Streams

Viele Javaprogramme benötigen Daten aus externen Quellen bzw. geben Daten an externe Dateien zurück. In Java erfolgt das Speichern wie Auslesen von Informationen über so genannte Streams; die entsprechenden Klassen und Methoden stellt das Paket java.io bereit. Ein Eingabestream überträgt Daten aus einer Quelle in ein Programm, ein Ausgabestream sendet Daten aus einem Programm an ein Ziel.

Byte-Streams

transportieren ganze, vorzeichenlose Zahlen (int) im Wertebereich zwischen 0 und 255. Verschiedenste Daten, z.B. numerische Werte, ausführbare Programme, Bilder sowie Klassendateien im Byte-Code, können im Byte-Format übertragen werden.

Eingabestrom

FileInputStream(String)

Konstruktor zur Erstellung eines Eingabestroms

> Das String-Argument steht für den Namen der Quelle (Datei). Absolute oder relative Adressen von Dateien trennen Verzeichnis und Dateinamen je nach Betriebssystem mit Schrägstrich oder umgekehrtem Schrägstrich voneinander; diese Darstellung ist nicht plattformunabhängig. Um die Adressierung zu vereinheitlichen, sollte man Dateinamen und Verzeichnis mit der statischen Konstanten File.separator trennen, indem man letztere mit dem Verkettungsoperator + in den String einfügt.

read()

Rückgabe des aktuellen Byte im Stream in Form einer ganzen Zahl,
Rückgabe des Wertes –1 am Ende des Streams

read(byte[ ], int, int)

liest mehrere Bytes aus einem Stream aus.

Die Argumente der Methode stehen für:

- Byte-Array zur Speicherung der Daten
- Element des Arrays zur Speicherung des ersten Bytes
- Zahl der zu lesenden Bytes

Die Methode gibt die Zahl der gelesenen Bytes zurück oder den Wert –1, wenn keine Bytes vor dem Ende des Streams gelesen wurden.

close( )

schließt den Stream nach Übertragung der Information und gibt Systemressourcen frei, die der offenen Datei zugeordnet sind.

Ausgabestrom

FileOutputStream(String)
Konstruktor zu Erstellung eines Ausgabestroms (analog zu FileInputStream)

In der Kombination FileOutputStream(String, boolean) setzt der Ausgabestrom Daten an das Ende einer existierenden Datei (true) bzw. überschreibt diese (false).

write(byte[ ], int, int)
schreibt analog zur entsprechenden read-Methode mehrere Bytes in den Ausgabestrom;

write(int)
schreibt ein einzelnes Byte in den Strom.

close()
schließt den Strom.
z.B.

```
import java.io.*;
    public class WriteBytes {
        public static void main(String[ ] arguments) {
            int[ ] data = { 0, 64, 128, 255 } ;
            FileOutputStream file = new FileOutputStream("datei");
            for (int i = 0; i < data.length; i++)
            file.write(data[i]);
            file.close( );
        }
    }
```

Gepufferte Streams
erhöhen die Effizienz der Datenübertragung

BufferedInputStream(InputStream)
BufferedOutputStream(OutputStream)
z. B.

```
FileOutputStream file = new FileOutputStream("datei");
BufferedOutputStream buff = new BufferedOutputStream(file);
```

Daten Streams

Um primitive Datentypen (boolean, byte, double, float, int, long, short) direkt aus dem Stream zu lesen bzw. in den Stream zu schreiben, verwendet man Datenstreams:

DataInputStream(InputStream)
DataOutputStream(OutputStream)

zugehörige Methoden sind:

- readBoolean( ), writeBoolean(boolean)
- readByte( ), writeByte(integer)
- readDouble( ), writeDouble(double)
- readInt( ), writeInt(int)
- readLong( ), writeLong(long)
- readShort( ), writeShort(int)

Zeichen-Streams
sind ein spezieller Typ von Byte-Stream, der nur Textdaten z.B. Textdateien, HTML-Dokumente oder Java-Quelldateien im ASCII-Zeichensatz oder in Unicode verarbeitet.

FileReader(String dateiName) bzw. FileReader(File datei)
Konstruktor für einen Zeichen-Eingabestrom

read( )
Rückgabe des aktuellen Zeichens als Integer

read(char[ ], int, int)
Auslesen mehrerér Zeichen aus einem Stream, Speicherung im Zeichenarray

BufferedReader(Reader)
Puffern von Daten in Kombination mit FileReader

readLine( )
Rückgabe der aktuellen Textzeile im Stream in Form eines String-Objektes
Das Streamende bewirkt den Rückgabewert null.

- Zeilenvorschub-Zeichen: '\n'
- Wagenrücklauf-Zeichen: '\r'

Das folgende Beispiel liest mit einem gepufferten Zeichenstream den Text der eigenen Quelldatei und zeigt diesen zeilenweise an.:

```
import java.io.*;
public class ReadSource {
        public static void main(String[ ] arguments) {
                try {
                        FileReader file = new
                                FileReader("ReadSource.java");
                        BufferedReader buff = new
                                BufferedReader(file);
                        boolean eof = false;
                        while (!eof) {
                                String line = buff.readLine( );
                                if (line == null)
                                        eof = true;
                                else
                                        System.out.println(line);
                        }
                        buff.close();
                }  catch (IOException e) {
                        System.out.println("Error -- " + e.toString());
                }
        }
}
```