Einführung zur Aufgabengruppe 2

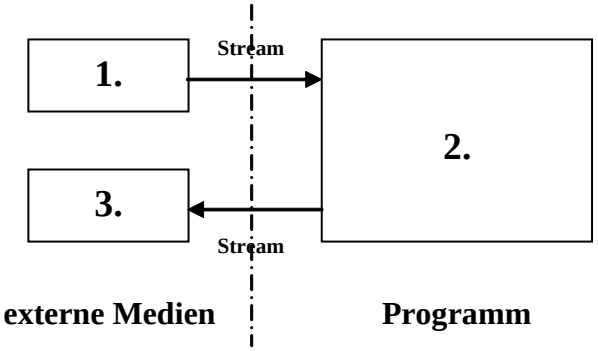
Alle Literaturhinweise beziehen sich auf die Vorlesung
Modellierung und Programmierung 1

1 Dateiverwaltung, das Stream-Konzept

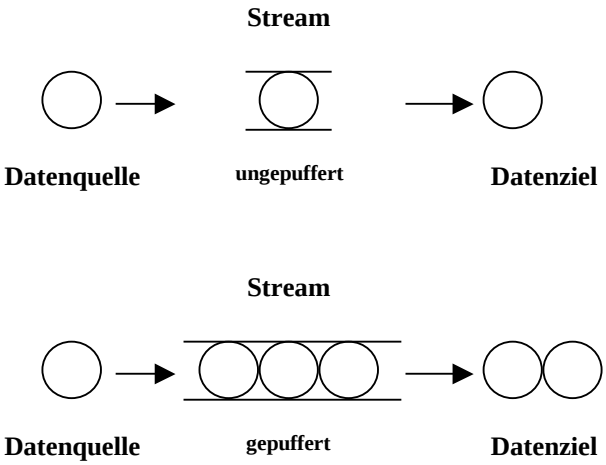
1.1 Datenströme (s. Kapitel Streams)

Jedes informationsverarbeitende System arbeitet nach dem **EVA-Prinzip**:

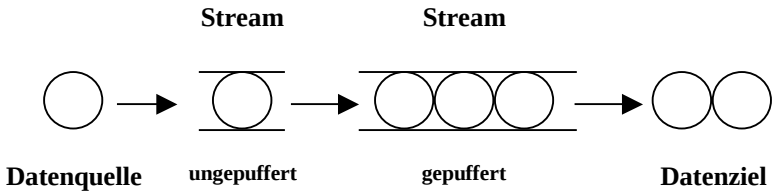
1. **E** ingabe von *Informationen*
2. **V** erarbeitung von *Informationen* entsprechend vorgegebener Arbeitsanweisungen
3. **A** usgabe von *Informationen*



Ein Datenstrom (**Stream**) transportiert Daten zwischen den Ein-, Ausgabemedien und dem Programm. Er kann *eine* Dateneinheit (**ungepuffert**) oder *mehrere* Dateneinheiten (**gepuffert**) aufnehmen.



Mehrere Datenströme können *hintereinandergeschaltet* werden (**Pipe**, **Pipeline** bzw. **Filter-Stream**).



1.1.1 Datenströme in Java, Paket `java.io.*`

Aus der Sicht der *Programme* wird unterschieden:

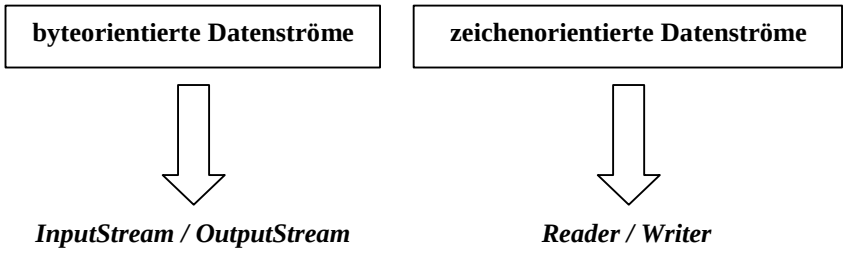
- **Eingabedatenströme** zum Einlesen von Daten
- **Ausgabedatenströme** zum Ausgeben von Daten

Aus der Sicht der *transportierten Dateneinheiten* wird unterschieden:

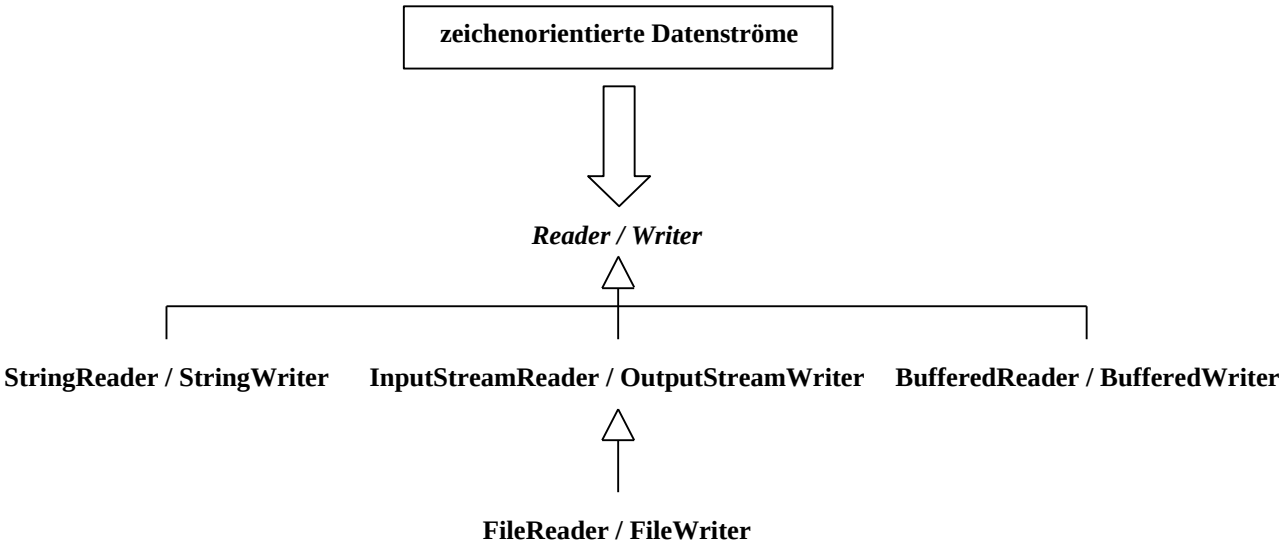
- **Byteströme** (8-Bit-Dateneiheiten, Datentype **`byte`**)
- **Zeichenströme** (16-Bit-Dateneinheiten, Datentype **`char`**)

Daraus ergeben sich vier Funktionalitäten, die im Paket **`java.io.*`** durch vier Hierarchien von Klassen abgedeckt werden:

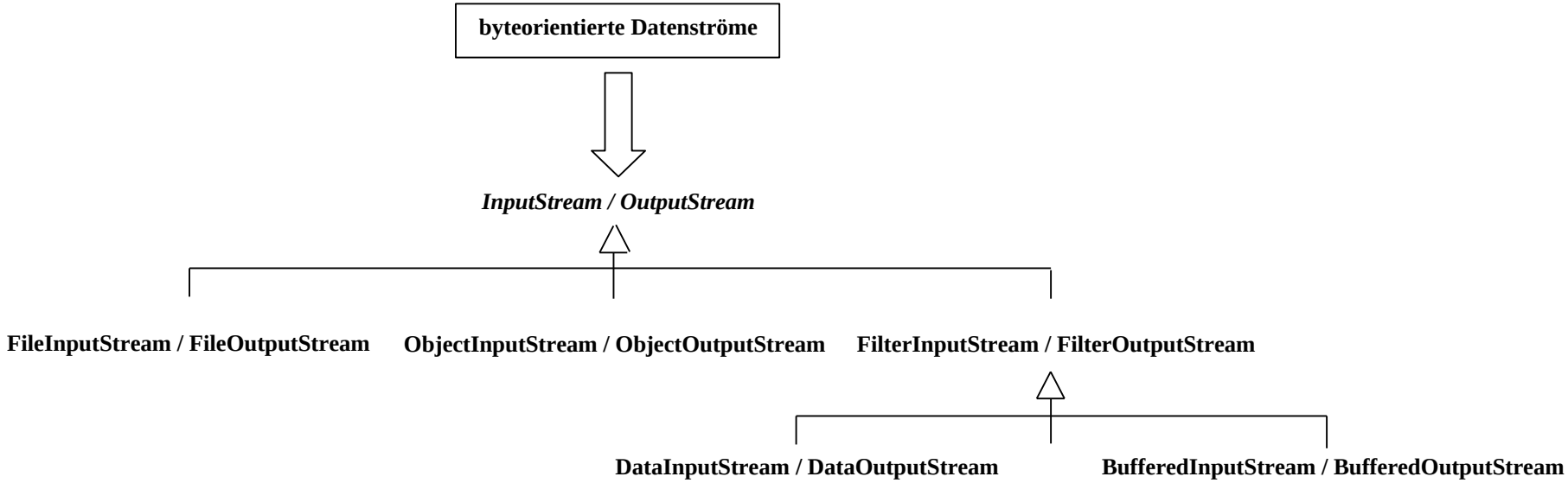
- **`InputStream`** ist die *abstrakte* Oberklasse für alle *byteorientierten* Klassen zum *Einlesen* von Daten.
- **`OutputStream`** ist die *abstrakte* Oberklasse für alle *byteorientierten* Klassen zum *Schreiben* von Daten.
- **`Reader`** ist die *abstrakte* Oberklasse für alle *zeichenorientierten* Klassen zum *Einlesen* von Daten.
- **`Writer`** ist die *abstrakte* Oberklasse für alle *zeichenorientierten* Klassen zum *Schreiben* von Daten.



1.1.2 Überblick über zeichenorientierte Datenströme

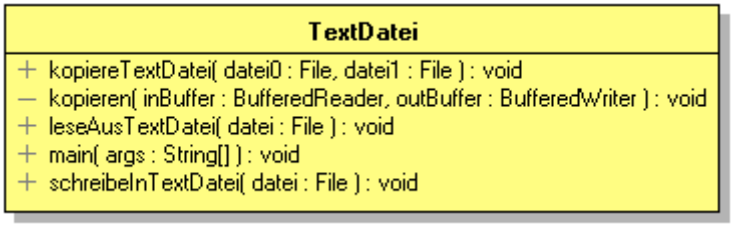


1.1.3 Überblick über byteorientierte Datenströme

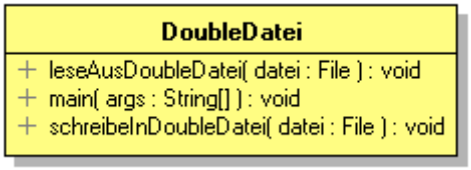


1.1.4 Beispiele für Datenströme

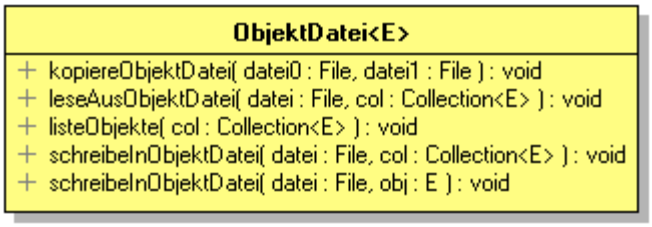
Beispiel Textdatenströme **TextDatei**



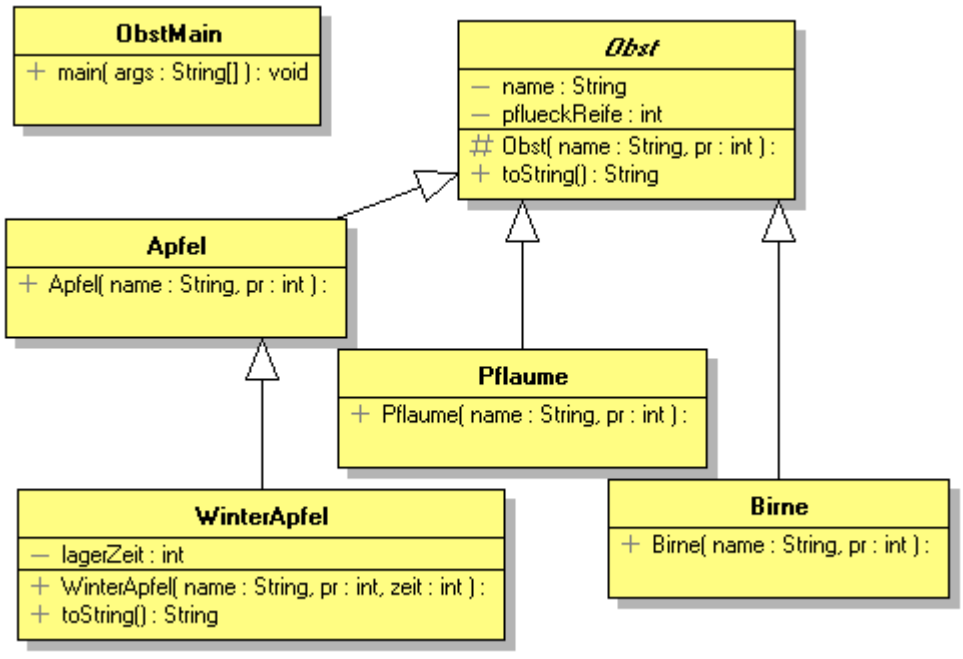
Beispiel Elementardatenströme **DoubleDatei**



Beispiel Referenzdatenströme **ObjektDatei**



Beispiel Referenzdatenströme zur Verwaltung von Obstarten **Obst**



1.2 Übersicht über häufig verwendete Datenströme

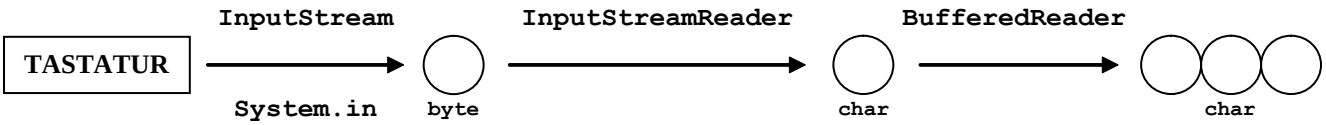
Textdaten (gepuffert)

Methoden

```
read(); readLine(); write(); newLine();
```

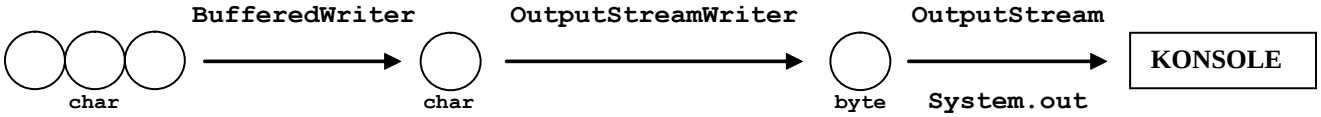
Lesen von Tastatur

```
BufferedReader inBuffer =
  new BufferedReader( new InputStreamReader( System.in));
```



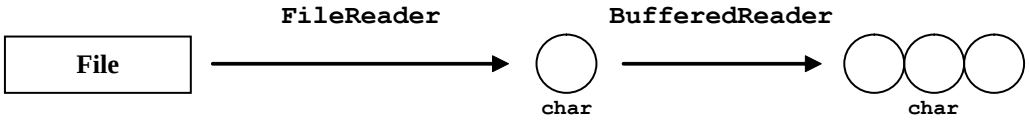
Schreiben auf Konsole

```
BufferedWriter outBuffer =
  new BufferedWriter( new OutputStreamWriter( System.out));
```



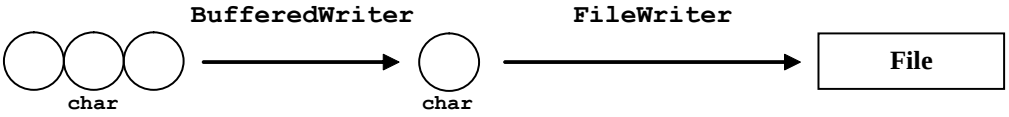
Lesen von Datei

```
BufferedReader inBuffer =
  new BufferedReader( new FileReader( File datei));
```



Schreiben auf Datei

```
BufferedWriter outBuffer =
  new BufferedWriter( new FileWriter( File datei));
```



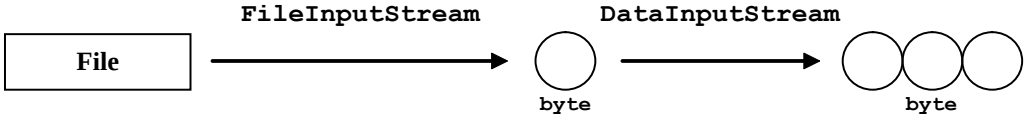
Elementardatentypen (ungepuffert)

Methoden

`read`*`Ttt`*`(); write`*`Ttt`*`();` (*`Ttt`* : Elementardatentyp *`ttt`*)

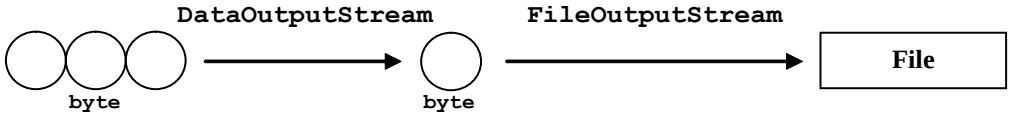
Lesen von Datei

```
DataInputStream inDatei =
                new DataInputStream( new FileInputStream( File datei));
```



Schreiben auf Datei

```
DataOutputStream outDatei =
               new DataOutputStream(new FileOutputStream( File datei));
```



Referenzdatentypen (ungepuffert)

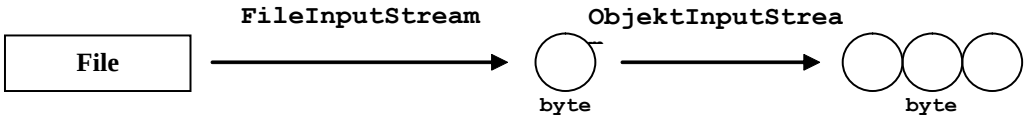
Methoden

```
readObject(); writeObject();
```

`read`*`Ttt`*`(); write`*`Ttt`*`();` (*`Ttt`* : Elementatdatentyp *`ttt`*)

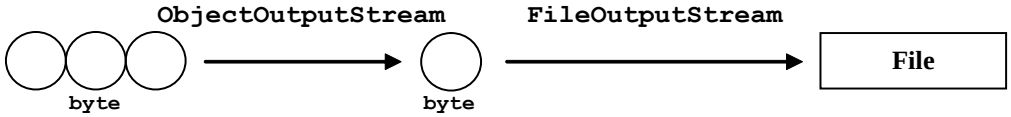
Lesen von Datei

```
ObjectInputStream inDatei =
             new ObjectInputStream( new FileInputStream( File datei));
```



Schreiben auf Datei

```
ObjectOutputStream outDatei =
           new ObjectOutputStream( new FileOutputStream( File datei));
```



1.3 Das Paket Tools

In diesem Tool wurden u.a. einige Tastatureingaben zusammengefasst (**Installation**).

Paket `Tools` **`Tools.zip`**[1]
Dokumentation des Pakets `Tools` **Dokumentation**

`Tools.IO.IOTools` Klasse für Tastatureingaberoutinen

Methoden der Klasse `IOTools`:

Lesen eines Zeichen:	`readChar()`
mit Eingabeaufforderung:	`readChar( String prompt)`
Lesen eines Wortes:	`readString()`
mit Eingabeaufforderung:	`readString( String prompt)`
Lesen einer Zeile:	`readLine()`
mit Eingabeaufforderung:	`readLine( String prompt)`
Lesen einer kurzen ganzen Zahl:	`readShort()`
mit Eingabeaufforderung:	`readShort( String prompt)`
Lesen einer ganzen Zahl:	`readInteger()`
mit Eingabeaufforderung:	`readInteger( String prompt)`
Lesen einer langen ganzen Zahl:	`readLong()`
mit Eingabeaufforderung:	`readLong( String prompt)`
Lesen einer gebrochenen Zahl:	`readFloat()`
mit Eingabeaufforderung:	`readFloat( String prompt)`
Lesen einer gebrochenen Zahl doppelter Genauigkeit:	`readDouble()`
mit Eingabeaufforderung:	`readDouble( String prompt)`
Lesen eines Wahrheitswertes	`readBoolean()`
mit Eingabeaufforderung:	`readBoolean( String prompt)`

[1] s. Dietmar Ratz, Jens Scheffler, Detlef Seese: *Grundkurs Programmieren in Java, Bd 1 - Der Einstieg in Programmierung und Objektorientierung*, Hanser Verlag.

2 Beispiel „XOR – Codierer“

2.1 Aufgabenstellung

Eines der einfachsten Verfahren der Verschlüsselungstechnik ist die Verschlüsselung mit dem binären Operator XOR (exklusiv Oder). Hierbei erfolgt die Verschlüsselung byteweise. Der Schlüssel ist eine 8 Bit lange binäre Zahl, die dem Benutzer bekannt ist. Die Besonderheit dieser Verschlüsselung besteht darin, dass sowohl für die Verschlüsselung als auch für die Entschlüsselung ein und dieselbe Methode verwendet werden kann.

Beispiel

Encodieren mit Schlüssel 16

A	0100 0001 ^ 0001 0000 = 0101 0001	Q
u	0111 0101 ^ 0001 0000 = 0110 0101	e
t	0111 0100 ^ 0001 0000 = 0110 0100	d
o	0110 1111 ^ 0001 0000 = 0111 1111	•

Decodieren mit Schlüssel 16

Q	0101 0001 ^ 0001 0000 = 0100 0001	A
e	0110 0101 ^ 0001 0000 = 0111 0101	u
d	0110 0100 ^ 0001 0000 = 0111 0100	t
•	0111 1111 ^ 0001 0000 = 0110 1111	o

ASCII - American Standard Code for Information Interchange[2]

dec	hex	char	dec	hex	char	dec	hex	char	dec	hex	char
000	00	NUL	032	20		064	40	@	096	60	`
001	01	SOH	033	21	!	065	41	**A**	097	61	a
002	02	STX	034	22	"	066	42	B	098	62	b
003	03	ETX	035	23	#	067	43	C	099	63	c
004	04	EOT	036	24	$	068	44	D	100	64	***d***
005	05	ENQ	037	25	%	069	45	E	101	65	***e***
006	06	ACK	038	26	&	070	46	F	102	66	f
						...					
014	0E	SO	046	2E	.	078	4E	N	110	6E	n
015	0F	SI	047	2F	/	079	4F	O	111	6F	**o**
016	10	DLE	048	30	0	080	50	P	112	70	p
017	11	DC1	049	31	1	081	51	***Q***	113	71	q
018	12	DC2	050	32	2	082	52	R	114	72	r
019	13	DC3	051	33	3	083	53	S	115	73	s
020	14	DC$	052	34	4	084	54	T	116	74	**t**
021	15	NAK	053	35	5	085	55	U	117	75	**u**
022	16	SYN	054	36	6	086	56	V	118	76	v
						...					
030	1E	RS	062	3E	>	094	5E	^	126	7E	~
031	1F	US	063	3F	?	095	5F	_	127	7F	□

[2] http://www.medizin.uni-koeln.de/kai/imsie/kurse/html/asciitab.html

Daten und Datenströme

Die originalen und die verschlüsselten Daten werden in Dateien verwaltet, d.h. das Programm muss die Fähigkeit haben, Dateien zu lesen und zu schreiben. Da die Verschlüsselung mit XOR byteweise erfolgt, ist es sinnvoll, die Daten als Folgen von Bytes zu betrachten. Damit ist der ungepufferter byteweise Zugriff folgerichtig. Unter der Voraussetzung, dass die Dateinamen bekannt sind, kann das Lesen und Schreiben durch die nachstehenden Anweisungen geschehen:

```
byte[] bytes = new byte
               [(int)(( new File( dateiName)).length())];

FileInputStream in = new FileInputStream( dateiName);

in.read( bytes);
in.close();
```

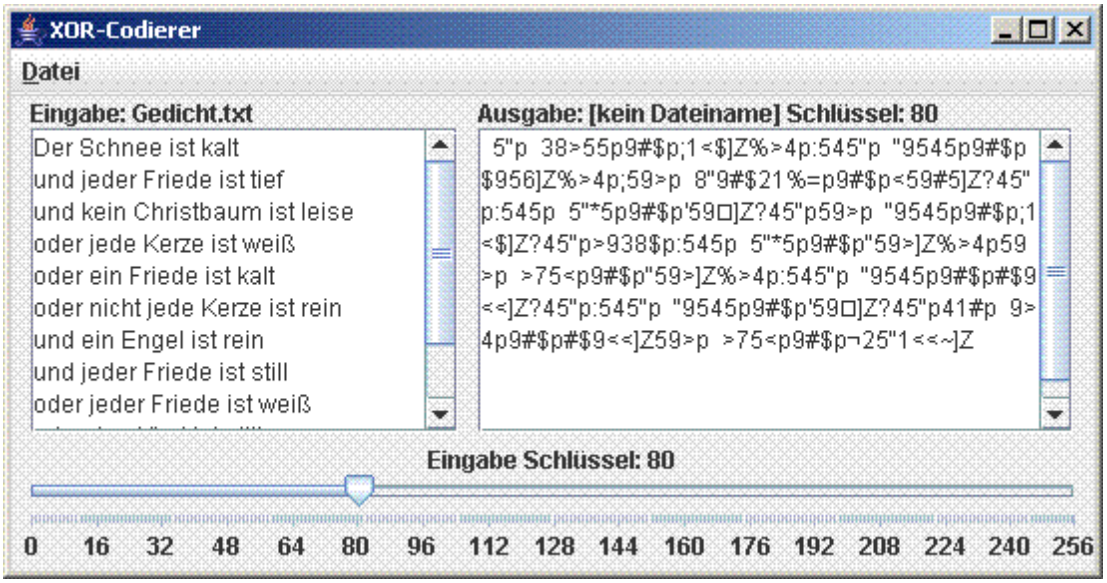
bzw.

```
FileOutputStream out = new FileOutputStream( dateiName);

if( bytes != null) out.write( bytes);
out.close();
```

Modellierung und Programmierung **Klassendiagramm**, **Dokumentation**, ***XOR.zip***

2.2 Zusatz: Benutzerschnittstelle zum XOR - Codierer



Modellierung und Programmierung mit MVC-Architektur

Klassendiagramm, **Dokumentation**, ***XOR_MVC.zip***