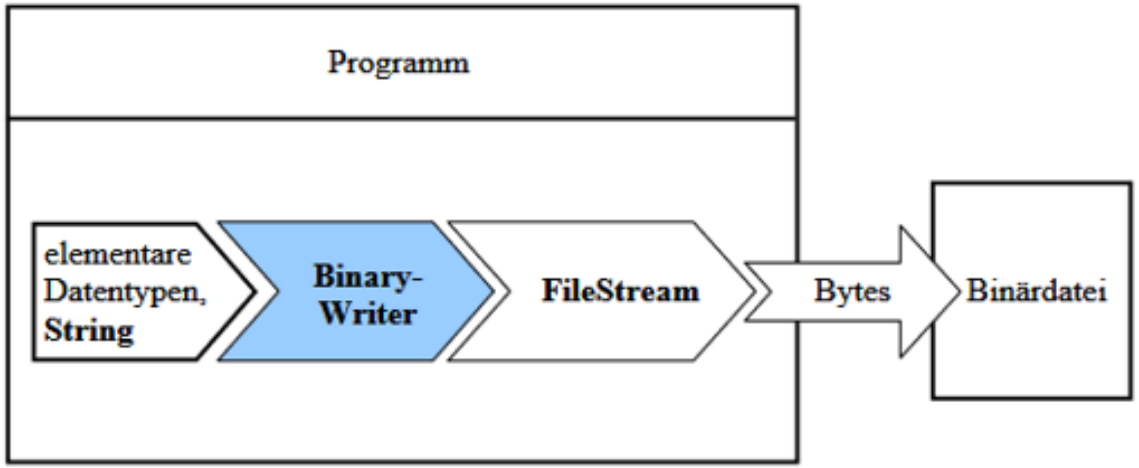
  - public BinaryWriter(Stream ausgabestrom);
  - public BinaryReader(Stream eingabestrom);

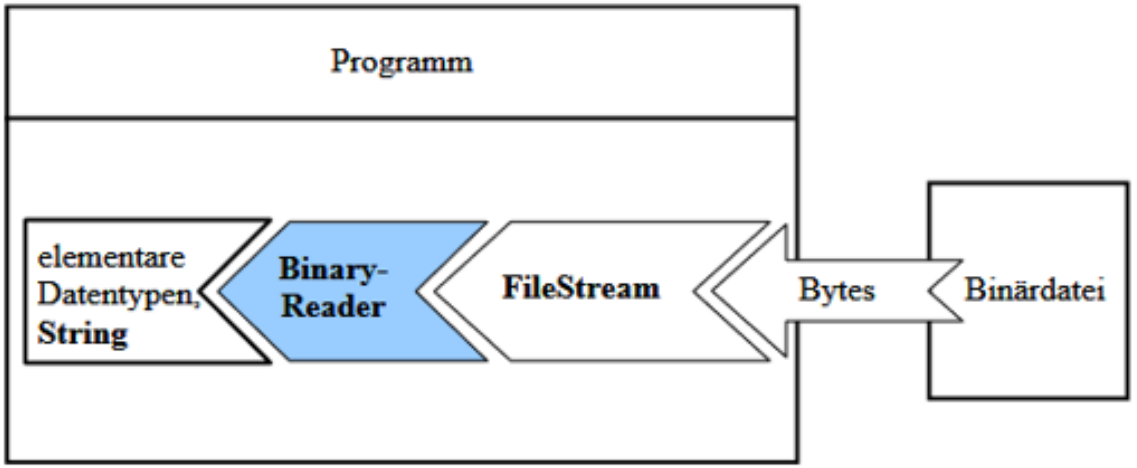
BinärReader & BinärWriter

Benutzen auch einen FileStream

Schreiben Daten als Binärdaten in Dateien

```
NUL NUL ?BELc:\Temp
NUL NUL NUL SOH
```



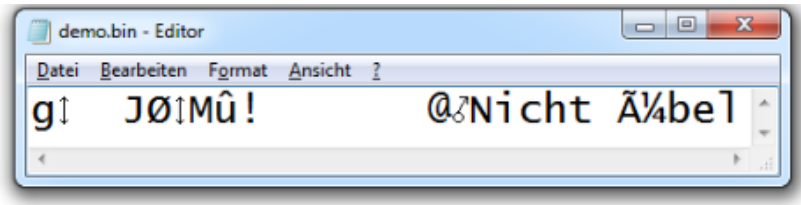


BinaryWriter

Die Ausgabe des Programms:

```
4711
3,1415926
Nicht übel
```

```
using System;
using System.IO;
class BinWrtRd {
   static void Main() {
      String name = "demo.bin";
      FileStream fso = new FileStream(name, FileMode.Create);
      BinaryWriter bw = new BinaryWriter(fso);
      bw.Write(4711);
      bw.Write(3.1415926);
      bw.Write("Nicht übel");
      bw.Close();

      FileStream fsi = new FileStream(name, FileMode.Open, FileAccess.Read);
      BinaryReader br = new BinaryReader(fsi);
      Console.WriteLine(br.ReadInt32() + "\n" +
                        br.ReadDouble() + "\n" +
                        br.ReadString());
      br.Close();
   }
}
```



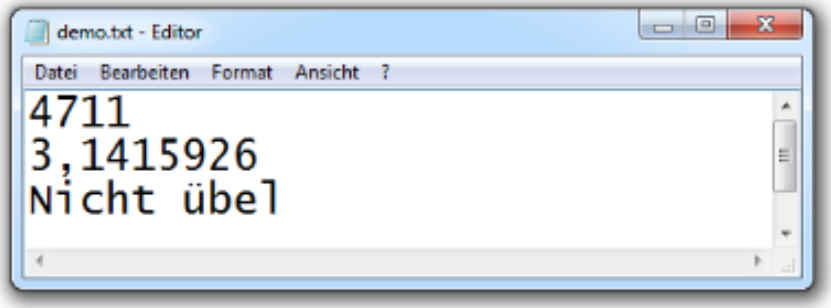
Teste StreamWriter & StreamReader

für selbigen Quellcode:

```
sw.WriteLine(4711);
sw.WriteLine(3.1415926);
sw.WriteLine("Nicht übel");
```

StreamWriter & StreamReader

```
using System;
using System.IO;

class StreamWrtRd {
   static void Main() {
      String name = "demo.txt";
      StreamWriter sw = new StreamWriter(name);
      sw.WriteLine(4711);
      sw.WriteLine(3.1415926);
      sw.WriteLine("Nicht übel");
      sw.Close();

      StreamReader sr = new StreamReader(
                              new FileStream(name, FileMode.Open, FileAccess.Read));
      Console.WriteLine("Inhalt der Datei {0}:\n",
                        ((FileStream)sr.BaseStream).Name);
      for (int i = 0; sr.Peek() >= 0; i++ ) {
         Console.WriteLine("{0}:\t{1}", i, sr.ReadLine());
      }
      sr.Close();
   }
}
```




```
1:      4711
2:      3,1415926
3:      Nicht übel
```

BinaryFormatter

Objekt unter .NET serialisierbar machen

http://openbook.rheinwerk-verlag.de/visual_csharp_2010/visual_csharp_2010_13_002.htm

Serialisierung mit BinaryFormatter

- Klasse mit dem Attribut Serializable markieren
  - `[Serializable()]`
  - `public class Person {...}`
- Fehlt das Attribut, wird die Ausnahme **SerializationException** ausgelöst

MyObjekt Serialisieren

- Objekt mit mehreren Attributen serialisieren

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

```
public static void WriteBinary()
{
    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();
}
```

MyObject deserialisieren (einlesen)

- Object mit mehreren Attributen einlesen und retour konvertieren.

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

    // Here's the proof
    Console.WriteLine("n1: {0}", obj.n1);
    Console.WriteLine("n2: {0}", obj.n2);
    Console.WriteLine("str: {0}", obj.str);
}
```

Examples

StreamReader & Writer – Ten Numbers

FileStream - Simultanes Lesen & Schreiben

SR & SW with FileStream - Using FileStream

Using-Block with SR & SW - 1x1 Tabelle

BinaryReader & -Writer

BinaryFormatter

Example - Ten Numbers

Schreibe 1..10 in eine Datei.

Öffne die Datei und gib den Inhalt in der Console aus.

Example - Simultaneously Read and Write

Read and Write operations are not possible on the same FileStream object simultaneously. If you are already reading from a file, create a separate FileStream object to write to the same file, as shown:

Understand StreamWriter using FileStream

```
//Create object of FileInfo for specified path
FileInfo fi = new FileInfo(@"D:\DummyFile.txt");

//Open file for Read\Write
FileStream fs = fi.Open(FileMode.OpenOrCreate,
    FileAccess.Write, FileShare.Read);

//Create StreamWriter object to write string to FileSream
StreamWriter sw = new StreamWriter(fs);
sw.WriteLine("Another line from streamwriter");
sw.Close();
```

Understand StreamReader using FileStream

```
//Create object of FileInfo for specified path
FileInfo fi = new FileInfo(@"D:\DummyFile.txt");

//Open file for Read\Write
FileStream fs = fi.Open(FileMode.OpenOrCreate,
    FileAccess.Read, FileShare.Read);

//Create object of StreamReader by passing
//FileStream object on which it needs to operates on
StreamReader sr = new StreamReader(fs);

//Use ReadToEnd method to read all the content from file
string fileContent = sr.ReadToEnd();

//Close StreamReader object after operation
sr.Close();
fs.Close();
```

Example - Using FileStream

"No, you can't have my old file cabinet. I'm using it as a paperweight for all the stuff I'm too lazy to file!"

Nutze FileStream mit StreamReader & StreamWriter.

Teste Read & Write und Peek aus.

Gehe in 2-3 Gruppen – lese den Quellcode deiner Mitschüler und stelle fest was in die Datei geschrieben wurde.

```
//Create FileInfo object for DummyFile.txt
FileInfo fi = new FileInfo(@"D:\DummyFile.txt");

//open DummyFile.txt for read operation
FileStream fsToRead = fi.Open(FileMode.OpenOrCreate,
    FileAccess.ReadWrite, FileShare.ReadWrite);

//open DummyFile.txt for write operation
FileStream fsToWrite = fi.Open(FileMode.OpenOrCreate,
    FileAccess.ReadWrite, FileShare.ReadWrite);

//get the StreamReader
StreamReader sr = new StreamReader(fsToRead);
//read all texts using StreamReader object
string fileContent = sr.ReadToEnd();
Console.WriteLine("Content:\n-------------\n" + fileContent);

//get the StreamWriter
StreamWriter sw = new StreamWriter(fsToWrite);
//write some text using StreamWriter
fsToWrite.Seek(0, SeekOrigin.End);
sw.WriteLine("\nAnother line from streamwriter");
sw.Close();

//read again
fsToRead.Seek(0, SeekOrigin.Begin);
fileContent = sr.ReadToEnd();
Console.WriteLine("Content:\n-------------\n" + fileContent);
sr.Close();

//close all Stream objects
fsToRead.Close();
fsToWrite.Close();
```

Open two FileStreams one for reading another one for writing && StreamReader StreamWriter

Test Peek()

Example - 1x1

Erstelle ein Programm, welches das kleine 1x1 in eine Datei schreibt und dieses wieder von der Datei einlesen kann und in der Console ausgibt.

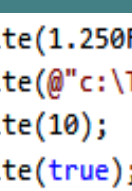
Nutze StreamReader & StreamWriter in einem Using-Block

```
Please find the

1x1= 1
1x2= 2
1x3= 3
1x4= 4
1x5= 5
1x6= 6
1x7= 7
1x8= 8
1x9= 9
1x10= 10
==============
2x1= 2
2x2= 4
2x3= 6
2x4= 8
2x5= 10
2x6= 12
2x7= 14
2x8= 16
2x9= 18
2x10= 20
==============
3x1= 3
3x2= 6
3x3= 9
3x4= 12
3x5= 15
3x6= 18
3x7= 21
3x8= 24
3x9= 27
3x10= 30
==============
```

Excercise
Binary Reader & Writer

Schreibe 4 Werte (Kommazahl, Zeichenkette, ganze Zahl, Wahrheitswert) in eine Binärdatei, lese diese wieder ein.

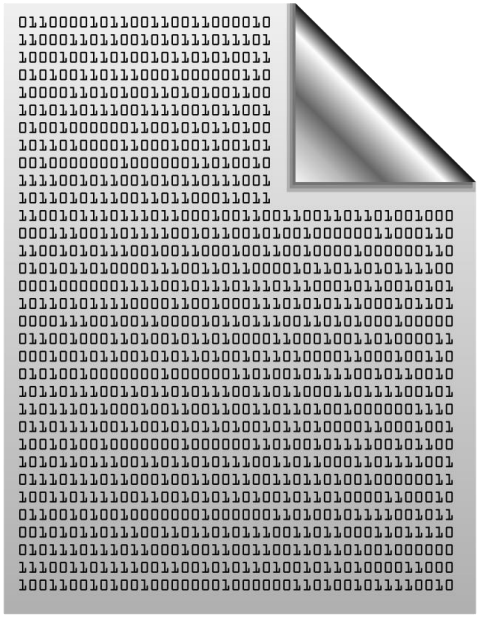
```
writer.Write(1.250F);
writer.Write(@"c:\Temp");
writer.Write(10);
writer.Write(true);
```



```
float aspectRatio;
string tempDirectory;
int autoSaveTime;
bool showStatusBar;
```

AppSettings.dat

```
00000000  00 00 A0 3F 07 63 3A 5C  54 65 6D 70 0A 00 00 00  ...?.c:\Temp....
00000010  01                                                .
```



Excercise BinaryFormatter

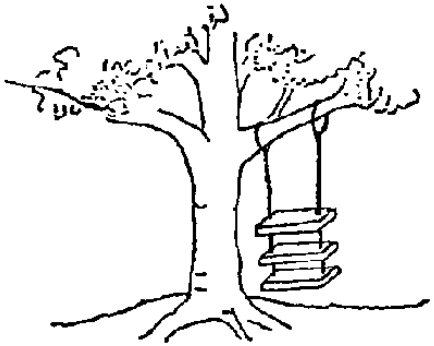
Personen serialisieren und deserialisieren

Erstelle eine Klasse Person mit Name und Alter, serialisiere diese Person in eine Datei mit dem Binary Formatter. Lies anschließend diese Person wieder ein.

Erstelle eine Liste von Personen, schreibe die gesamte Liste mit allen Personen mit dem BinaryFormatter in eine Datei.

Solutions

zu den Aufgabenstellungen



Solution - Ten Numbers

Schreibe 1..10 in eine Datei.

Öffne die Datei und gib den Inhalt in der Console aus.

StreamWriter

```
public static void JustWrite(string filename)
{
    Int64 x;
    try
    {
        //Open the File
        StreamWriter sw = new StreamWriter(filename, true, Encoding.ASCII);

        //Writeout the numbers 1 to 10 on the same line.
        for (x = 0; x < 10; x++)
        {
            sw.Write(x);
        }

        //close the file
        sw.Close();
    }
    catch (Exception e)
    {
        Console.WriteLine("Exception: " + e.Message);
    }
    finally
    {
        Console.WriteLine("Executing finally block.");
    }
}
```

Nutze StreamWriter um eine Datei filename zu erzeugen und eine Reihe von Zahlen darin zu speichern.

StreamReader

```
public static void JustRead(string filename)
{
    String line;
    try
    {
        //Pass the file path and file name to the StreamReader constructor
        StreamReader sr = new StreamReader(filename);

        //Read the first line of text
        line = sr.ReadLine();

        //Continue to read until you reach end of file
        while (line != null)
        {
            //write the lie to console window
            Console.WriteLine(line);
            //Read the next line
            line = sr.ReadLine();
        }

        //close the file
        sr.Close();
        Console.ReadLine();
    }
    catch (Exception e)
    {
        Console.WriteLine("Exception: " + e.Message);
    }
    finally
    {
        Console.WriteLine("Executing finally block.");
    }
}
```

Nutze StreamReader um eine Datei filename zu öffnen und zeilenweise aus der Datei zu lesen und in der Console auszugeben.

Solution - 1x1

Erstelle ein Programm, welches das kleine 1x1 in eine Datei schreibt und dieses wieder von der Datei einlesen kann und in der Console ausgibt.

Nutze StreamReader & StreamWriter in einem Using-Block

```
Please find the below generated table of 1 to 10

1x1= 1
1x2= 2
1x3= 3
1x4= 4
1x5= 5
1x6= 6
1x7= 7
1x8= 8
1x9= 9
1x10= 10
==============
2x1= 2
2x2= 4
2x3= 6
2x4= 8
2x5= 10
2x6= 12
2x7= 14
2x8= 16
2x9= 18
2x10= 20
==============
3x1= 3
3x2= 6
3x3= 9
3x4= 12
3x5= 15
3x6= 18
3x7= 21
3x8= 24
3x9= 27
3x10= 30
==============
```

Example 1x1 = 1

- Erstelle ein Programm, welches das kleine 1x1 in eine Datei schreibt und dieses wieder von der Datei einlesen kann und in der Console ausgibt.

- Nutze StreamReader & StreamWriter in einem Using-Block

```
static void ReadFromFile()
{
    using (StreamReader sr = File.OpenText(@"table.tbl"))...
}

static void WriteToFile()
{
    using (StreamWriter sw = File.CreateText(@"table.tbl"))
    {
        sw.WriteLine("Please find the below generated table of 1 to 10");
        sw.WriteLine("");
        for (int i = 1; i <= 10; i++)...
        Console.WriteLine("Table successfully written on file.");
    }
}
```

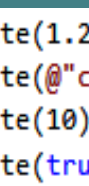
Lösung 1x1

```
static void ReadFromFile()
{
    using (StreamReader sr = File.OpenText(@"table.tbl"))
    {
        string tables = null;

        while ((tables = sr.ReadLine()) != null)
        {
            Console.WriteLine("{0}", tables);
        }
        Console.WriteLine("Table Printed.");
    }
}

static void WriteToFile()
{
    using (StreamWriter sw = File.CreateText(@"table.tbl"))
    {
        sw.WriteLine("Please find the below generated table of 1 to 10");
        sw.WriteLine("");
        for (int i = 1; i <= 10; i++)
        {
            for (int j = 1; j <= 10; j++)
            {
                sw.WriteLine("{0}x{1}= {2}", i, j, (i * j));
            }
            sw.WriteLine("==============");
        }
        Console.WriteLine("Table successfully written on file.");
    }
}
```

Solution
Binary Reader & Writer

Schreibe 4 Werte (Kommazahl, Zeichenkette, ganze Zahl, Wahrheitswert) in eine Binärdatei, lese diese wieder ein.

```
writer.Write(1.250F);
writer.Write(@"c:\Temp");
writer.Write(10);
writer.Write(true);
```



```
float aspectRatio;
string tempDirectory;
int autoSaveTime;
bool showStatusBar;
```

AppSettings.dat

```
00000000  00 00 A0 3F 07 63 3A 5C  54 65 6D 70 0A 00 00 00  ...?.c:\Temp....
00000010  01                                                .
```

BinärWriter

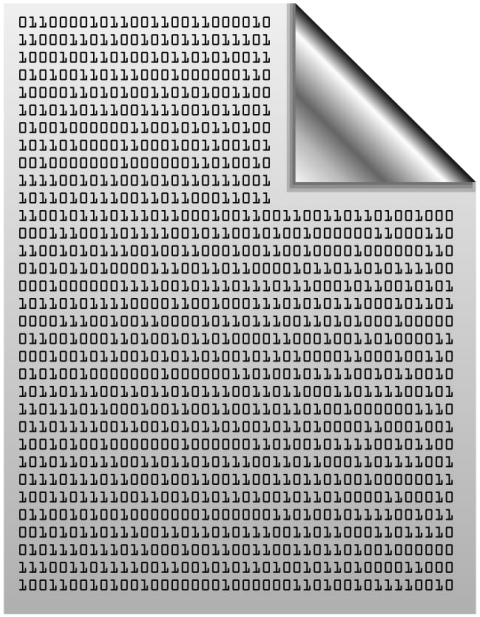
```
const string fileName = "AppSettings.dat";

public static void TestBinary()
{
    WriteDefaultValues();
    DisplayValues();
}
public static void WriteDefaultValues()
{
    using (BinaryWriter writer = new BinaryWriter(
        File.Open(fileName, FileMode.Create)))
    {
        writer.Write(1.250F);
        writer.Write(@"c:\Temp");
        writer.Write(10);
        writer.Write(true);
    }
}
```

'AppSettings.dat'

NUL NUL ?BELc:\Temp
NUL NUL NUL SOH

BinärReader

```
public static void DisplayValues()
{
    float aspectRatio;
    string tempDirectory;
    int autoSaveTime;
    bool showStatusBar;

    if (File.Exists(fileName))
    {
        using (BinaryReader reader = new BinaryReader(
            File.Open(fileName, FileMode.Open)))
        {
            aspectRatio = reader.ReadSingle();
            tempDirectory = reader.ReadString();
            autoSaveTime = reader.ReadInt32();
            showStatusBar = reader.ReadBoolean();
        }

        Console.WriteLine("Aspect ratio set to: " + aspectRatio);
        Console.WriteLine("Temp directory is: " + tempDirectory);
        Console.WriteLine("Auto save time set to: " + autoSaveTime);
        Console.WriteLine("Show status bar: " + showStatusBar);
    }
}
```



Solution BinaryFormatter

Personen serialisieren und deserialisieren

Erstelle eine Klasse Person mit Name und Alter, serialisiere diese Person in eine Datei mit dem Binary Formatter. Lies anschließend diese Person wieder ein.

Erstelle eine Liste von Personen, schreibe die gesamte Liste mit allen Personen mit dem BinaryFormatter in eine Datei.

Beispiel Person

- Person mit Name und Alter serialisieren und deserialisieren

```
[Serializable()]
class Person {
  public string Name {get; set;}
  private int _Alter;
  // ----- Konstruktor -----
  public Person(int alter, string name) {
    _Alter = alter;
    Name = name;
  }
  public int Alter {
    get { return _Alter; }
  }
}
```

```
using System.Runtime.Serialization.Formatters.Binary;
...
Person pers = new Person(56, "Schmidt");
FileStream stream;
stream = new FileStream(@"D:\MyPerson.dat", FileMode.Create);
BinaryFormatter formatter = new BinaryFormatter();
formatter.Serialize(stream, pers);
stream.Close();
```

```
Person pers;
BinaryFormatter formatter = new BinaryFormatter();
FileStream stream = new FileStream(@"D:\MyPerson.dat", FileMode.Open);
pers = (Person)formatter.Deserialize(stream);
```

Mehrere Objekte Serialisieren

- Erstelle eine Liste von Personen und Firmen, speichere diese in einer Arraylist und serialisiere diese. Lies anschließend die gesamte Liste wieder ein

```
// serialisierbare Klassen
 [Serializable()]
class Person {
  public string Name { get; set; }
  private int _Alter;
  // ----- Konstruktor -----
  public Person(string name, int alter) {
    Name = name;
    _Alter = alter;
  }
  // Eigenschaftsmethode
  public int Alter {
    get { return _Alter; }
  }
}

 [Serializable()]
class Firma {
  public string Name { get; set; }
  //----- Konstruktor -----
  public Firma(string name) {
    Name = name;
  }
}
```

Serialisieren von Firmen und Personen

```
static void Main(string[] args) {
  ArrayList arrList = new ArrayList();
  Person pers1 = new Person("Fische", 23);
  Person pers2 = new Person("Beate", 38);
  Firma firma1 = new Firma("Tollsoft");
  Firma firma2 = new Firma("Microsoft");
  // Objekte einer ArrayList übergeben
  arrList.Add(pers1);
  arrList.Add(pers2);
  arrList.Add(firma1);
  arrList.Add(firma2);
  // ArrayList-Objekt serialisieren
  SaveObject(arrList);
  ArrayList newArrList = new ArrayList();
  // Deserialisieren und die Objektdaten anzeigen
  GetObject(ref newArrList);
  foreach (object obj in newArrList) {
    if (obj is Person) {
      Console.Write("{0}/", ((Person)obj).Name);
      Console.WriteLine(((Person)obj).Alter);
    }
    else if (obj is Firma)
      Console.WriteLine(((Firma)obj).Name);
  }
  Console.ReadLine();
}
```

Serialisieren

- Eine gesamte Liste von unterschiedlichen Typen als ArrayList serialisieren

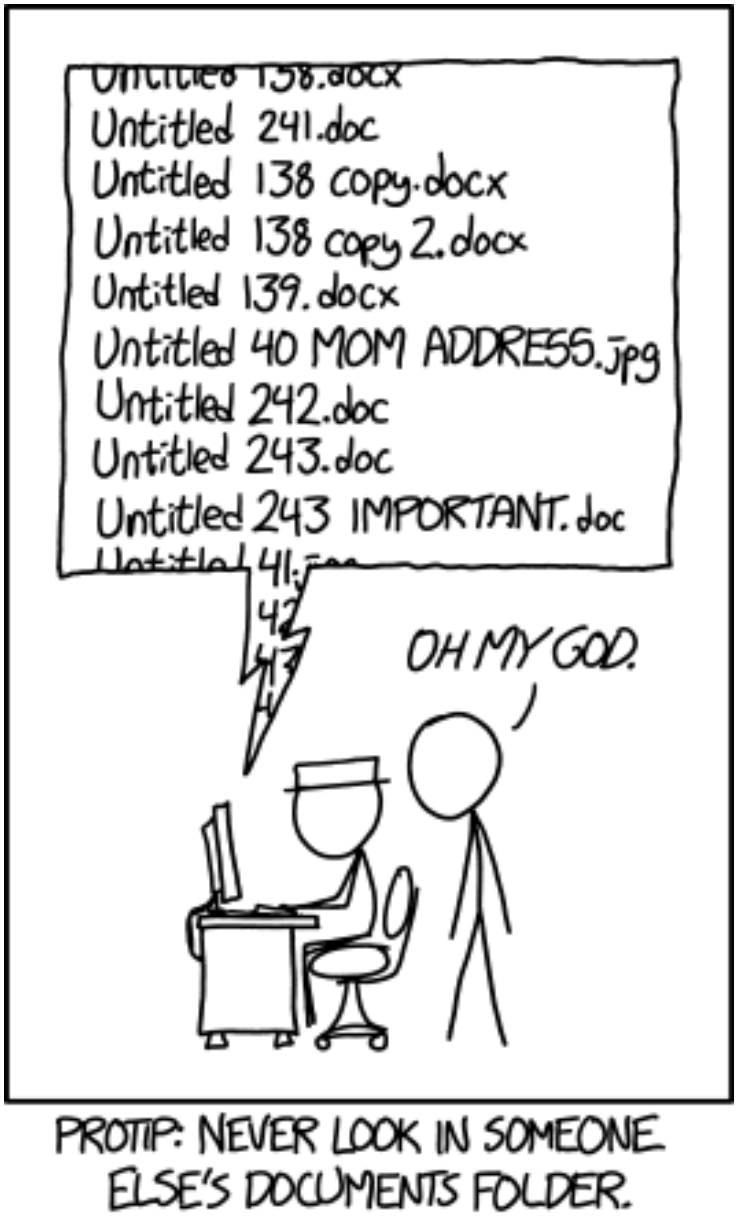
```
// ArrayList-Objekt speichern
public static void SaveObject(ArrayList arr) {
  FileStream fs = new FileStream(@"D:\Objektliste.ifn", FileMode.Create);
  BinaryFormatter binFormatter = new BinaryFormatter();
  binFormatter.Serialize(fs, arr);
  fs.Close();
}
```

Deserialisieren

- Einlesen einer Arraylist

```
// Datei lesen und das ArrayList-Objekt in den Referenzparameter
// schreiben - der Rückgabewert gibt Aufschluss über den
// Erfolg der Deserialisierung
public static void GetObject(ref ArrayList arr) {
  FileStream fs = new FileStream(@"D:\Objektliste.ifn", FileMode.Open); ;
  try {
    BinaryFormatter formatter = new BinaryFormatter();
    arr = (ArrayList)formatter.Deserialize(fs);
  }
  catch (SerializationException e) {
    // die Datei kann nicht deserialisiert werden
    Console.WriteLine(e.Message);
  }
  catch (IOException e){
  // Beim Versuch, die Datei zu öffnen, ist ein Fehler aufgetreten.
  Console.WriteLine(e.Message);
  }
}
```

End of Files...


Untitled 241.doc
Untitled 138 copy.docx
Untitled 138 copy 2.docx
Untitled 139.docx
Untitled 40 MOM ADDRESS.jpg
Untitled 242.doc
Untitled 243.doc
Untitled 243 IMPORTANT.doc
OH MY GOD.
PROTIP: NEVER LOOK IN SOMEONE ELSE'S DOCUMENTS FOLDER.