07-Input/Output-1

Objektorientierte Programmierung | Matthias Tichy

Lernziele

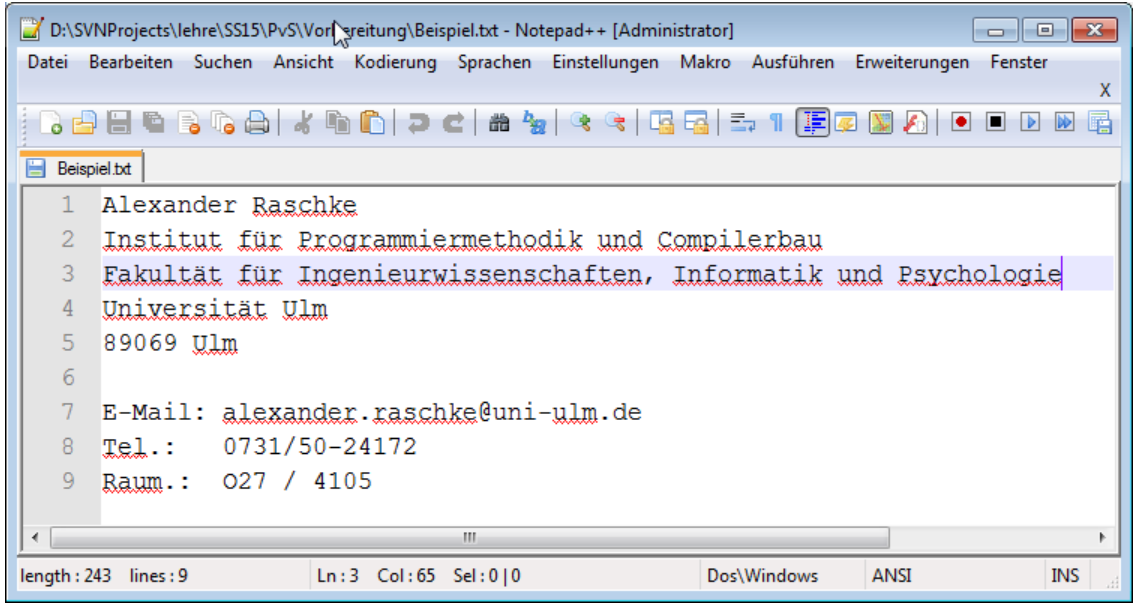
- Dateiformate
- Input/Output in Java
  - Streams API
  - Decorator-Pattern

Dateiformate

Dateiinhalt

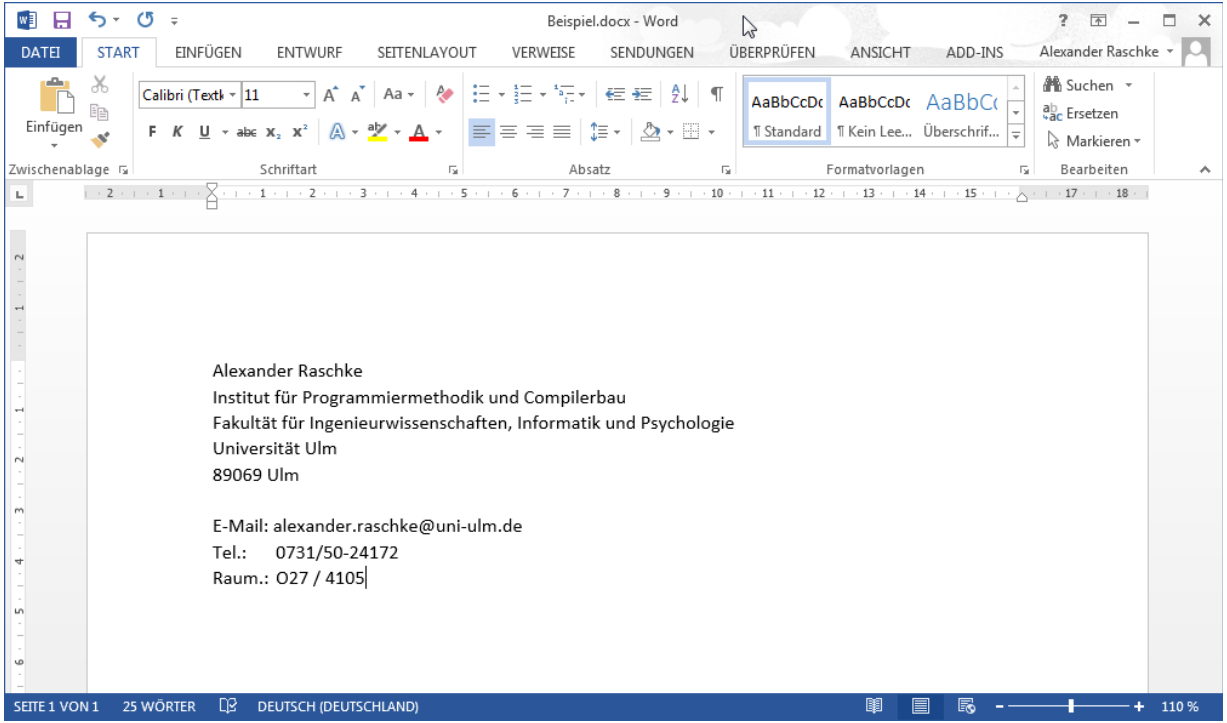
```
01000001011011000110010101111000011000010110111001100100011001010111001000100000010100100110000
01011100110110001101101000011010110110010100001101000010100100100101101110011100110111010001100
10010111010001110101011110100001000000110011011111110001110010001000000101000001110010011011110
10011101110010011000010110110101101101011010010110010101110010011011010110010101110100011010000
01101111011001000110100101101011001000000111010101101110011001000010000001000001101101111011011
01011100000110100101101100011001010111001001100010011000010111010100001101000010100100011001100
00010110101101110101011011000111010011100100011101000010000001100110111111000111001000100000001
00100101101110011001110110010101101110011010010110010101110101011100100111011101101001011100011
01110011011001010110111001110011011000110110101000011000001011001100111010001100101011011100010111
00001000000100100101101110011001100110111101110010011011010110000010111010001101001011010110010
00000111010101101110011001000010000001010000011100110111100101100011011010000110111101101100001
10111101100111011010010110010100001101000010100101010101101110011010010111011001100101011100101
01110011011010010111010011100100011101000010000001010101011011000110110100001101000010100011110
00001110010011000000110110001110010010000001010101011011000110110100001101000010100000110100000
10100100010100101101010011010110000010110100101101100000111010001000000110000101101100011001010
11100001100001011011100110010001100101011100100010111001110010011000010111001101100011011010000
01101011011001010100000001110101011011100110100100101110101110101011011000110110100101110011001
00011001010000110100001010010101000110010101101100001011100011101000100000000010010011000000011
01110011001100110001001011110011010100110000001011010011001000110100001100010011011100110010000
00110100001010010100100110000010111010101101101001011100011101000100000000010010100111100110010
00110111001000000010111100100000001101000011000100110000000110101
```

Dateiinhalt



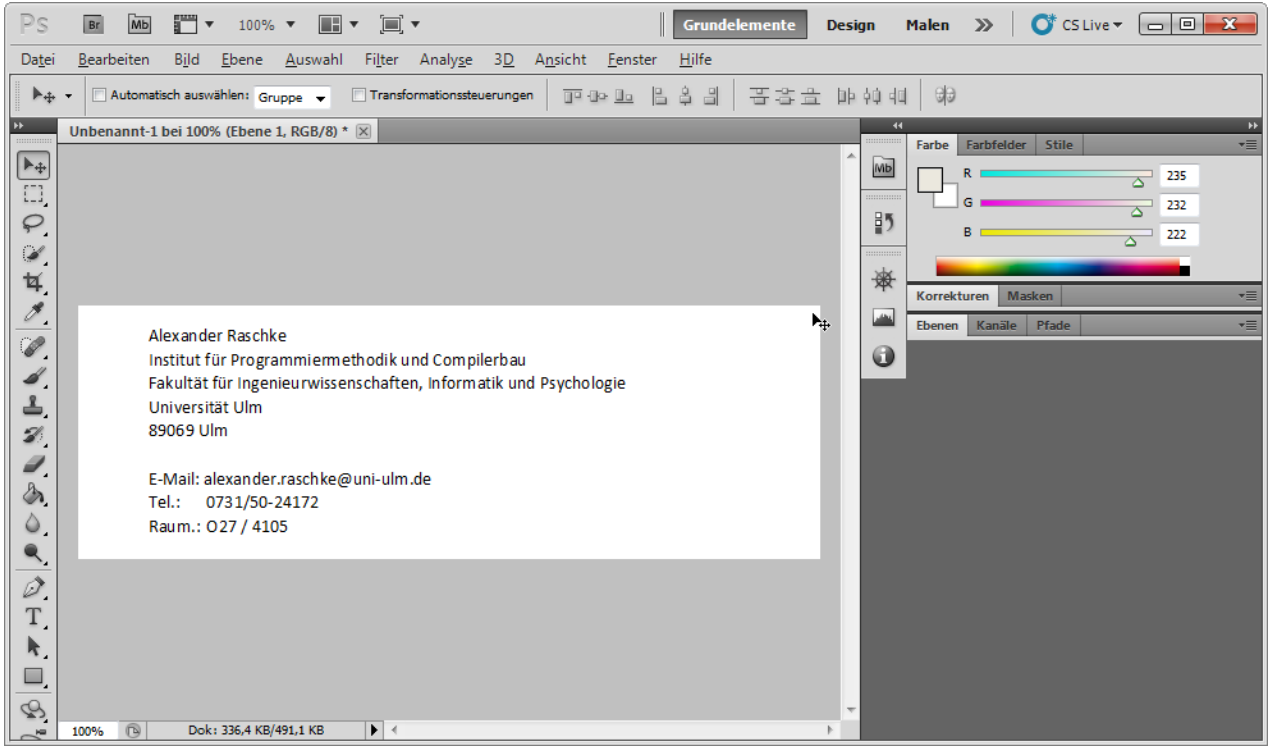
Dateiinhalt

Dateiinhalt



Dateiinhalt

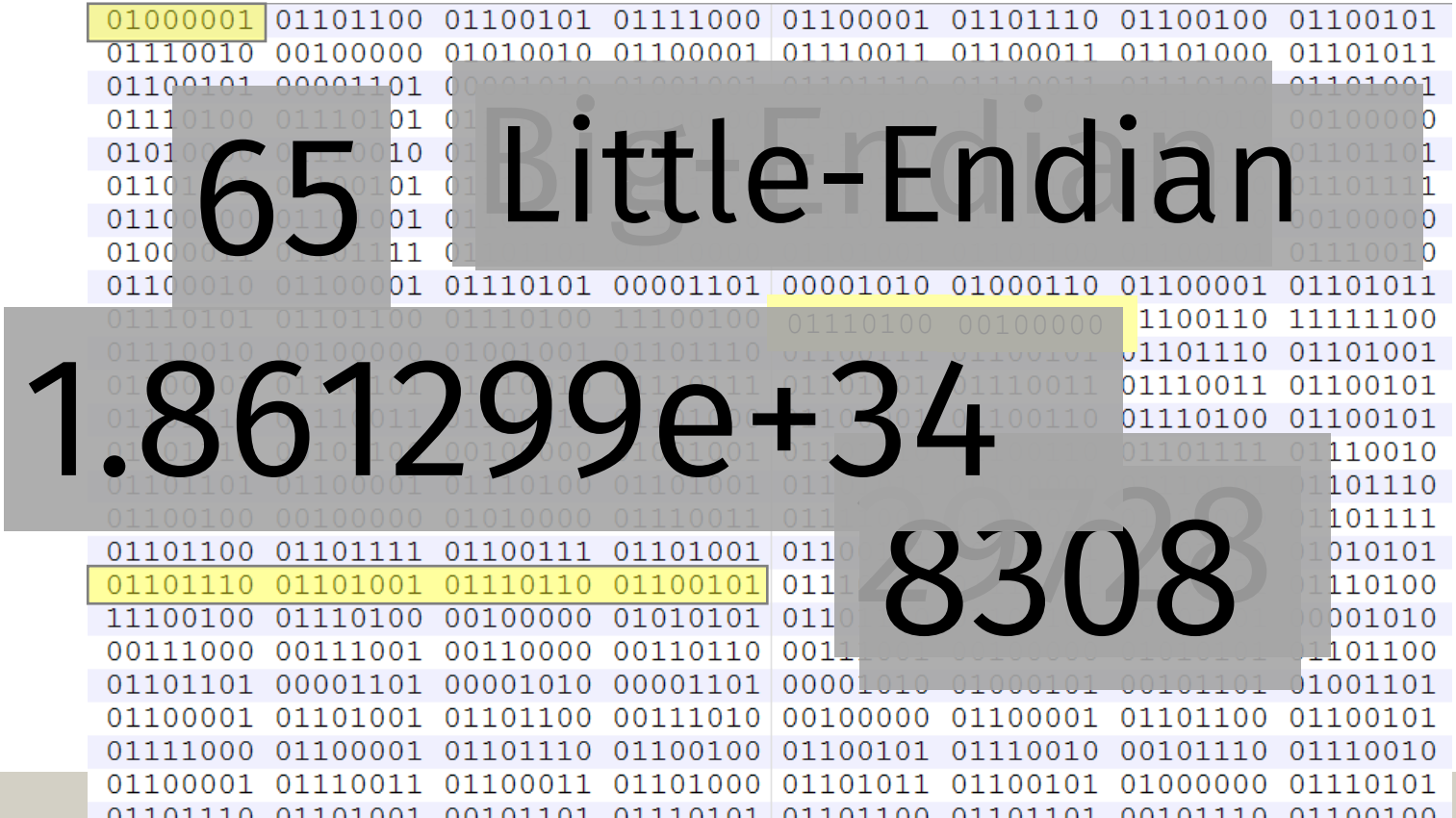
Dateiinhalt



Dateiinhalt

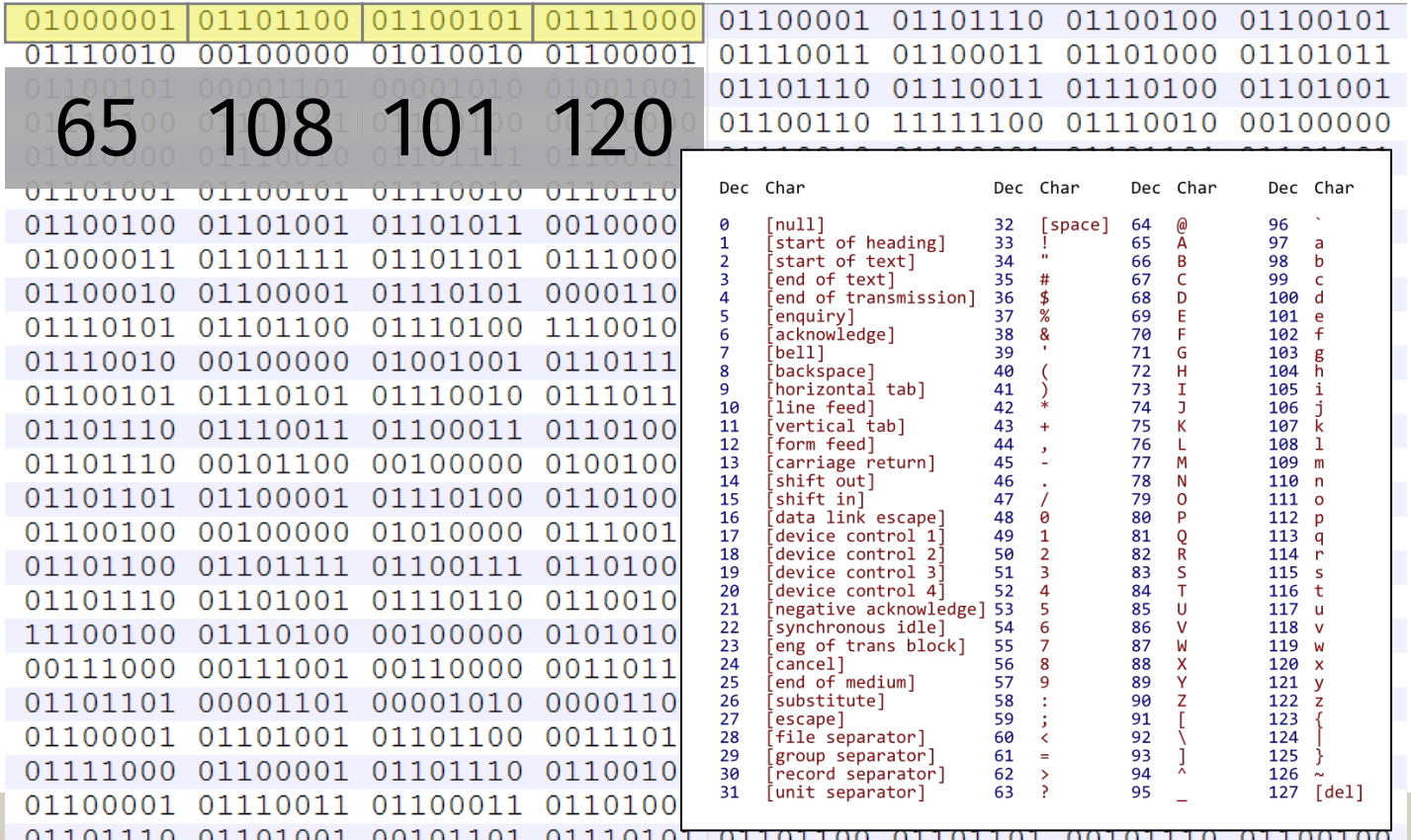
```
01000001011011000110010101111000011000010110111001100100011001010111001000100000010100100110000
01011100110110001101101000011010110110010100001101000010100100100101101110011100110111010001100
10010111010001110101011110100001000000110011011111110001110010001000000101000001110010011011110
10011101110010011000010110110101101101011010010110010101110010011011010110010101110100011010000
01101111011001000110100101101011001000000111010101101110011001000010000001000001101101111011011
01011100000110100101101100011001010111001001100010011000010111010100001101000010100100011001100
00010110101101110101011011000111010011100100011101000010000001100110111111000111001000100000001
00100101101110011001110110010101101110011010010110010101110101011100100111011101101001011100011
01110011011001010110111001110011011000110110101000011000001011001100111010001100101011011100010111
00001000000100100101101110011001100110111101110010011011010110000010111010001101001011010110010
00000111010101101110011001000010000001010000011100110111100101100011011010000110111101101100001
10111101100111011010010110010100001101000010100101010101101110011010010111011001100101011100101
01110011011010010111010011100100011101000010000001010101011011000110110100001101000010100011110
00001110010011000000110110001110010010000001010101011011000110110100001101000010100000110100000
10100100010100101101010011010110000010110100101101100000111010001000000110000101101100011001010
11100001100001011011100110010001100101011100100010111001110010011000010111001101100011011010000
01101011011001010100000001110101011011100110100100101110101110101011011000110110100101110011001
00011001010000110100001010010101000110010101101100001011100011101000100000000010010011000000011
01110011001100110001001011110011010100110000001011010011001000110100001100010011011100110010000
00110100001010010100100110000010111010101101101001011100011101000100000000010010100111100110010
00110111001000000010111100100000001101000011000100110000000110101
```

Dateiinhalt



Zeichentabellen

Text?



Dec	Char	Dec	Char	Dec	Char	Dec	Char
0	[null]	32	[space]	64	@	96	`
1	[start of heading]	33	!	65	A	97	a
2	[start of text]	34	"	66	B	98	b
3	[end of text]	35	#	67	C	99	c
4	[end of transmission]	36	$	68	D	100	d
5	[enquiry]	37	%	69	E	101	e
6	[acknowledge]	38	&	70	F	102	f
7	[bell]	39	'	71	G	103	g
8	[backspace]	40	(	72	H	104	h
9	[horizontal tab]	41	)	73	I	105	i
10	[line feed]	42	*	74	J	106	j
11	[vertical tab]	43	+	75	K	107	k
12	[form feed]	44	,	76	L	108	l
13	[carriage return]	45	-	77	M	109	m
14	[shift out]	46	.	78	N	110	n
15	[shift in]	47	/	79	O	111	o
16	[data link escape]	48	0	80	P	112	p
17	[device control 1]	49	1	81	Q	113	q
18	[device control 2]	50	2	82	R	114	r
19	[device control 3]	51	3	83	S	115	s
20	[device control 4]	52	4	84	T	116	t
21	[negative acknowledge]	53	5	85	U	117	u
22	[synchronous idle]	54	6	86	V	118	v
23	[eng of trans block]	55	7	87	W	119	w
24	[cancel]	56	8	88	X	120	x
25	[end of medium]	57	9	89	Y	121	y
26	[substitute]	58	:	90	Z	122	z
27	[escape]	59	;	91	[	123	{
28	[file separator]	60	<	92	\	124	|
29	[group separator]	61	=	93	]	125	}
30	[record separator]	62	>	94	^	126	~
31	[unit separator]	63	?	95	_	127	[del]

ASCII

- American Standard Code for Information Interchange

- erste Version 1963
- Version von 1968 bis heute gültig

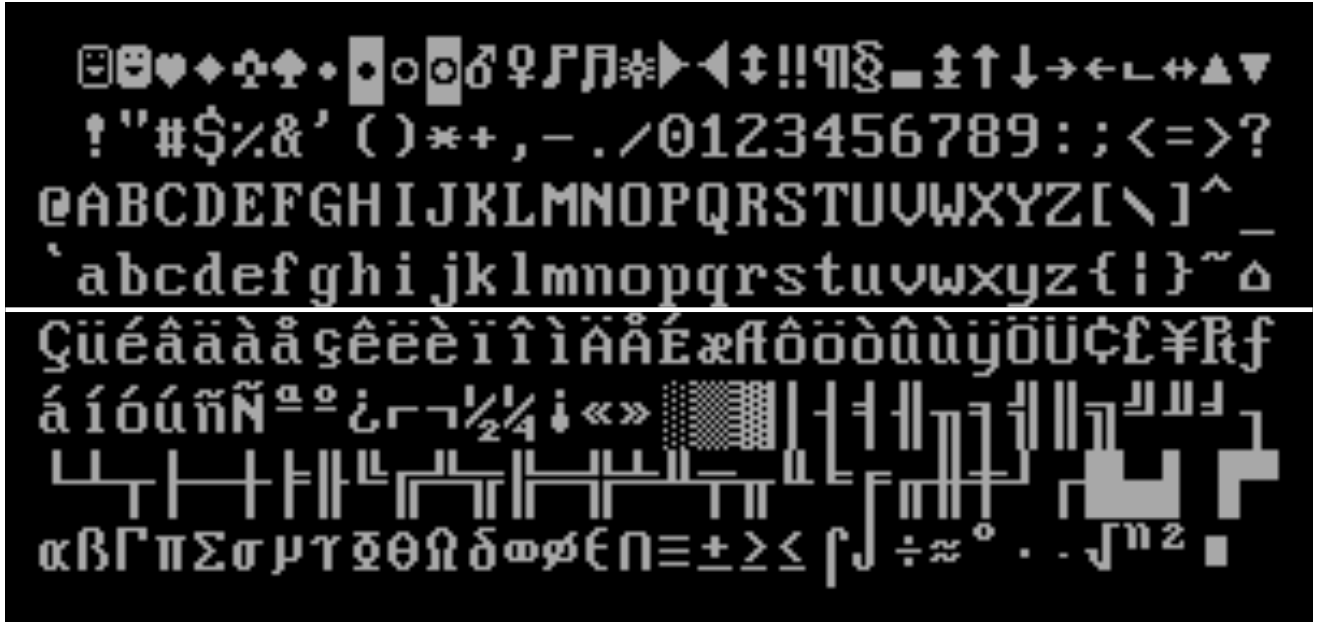
- 7-bit Zeichenkodierung

- ursprünglich für Fernschreiber

ASCII

Dec	Char	Dec	Char	Dec	Char	Dec	Char
0	[null]	32	[space]	64	@	96	`
1	[start of heading]	33	!	65	A	97	a
2	[start of text]	34	"	66	B	98	b
3	[end of text]	35	#	67	C	99	c
4	[end of transmission]	36	$	68	D	100	d
5	[enquiry]	37	%	69	E	101	e
6	[acknowledge]	38	&	70	F	102	f
7	[bell]	39	'	71	G	103	g
8	[backspace]	40	(	72	H	104	h
9	[horizontal tab]	41	)	73	I	105	i
10	[line feed]	42	*	74	J	106	j
11	[vertical tab]	43	+	75	K	107	k
12	[form feed]	44	,	76	L	108	l
13	[carriage return]	45	-	77	M	109	m
14	[shift out]	46	.	78	N	110	n
15	[shift in]	47	/	79	O	111	o
16	[data link escape]	48	0	80	P	112	p
17	[device control 1]	49	1	81	Q	113	q
18	[device control 2]	50	2	82	R	114	r
19	[device control 3]	51	3	83	S	115	s
20	[device control 4]	52	4	84	T	116	t
21	[negative acknowledge]	53	5	85	U	117	u
22	[synchronous idle]	54	6	86	V	118	v
23	[eng of trans block]	55	7	87	W	119	w
24	[cancel]	56	8	88	X	120	x
25	[end of medium]	57	9	89	Y	121	y
26	[substitute]	58	:	90	Z	122	z
27	[escape]	59	;	91	[	123	{
28	[file separator]	60	<	92	\	124	|
29	[group separator]	61	=	93	]	125	}
30	[record separator]	62	>	94	^	126	~
31	[unit separator]	63	?	95	_	127	[del]

Codepage 437



Codepage 737

	…0	…1	…2	…3	…4	…5	…6	…7	…8	…9	…A	…B	…C	…D	…E	…F
8…	A 391	B 392	Γ 393	Δ 394	E 395	Z 396	H 397	Θ 398	I 399	K 39A	Λ 39B	M 39C	N 39D	Ξ 39E	O 39F	Π 3A0
9…	P 3A1	Σ 3A3	T 3A4	Y 3A5	Φ 3A6	X 3A7	Ψ 3A8	Ω 3A9	α 3B1	β 3B2	γ 3B3	δ 3B4	ε 3B5	ζ 3B6	η 3B7	θ 3B8
A…	ι 3B9	κ 3BA	λ 3BB	μ 3BC	ν 3BD	ξ 3BE	ο 3BF	π 3C0	ρ 3C1	σ 3C3	ς 3C2	τ 3C4	υ 3C5	φ 3C6	χ 3C7	ψ 3C8
B…	░ 2591	▒ 2592	▓ 2593	│ 2502	┤ 2524	╡ 2561	╢ 2562	╖ 2556	╕ 2555	╣ 2563	║ 2551	╗ 2557	╝ 255D	╜ 255C	╛ 255B	┐ 2510
C…	└ 2514	┴ 2534	┬ 252C	├ 251C	─ 2500	┼ 253C	╞ 255E	╟ 255F	╚ 255A	╔ 2554	╩ 2569	╦ 2566	╠ 2560	═ 2550	╬ 256C	╧ 2567
D…	╨ 2568	╤ 2564	╥ 2565	╙ 2559	╘ 2558	╒ 2552	╓ 2553	╫ 256B	╪ 256A	┘ 2518	┌ 250C	█ 2588	▄ 2584	▌ 258C	▐ 2590	▀ 2580
E…	ω 3C9	ά 3AC	έ 3AD	ή 3AE	ϊ 3CA	ί 3AF	ό 3CC	ύ 3CD	ϋ 3CB	ώ 3CE	Ά 386	Έ 388	Ή 389	Ί 38A	Ό 38C	Ύ 38E
F…	Ώ 38F	± B1	≥ 2265	≤ 2264	Ϊ 3AA	Ϋ 3AB	÷ F7	≈ 2248	° B0	∙ 2219	· B7	√ 221A	ⁿ 207F	² B2	■ 25A0	NBSP A0

ISO-Norm 8859

- Versuch der Vereinheitlichung der "Wildwuchse"
- Unterschiedliche Codetabellen (-0 bis -15)
- teilweise wieder Rückportiert auf Codepages (MSDOS-850, Windows-1252)

Unicode

- 1. Version: 1991

- ursprünglich: maximal 65.536 Zeichen (= 16 bit)
- ab 1996: maximal 1.114.112 Zeichen (21 bit)
- eingeteilt in 17 Planes mit jeweils 65536 Zeichen
- Version 15.0: 149,186 Zeichen

- unicode.org, unicode-table.com

UTF-16

- kodiert erste Plane 00 direkt
- weitere Planes durch zwei 16-bit Blöcke
- Beispiel: Violinschlüssel mit Unicode U+1D11E

```
    01        D1        1E
00000001 11010001 00011110        -10000x0
00000000 11010001 00011110        2 mal 10bit
0000 0000 1101 0001 0001 1110      2 mal 16bit bilden
1101 1000 0011 0100 1101 1101 0001 1110
   D8        34        DD        1E
```

- Standard für interne Zeichenrepräsentation in Java und .NET

UTF-8

- Universal Character Set Transformation Format
- aktuellste Definition von 2003: ISO 10646-1

- Kodierung:
  - UTF-8 bis zu 7-bit ist identisch mit ASCII
  - bei Zeichen mit höheren Werten ist das höchste Bit = 1
  - Folgebytes werden mit 10 eingeleitet
- Wertebereiche (Hexadezimal):

```
0000 0080 – 0000 07FF: 110xxxxx 10xxxxxx
0000 0800 – 0000 FFFF: 1110xxxx 10xxxxxx 10xxxxxx
0000 8000 – 0010 FFFF: 11110xxx 10xxxxxx 10xxxxxx 10xxxxxx
```

UTF-8

- Eigenschaften:
  - meist kompaktere Darstellung als UTF-16
  - kompatibel mit ASCII
  - Folge-Bytes niemals ASCII (daher gleich erkennbar)
  - Sortierbarkeit bleibt erhalten
  - einfache Kodierungsfunktion
- Standard im Web
- Auch auf vielen Systemen mittlerweile Standard

UTF-8

- Beispiel: Violinschlüssel mit Unicode U+1D11E

```
     01        D1        1E
00000001 11010001 00011110        benötigte Bits: 21
```

Kodierung in 4 Multibytes:

```
11110000 10011101 10000100 10011110
```

- Typische Darstellungsprobleme beim Lesen von UTF-8 Dateien mit Windows-1252
  - UniversitÃ¤t → Universität
  - HÃ¶he → Höhe

Text- vs. Binärformat

Dateiformate

eine Frage der Interpretation des Inhalts

- Textdatei
  - Codierung
  - Zeilenende
- Binärdatei
  - Endianess
  - Formate ➔ Interpretation des Inhalts
  - Dateiformatspezifikation
    - welches Byte steht an welcher Stelle und hat welche Bedeutung?
    - Bsp.: PNG: http://www.libpng.org/pub/png/spec/1.2/

Zeilenende

- ASCII-Code 13 = <Carriage Return> = <CR> = \r
- ASCII-Code 10 = <Line Feed> = <LF> = \n

- Ursprung:

Formate Vor-/Nachteile

Textformat	Binärformat
Kleine Fehler evtl. nicht schlimm	Ein fehlerhaftes Byte zerstört zugrunde-liegendes Format
Verarbeitung bedarf Vorverarbeitung (Parsen)	Werte liegen direkt vor
Menschenlesbar	Maschinenlesbar

Zugriff auf Dateien

- variable Längen
  - Längenangabe am Anfang
  - Schlusszeichen am Ende
  - Position eines Datums in einer Datei kann nicht vorhergesagt werden (➔ nur **sequentieller** Zugriff möglich)
  - bei Änderung, muss komplette Datei neu geschrieben werden
- Blöcke mit festen Längen
  - evtl. Platzverschwendung wenn weniger benötigt
  - direkter, schneller, **wahlfreier** Zugriff möglich (lesend und schreibend)

Input/Output

Datenflüsse in Java

- Datenflüsse von und nach Programmen als **Stream** bezeichnet

- Generische Interfaces:
  - `InputStream`
    `read, skip, reset, close`
  - `OutputStream`
    `write, flush, close`

Zugriff auf Dateien in Java

- Byte Streams:
  binäre Rohdaten
- File Streams:
  Lese- / Schreiboperationen auf Dateien
- Buffered Streams:
  Puffert die Schreib- und Leseoperationen
- Character Streams:
  Textdateien (inkl. Encoding)
- Standard-I/O :
  Standardein- und -ausgabe von der Konsole
- Data Streams:
  binäre Datenströme von Basisdatentypen und Strings
- Object Streams:
  binäre Datenströme von beliebigen Objekten

Achtung

Demo

- de.uni_ulm.sp.oop.io
  - CopyBytes

Exceptions

Beispiel

CopyBytes

```
in = new FileInputStream("test.pdf");
out = new FileOutputStream("out.pdf");
int c;
// read as long there is something to read
// (end of file (EOF) is marked by -1)
c = in.read();
while (c != -1) {
        // and write immediately the byte read
        out.write(c);
        c = in.read();
}
in.close();
out.close();
```

Was kann alles schief gehen?

Exceptions

CopyBytes

```
try {
        in = new FileInputStream("test.pdf");
        out = new FileOutputStream("out.pdf");
        int c;
        c = in.read();
        while (c != -1) {
                out.write(c);
                c = in.read();
        }
} catch (IOException ex) {
        logger.error("Problem during copying", ex);
} finally {
        if (in != null) { in.close();}
        if (out != null) {out.close();}
}
```

Try Block
- Happy Path

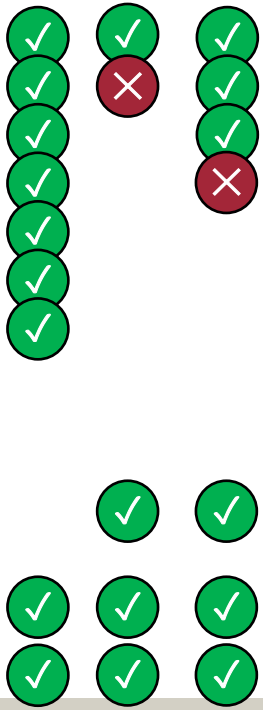
Catch Block
- Fehlerbehandlung

Finally Block
- Passiert immer

Fehler an verschiedenen Stellen

CopyBytes

```
try {
        in = new FileInputStream("test.pdf");
        out = new FileOutputStream("out.pdf");
        int c;
        c = in.read();
        while (c != -1) {
                out.write(c);
                c = in.read();
        }
} catch (IOException ex) {
        logger.error("Problem during copying", ex);
} finally {
        if (in != null) { in.close();}
        if (out != null) {out.close();}
}
```



Demo

- de.uni_ulm.sp.oop.io
  - CopyBytes

Anmerkungen

- Schließen vergessen?
- try-with-resources-Statement ab Java 7:

```
try (FileInputStream in = new FileInputStream("beispiel.txt");
     FileOutputStream out = new FileOutputStream("out.txt")) {
    int c;
    while ((c = in.read()) != -1) {
       out.write(c);
    }
}
```

- deklarierte Variablen nur im try-Block bekannt, nicht in catch- oder finally-Blöcken
- Voraussetzung: AutoCloseable-Interface

Demo

- de.uni_ulm.sp.oop.io
  - BufferedCopy

Throws declaration

CopyBytesRefactored

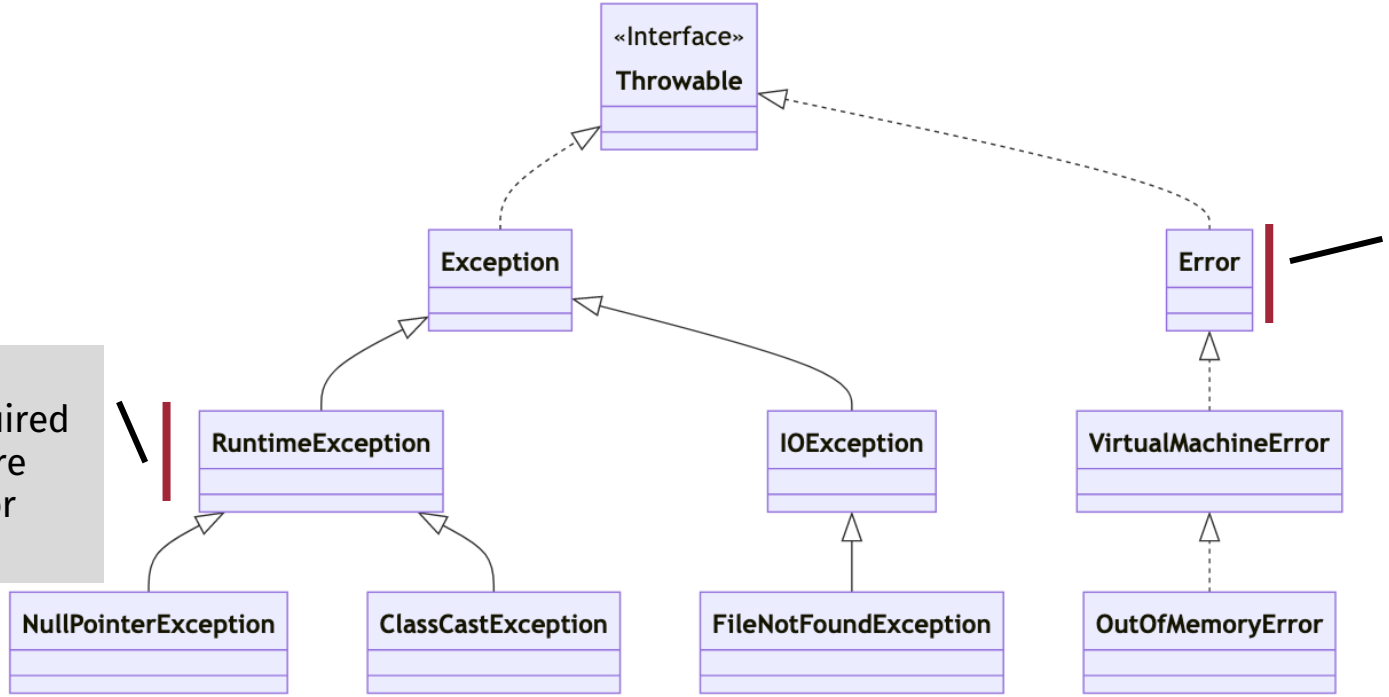
Kein catch → throws declaration

```
private static void copyFile() throws IOException {
        try (InputStream in = new BufferedInputStream(new FileInputStream("test.pdf"));
        OutputStream out = new BufferedOutputStream(new FileOutputStream("out.pdf"))) {
                int c;
                c = in.read();
                while (c != -1) {
                        out.write(c);
                        c = in.read();
                }
        }
}
```

Hier catch nötig

```
public static void main(String[] args) {
        try {
                copyFile();
        } catch (IOException ex) {
                logger.error("Problem ...", ex);
        }
        System.out.println("Fertig!");
}
```

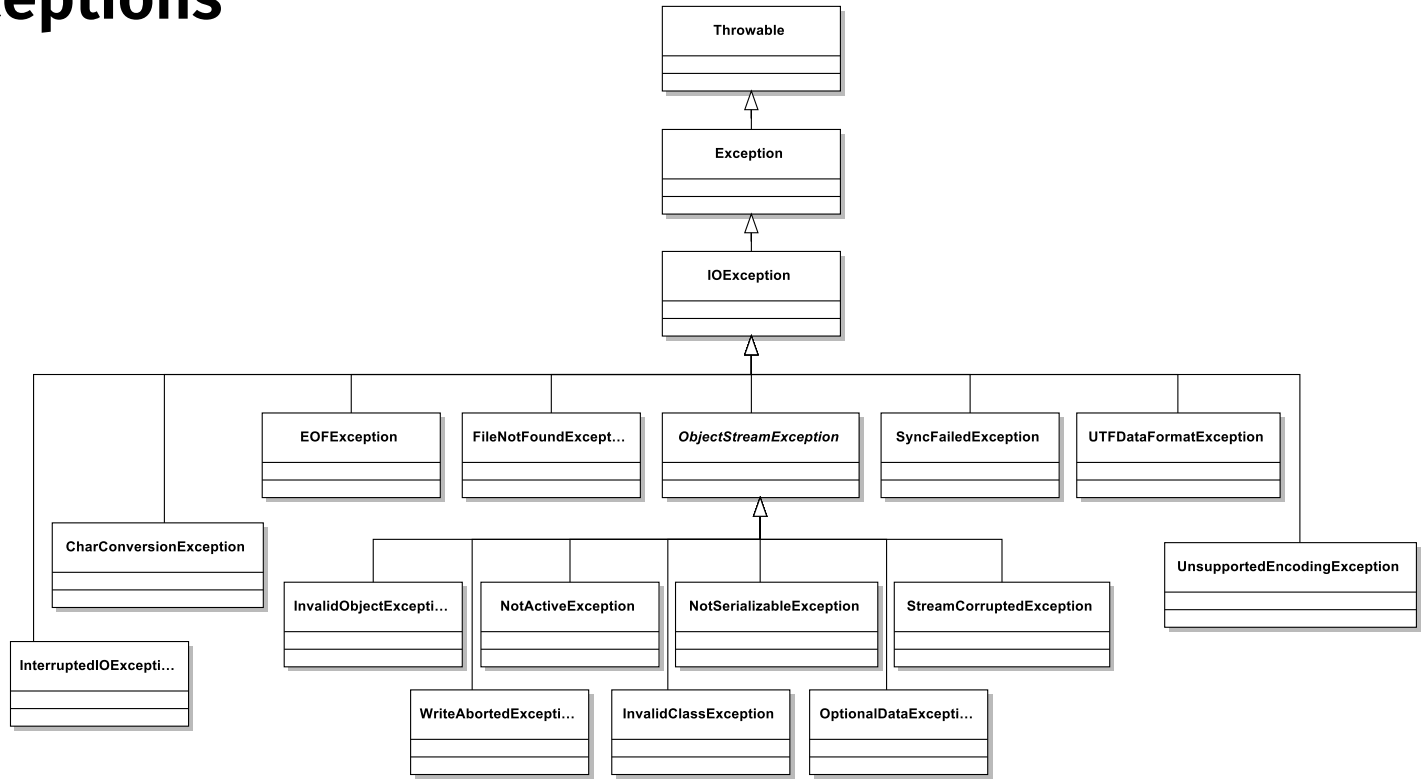
Throwable / Exception / Error



Should Not Catch

Unchecked
- Not required to declare throws or catch

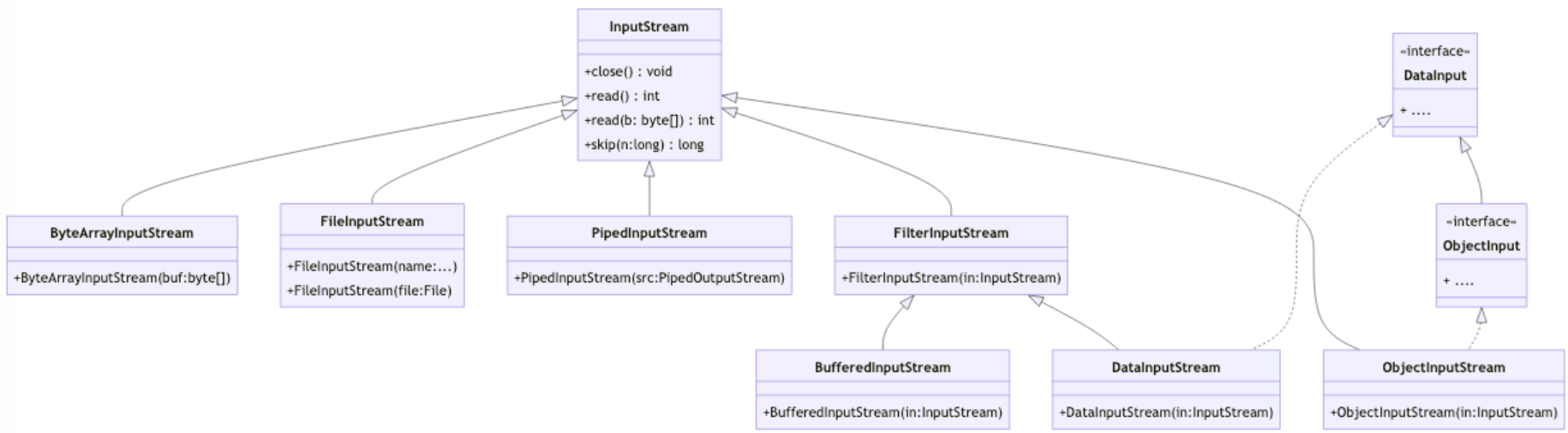
Exceptions


Throwable
Exception
IOException
EOFException
FileNotFoundExcept…
ObjectStreamException
SyncFailedException
UTFDataFormatException
CharConversionException
InvalidObjectExcepti…
NotActiveException
NotSerializableException
StreamCorruptedException
UnsupportedEncodingException
InterruptedIOExcepti…
WriteAbortedExcepti…
InvalidClassException
OptionalDataExcepti…

Input / Output

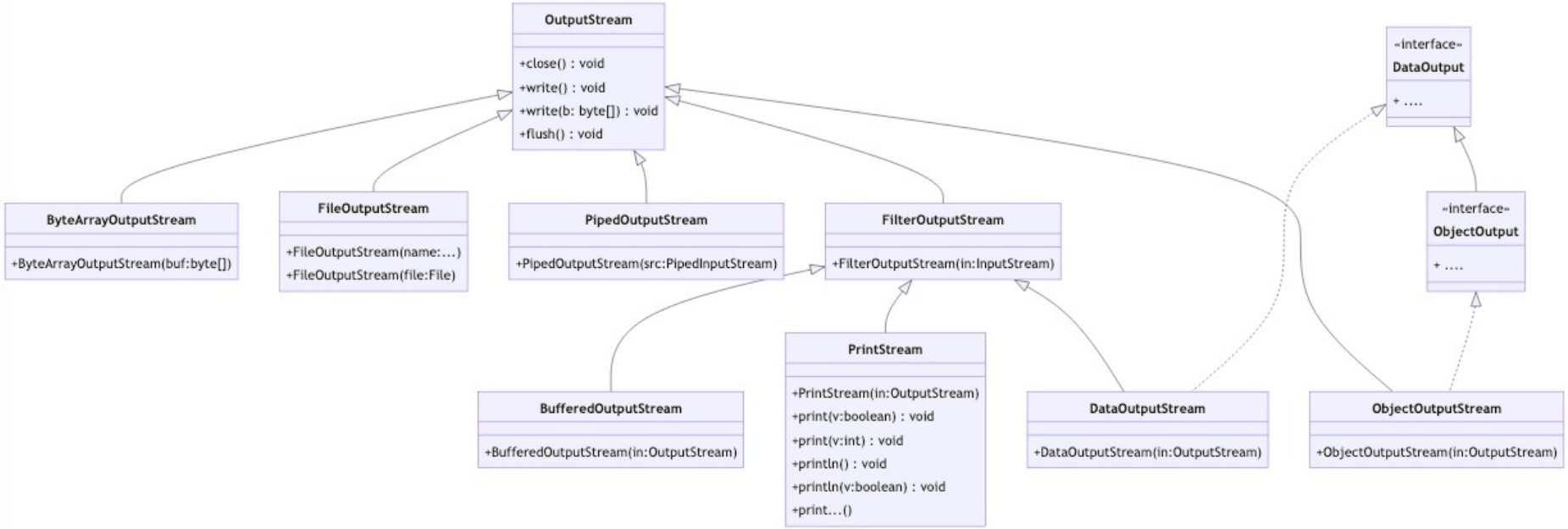
java.io Klassen

InputStreams – byte-basiert



java.io Klassen

OutputStream – byte-basiert



z.B. weitere Streams

```
package java.util.zip;

public class InflaterInputStream extends FilterInputStream {
  ...
}

public class ZipInputStream extends InflaterInputStream implements ZipConstants
{
  ...
}

public class GZIPInputStream extends InflaterInputStream {
...
}
```

z.B. weitere Streams

```
package javax.crypto;

public class CipherInputStream extends FilterInputStream {
  public CipherInputStream(InputStream is, Cipher c) {
    ...
  }
}

public class CipherOutputStream extends FilterOutputStream {
  public CipherOutputStream(OutputStream os, Cipher c) {
    ...
  }
}
```

System.out.println(...)

```
package java.lang;

public final class System {

  public static final PrintStream out = null;

  ...
```

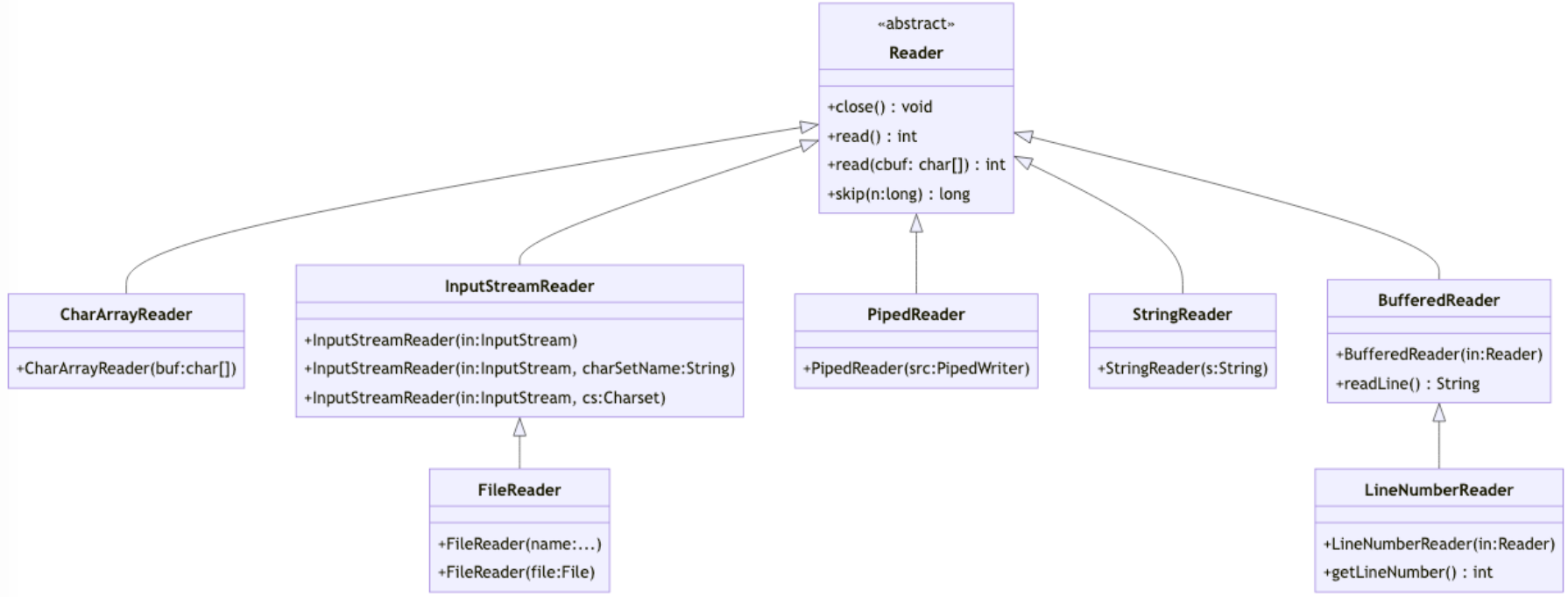
z.B. weitere Nutzung von Streams

```
package java.net;

public abstract class URLConnection {

  public InputStream getInputStream() throws IOException {...}
  public OutputStream getOutputStream() throws IOException {...}

  ...
```


Rückgabetyp ist ein Interface
Nutzer muss nicht die konkrete Klasse kennen → einfache Änderbarkeit

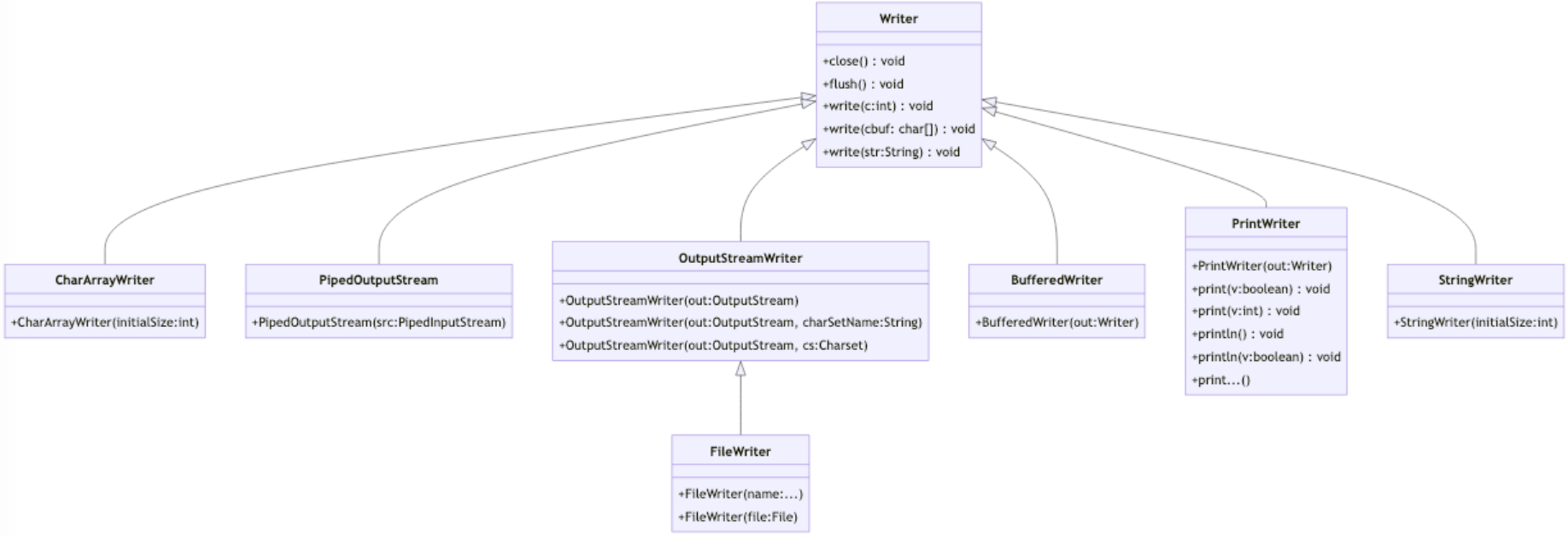
java.io Klassen

Reader – character-basiert



java.io Klassen

Writer – character-basiert



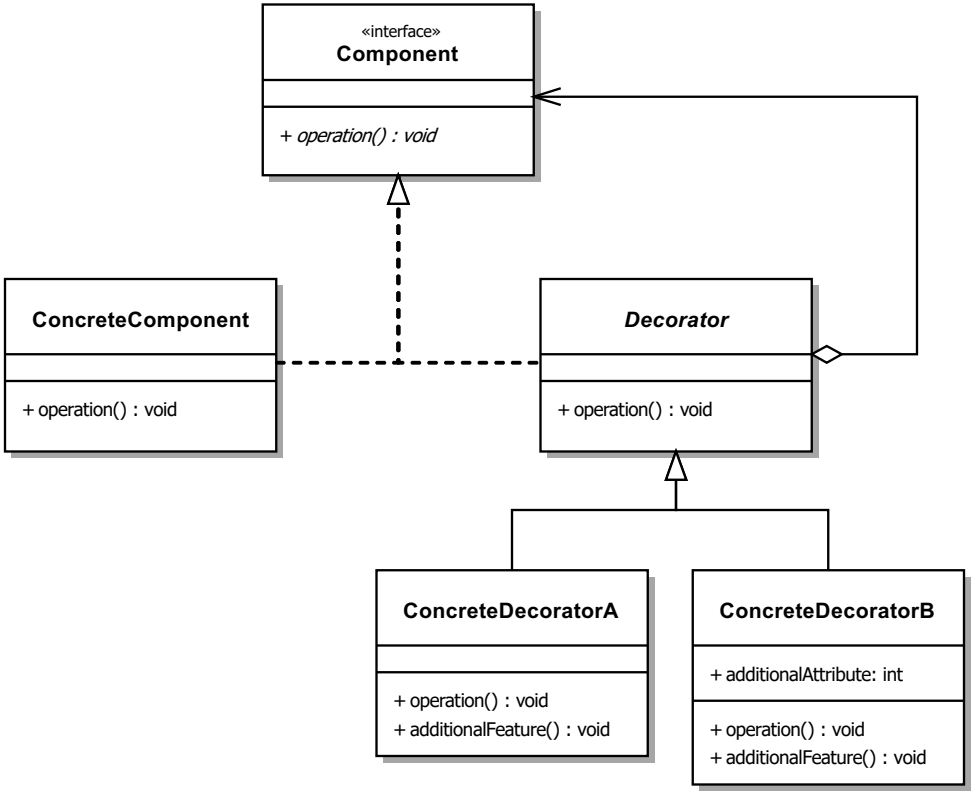
Pattern?

- Problem:
  unterschiedliche Klassen sollen mit verschiedenen Eigenschaften angereichert werden

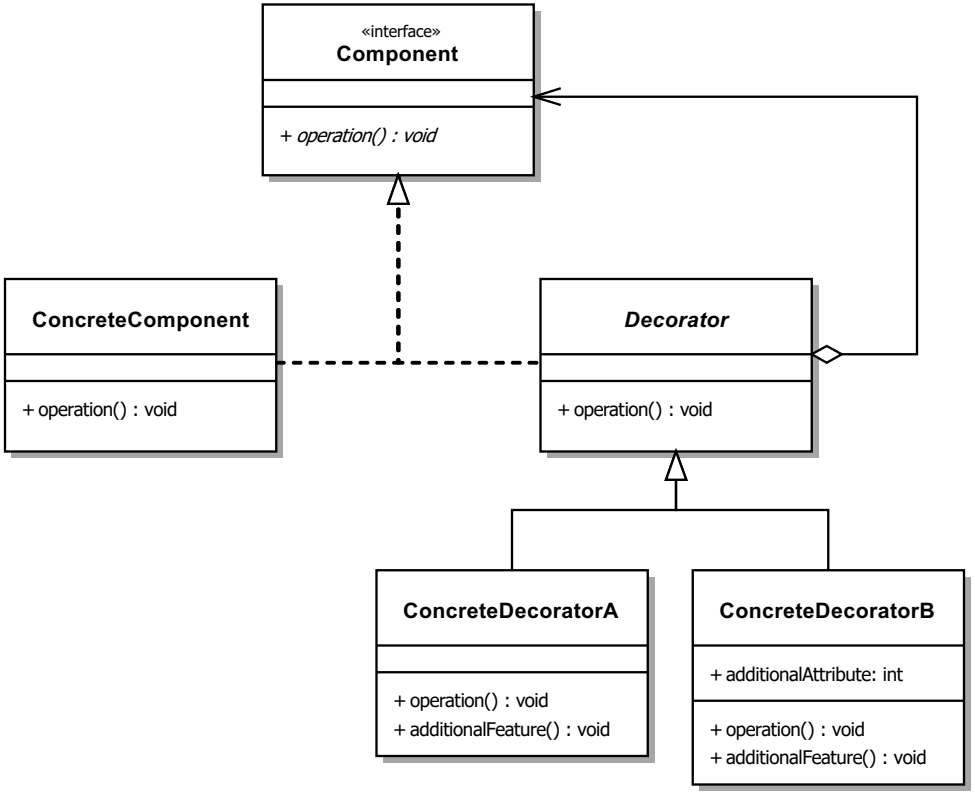
- naive Lösung:
  für jede Kombination eine neue Klasse

- bessere Lösung:
  Decorator Pattern

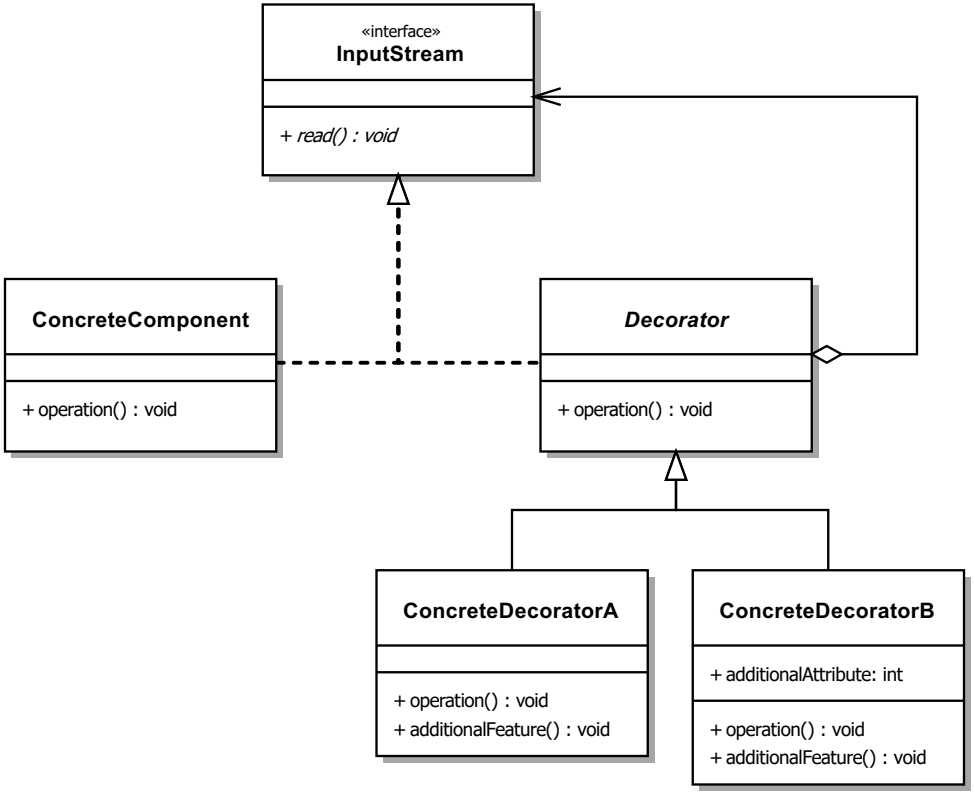
Decorator Pattern



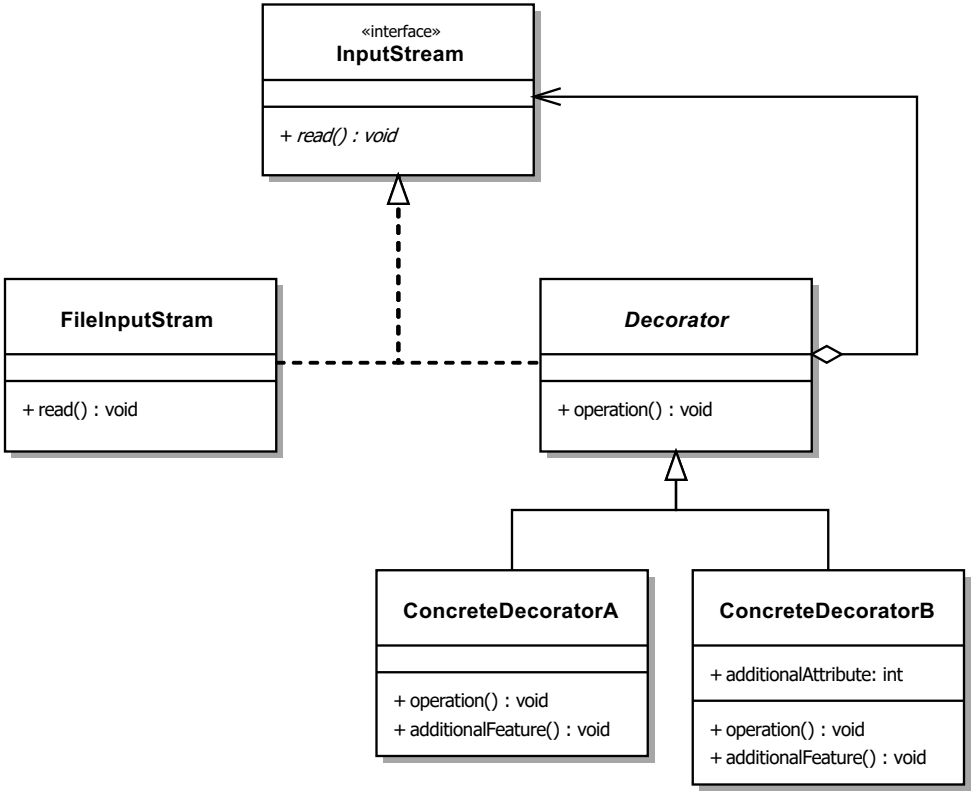
Decorator Pattern in java.io



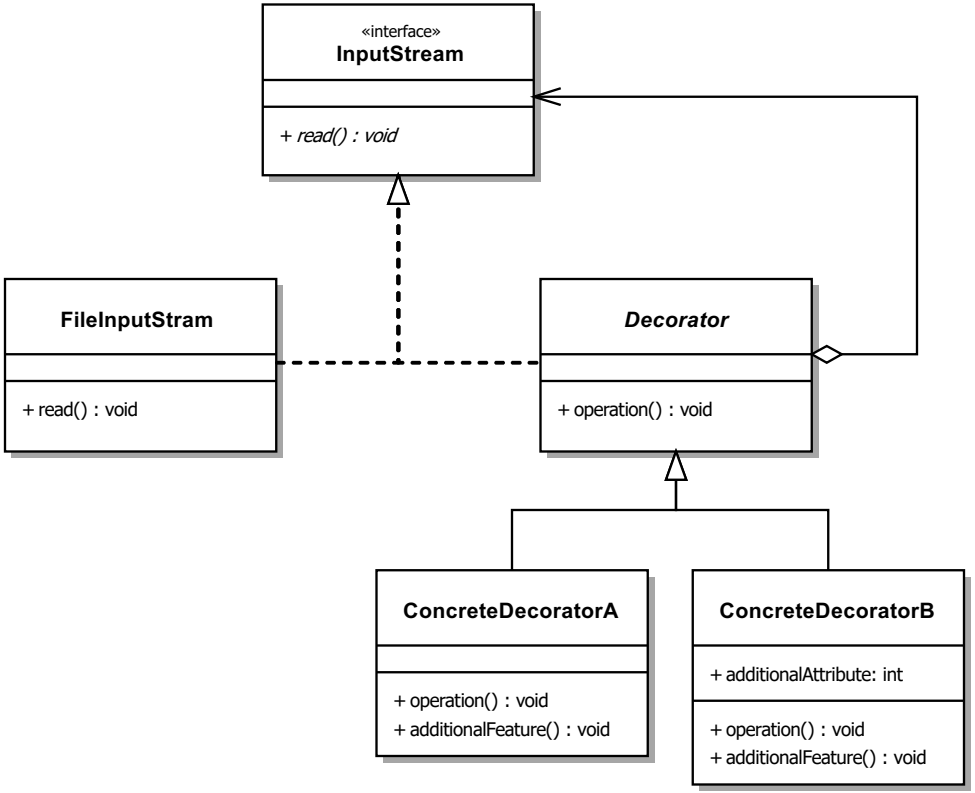
Decorator Pattern in java.io



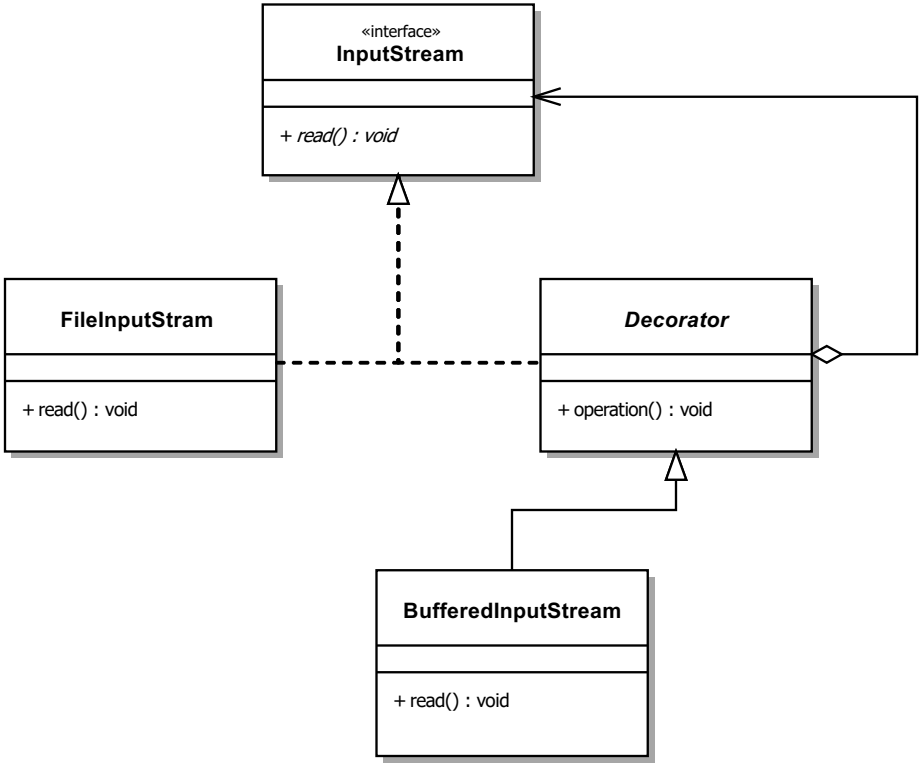
Decorator Pattern in java.io



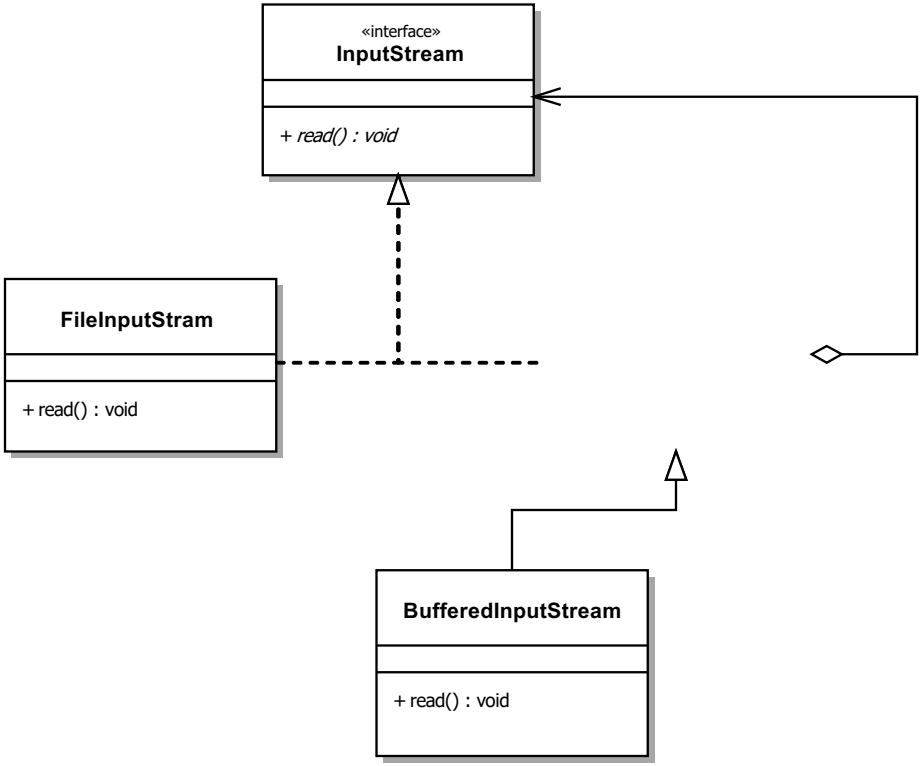
Decorator Pattern in java.io



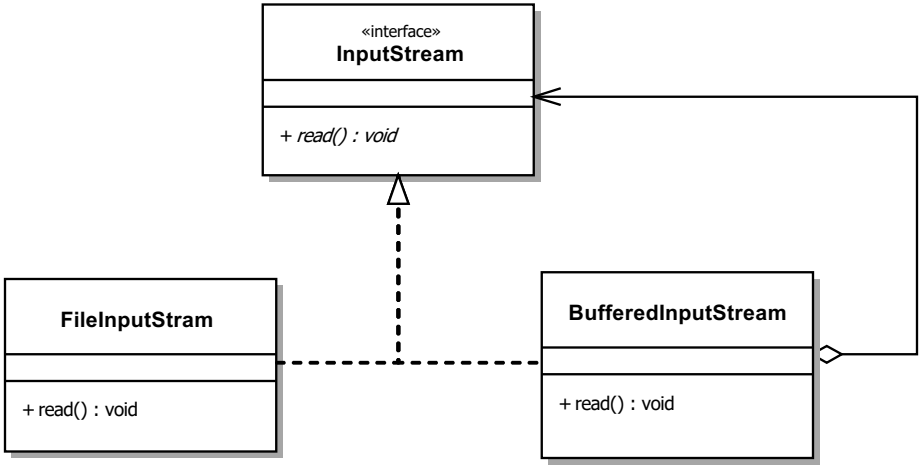
Decorator Pattern in java.io



Decorator Pattern in java.io



Decorator Pattern in java.io



Vorteile?

- Man kann "Kaskaden" bilden und dennoch transparent auf die Interfaces zugreifen:

```
new BufferedInputStream(
      new FileInputStream("Datei.txt"));
```

Code-Beispiele

de.uni-ulm.sp.oop.io

- CopyBytes
- BufferedCopyBytes
- BufferedTextCopy
- DataInput
- DataOutput
- SerializeExample

Lernziele

- Dateiformate
- Input/Output in Java
  - Streams API
  - Decorator-Pattern