Ein- und Ausgabe von Dateien – Teil 2

Ingo Köster

Diplom-Informatiker (FH)

Abstrakte Klasse `Stream`

- Basisklasse für einige der Stream-Klassen
- Stellt alle fundamentalen Eigenschaften und Methoden bereit, die von den abgeleiteten Klassen geerbt werden
- Die von Stream abgeleiteten Klassen unterstützen mit ihren Methoden nur Operationen auf Bytesequenzen
- Der Inhalt eines solchen Stroms muss noch interpretiert werden

Abgeleitete Klassen von `Stream`

- `FileStream`
- `TextReader & TextWriter`
- `StreamReader & StreamWriter`
- `StringReader & StringWriter`
- `BinaryReader & BinaryWriter`
- etc.

- Stellen entsprechende `Read` und/oder `Write` Methoden zur Verfügung

Beispiel

- `StreamReader` und `StreamWriter` stellen Funktionen bereit, um aus Streams zu lesen und zu schreiben

```
FileStream fs = File.Open(@"c:\text.txt", FileMode.Open);

StreamReader sr = new StreamReader(fs);
Console.WriteLine(sr.ReadToEnd());

fs.Close(); // Schließt auch den StreamReader
```

BufferedStream

- Wird benutzt, um Daten eines anderen E/A-Datenstroms zu puffern
- Ein Puffer ist ein Block von Bytes im Arbeitsspeicher
- Dient zum Zwischenspeichern von Datenströmen (Cache)
  - Ziel: Anzahl der Aufrufe an das Betriebssystem zu verringern
- Wird immer im Zusammenhang mit anderen Stream-Klassen eingesetzt

Beispiel BufferedStream

```
› FileStream fs = File.Open(@"c:\datei.txt",
  FileMode.OpenOrCreate);

› BufferedStream bs = new BufferedStream(fs);

› StreamWriter sw = new StreamWriter(bs);

› sw.WriteLine("Daten");

› sw.Close();    // Daten in BufferedStream schreiben
› bs.Close();
```

Kompression

- Byte-Ströme komprimieren und dekomprimieren
  - Aus Namensraum `System.IO.Compression`

- `DeflateStream`
  - Komprimieren und Dekomprimieren von Streams mit dem Deflate-Algorithmus

- `GZipStream`
  - Ebenfalls Deflate-Algorithmus
  - GZIP-Datenformat
  - Kann für die Verwendung anderer Komprimierungsformate erweitert werden
  - Bessere Wahl wenn Daten von anderer Software genutzt werden soll

MemoryStream

› Häufig sind Dateien oder Netzwerkverbindungen das Ziel von Datenströmen

› Daten können jedoch bewusst temporär im Hauptspeicher gespeichert werden

› Ersetzt ggf. die Notwendigkeit Daten in einer temporären Datei zu speichern

› Verhindert z.B. dass eine Datei längere Zeit geöffnet sein muss, wenn diese in einen `MemoryStream` gelesen wird

Alternatives schließen von Streams

› Mittels des using-Statements werden geöffnete Streams beim Verlassen des Blocks geschlossen

```
using (FileStream stream = File.Open(…))
{
    // Hier mit dem Stream arbeiten
}
```