7
Dateien und Datenströme (Streams)

Jörn Loviscach

Versionsstand: 21. März 2014, 22:57

Die nummerierten Felder sind absichtlich leer, zum Ausfüllen beim Ansehen der Videos:
http://www.j3L7h.de/videos.html

Bitte hier notieren, was beim Bearbeiten unklar geblieben ist

0 Ansätze zum Umgang mit Dateien

Festplatten, optische Medien (CD, DVD, Blu-Ray), USB-Sticks oder Speicher in der „Cloud“ (= Server irgendwo im Internet) leisten Einiges, was der normale Hauptspeicher (RAM) nicht beherrscht:

[1]

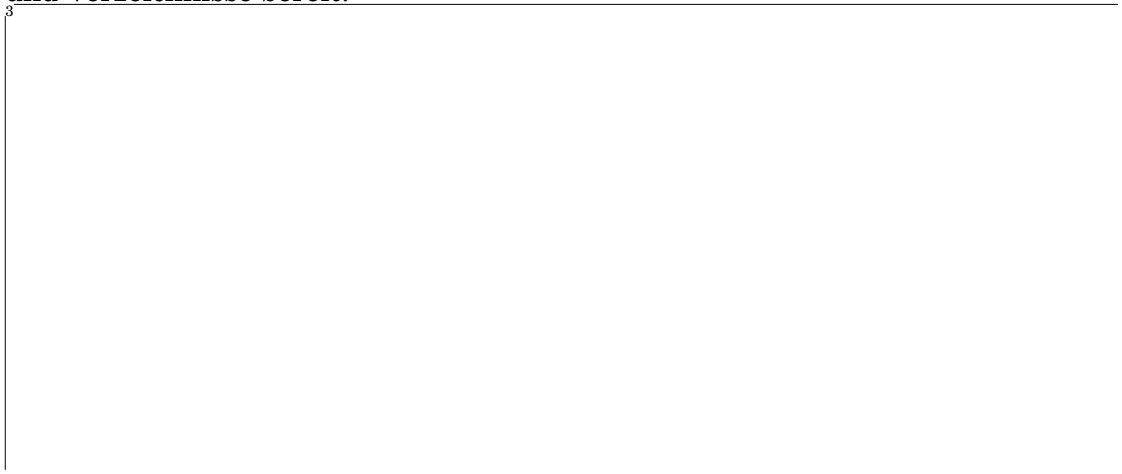
Diese Medien wissen aber nichts über unsere Variablen, sondern speichern nur Folgen von Einsen und Nullen, immerhin meist unterteilt in Dateien [files] und sortiert in Verzeichnisse [directories] = Ordner [folder].

Der Weg aus einen Programm zu einer Datei und zurück ist nicht völlig trivial. Man kann grob drei Ansätze erkennen:[2]

In jedem der drei Ansätze gibt es massiv Möglichkeiten, dass Exceptions auftreten!

1 Direktes Arbeiten mit Dateien und Verzeichnissen

.NET stellt im Namensraum `System.IO` geradlinige Hilfsklassen für Dateien und Verzeichnisse bereit:[3]



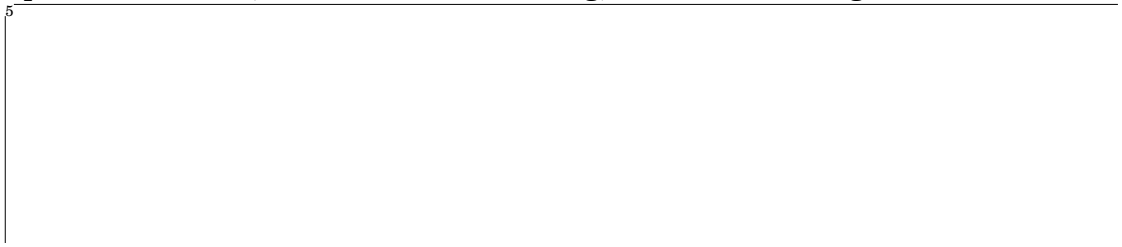
Oft steht man vor dem Problem, alle Unterordner, Unterunterordner, ... eines gegebenen Verzeichnisses durchsuchen zu müssen. Das ist *der* Job für Rekursion: Eine Funktion ruft sich selbst auf.[4]

Allerdings kann man die meisten solchen Suchanfragen mit der Methode `EnumerateFiles` der Klasse `DirectoryInfo` erledigen.

2 Datenströme, Streams

Dateien komplett in den Speicher zu laden oder komplett aus Datenstrukturen im Speicher zu füllen, ist eine einfache Lösung, aber oft nicht möglich:

[5]



„Wahlfreien“ Zugriff [random access] hat man halt nur auf das lokale RAM, daher dessen Name Random-Access Memory. Dateien dagegen bieten einen „sequentiellen“ (fortlaufenden) Zugriff: Am besten fängt man vorne an und hört hinten auf. Es gibt einen Cursor, der bestimmt, welches Element gerade geschrieben oder gelesen wird. Der wandert nach jedem Schreiben/Lesen automatisch weiter:

[6]

Datenströme [streams] sind das informatische Modell dafür – eine Abstraktion, so wie die Queue die Warteschlange vor dem Schalter abstrahiert. Der Vorteil der Abstraktion ist, dass man ganz andere Datenquellen/Datenspeicher mit demselben Modell behandeln kann, also dafür nichts Neues lernen muss. Im Objektkatalog sieht man, welche Klassen von der (abstrakten!) Klasse `Stream` des Namensraums `System.IO` abgeleitet sind: Kompression, Verschlüsselung und vieles mehr lässt sich mit Streams erledigen.

Alle Kindklassen erben die Funktionalität des `Stream`, insbesondere:

[7]

Wir betrachten hier nur die Kindklasse `FileStream` für Dateien. Das Wesentliche ist deren Konstruktor: Der nimmt zum Beispiel den Pfad der zu lesenden oder

schreibenden Datei, einen Modus und eine Zugriffsart entgegen.

[8]

Mit den reinen Streams zu arbeiten, ist nicht allzu lustig, weil diese nur Bytes verarbeiten, keine anderen Datentypen. Ein `BinaryWriter` und ein `BinaryReader` können alle einfachen Datentypen passend übersetzen. Sie sind hilfreich, wenn man bestehende Datenformate schreiben und lesen will:

[9]

Ein `StreamWriter` und ein `StreamReader` übersetzen Zeichenketten in/aus den Bytes des Streams:

[10]

Die Konstruktoren von `StreamWriter` und `StreamReader` können dabei auch direkt einen Dateipfad verarbeiten statt einen `FileStream`.

In C++ schreibt man das Lesen aus bzw. das Schreiben in Streams ganz auffällig in der Art `theStream << 42 << 3.14 << "abc";` und `theStream >> a >> b;`. Java und C# sind zu der üblichen Schreibweise mit Methodenaufrufen zurückgekehrt.

3 Serialisierung

Im einfachsten Fall schreibt man ein neues Programm, das sich nicht um alte Datenformate scheren muss und nur seine eigenen Objekte in Datenströme speichern („serialisieren“) und dann wieder daraus laden („deserialisieren“) muss.

In C# muss ein `[Serializable]` vor der Klassendefinition dem Compiler anzeigen, dass Instanzen dieser Klasse gespeichert und geladen werden können. Das Übersetzen in/aus Bytes übernimmt ein Formatierer, zum Beispiel der `SoapFormatter`, für den die dll `System.Runtime.Serialization.Formatters.Soap` als Verweis in das Projekt einzubinden ist:

[11]

Dieser Formatierer schreibt eine XML-Datei, die man als Text lesen kann – und bearbeiten kann, was manche Problemen lösen hilft.

Java unterstützt eine vergleichbare Serialisierung; C++ ist dafür auf besondere Klassenbibliotheken angewiesen.