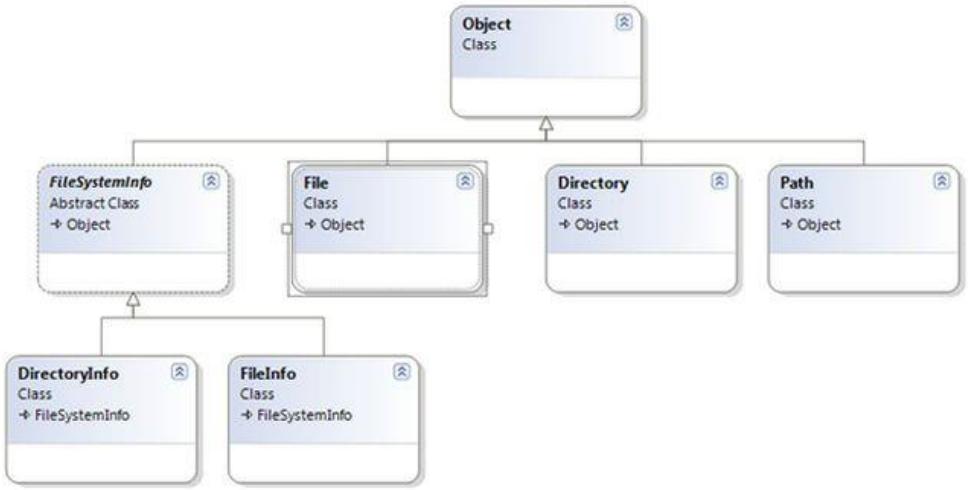
Arbeiten mit Dateien

File & Directory

SEW2



Übersicht

- Dateien & Streams
- Laufwerke, Verzeichnisse und Dateien
- Klasse File & FileInfo
  - Exists / Open
  - FileMode / FileAccess / FileShare
- Klasse Directory & DirectoryInfo
  - Path
- Streams
  - StreamReader & StreamWriter

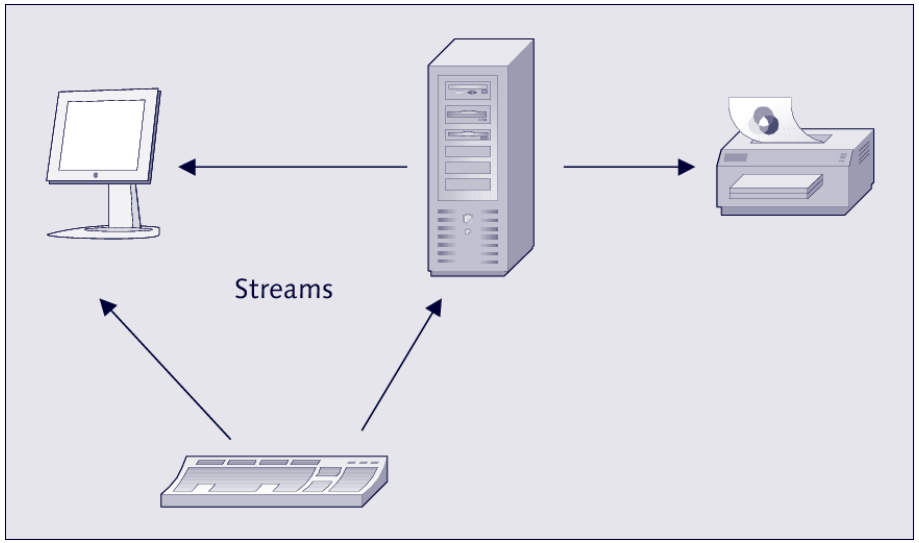
http://openbook.rheinwerk-verlag.de/visual_csharp_2010/visual_csharp_2010_12_001.htm

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



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

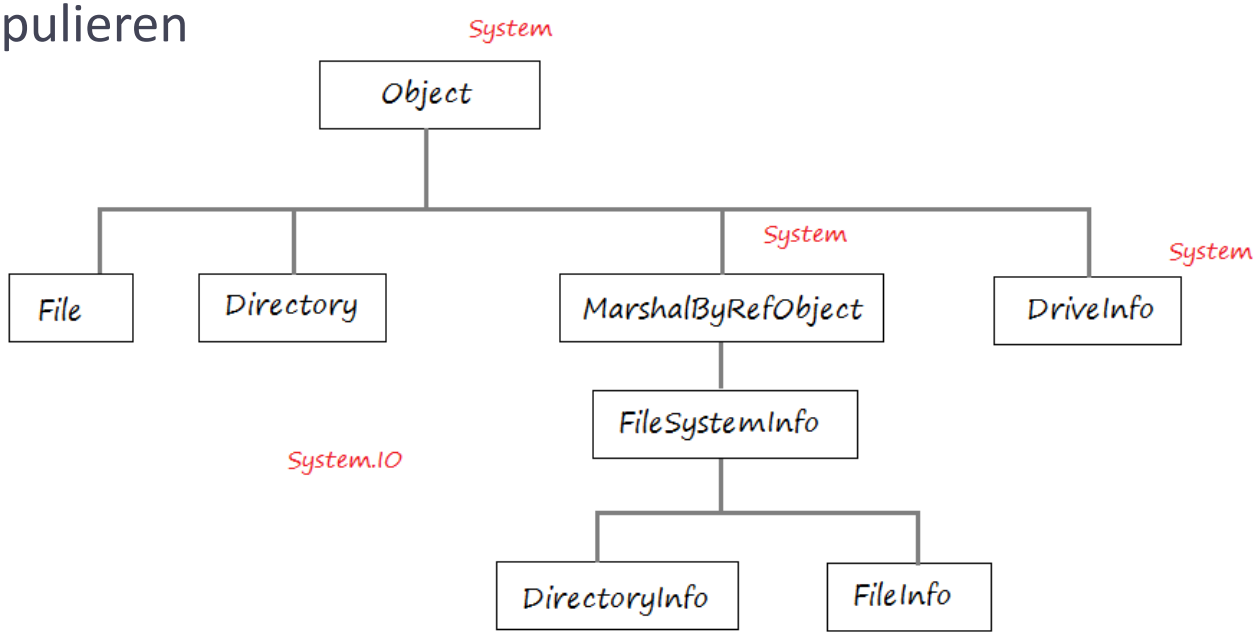
Namespaces der Ein- bzw. Ausgabe

- 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

Verzeichnisse & Dateien

bearbeiten & manipulieren



Verzeichnisse und Dateien

- File
  - beinhaltet statische Methoden
- FileInfo
  - beinhaltet Instanzmethoden
- Directory
  - beinhaltet statische Methoden
- DirectoryInfo
  - beinhaltet Instanzmethoden
- Path
  - beinhaltet statische Methoden

Die Klasse »File«

- nicht ableitbare Klasse File
- stellt nur statische Methoden zur Verfügung
  - Methoden von FileInfo sind Instanzmethoden
    - FileInfo i = new FileInfo() i.Methode();

- File & FileInfo sind funktionell ähnlich
  - Datei erstellen kopieren, löschen usw.
  - Attribute einer Datei lesen oder setzen
  - Datei öffnen zum Lesen und schreiben

Die Klasse »File«

Methode	Rückgabetyp	Beschreibung
AppendAllText	void	Öffnet eine Datei und fügt die angegebene Zeichenfolge an die Datei an.
AppendText	StreamWriter	Hängt Text an eine existierende Datei an.
Copy	void	Kopiert eine bestehende Datei an einen anderen Speicherort.
Create	FileStream	Erzeugt eine Datei in einem angegebenen Pfad.
CreateText	StreamWriter	Erstellt oder öffnet eine Textdatei.
Delete	**void**	**Löscht eine Datei.**
Exists	**Boolean**	**Gibt einen booleschen Wert zurück, der false ist, wenn die angegebene Datei nicht existiert.**
GetAttributes	FileAttributes	Liefert das Bitfeld der Dateiattribute.
GetCreationTime	DateTime	Liefert das Erstellungsdatum und die Uhrzeit einer Datei.

Die Klasse »File«

Methode	Rückgabetyp	Beschreibung
GetLastAccessTime	DateTime	Liefert Datum und Uhrzeit des letzten Zugriffs.
GetLastWriteTime	DateTime	Liefert Datum und Uhrzeit des letzten Schreibzugriffs.
Move	void	Verschiebt eine Datei in einen anderen Ordner oder benennt sie um.
Open	**FileStream**	**Öffnet eine Datei.**
OpenRead	FileStream	Öffnet eine Datei zum Lesen.
OpenText	StreamReader	Öffnet eine Textdatei zum Lesen.
OpenWrite	FileStream	Öffnet eine Datei zum Schreiben.
ReadAllBytes	byte[]	Öffnet eine Binärdatei und liest den Inhalt der Datei in ein Byte-Array ein.
ReadAllLines	**string[]**	**Öffnet eine Textdatei und liest alle Zeilen der Datei in ein Zeichenfolgen-Array ein.**

Die Klasse »File«

Methode	Rückgabetyp	Beschreibung
ReadAllText	string	Öffnet eine Textdatei, liest alle Zeilen der Datei in eine Zeichenfolge ein und schließt dann die Datei.
SetAttributes	void	Setzt Dateiattribute.
SetCreationTime	void	Setzt Erstellungsdatum und -uhrzeit.
SetLastAccessTime	void	Setzt Datum und Uhrzeit des letzten Zugriffs.
SetLastWriteTime	void	Setzt Datum und Uhrzeit des letzten Schreibzugriffs.
WriteAllBytes	void	Erstellt eine neue Datei und schreibt das angegebene Byte-Array in die Datei.
WriteAllLines	**void**	**Erstellt eine neue Datei und schreibt das angegebene Zeichenfolgen-Array in die Datei.**
WriteAllText	void	Erstellt eine neue Datei und schreibt das angegebene Zeichenfolgen-Array in die Datei.

Prüfen, ob eine Datei existiert

- Prüfen ob eine Datei existiert,
  bevor man probiert darauf zuzugreifen.

- File.Exists liefert booleschen Wert

```
string path = @"C:\MyFile.txt";
if (File.Exists(path)) {
  // Datei existiert im angegebenen Pfad
}
```

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

Konstanten der Enumeration »FileMode«

FileMode-Konstante	Beschreibung
Append	Öffnet eine bestehende Datei und setzt den Dateizeiger an das Dateiende. Damit wird das Anhängen von Dateninformationen an die Datei ermöglicht. Existiert die Datei noch nicht, wird sie erzeugt.
Create	Erzeugt eine neue Datei. Existiert bereits eine gleichnamige Datei, wird diese überschrieben.
CreateNew	Erzeugt in jedem Fall eine neue Datei. Existiert im angegebenen Pfad bereits eine gleichnamige Datei, wird die Ausnahme IOException ausgelöst.
Open	Öffnet eine bestehende Datei. Wird diese unter der Pfadangabe nicht gefunden, kommt es zur Ausnahme FileNotFoundException.
OpenOrCreate	Öffnet eine bestehende Datei. Sollte diese im angegebenen Pfad nicht existieren, wird eine neue erzeugt.
Truncate	Öffnet eine Datei und löscht deren Inhalt. Nachfolgende Leseoperationen führen dazu, dass eine Ausnahme ausgelöst wird.

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.

Codezeile zum 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

In eine Datei schreiben oder lesen

```
File.WriteAllText(
    @"C:\MyTextFile.txt", text);
```

- Inhalt der Variablen text
  in die angegebene Datei schreiben
- Existiert die Datei schon,
  wird diese überschrieben

```
Console.WriteLine(
    File.ReadAllText(
    @"C:\MyTextFile.txt"));
```

- Gibt Inhalt einer Datei in der Console aus.

Klasse »FileInfo«

- File veröffentlicht nur statische Methoden,
- FileInfo-Methoden beziehen sich auf eine konkrete Instanz
  - ist eine nicht ableitbare Klasse (sealed)
  - ist das Pendant zur Klasse File

```
FileInfo myFile =
   new FileInfo(@"C:\MyDocuments\MyFile.txt");
```

Directory und DirectoryInfo

- Directory mit statischen Methoden
- DirectoryInfo - konkrete Instanz
  - Directory kann
    - Ordner anlegen, löschen oder verschieben
    - Dateinamen von Verzeichnis abrufen
    - verzeichnisspezifische Eigenschaften abfragen
      - Erstellungsdatum
      - Datum des letzten Zugriffs

Directory & File vs … Info

- Operative Berechtigungen
  sind Voraussetzung für einen Dateizugriff
  - Anwender keine Berechtigung -> Zugriff abgelehnt

  - Klassen File und Directory
    - prüfen bei jedem Zugriff erneut
    - belasten so das System unnötig

  - DirectoryInfo und FileInfo
    - prüfen einmalig

Klasse »Path« - Pfadangabe

- Speicherort einer Datei oder eines Verzeichnisses
- Methoden der Klasse Path (statisch)
  - filtern eine Pfadangabe nach
    - zB: Verzeichnisname, Dateiname, Dateiendung, …

Methode	Beschreibung
GetDirectoryName	Liefert aus einer gegebenen Pfadangabe das Verzeichnis zurück.
GetExtension	**Liefert aus einer gegebenen Pfadangabe die Dateierweiterung einschließlich des führenden Punktes zurück.**
GetFileName	**Liefert den vollständigen Dateinamen zurück.**
GetFileNameWithoutExtension	Liefert den Dateinamen ohne Dateierweiterung zurück.
GetFullPath	Liefert die komplette Pfadangabe zurück.
GetPathRoot	Liefert das Stammverzeichnis.

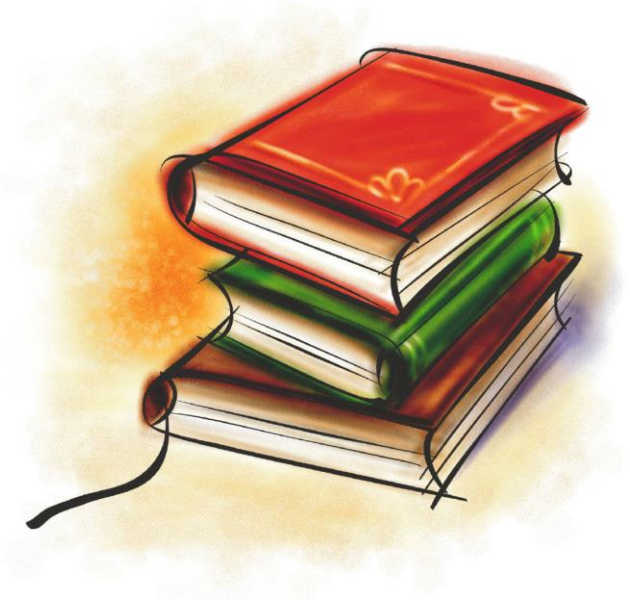
Using File-Methods

```
//Check whether file is exists or not at particular location
bool isFileExists = File.Exists(@"C:\ DummyFile.txt"); // returns false

//Copy DummyFile.txt as new file DummyFileNew.txt
File.Copy(@"C:\DummyFile.txt", @"D:\NewDummyFile.txt");

//Get when the file was accessed last time
DateTime lastAccessTime = File.GetLastAccessTime(@"C:\DummyFile.txt");

//get when the file was written last time
DateTime lastWriteTime = File.GetLastWriteTime(@"C:\DummyFile.txt");

// Move file to new location
File.Move(@"C:\DummyFile.txt", @"D:\DummyFile.txt");

//Open file and returns FileStream for reading bytes from the file
FileStream fs = File.Open(@"D:\DummyFile.txt", FileMode.OpenOrCreate);

//Open file and return StreamReader for reading string from the file
StreamReader sr = File.OpenText(@"D:\DummyFile.txt");

//Delete file
File.Delete(@"C:\DummyFile.txt");
```

Example 1 - Book

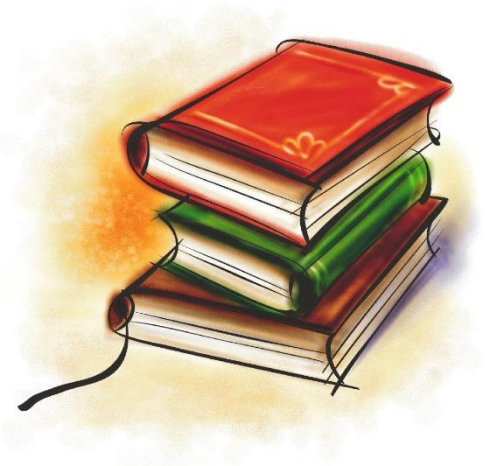
File.ReadAllLine

File.WriteAllLine



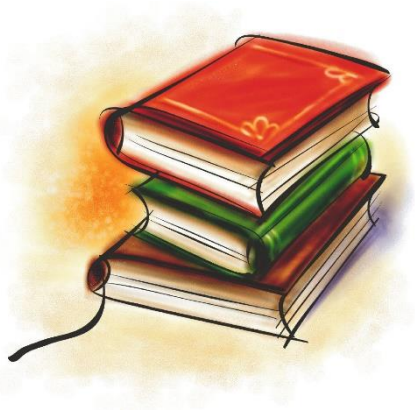
Lesen und Schreiben in CSV

```
public class Book
{
    public decimal Price { get; set; }
    public string Title { get; set; }
    public string Author { get; set; }

}

public static void TestBookToCsvAndBack()
{
    Book b = new Book();
    b.Price = 34.5m;
    b.Author = "Ella Kensington";
    b.Title = "Mary";
    BookExample.WriteBook(b);
    BookExample.ReadBook();
}

public static string ToCsv(Book b)
{
    return String.Format("{0};{1};{2}", b.Price, b.Title, b.Author);
}

public static string ToBook(Book b)
{
    return String.Format("{1} von {2} kostet: {0}", b.Price, b.Title, b.Author);
}
```



Lesen & Schreiben mit File

```
//Book
//Decimal;String;String
//34.5;Mary;Ella Kensington
public static void WriteBook(Book b)
{
    File.WriteAllText("books.csv", ToCsv(b));
}

public static void ReadBook()
{
    Book b = new Book();
    string[] lines = File.ReadAllLines("books.csv");
    string[] words;
    for (int i = 0; i < lines.Length; i++)
    {
        words = lines[i].Split(';');
        b.Price = Convert.ToDecimal(words[0]);
        b.Title = words[1];
        b.Author = words[2];
        Console.WriteLine("Book: {0}" , ToBook(b) );
    }
}
```



```
24,5;Robin;Ella Kensington
12,9;Mysterio;Ella Kensington
14,0;Sieben Botschaften;Ella Kensington
25,9;Gespräche mit Gott;Neal Walsch
34,5;Mary;Ella Kensington
```

Example 2 - Car

File.ReadAllLine

File.WriteAllLine

File.AppendAllText

Lesen und Schreiben

```
class Car {
    public string Color { get; set; }
    public string Brand { get; set; }
    //add 3 attributes

    public override string ToString() {
        return $"Auto der Marke {Brand} ist {Color}.";
    }
    public string ToCsv() {
        return $"{Brand};{Color}";
    }

    public void AppendOneCarToCsv(string filename) {
        File.AppendAllText(filename, this.ToCsv());
    }

    public static void WriteCsv(string filename, Car[] cars) ...

    public static Car[] ReadCsv(string filename) ...

    public static void TestCars() ...

}
```

```
public static void TestCars() {
    //Cars erstellen und in ein Array speichern
    Car[] cars = new Car[3];
    cars[0] = new Car() { Brand = "BMW", Color = "Blau" };
    cars[1] = new Car() { Brand = "Audi", Color = "Rot" };
    cars[2] = new Car() { Brand = "Fiat", Color = "Blau" };
    //Array in csv-Datei speichern
    Car.WriteCsv("cars.csv", cars);
    //Einzelne Instanz zur Csv-Datei hinzufügen
    Car c1 = new Car() { Color = "Green", Brand = "Toyota" };
    c1.AppendOneCarToCsv("cars.csv");
    //Cars von csv Laden und ausgeben
    Car[] readCars = Car.ReadCsv("cars.csv");
    foreach (var item in readCars) {
        Console.WriteLine(item.ToString());
    }
}
```

Lesen und Schreiben mit Cars

```
public static Car[] ReadCsv(string filename) {
    string[] lines = File.ReadAllLines(filename);
    string[] words;
    Car[] cars = new Car[lines.Length];
    for (int i = 0; i < lines.Length; i++) {
        Car b = new Car();
        words = lines[i].Split(';');
        b.Brand = words[0];
        b.Color = words[1];
        cars[i] = b;
    }
    return cars;
}
```

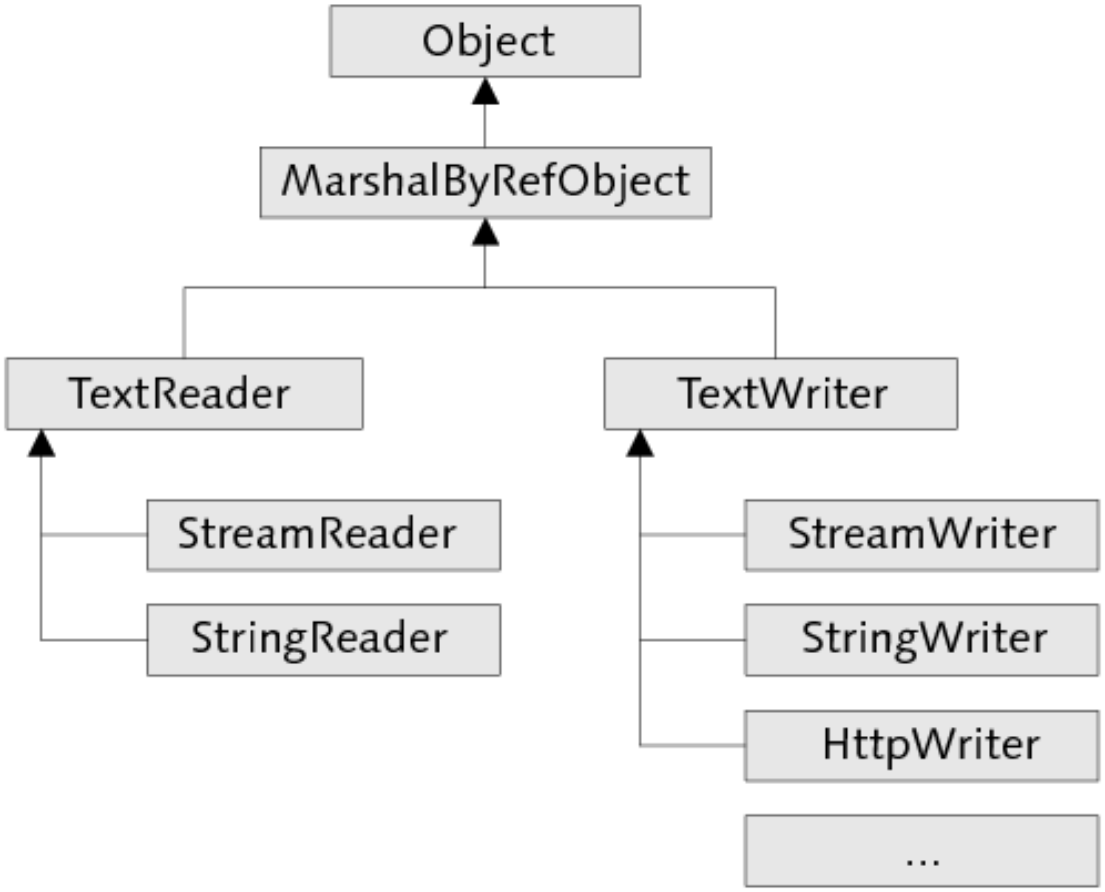
```
public static void WriteCsv(string filename, Car[] cars) {
    string[] carsAsCsv = new string[cars.Length];
    for (int i = 0; i < cars.Length; i++) {
        carsAsCsv[i] = cars[i].ToCsv();
    }
    File.WriteAllLines(filename, carsAsCsv);
}
```

StreamReader & StreamWriter

SEW2

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 - Konstruktor

- Parameter: Pfadangabe zu einer Datei

```
public StreamWriter(string);
```

```
StreamWriter myStreamWriter =
new StreamWriter(@"D:\MyText.txt");
```

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

- int: (letzter Parameter)
  - Größe des Puffers beeinflussen / setzen

Schreiben in den Datenstrom

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

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

Close - StreamWriter & Streams

- **StreamWriter** müssen aufgrund ihres lokalen Puffers unbedingt *vor* dem zugrunde liegenden **Stream**-Objekt geschlossen werden
  - was nur durch einen **Close()** - Aufruf
    - oder einen äquivalenten **Dispose()** - Aufruf

    an das **StreamWriter**-Objekt sichergestellt ist

  - mit **using**-Block findet dies automatisiert statt

Eigenschaften der Klasse »StreamWriter«

Standardmäßig wird als Encoding UTF-8 verwendet.

Das Encoding kann explizit angegeben werden:

```
StreamWriter sw = new StreamWriter(@"C:\NewFile.txt",
                                   false, Encoding.Unicode);
```

```
//Liefert das aktuelle Encoding-Schema zurück:
Console.WriteLine("Format: {0}", sw.Encoding);
```

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

Konstanten der Klasse 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 in der Little-Endian-Bytereihenfolge ab.
UTF32	Ruft eine Codierung für das UTF-32-Format in der Little-Endian-Bytereihenfolge 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

Hello World

```
public static void HalloWelt()
{
    try
    {
        //Pass the filepath and filename to the StreamWriter Constructor
        StreamWriter sw = new StreamWriter("Test.txt");

        //Write a line of text
        sw.WriteLine("Hello World!!");

        //Write a second line of text
        sw.WriteLine("From the StreamWriter class");

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

Nutze StreamWriter um eine Datei „Test.txt“ zu erzeugen und „Hallo Welt“ darin zu speichern.

Klasse StreamReader

Lesen einer Textdatei

StreamReader

- Ermöglicht Daten
  einer bestimmten Kodierung zu lesen
  - Gegenstück zum StreamWriter

- Konstruktor:
  - Pfadangabe & Encoding & Puffergröße setzen
- StreamReader zum Lesen aus Datenströmen nutzen

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

Methoden von StreamReader

Methode	Beschreibung
Peek	Liest ein Zeichen aus dem Strom und liefert den int-Wert zurück, der das Zeichen repräsentiert, verarbeitet das Zeichen aber nicht. Der Zeiger wird nicht auf die Position des folgenden Zeichens gesetzt, wenn Peek aufgerufen wird, sondern verbleibt in seiner Stellung. Verweist der Zeiger hinter den Datenstrom, ist der Rückgabewert –1.
Read	Liest ein oder mehrere Zeichen aus dem Strom und liefert den int-Wert zurück, der das Zeichen repräsentiert. Ist kein Zeichen mehr verfügbar, ist der Rückgabewert –1. Der Positionszeiger verweist auf das nächste zu lesende Zeichen. Eine zweite Variante dieser überladenen Methode liefert die Anzahl der eingelesenen Zeichen.
ReadLine	Liest eine Zeile aus dem Datenstrom – entweder bis zum Zeilenumbruch oder bis zum Ende des Stroms. **Der Rückgabewert ist vom Typ string.**
ReadToEnd	Liest von der aktuellen Position des Positionszeigers bis zum Ende des Stroms alle Zeichen ein. **Der Rückgabewert ist vom Typ string.**

Read mit int als Return-Typ

- Read liefert einen int, der das Zeichen „repräsentiert“ -> cast to char

```
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());
    }
}
```

Autoflush

- boolesche **StreamWriter**-Eigenschaft **AutoFlush** wird festgelegt:
  - ob Zeichen sofort in den Ausgabestrom wandern bei jedem **Write()** oder **WriteLine()**
    - bei bestimmten Ausgabegeräten sinnvoll

    oder
  - zwischengepuffert werden sollen
    - dies bringt höhere Performanz

- Voreinstellung: **false**

Using-Block

für StreamReader & -Writer

Using-Block

```
public static void TestUsingWriter(String path)
{
    using (StreamWriter sr = File.AppendText(path))
    {
        sr.WriteLine("What's a usingblock?!");
        sr.Close();

        Console.WriteLine(File.ReadAllText(path));
    }
}
```

Using sorgt für das Schließen der Streams automatisch

```
public static void TestUsingReader(String path)
{

    using (StreamReader sr = File.OpenText(path))
    {
        String s = "";

        while ((s = sr.ReadLine()) != null)
        {
            Console.WriteLine(s);
        }
    }
}
```

https://www.guru99.com/c-sharp-stream.html

Fortsetzung mit Streams

FileStream

StringReader / -Writer

BinaryReader / -Writer

BinaryFormatter