STREAMS

Vincent Uhlmann
IT-Akademie Dr. Heuer GmbH

SYSTEM.IO

- File
- FileInfo
- Directory
- DirectoryInfo
- DriveInfo
- DriveType
- FileSystemInfo
- Path
- FileSystemWatcher

FILE

- Stellt Methoden für z.B. die Erstellung und das Kopieren, Löschen, Verschieben und Öffnen von Dateien bereit

```
if(File.Exists("C:\\Users\\vu\\Desktop\\test.txt"))
{
    Console.WriteLine("Datei existiert");
}
```

FILEINFO

- Stellt Methoden und Eigenschaften für die Arbeit mit Dateien bereit, einschließlich der Möglichkeit, Dateiinformationen wie Pfad, Größe und Erstellungsdatum abzurufen

```
FileInfo info = new
FileInfo("C:\\Users\\vu\\Desktop\\test.txt");

Console.WriteLine(info.LastAccessTime);
```

DRIVEINFO

- Stellt Methoden und Eigenschaften für die Arbeit mit Verzeichnissen bereit, einschließlich der Möglichkeit, Verzeichnisinformationen wie Pfad, Erstellungsdatum und Zugriffsrechte abzurufen

```
var drives = DriveInfo.GetDrives();

foreach (var drive in drives)
{
    Console.WriteLine(drive.Name);
    Console.WriteLine(drive.DriveType);
    Console.WriteLine(drive.DriveFormat);
    Console.WriteLine(drive.AvailableFreeSpace);
}
```

PATH

- Führt Operationen auf String Instanzen aus, die Datei- oder Verzeichnispfadinformationen enthalten. Diese Operationen werden plattformübergreifend durchgeführt

```
string path = "C:\\Users\\vu\\Desktop\\test.txt";

if (Path.HasExtension(path))
{
    Console.WriteLine("{0} has an extension.", path);
}

var tempFolder = Path.GetTempPath();
var tempFile = Path.GetTempFileName();
```

FILESYSTEMWATCHER

- Überwacht und löst Ereignisse aus, wenn ein Verzeichnis oder eine Datei in einem Verzeichnis geändert werden

```
static void Main()
{
    FileSystemWatcher watcher = new FileSystemWatcher("C:\\Users\\vu\\Desktop");
    watcher.Created += FileCreated;
    watcher.EnableRaisingEvents = true;
    Console.Read();
}

private static void FileCreated(object sender, FileSystemEventArgs e)
{
    Console.WriteLine("Datei wurde erstellt");
}
```

STREAMS

- Ermöglichen das sequenzielle Lesen und Schreiben von Daten
- Streams repräsentieren eine Sequenz von Bytes, die kontinuierlich gelesen oder geschrieben werden können
- Sind im Namespace System.IO zu finden
- Ermöglichen den Zugriff auf verschiedene Datenquellen (Netzwerkverbindungen, Dateien etc.)
- Es gibt verschiedenen Stream Implementationen (Klassen) für verschiedene Anforderungen
- Ermöglicht das Lesen von Daten in „Stücken“, ohne die ganze Datei in den Arbeitsspeicher zu laden

STREAM KLASSE

- Abstrakte Basisklasse für das Lesen und Schreiben von Bytes
- Enthält Methoden wie Read, Write, Seek usw. für den Zugriff auf Daten

- Implementationen
- FileStream
- MemoryStream
- NetworkStream
- …

IMPLEMENTATIONEN

- StringReader & StringWriter
- StreamReader & StreamWriter
- BinaryReader & BinaryWriter
- …

FILESTREAM

- Stellt einen Stream für eine Datei bereit, wobei synchrone und asynchrone Lese- und Schreibvorgänge unterstützt werden
- Erster Parameter ist der Name der Datei, einschließlich Pfad und Dateiendung
- Zweiter Parameter legt den Modus fest, in dem die Datei geöffnet werden soll (Open, Create, OpenCreate, …)
- Close schließt den Stream und gibt alle Ressourcen frei

```
string path = @"C:\Users\vu\Desktop\test.txt";

FileStream fs = new FileStream(path,
FileMode.OpenOrCreate);

byte[] info = Encoding.UTF8.GetBytes("Peter");

fs.Write(info, 0, info.Length);

fs.Close();
```

FILESTREAM

- Dritter Parameter legt fest, auf welche Weise auf die Datei zugegriffen werden soll (Read, Write, ReadWrite)
- Vierter Parameter legt fest, welche Zugriffsrechte anderen auf diese spezifische Datei gewährt werden soll (None, Read, Write, ReadWrite, …)

```
string path = @"C:\Users\vu\Desktop\test.txt";

FileStream fs = new FileStream(path,
FileMode.OpenOrCreate, FileAccess.ReadWrite,
FileShare.Read);

byte[] info = Encoding.UTF8.GetBytes("Peter");

fs.Write(info, 0, info.Length);

fs.Close();
```

MEMORYSTREAM

- Liest oder schreibt Bytes, die im Speicher liegen
- Nützlich für temporäre Speicherung
- Erster Parameter im Konstruktor ist die Kapazität in Bytes
- Weitere Parameter legen z.B. fest, ob der Stream nur Lesbar seien soll

```
byte[] writeString = Encoding.UTF8.GetBytes("Peter ist toll");

MemoryStream memoryStream = new MemoryStream(100);

memoryStream.Write(writeString, 0, writeString.Length);

memoryStream.Position = 0;

byte[] readString = new byte[100];
memoryStream.Read(readString, 0, 100);

string s = Encoding.UTF8.GetString(readString);

Console.WriteLine("Gelesen: " + s);
memoryStream.Close();
```

STRINGREADER

- StringReader ist eine weitere Klasse, die von TextReader erbt und zum Lesen aus einem String verwendet wird
- Ermöglicht das synchrone oder asynchrone Lesen eines Strings. Es kann Zeichen, Zeilen oder den gesamten String lesen
- Wird mit einem String initialisiert aus dem gelesen/geschrieben werden soll

```
StringReader reader = new StringReader("Hallo Peter");

string line = await reader.ReadToEndAsync();

Console.WriteLine(line);

reader.Close();
```

STREAMREADER

- StreamReader erbt von TextReader und bietet Implementierungen für das Lesen aus einem Stream
- Wird verwendet, um Text oder große Sätze aus einer Datei zu lesen. Es bietet Methoden wie Read und ReadLine, um Daten aus dem Stream zu lesen
- Wird mit einem Stream wie z.B. einem FileStream erstellt

```
string path = @"C:\Users\vu\Desktop\test.txt";

FileStream fs = new FileStream(path,
FileMode.OpenOrCreate);

StreamReader reader = new StreamReader(fs);

string? line = await reader.ReadLineAsync();

Console.WriteLine(line);

reader.Close();
```

BINARYREADER / BINARYWRITER

- Geeignet für das Lesen und Schreiben von binären Daten, wie z.B. Bildern, Audio oder jeglicher nicht-textueller Information
- Operiert auf Byte-Ebene, was bedeutet, dass Sie einzelne Bytes oder Blöcke von Bytes von und zu einer Datei lesen und schreiben können

```
string path = @"C:\Users\vu\Desktop\test.txt";

FileStream fs = new FileStream(path,
FileMode.OpenOrCreate);

BinaryWriter bw = new BinaryWriter(fs);

bw.Write("Hallo");
bw.Write(5);
bw.Write("Peter");
bw.Write(6.0);

bw.Close();
```

KOMPRESSION

- Byte-Ströme können komprimiert und dekomprimiert werden
- Namespace System.IO.Compression
- Algorithmen wie Deflate
- Streams sind z.B. DeflateStream oder GZipStream

```
GZipStream gzipStream = new GZipStream(destinationStream,
CompressionMode.Compress);

sourceStream.CopyTo(gzipStream);

gzipStream.Close();
```