Java-API: Stream-Klassen

Die API-Bibliothek **`java.io`** bietet zahlreiche Stream-Klassen mit vielen Ein- und Ausgabefunktionen, mit deren Hilfe alle denkbaren Arten von Datenübertragungen und -konvertierungen realisiert werden können.

http://download.oracle.com/javase/tutorial/essential/io/index.html

Prinzipiell lassen sich zwei Klassenhierarchien für zwei unterschiedliche Datenformate unterscheiden:

- Byteorientierte Input-Output-Streams
- unicodebasierte Reader- / Writer-Streams (Unicode-Zeichenformat).

Innerhalb dieser Hierarchien treten die Streamklassen jeweils paarweise auf; es gibt Klassen zum Einlesen (...`InputStream`) und es gibt Klassen zum Ausgeben eines Datenstroms (...`OutputStream`).

> Bei allen Datenströmen ist zu beachten, dass Laufzeitausnahmen (Exceptions) auftreten können, die abgefangen werden müssen. Dies kann innerhalb eines `try/catch` -Blocks geschehen oder bei der Deklaration der aufrufenden Methode (`throws IOException`).

Die Basisklassen der großen Stream-Hierarchien sind abstrakte Klassen, d.h. sie können nicht direkt zur Objekterzeugung genutzt werden, sondern in einem Programm müssen konkrete Unterklassen verwendet werden (siehe Kapitel Vererbung)

Um Stream-Klassen an den Verwendungszweck anzupassen, werden in der Regel geeignete Basisklassen 'übereinander gelegt' (gefiltert). Zum Beispiel kann ein Datenstrom aus einer Datei gelesen werden, indem ein `FileInputStream` über einen `InputStream` gelegt wird. Dies geschieht, indem im Konstruktoraufruf ein spezialisiertes Stream-Objekt erzeugt und an das allgemeinere Streamobjekt übergeben wird.

Für einige häufig vorkommende Einsatzzwecke (Tastatureingabe, Bildschirm-ausgabe, Dateifunktionen) werden im Folgenden Abschnitt einige Musterbeispiele gezeigt.

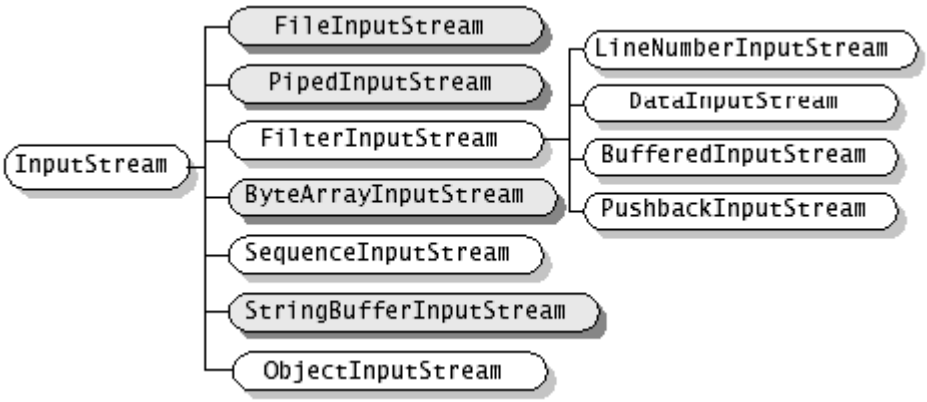
Byteformat (InputStream / OutputStream)

Spezialisierte Klassen für den Datenaustausch im *Byteformat* nutzen die Grundeinheit von 8 Bits (Byteformat), die auch zum Datenaustausch in Netzwerken und im Internet verwendet wird. In diesem Format können (bei entsprechender Vorbereitung) binäre Dateien und Java-Klassen (class-Dateien) übertragen werden.

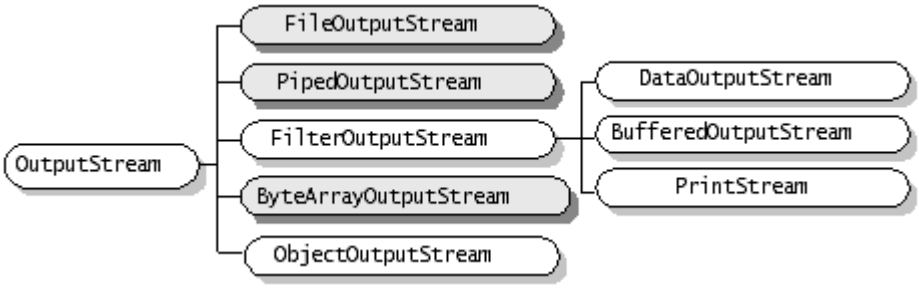
Zu den byteorientierten Datenströmen gehören auch die Klassen zur Serialisierung in Dateien (FileStreams), in Netzwerkverbindungen und auch zur Serialisierung von Objekten (ObjectStreams).

Die byteorientierten Datenströme leiten sich her von den abstrakten Basisklassen

- **`InputStream`**



- **`OutputStream`**



Beispiele für byteorientierte Datenströme

1. Tastatureingabe, Bildschirmausgabe

Die folgenden Beispielprogramme lesen Zeichen von der Tastatur ein (Input) und geben sie auf dem Bildschirm aus (Output). Hierbei können die standardmäßig stets verfügbaren Basis-Ein- und Ausgabeströme namens `System.in` und `System.out` verwendet werden. Beim Einlesen ist zu beachten, dass die häufig verwendete Leseoperation `read()` einen eingelesenen Wert immer als Integer (Ganzzahl, ASCII-Wert) zurückgibt. Vor seiner Weiterverwendung als Buchstabe muss dieser Wert in den Zeichentyp **char** gewandelt werden. Bei Zahleneingabe (vor allem als Gleitpunktzahl) muss entsprechend vorgegangen werden.

Buchstaben einlesen

```
import java.io.*;

class InputOutput1a {

    public static void main(String[] a) throws Exception {

      System.out.println("Bitte geben Sie etwas ein "+
                         "und schliessen mit <return>");

      int input=0;

        while ((input = System.in.read())!= 13) {

             System.out.println( (char)input );
        }
    }
 }
```

String einlesen

Im zweiten Beispiel wird das eingelesene Byte an einen bereits vorhandenen Leerstring angehängt.

```
import java.io.*;

class InputOutput1b {

    public static void main(String[] a) throws Exception {

      System.out.println("Bitte geben Sie einen String ein "+
                         "und schliessen mit <return>");

      String s="";
      int input=0;

      while ((input = System.in.read())!= 13) {

            s = s + (char)input;
      }
      System.out.println("Ausgabe: " + s);
    }
 }
```

Dieses Verfahren ist jedoch nicht sehr effizient, da das verwendete String-Objekt bei jedem Schleifendurchlauf neu angelegt wird (auch wenn dies am Quelltext nicht erkenntlich ist). Es empfiehlt sich vielmehr, einen StringBuffer anzulegen, an den das eingelesene Zeichen mit append angehängt wird.

```
import java.io.*;

class InputOutput1c {

    public static void main(String[] a) throws Exception {


      System.out.println("Bitte geben Sie einen String ein "+
                         "und schliessen mit <return>");

      StringBuffer i= new StringBuffer();

      int input=0;

      while ((input = System.in.read())!= 13) {

            i.append((char)input);
         }
      System.out.println("Ausgabe: " + i);
    }
 }
```

Zahlen einlesen

Beim Einlesen von Zahlen wird ähnlich wie oben verfahren. Der Inhalt des Stringbuffers muss anschliessend mit der Methode `Integer.parseInt( )` in eine berechenbare Zahl umgewandelt werden.

```
import java.io.*;

class InputOutput1d {

    public static void main(String[] a) throws Exception {

      System.out.println("Bitte geben Sie eine Ganzzahl ein "+
                         "und schliessen mit <return>");

      StringBuffer i = new StringBuffer();
      int input=0;

      while ((input = System.in.read())!= 13) {

            i.append((char)input);
      }

      int i = Integer.parseInt(i.toString());

      System.out.println("Ausgabe: " + i);
    }
 }
```

Analog dazu wird eine Dezimalzahl mit der Methode `Double.parseDouble( )` in einen berechenbaren Wert umgewandelt.

```
import java.io.*;

class InputOutput1e {

    public static void main(String[] a) throws Exception {

      InputStream in = System.in;

      System.out.println("Bitte geben Sie eine Dezimalzahl ein "+
                         "und schliessen mit <return>");

      StringBuffer i = new StringBuffer();
      int input=0;

      while ((input = in.read())!= 13) {

            i.append((char)input);
      }

      double d = Double.parseDouble(i.toString());

      System.out.println("Ausgabe: " + d);
    }
 }
```