

JAVA für Fortgeschrittene

Einheit 01: Java Input/Output

L.Raed

Ludwig-Maximilians-Universität München
Institut für Informatik
Programmierung und Softwaretechnik
Prof. Wirsing

14. März 2011



Die Lernziele der heutigen Einheit

- 1 Einführung in Input/Output Package: java.io
- 2 Lesen/Schreiben in Text-Format
- 3 Lesen/Schreiben in Binär-Format
- 4 Objekt Serialisieren
- 5 Neue I/O Package: java.nio
- 6 Reguläre Ausdrücke

Die Lernziele der heutigen Einheit

- 1 Einführung in Input/Output Package: java.io
- 2 Lesen/Schreiben in Text-Format
- 3 Lesen/Schreiben in Binär-Format
- 4 Objekt Serialisieren
- 5 Neue I/O Package: java.nio
- 6 Reguläre Ausdrücke

Die Lernziele der heutigen Einheit

- 1 Einführung in Input/Output Package: java.io
- 2 Lesen/Schreiben in Text-Format
- 3 Lesen/Schreiben in Binär-Format
- 4 Objekt Serialisieren
- 5 Neue I/O Package: java.nio
- 6 Reguläre Ausdrücke

Die Lernziele der heutigen Einheit

- 1 Einführung in Input/Output Package: java.io
- 2 Lesen/Schreiben in Text-Format
- 3 Lesen/Schreiben in Binär-Format
- 4 Objekt Serialisieren
- 5 Neue I/O Package: java.nio
- 6 Reguläre Ausdrücke

Die Lernziele der heutigen Einheit

- 1 Einführung in Input/Output Package: java.io
- 2 Lesen/Schreiben in Text-Format
- 3 Lesen/Schreiben in Binär-Format
- 4 Objekt Serialisieren
- 5 Neue I/O Package: java.nio
- 6 Reguläre Ausdrücke

Die Lernziele der heutigen Einheit

- 1 Einführung in Input/Output Package: java.io
- 2 Lesen/Schreiben in Text-Format
- 3 Lesen/Schreiben in Binär-Format
- 4 Objekt Serialisieren
- 5 Neue I/O Package: java.nio
- 6 Reguläre Ausdrücke

Die Lernziele der heutigen Einheit

- 1 Einführung in Input/Output Package: java.io
- 2 Lesen/Schreiben in Text-Format
- 3 Lesen/Schreiben in Binär-Format
- 4 Objekt Serialisieren
- 5 Neue I/O Package: java.nio
- 6 Reguläre Ausdrücke

Gliederung

- 1 Einführung in java.io
 - Byte/Unicode-Streams
 - Stream-Filter Kombinieren
- 2 Objekte in Textformat speichern
 - Daten in Textformat schreiben/lesen
- 3 Binäre Daten
 - Schreiben/Lesen in binärem Format
 - Random-Access

Was ist ein Stream

- Was ist ein Stream?
 - Ein Stream ist eine Quelle aus der bytes gelesen werden.
 - Ein Stream ist ein Ziel, in das bytes geschrieben werden.
 - Ein Stream kann eine Datei, Netzwerkverbindung oder Memory Blocks sein.
 - Es wird zwischen Input und Output Stream unterschieden.
 - Es gibt Byte-orientierte und Unicode-orientierte Streams.
- Definition von Input- und Output Streams
 - **Input-Stream**: ist ein Objekt, aus dem bytes gelesen werden.
 - **Output-Stream**: ist ein Objekt, in das bytes geschrieben werden.
 - Wenn wir aus einer Datei lesen → brauchen wir ein InputStream Objekt.
 - Wenn wir in eine Datei schreiben → brauchen wir ein OutputStream Objekt.
- Java Bibliothek für Input/Output Streams
 - `java.io.*`: enthält die meisten nützlichen Input/Output Klassen/Interfaces
 - `java.nio.*`: ab Java 1.4 und enthält z.B Character set, NonBlocking I/O.

Was ist ein Stream

- Was ist ein Stream?
 - Ein Stream ist eine Quelle aus der bytes gelesen werden.
 - Ein Stream ist ein Ziel, in das bytes geschrieben werden.
 - Ein Stream kann eine Datei, Netzwerkverbindung oder Memory Blocks sein.
 - Es wird zwischen Input und Output Stream unterschieden.
 - Es gibt Byte-orientierte und Unicode-orientierte Streams.
- Definition von Input- und Output Streams
 - **Input-Stream**: ist ein Objekt, aus dem bytes gelesen werden.
 - **Output-Stream**: ist ein Objekt, in das bytes geschrieben werden.
 - Wenn wir aus einer Datei lesen → brauchen wir ein InputStream Objekt.
 - Wenn wir in eine Datei schreiben → brauchen wir ein OutputStream Objekt.
- Java Bibliothek für Input/Output Streams
 - `java.io.*`: enthält die meisten nützlichen Input/Output Klassen/Interfaces
 - `java.nio.*`: ab Java 1.4 und enthält z.B Character set, NonBlocking I/O.

Was ist ein Stream

- Was ist ein Stream?
 - Ein Stream ist eine Quelle aus der bytes gelesen werden.
 - Ein Stream ist ein Ziel, in das bytes geschrieben werden.
 - Ein Stream kann eine Datei, Netzwerkverbindung oder Memory Blocks sein.
 - Es wird zwischen Input und Output Stream unterschieden.
 - Es gibt Byte-orientierte und Unicode-orientierte Streams.
- Definition von Input- und Output Streams
 - **Input-Stream**: ist ein Objekt, aus dem bytes gelesen werden.
 - **Output-Stream**: ist ein Objekt, in das bytes geschrieben werden.
 - Wenn wir aus einer Datei lesen → brauchen wir ein InputStream Objekt.
 - Wenn wir in eine Datei schreiben → brauchen wir ein OutputStream Objekt.
- Java Bibliothek für Input/Output Streams
 - `java.io.*`: enthält die meisten nützlichen Input/Output Klassen/Interfaces
 - `java.nio.*`: ab Java 1.4 und enthält z.B Character set, NonBlocking I/O.

Was ist ein Stream

- Was ist ein Stream?
 - Ein Stream ist eine Quelle aus der bytes gelesen werden.
 - Ein Stream ist ein Ziel, in das bytes geschrieben werden.
 - Ein Stream kann eine Datei, Netzwerkverbindung oder Memory Blocks sein.
 - Es wird zwischen Input und Output Stream unterschieden.
 - Es gibt Byte-orientierte und Unicode-orientierte Streams.
- Definition von Input- und Output Streams
 - **Input-Stream**: ist ein Objekt, aus dem bytes gelesen werden.
 - **Output-Stream**: ist ein Objekt, in das bytes geschrieben werden.
 - Wenn wir aus einer Datei lesen → brauchen wir ein InputStream Objekt.
 - Wenn wir in eine Datei schreiben → brauchen wir ein OutputStream Objekt.
- Java Bibliothek für Input/Output Streams
 - `java.io.*`: enthält die meisten nützlichen Input/Output Klassen/Interfaces
 - `java.nio.*`: ab Java 1.4 und enthält z.B Character set, NonBlocking I/O.

Was ist ein Stream

- Was ist ein Stream?
 - Ein Stream ist eine Quelle aus der bytes gelesen werden.
 - Ein Stream ist ein Ziel, in das bytes geschrieben werden.
 - Ein Stream kann eine Datei, Netzwerkverbindung oder Memory Blocks sein.
 - Es wird zwischen Input und Output Stream unterschieden.
 - Es gibt Byte-orientierte und Unicode-orientierte Streams.
- Definition von Input- und Output Streams
 - **Input-Stream**: ist ein Objekt, aus dem bytes gelesen werden.
 - **Output-Stream**: ist ein Objekt, in das bytes geschrieben werden.
 - Wenn wir aus einer Datei lesen → brauchen wir ein InputStream Objekt.
 - Wenn wir in eine Datei schreiben → brauchen wir ein OutputStream Objekt.
- Java Bibliothek für Input/Output Streams
 - **java.io.***: enthält die meisten nützlichen Input/Output Klassen/Interfaces
 - **java.nio.***: ab Java 1.4 und enthält z.B Character set, NonBlocking I/O.

Byte und Unicode-Orientierte Streams

● Byte-orientierte Streams

- Schreiben/Lesen ein Byte, eine Folge von Bytes.
- Java Wurzel-Hierarchie: OutputStream, InputStream.
- DataOutputStream/DataInputStream schreiben/lesen binäre Daten.

● Unicode-orientierte Streams

- Schreiben/Lesen Unicode-Text
- Ein Unicode-Zeichen braucht für die Kodierung mehrere bytes.
- Java Wurzel-Hierarchie: Reader, Writer.
- PrintWriter/Scanner schreiben/lesen Unicode Texte/Zahlen.
- BufferedWriter/BufferedReader lesen/schreiben Unicode Texte.
- Reader/Writer lesen/schreiben basierend auf 2-byte Unicode code unit.

Byte und Unicode-Orientierte Streams

● Byte-orientierte Streams

- Schreiben/Lesen ein Byte, eine Folge von Bytes.
- Java Wurzel-Hierarchie: OutputStream, InputStream.
- DataOutputStream/DataInputStream schreiben/lesen binäre Daten.

● Unicode-orientierte Streams

- Schreiben/Lesen Unicode-Text
- Ein Unicode-Zeichen braucht für die Kodierung mehrere bytes.
- Java Wurzel-Hierarchie: Reader, Writer.
- PrintWriter/Scanner schreiben/lesen Unicode Texte/Zahlen.
- BufferedWriter/BufferedReader lesen/schreiben Unicode Texte.
- Reader/Writer lesen/schreiben basierend auf 2-byte Unicode code unit.

Byte und Unicode-Orientierte Streams

● Byte-orientierte Streams

- Schreiben/Lesen ein Byte, eine Folge von Bytes.
- Java Wurzel-Hierarchie: OutputStream, InputStream.
- DataOutputStream/DataInputStream schreiben/lesen binäre Daten.

● Unicode-orientierte Streams

- Schreiben/Lesen Unicode-Text
- Ein Unicode-Zeichen braucht für die Kodierung mehrere bytes.
- Java Wurzel-Hierarchie: Reader, Writer.
- PrintWriter/Scanner schreiben/lesen Unicode Texte/Zahlen.
- BufferedWriter/BufferedReader lesen/schreiben Unicode Texte.
- Reader/Writer lesen/schreiben basierend auf 2-byte Unicode code unit.

Byte und Unicode-Orientierte Streams

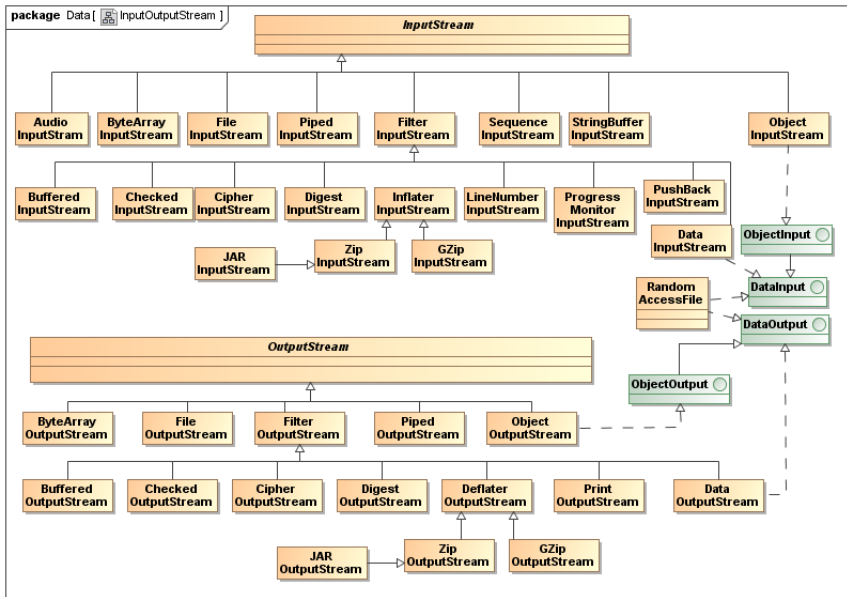
• Byte-orientierte Streams

- Schreiben/Lesen ein Byte, eine Folge von Bytes.
- Java Wurzel-Hierarchie: OutputStream, InputStream.
- DataOutputStream/DataInputStream schreiben/lesen binäre Daten.

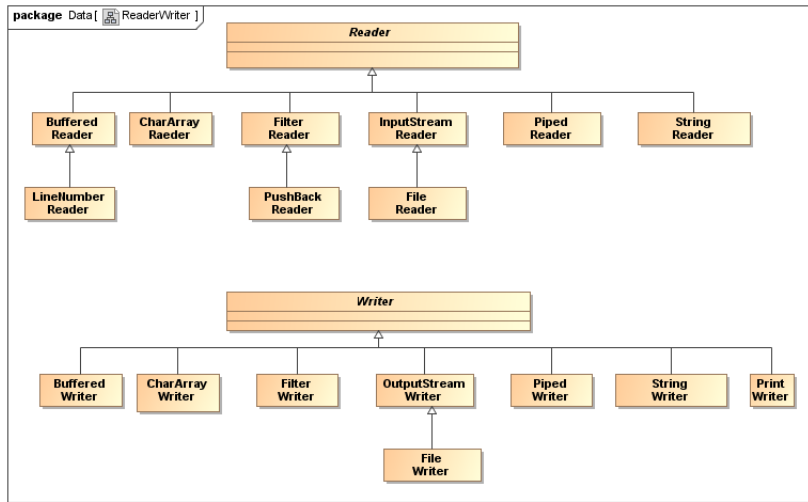
• Unicode-orientierte Streams

- Schreiben/Lesen Unicode-Text
- Ein Unicode-Zeichen braucht für die Kodierung mehrere bytes.
- Java Wurzel-Hierarchie: Reader, Writer.
- PrintWriter/Scanner schreiben/lesen Unicode Texte/Zahlen.
- BufferedWriter/BufferedReader lesen/schreiben Unicode Texte.
- Reader/Writer lesen/schreiben basierend auf 2-byte Unicode code unit.

Vererbungshierarchie: Byte-orientierte Streams



Vererbungshierarchie: Unicoe-orientierte Streams



Byte-orientierte Streams Methoden

#	InputStream Methoden	OutputStream Methoden
1	abstract int read()	abstract void write(int n)
2	int read(byte[] b)	void write(byte[] b)
3	int read(byte[] b, int off, int len)	void write(byte[] b, int off, int len)
4	void close()	void close()
5	int available()	void flush()
6	int skip(long n)	-
7	void mark(int readlimit)	-
8	void reset()	-
9	boolean markSupported()	

InputStream aus dem JAVA API 6

Constructor Summary

[InputStream](#) ()

Method Summary

int	available ()	Returns an estimate of the number of bytes that can be read (or skipped over) from this input stream without blocking by the next invocation of a method for this input stream.
void	close ()	Closes this input stream and releases any system resources associated with the stream.
void	mark (int readlimit)	Marks the current position in this input stream.
boolean	markSupported ()	Tests if this input stream supports the <code>mark</code> and <code>reset</code> methods.
abstract int	read ()	Reads the next byte of data from the input stream.
int	read (byte[] b)	Reads some number of bytes from the input stream and stores them into the buffer array b.
int	read (byte[] b, int off, int len)	Reads up to len bytes of data from the input stream into an array of bytes.
void	reset ()	Repositions this stream to the position at the time the <code>mark</code> method was last called on this input stream.
long	skip (long n)	Skips over and discards n bytes of data from this input stream.

OutputStream aus dem JAVA API 6

Constructor Summary

[OutputStream\(\)](#)

Method Summary

void	close()	Closes this output stream and releases any system resources associated with this stream.
void	flush()	Flushes this output stream and forces any buffered output bytes to be written out.
void	write(byte[] b)	Writes <code>b.length</code> bytes from the specified byte array to this output stream.
void	write(byte[] b, int off, int len)	Writes <code>len</code> bytes from the specified byte array starting at offset <code>off</code> to this output stream.
abstract void	write(int b)	Writes the specified byte to this output stream.

Regeln für byte-orientierte Input/OutputStream

- Das Blockierungsprinzip von `int read()` und `void write()`
 - `int read()`: wartet solange bis die bytes zum Lesen bereit sind.
 - `void write()`: wartet solange bis die bytes zum Schreiben bereit sind.
 - währends des Warten(Blockieren) können andere Threads weiter arbeiten.
- Blockierung umgehen/vermeiden
 - Die `available()` Abfrage vor dem Lesen kann die Blockierung vermeiden.

```
int bytesAvailable = in.available();
if (bytesAvailable > 0) {
    byte[] data = new byte[bytesAvailable];
    in.read(data); }
```
- Betriebssystem-Ressourcen wieder freisetzen: `close()`
 - `close()` muss am Ende vom Lesen/Schreiben aufgerufen.
 - `close()` leert den für das Output-Stream benutzten Buffer.
 - ofene Streams ohne `close()` belasten die Leistung des Rechners sehr.

Unicode-orientierte Streams Methoden

#	Reader Methoden	Writer Methoden
1	<code>abstract void close()</code>	<code>Writer append(char c)</code>
2	<code>void mark(int readAheadLimit)</code>	<code>Writer append(CharSequence csq)</code>
3	<code>boolean markSupported()</code>	<code>Writer append(CharSequence csq, int start, int end)</code>
4	<code>int read()</code>	<code>abstract void close()</code>
5	<code>int read(char[] cbuf)</code>	<code>abstract void flush()</code>
6	<code>abstract int read(char[]cbuf, int off, int len)</code>	<code>void write(char[] cbuf)</code>
7	<code>int read(CharBuffer target)</code>	<code>abstract void write(char[] cbuf, int off, int len)</code>
8	<code>boolean ready()</code>	<code>void write(int c)</code>
9	<code>void reset()</code>	<code>void write(String str)</code>
10	<code>long skip(long n)</code>	<code>void write(String str, int off, int len)</code>

Die abstrakte Klasse Reader aus dem JAVA API 6

Constructor Summary

protected	Reader ()	Creates a new character-stream reader whose critical sections will synchronize on the reader itself.
protected	Reader (Object lock)	Creates a new character-stream reader whose critical sections will synchronize on the given object.

Method Summary

abstract void	close ()	Closes the stream and releases any system resources associated with it.
void	mark (int readAheadLimit)	Marks the present position in the stream.
boolean	markSupported ()	Tells whether this stream supports the mark() operation.
int	read ()	Reads a single character.
int	read (char[] cbuf)	Reads characters into an array.
abstract int	read (char[] cbuf, int off, int len)	Reads characters into a portion of an array.
int	read (CharBuffer target)	Attempts to read characters into the specified character buffer.
boolean	ready ()	Tells whether this stream is ready to be read.
void	reset ()	Resets the stream.

Die abstrakte Klasse Writer aus dem JAVA API 6

Constructor Summary

protected	Writer () Creates a new character-stream writer whose critical sections will synchronize on the writer itself.
protected	Writer (Object lock) Creates a new character-stream writer whose critical sections will synchronize on the given object.

Method Summary

Writer	append (char c) Appends the specified character to this writer.
Writer	append (CharSequence csq) Appends the specified character sequence to this writer.
Writer	append (CharSequence csq, int start, int end) Appends a subsequence of the specified character sequence to this writer.
abstract void	close () Closes the stream, flushing it first.
abstract void	flush () Flushes the stream.
void	write (char[] cbuf) Writes an array of characters.
abstract void	write (char[] cbuf, int off, int len) Writes a portion of an array of characters.
void	write (int c) Writes a single character.
void	write (String str) Writes a string.

Einführung in Unicode-Zeichen

- **Verschiedene Sprachen haben unterschiedliche Standards**
 - ASCII für USA
 - ISO 8859-1 für Europa
 - KOI-8 für Russland
 - BIG-5 für China
- **Es entstehen Kodierungsprobleme**
 - die Standards benutzen unterschiedliche Kodierungslänge.
 - Kodierung: 1,2 oder mehrere Bytes für ein Zeichen.
 - Ein Wert könnte bei 2 Standards unterschiedliche Symbole darstellen.
- **Lösung der Kodierungsprobleme**
 - einheitliche Kodierung: Unicode Zeichen.

Unicode und UTF-16

- Unicode Zeichen

- benutzt zur Kodierung 2 bytes pro Zeichen.
- kann 65,536 Zeichen darstellen
- darstellbar als hexadezimal Wert von \u0000 bis \uFFFF.
- Zum Beispiel ist \u2122 das griechische Buchstabe π (pi).
- mit ostasiatischen Sonderzeichen gibt es mehr als 65,536 Zeichen.
- Lösung: UTF-16 Kodierung.

- UTF-16 Kodierung

- stellt alle Unicode-Werte(code points) durch variable Kodierungslänge.
- char in Java beschreibt ein „code unit“ in die UTF-16 Kodierung.

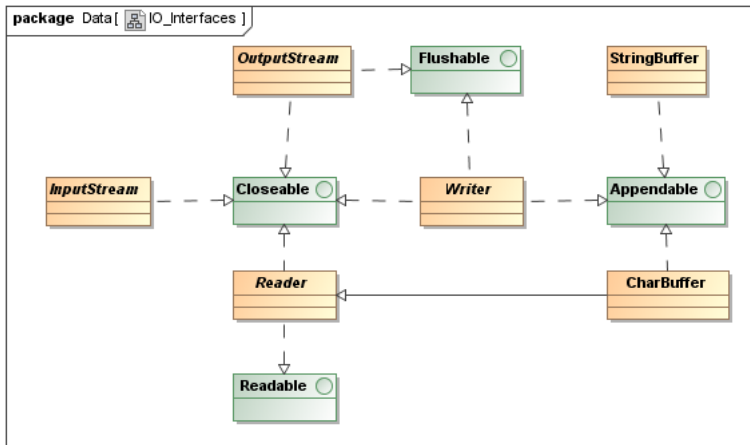
char und Sonderzeichen

- Der Datentyp `char`
 - wird für Einzelzeichen (characters) benutzt.
 - 'A' ist ein konstantes Zeichen mit dem Wert 65.
 - `char c='A'` oder `char c=65;` sind äquivalent.
- Darstellung vom Sonderzeichen
 - das Escape-Sequenz `\u` deutet die Kodierung von Unicode an.

Escape-Sequenz	Name	Unicode Wert
<code>\b</code>	Backspace (Backspace)	<code>\u0008</code>
<code>\t</code>	Tabulator (Tab)	<code>\u0009</code>
<code>\n</code>	Zeilenvorschub (Linefeed)	<code>\u000a</code>
<code>\r</code>	die Zeilenumschaltung (Carriage return)	<code>\u000d</code>
<code>\"</code>	das Anführungszeichen (Double quote)	<code>\u0022</code>
<code>'</code>	das einfache Anführungszeichen (Single quote)	<code>\u0027</code>
<code>\\</code>	der Backslash (Backslash)	<code>\u005c</code>

Die I/O Interfaces

Java 1.5 bietet folgende IO-Interfaces



Die I/O Interfaces

① `java.io.Closable` Interface: Methode `void close()`

- Closable ist eine Datenquelle/Datenziel das geschlossen werden kann.
- wird von `InputStream/OutputStream`, `Reader/Writer` implementiert.
- `void close()` throws `IOException` (falls ein I/O error passiert)
- `close()` schließt das Stream und gibt die beteiligen Ressourcen wieder frei.

② `java.io.Flushable` Interface: Methode `void flush()`

- Flushable ist ein Datenziel das geleert werden kann.
- wird von `OutputStream` und `Writer` implementiert.
- `void flush()` throws `IOException` (falls ein I/O error passiert)
- `flush()` leert das Stream und schreibt die gepufferte Daten in das Ziel.

Relevante lang Interfaces für I/O

- 3 **java.lang.Readable Interface: Methode `int read(CharBuffer cb)`**
 - Readable ist eine Zeichenquelle (source of characters).
 - CharBuffer stellt die Readable Characters für `read()` Aufrufer bereit.
 - Readable wird vom Reader implementiert.
 - `int read(CharBuffer cb)` throws `IOException` (falls ein I/O error passiert)
 - `read` liest characters und speichert sie in dem spezifizierten buffer `cb`.
- 4 **java.lang.Appendable Interface: Methode 3 `append` Methoden**
 - ein Objekt an dem `char/Sequence` angehängt werden können.
 - wird von `Writer/StringBuffer` und `CharBuffer` implementiert.
 - `Appendable append(char c)` throws `IOException`
 - `Appendable append(CharSequence csq)` throws `IOException`
 - `Appendable append(CharSequence csq, int start, int end)`
 - `append` Methoden hängen ein `char/CharSequence` oder `SubSequence` an.

Stream-Filter Kombination-Technik

- I/O benutzt die Technik des Dekorator-Design-Pattern.
 - Manche Streams können die Bytes nur extrahieren, andere nur verarbeiten.
 - FileInputStream kann z.B. Bytes holen aber keine Nummer lesen.
 - DataInputStream kann z.B Zahlen lesen aber nicht Daten aus Datei lesen.
 - Dekorator-Pattern: kombiniere DataInputStream mit FileInputStream
 - HighLevelStream h = new HighLevelStream(LowLevelStream);
- Low-Level Streams: zum gewinnen von Bytes
 - liefern Disk-Datei gebundenes Input und OutputStream.
 - lesen/schreiben nur bytes (byte-level Operationen)
 - Beispiel für Byte-orientiert: FileOutputStream/FileInputStream
 - Beispiel für Unicode-orientiert: FileWriter/FileReader.

```
FileInputStream fin = new FileInputStream("test.dat");  
byte b = (byte) fin.read();
```
- High-Level Streams: zum verarbeiten von Bytes
 - haben keine Methoden um Bytes zu gewinnen, sondern zum Verarbeiten.
 - Beispiel für Byte-orientiert: DataInputStream/DataOutputStream
 - Beispiel für Unicode-orientiert: BufferedWriter/BufferedReader.

Stream-Filter Kombination-Technik

- I/O benutzt die Technik des Dekorator-Design-Pattern.
 - Manche Streams können die Bytes nur extrahieren, andere nur verarbeiten.
 - FileInputStream kann z.B. Bytes holen aber keine Nummer lesen.
 - DataInputStream kann z.B. Zahlen lesen aber nicht Daten aus Datei lesen.
 - Dekorator-Pattern: kombiniere DataInputStream mit FileInputStream
 - HighLevelStream h = new HighLevelStream(LowLevelStream);
- Low-Level Streams: zum gewinnen von Bytes
 - liefern Disk-Datei gebundenes Input und OutputStream.
 - lesen/schreiben nur bytes (byte-level Operationen)
 - Beispiel für Byte-orientiert: FileOutputStream/FileInputStream
 - Beispiel für Unicode-orientiert: FileWriter/FileReader.

```
FileInputStream fin = new FileInputStream("test.dat");  
byte b = (byte) fin.read();
```
- High-Level Streams: zum verarbeiten von Bytes
 - haben keine Methoden um Bytes zu gewinnen, sondern zum Verarbeiten.
 - Beispiel für Byte-orientiert: DataInputStream/DataOutputStream
 - Beispiel für Unicode-orientiert: BufferedWriter/BufferedReader.

Stream-Filter Kombination-Technik

- I/O benutzt die Technik des Dekorator-Design-Pattern.
 - Manche Streams können die Bytes nur extrahieren, andere nur verarbeiten.
 - FileInputStream kann z.B. Bytes holen aber keine Nummer lesen.
 - DataInputStream kann z.B. Zahlen lesen aber nicht Daten aus Datei lesen.
 - Dekorator-Pattern: kombiniere DataInputStream mit FileInputStream
 - HighLevelStream h = new HighLevelStream(LowLevelStream);
- Low-Level Streams: zum gewinnen von Bytes
 - liefern Disk-Datei gebundenes Input und OutputStream.
 - lesen/schreiben nur bytes (byte-level Operationen)
 - Beispiel für Byte-orientiert: FileOutputStream/FileInputStream
 - Beispiel für Unicode-orientiert: FileWriter/FileReader.

```
FileInputStream fin = new FileInputStream("test.dat");
byte b = (byte) fin.read();
```
- High-Level Streams: zum verarbeiten von Bytes
 - haben keine Methoden um Bytes zu gewinnen, sondern zum Verarbeiten.
 - Beispiel für Byte-orientiert: DataInputStream/DataOutputStream
 - Beispiel für Unicode-orientiert: BufferedWriter/BufferedReader.

Stream-Filter Kombination-Technik

- I/O benutzt die Technik des Dekorator-Design-Pattern.
 - Manche Streams können die Bytes nur extrahieren, andere nur verarbeiten.
 - FileInputStream kann z.B. Bytes holen aber keine Nummer lesen.
 - DataInputStream kann z.B Zahlen lesen aber nicht Daten aus Datei lesen.
 - Dekorator-Pattern: kombiniere DataInputStream mit FileInputStream
 - HighLevelStream h = new HighLevelStream(LowLevelStream);
- Low-Level Streams: zum gewinnen von Bytes
 - liefern Disk-Datei gebundenes Input und OutputStream.
 - lesen/schreiben nur bytes (byte-level Operationen)
 - Beispiel für Byte-orientiert: FileOutputStream/FileInputStream
 - Beispiel für Unicode-orientiert: FileWriter/FileReader.

```
FileInputStream fin = new FileInputStream("test.dat");
byte b = (byte) fin.read();
```
- High-Level Streams: zum verarbeiten von Bytes
 - haben keine Methoden um Bytes zu gewinnen, sondern zum Verarbeiten.
 - Beispiel für Byte-orientiert: DataInputStream/DataOutputStream
 - Beispiel für Unicode-orientiert: BufferedWriter/BufferedReader.

Stream-Filter Kombination-Technik

- I/O benutzt die Technik des Dekorator-Design-Pattern.
 - Manche Streams können die Bytes nur extrahieren, andere nur verarbeiten.
 - FileInputStream kann z.B. Bytes holen aber keine Nummer lesen.
 - DataInputStream kann z.B. Zahlen lesen aber nicht Daten aus Datei lesen.
 - Dekorator-Pattern: kombiniere DataInputStream mit FileInputStream
 - HighLevelStream h = new HighLevelStream(LowLevelStream);
- Low-Level Streams: zum gewinnen von Bytes
 - liefern Disk-Datei gebundenes Input und OutputStream.
 - lesen/schreiben nur bytes (byte-level Operationen)
 - Beispiel für Byte-orientiert: FileOutputStream/FileInputStream
 - Beispiel für Unicode-orientiert: FileWriter/FileReader.


```
FileInputStream fin = new FileInputStream("test.dat");
byte b = (byte) fin.read();
```
- High-Level Streams: zum verarbeiten von Bytes
 - haben keine Methoden um Bytes zu gewinnen, sondern zum Verarbeiten.
 - Beispiel für Byte-orientiert: DataInputStream/DataOutputStream
 - Beispiel für Unicode-orientiert: BufferedWriter/BufferedReader.

Stream-Filter Kombination-Technik

- I/O benutzt die Technik des Dekorator-Design-Pattern.
 - Manche Streams können die Bytes nur extrahieren, andere nur verarbeiten.
 - FileInputStream kann z.B. Bytes holen aber keine Nummer lesen.
 - DataInputStream kann z.B. Zahlen lesen aber nicht Daten aus Datei lesen.
 - Dekorator-Pattern: kombiniere DataInputStream mit FileInputStream
 - HighLevelStream h = new HighLevelStream(LowLevelStream);
- Low-Level Streams: zum gewinnen von Bytes
 - liefern Disk-Datei gebundenes Input und OutputStream.
 - lesen/schreiben nur bytes (byte-level Operationen)
 - Beispiel für Byte-orientiert: FileOutputStream/FileInputStream
 - Beispiel für Unicode-orientiert: FileWriter/FileReader.

```
FileInputStream fin = new FileInputStream("test.dat");
byte b = (byte) fin.read();
```
- High-Level Streams: zum verarbeiten von Bytes
 - haben keine Methoden um Bytes zu gewinnen, sondern zum Verarbeiten.
 - Beispiel für Byte-orientiert: DataInputStream/DataOutputStream
 - Beispiel für Unicode-orientiert: BufferedWriter/BufferedReader.

Kombination mehrerer Filter

- Beispiel 1: Binäre Daten extrahieren & Zahlen lesen

```
FileInputStream fin = new FileInputStream("test.dat");  
DataInputStream din = new DataInputStream(fin);  
double s = din.readDouble();
```

- Beispiel 2: Binäre Daten gepuffert gewinnen & Zahlen lesen

```
DataInputStream din = new DataInputStream(  
new BufferedInputStream(  
new FileInputStream("test.dat")));
```

- Stream sind nicht gepuffert: jeweils ein byte wird gelesen.
- effizienter: blocks von Daten in Puffer speichern dann lesen.
- Lösung: DataInputStream mit BufferedInputStream kombinieren.
- Und BufferedInputStream mit FileInputStream kombinieren.

Kombination mehrerer Filter

- **Beispiel 3: Zwischen-Streams im Auge behalten: PushbackInputStream**

```
PushbackInputStream pbin = new PushbackInputStream(  
    new BufferedInputStream(  
        new FileInputStream("test.dat")));  
int b = pbin.read(); //nächstes Byte lesen.  
if (b != '<') pbin.unread(b); //wirf es zurück.
```

- **Beispiel 4: Zwischen-Streams im Auge behalten und Zalen lesen**

```
DataInputStream din = new DataInputStream(  
    pbin = new PushbackInputStream(  
        new BufferedInputStream(  
            new FileInputStream("test.dat"))));
```

- **Beispiel 5: Gezippte und gepufferte Streams lesen**

```
ZipInputStream zin = new ZipInputStream(  
    new FileInputStream("test.zip"));  
DataInputStream din = new DataInputStream(zin);
```

Textformat vs. Binary Format

- Speichern von int 1234 in binary Format
 - In hexadecimal notation: 00 00 04 D2
- Speichern von int 1234 in Text Format
 - UTF-Encoding: 00 31 00 32 00 33 00 34 (in hex)
 - ISO 8859-1 Encoding: 31 32 33 34
 - UTF, ISO werden "Character Sets" genannt.
- Textformat vs. Binary Format
 - Binary Format: schneller, effizienter, für Menschen schwer lesbar.
 - Text Format: langsamer, weniger effizient, für Menschen leicht lesbar.

Textformat vs. Binary Format

- Speichern von int 1234 in binary Format
 - In hexadecimal notation: 00 00 04 D2
- Speichern von int 1234 in Text Format
 - UTF-Encoding: 00 31 00 32 00 33 00 34 (in hex)
 - ISO 8859-1 Encoding: 31 32 33 34
 - UTF, ISO werden "Character Sets" genannt.
- Textformat vs. Binary Format
 - Binary Format: schneller, effizienter, für Menschen schwer lesbar.
 - Text Format: langsamer, weniger effizient, für Menschen leicht lesbar.

Textformat vs. Binary Format

- Speichern von int 1234 in binary Format
 - In hexadecimal notation: 00 00 04 D2
- Speichern von int 1234 in Text Format
 - UTF-Encoding: 00 31 00 32 00 33 00 34 (in hex)
 - ISO 8859-1 Encoding: 31 32 33 34
 - UTF, ISO werden "Character Sets" genannt.
- Textformat vs. Binary Format
 - Binary Format: schneller, effizienter, für Menschen schwer lesbar.
 - Text Format: langsamer, weniger effizient, für Menschen leicht lesbar.

Textformat vs. Binary Format

- Speichern von int 1234 in binary Format
 - In hexadecimal notation: 00 00 04 D2
- Speichern von int 1234 in Text Format
 - UTF-Encoding: 00 31 00 32 00 33 00 34 (in hex)
 - ISO 8859-1 Encoding: 31 32 33 34
 - UTF, ISO werden "Character Sets" genannt.
- Textformat vs. Binary Format
 - Binary Format: schneller, effizienter, für Menschen schwer lesbar.
 - Text Format: langsamer, weniger effizient, für Menschen leicht lesbar.

Textformat vs. Binary Format

- Speichern von int 1234 in binary Format
 - In hexadecimal notation: 00 00 04 D2
- Speichern von int 1234 in Text Format
 - UTF-Encoding: 00 31 00 32 00 33 00 34 (in hex)
 - ISO 8859-1 Encoding: 31 32 33 34
 - UTF, ISO werden "Character Sets" genannt.
- Textformat vs. Binary Format
 - Binary Format: schneller, effizienter, für Menschen schwer lesbar.
 - Text Format: langsamer, weniger effizient, für Menschen leicht lesbar.

Textformat vs. Binary Format

- Speichern von int 1234 in binary Format
 - In hexadecimal notation: 00 00 04 D2
- Speichern von int 1234 in Text Format
 - UTF-Encoding: 00 31 00 32 00 33 00 34 (in hex)
 - ISO 8859-1 Encoding: 31 32 33 34
 - UTF, ISO werden "Character Sets" genannt.
- Textformat vs. Binary Format
 - Binary Format: schneller, effizienter, für Menschen schwer lesbar.
 - Text Format: langsamer, weniger effizient, für Menschen leicht lesbar.

Daten in Textformat in eine Datei schreiben

1 **OutputStream mit einer Datei verbinden**

- `java.io.FileWriter`: verbindet OutputStream mit einer Festplatte-Datei.
- `FileWriter f = new FileWriter("test.txt");`
- Alternative: `FileWriter f = new FileWriter(new FileOutputStream("test.txt"));`

2 **Daten in Textformat schreiben**

- `java.io.BufferedWriter` zum Schreiben von Werten in Textformat bietet Java
- `java.util.PrintWriter` ab Java 5 zum Schreiben Text/Zahlen in Textformat
- `BufferedWriter out = new BufferedWriter(new FileWriter("test.txt"));`
- `PrintWriter out = new PrintWriter("test.txt"); // Ist möglich ohne FileWriter.`

Daten in Textformat in eine Datei schreiben

1 **OutputStream mit einer Datei verbinden**

- `java.io.FileWriter`: verbindet OutputStream mit einer Festplatte-Datei.
- `FileWriter f = new FileWriter("test.txt");`
- Alternative: `FileWriter f = new FileWriter(new FileOutputStream("test.txt"));`

2 **Daten in Textformat schreiben**

- `java.io.BufferedWriter` zum Schreiben von Werten in Textformat bietet Java
- `java.util.PrintWriter` ab Java 5 zum Schreiben Text/Zahlen in Textformat
- `BufferedWriter out = new BufferedWriter(new FileWriter("test.txt"));`
- `PrintWriter out = new PrintWriter("test.txt"); // Ist möglich ohne FileWriter.`

Daten in Textformat in eine Datei schreiben

1 **OutputStream mit einer Datei verbinden**

- `java.io.FileWriter`: verbindet OutputStream mit einer Festplatte-Datei.
- `FileWriter f = new FileWriter("test.txt");`
- Alternative: `FileWriter f = new FileWriter(new FileOutputStream("test.txt"));`

2 **Daten in Textformat schreiben**

- `java.io.BufferedWriter` zum Schreiben von Werten in Textformat bietet Java
- `java.util.PrintWriter` ab Java 5 zum Schreiben Text/Zahlen in Textformat
- `BufferedWriter out = new BufferedWriter(new FileWriter("test.txt"));`
- `PrintWriter out = new PrintWriter("test.txt"); // Ist möglich ohne FileWriter.`

Daten in Textformat in eine Datei schreiben

1 **OutputStream mit einer Datei verbinden**

- `java.io.FileWriter`: verbindet OutputStream mit einer Festplatte-Datei.
- `FileWriter f = new FileWriter("test.txt");`
- Alternative: `FileWriter f = new FileWriter(new FileOutputStream("test.txt"));`

2 **Daten in Textformat schreiben**

- `java.io.BufferedWriter` zum Schreiben von Werten in Textformat bietet Java
- `java.util.PrintWriter` ab Java 5 zum Schreiben Text/Zahlen in Textformat
- `BufferedWriter out = new BufferedWriter(new FileWriter("test.txt"));`
- `PrintWriter out = new PrintWriter("test.txt"); // Ist möglich ohne FileWriter.`

Daten in Textformat aus einer Datei lesen

1 **InputStream mit einer Datei verbinden**

- `java.io.FileReader` verbindet `InputStream` mit einer Festplatte-Datei.
- `FileReader f = new FileReader("test.txt");`
- Alternative: `FileReader f = new FileReader(new FileInputStream("test.txt"));`

2 **Daten in Textformat lesen**

- `java.io.BufferedReader` zum Lesen von Daten in Textformat.
- `java.util.Scanner` Ab Java 5 zum Lesen von Text/Zahlen aus Textformat
- `BufferedReader in = new BufferedReader(new FileReader("test.txt"));`
- `Scanner in = new Scanner("test.txt"); // Ist möglich ohne FileReader`

Daten in Textformat aus einer Datei lesen

1 **InputStream mit einer Datei verbinden**

- `java.io.FileReader` verbindet `InputStream` mit einer Festplatte-Datei.
- `FileReader f = new FileReader("test.txt");`
- Alternative: `FileReader f = new FileReader(new FileInputStream("test.txt"));`

2 **Daten in Textformat lesen**

- `java.io.BufferedReader` zum Lesen von Daten in Textformat.
- `java.util.Scanner` Ab Java 5 zum Lesen von Text/Zahlen aus Textformat
- `BufferedReader in = new BufferedReader(new FileReader("test.txt"));`
- `Scanner in = new Scanner("test.txt"); // Ist möglich ohne FileReader`

Daten in Textformat aus einer Datei lesen

1 **InputStream mit einer Datei verbinden**

- `java.io.FileReader` verbindet `InputStream` mit einer Festplatte-Datei.
- `FileReader f = new FileReader("test.txt");`
- Alternative: `FileReader f = new FileReader(new FileInputStream("test.txt"));`

2 **Daten in Textformat lesen**

- `java.io.BufferedReader` zum Lesen von Daten in Textformat.
- `java.util.Scanner` Ab Java 5 zum Lesen von Text/Zahlen aus Textformat
- `BufferedReader in = new BufferedReader(new FileReader("test.txt"));`
- `Scanner in = new Scanner("test.txt"); // Ist möglich ohne FileReader`

Daten in Textformat aus einer Datei lesen

1 **InputStream mit einer Datei verbinden**

- `java.io.FileReader` verbindet InputStream mit einer Festplatte-Datei.
- `FileReader f = new FileReader("test.txt");`
- Alternative: `FileReader f = new FileReader(new FileInputStream("test.txt"));`

2 **Daten in Textformat lesen**

- `java.io.BufferedReader` zum Lesen von Daten in Textformat.
- `java.util.Scanner` Ab Java 5 zum Lesen von Text/Zahlen aus Textformat
- `BufferedReader in = new BufferedReader(new FileReader("test.txt"));`
- `Scanner in = new Scanner("test.txt"); // Ist möglich ohne FileReader`

Objekte in Unicode-Text speichern

- Ein Objekt in Text-Format speichern

- FileWriter zum Erzeugen der Datei (OutputStream) benutzen.
- BufferedWriter zum Schreiben des Objektzustandes benutzen.
- BufferedWriter mit FileWriter also kombinieren.

```
FileWriter f = new FileWriter("test.txt");
BufferedWriter out = new BufferedWriter(f);
out.write(V1 separator V2 separator Vn);
```

- Ein Objekt aus Text-Format Rekonstruieren

- FileReader zum Gewinn der Daten aus einer Datei benutzen.
- BufferedReader zum Lesen von Text aus den Daten benutzen.
- BufferedReader mit FileReader also kombinieren.

```
FileReader f = new FileReader("test.txt");
BufferedReader in = new BufferedReader(f);
String line = in.readLine();
String[] value = line.split(separator);
V1 = value[1]; // Für Strings
Vn = Integer.parseInt(value[n]); // Zahl
```

Die java.io.BufferedWriter Klasse

• Die BufferedWriter Klasse

- muss mit einem Output-Source kombiniert werden.
- ermöglicht das Schreiben von Daten in Text-Format.
- kann keine Zahlen wie z.B. long, float, double schreiben.
- Zahlen müssen vorher in String konvertiert werden.
- Puffert Zeichen, um sie effizienter zu schreiben.
- Standard Puffer Größe ist für die meisten Aufgabe ausreichen.
- BufferedWriter write Methoden schreiben Zeichen/Array von Zeichen.
- BufferedWriter write Methoden können IOException auswerfen.

• BufferedWriter Beispiel

```
FileWriter f = new FileWriter("test.txt");  
BufferedWriter out = new BufferedWriter(f);  
String name="Lara Hennicker"  
double gehalt = 250;  
out.write(name + " " + gehalt);  
out.close();
```

BufferedWriter aus dem JAVA-API 6

Constructor Summary

BufferedWriter([Writer](#) out)

Creates a buffered character-output stream that uses a default-sized output buffer.

BufferedWriter([Writer](#) out, int sz)

Creates a new buffered character-output stream that uses an output buffer of the given size.

Method Summary

void [close](#)()

Closes the stream, flushing it first.

void [flush](#)()

Flushes the stream.

void [newLine](#)()

Writes a line separator.

void [write](#)(char[] cbuf, int off, int len)

Writes a portion of an array of characters.

void [write](#)(int c)

Writes a single character.

void [write](#)([String](#) s, int off, int len)

Writes a portion of a String.

Die java.io.BufferedReader Klasse

• Die BufferedReader Klasse

- muss muss mit einem Input-Source kombiniert werden.
- ermöglicht das Lesen von Daten in Text-Format.
- kann keine Zahlen wie z.B. int, long, double lesen.
- BufferedReader read Methoden lesen Zeichen/Array von Zeichen.
- readLine() liefert null, wenn keine input mehr vorhanden ist.
- BufferedReader read Methoden können IOException auswerfen.

• BufferedReader Beispiel

```
FileReader f = new FileReader("test.txt");
BufferedReader in = new BufferedReader(f);
String name ;
while( (name=in.readLine()) != null){
do something with the data.
}
out.close();
```

BufferedReader aus dem JAVA-API 6

Constructor Summary

[BufferedReader](#)([Reader](#) in)

Creates a buffering character-input stream that uses a default-sized input buffer.

[BufferedReader](#)([Reader](#) in, int sz)

Creates a buffering character-input stream that uses an input buffer of the specified size.

Method Summary

void [close](#)()

Closes the stream and releases any system resources associated with it.

void [mark](#)(int readAheadLimit)

Marks the present position in the stream.

boolean [markSupported](#)()

Tells whether this stream supports the mark() operation, which it does.

int [read](#)()

Reads a single character.

int [read](#)(char[] cbuf, int off, int len)

Reads characters into a portion of an array.

[String](#) [readLine](#)()

Reads a line of text.

boolean [ready](#)()

Tells whether this stream is ready to be read.

void [reset](#)()

Resets the stream to the most recent mark.

long [skip](#)(long n)

Die java.util.PrintWriter Klasse

• Die PrintWriter Klasse

- ermöglicht das Schreiben von Strings und Zahlen in Text-Format.
- ihre Methoden: print, println, printf schreiben ein output in Text-Format.
- PrintWriter Methoden werfen kein Exception, deshalb checkError() aufrufen.
- boolean checkError() Überprüfen ob etwas beim Schreiben nicht stimmte.

• PrintWriter Beispiel

```
PrintWriter pwOut = new PrintWriter("test.txt");
String name="Lara Hennicker"
int alter = 25;
pwOut.print(name);
pwOut.print(' ');
pwOut.println(alter);
```

PrintWriter Konstruktoren (Aus dem JAVA-API 6)

Constructor Summary

`PrintWriter`([File](#) file)

Creates a new `PrintWriter`, without automatic line flushing, with the specified file.

`PrintWriter`([File](#) file, [String](#) csn)

Creates a new `PrintWriter`, without automatic line flushing, with the specified file and charset.

`PrintWriter`([OutputStream](#) out)

Creates a new `PrintWriter`, without automatic line flushing, from an existing `OutputStream`.

`PrintWriter`([OutputStream](#) out, boolean autoFlush)

Creates a new `PrintWriter` from an existing `OutputStream`.

`PrintWriter`([String](#) fileName)

Creates a new `PrintWriter`, without automatic line flushing, with the specified file name.

`PrintWriter`([String](#) fileName, [String](#) csn)

Creates a new `PrintWriter`, without automatic line flushing, with the specified file name and charset.

`PrintWriter`([Writer](#) out)

Creates a new `PrintWriter`, without automatic line flushing.

`PrintWriter`([Writer](#) out, boolean autoFlush)

Creates a new `PrintWriter`.

PrintWriter Methoden I (Aus dem JAVA-API 6)

Method Summary

PrintWriter	append (char c) Appends the specified character to this writer.
PrintWriter	append (CharSequence csq) Appends the specified character sequence to this writer.
PrintWriter	append (CharSequence csq, int start, int end) Appends a subsequence of the specified character sequence to this writer.
boolean	checkError () Flushes the stream if it's not closed and checks its error state.
protected void	clearError () Clears the error state of this stream.
void	close () Closes the stream and releases any system resources associated with it.
void	flush () Flushes the stream.
PrintWriter	format (Locale l, String format, Object ... args) Writes a formatted string to this writer using the specified format string and arguments.
PrintWriter	format (String format, Object ... args) Writes a formatted string to this writer using the specified format string and arguments.
void	print (boolean b) Prints a boolean value.
void	print (char c) Prints a character.
void	print (char[] s) Prints an array of characters.

PrintWriter Methoden II (Aus dem JAVA-API 6)

`void print(double d)`
Prints a double-precision floating-point number.

`void print(float f)`
Prints a floating-point number.

`void print(int i)`
Prints an integer.

`void print(long l)`
Prints a long integer.

`void print(Object obj)`
Prints an object.

`void print(String s)`
Prints a string.

`PrintWriter printf(Locale l, String format, Object... args)`
A convenience method to write a formatted string to this writer using the specified format string and arguments.

`PrintWriter printf(String format, Object... args)`
A convenience method to write a formatted string to this writer using the specified format string and arguments.

`void println()`
Terminates the current line by writing the line separator string.

`void println(boolean x)`
Prints a boolean value and then terminates the line.

`void println(char x)`
Prints a character and then terminates the line.

`void println(char[] x)`
Prints an array of characters and then terminates the line.

PrintWriter Methoden III (Aus dem JAVA-API 6)

void	println (double x) Prints a double-precision floating-point number and then terminates the line.
void	println (float x) Prints a floating-point number and then terminates the line.
void	println (int x) Prints an integer and then terminates the line.
void	println (long x) Prints a long integer and then terminates the line.
void	println (Object x) Prints an Object and then terminates the line.
void	println (String x) Prints a String and then terminates the line.
protected void	setError () Indicates that an error has occurred.
void	write (char[] buf) Writes an array of characters.
void	write (char[] buf, int off, int len) Writes A Portion of an array of characters.
void	write (int c) Writes a single character.
void	write (String s) Writes a string.
void	write (String s, int off, int len) Writes a portion of a string.

Die java.util.Scanner Klasse

- Die Scanner Klasse

- ermöglicht das Lesen von Strings und Zahlen in Text-Format.
- `nextLine()` liefert null, wenn keine input mehr vorhanden ist.
- `nextInt()`, `nextLong()`, `nextDouble()` zum lesen von Zahlen in Textformat.
- Lesen aus der Console `scanner in = new Scanner(System.in);`
- Scanner Methoden werfen kein `IOException` aus.

- `PrintWriter` Beispiel

```
Scanner in = new Scanner("test.txt");  
String entry = in.nextLine();  
double id = in.nextDouble();
```

Scanner Konstruktoren (Aus dem JAVA-API 6)

Constructor Summary

Scanner([File](#) source)

Constructs a new `Scanner` that produces values scanned from the specified file.

Scanner([File](#) source, [String](#) charsetName)

Constructs a new `Scanner` that produces values scanned from the specified file.

Scanner([InputStream](#) source)

Constructs a new `Scanner` that produces values scanned from the specified input stream.

Scanner([InputStream](#) source, [String](#) charsetName)

Constructs a new `Scanner` that produces values scanned from the specified input stream.

Scanner([Readable](#) source)

Constructs a new `Scanner` that produces values scanned from the specified source.

Scanner([ReadableByteChannel](#) source)

Constructs a new `Scanner` that produces values scanned from the specified channel.

Scanner([ReadableByteChannel](#) source, [String](#) charsetName)

Constructs a new `Scanner` that produces values scanned from the specified channel.

Scanner([String](#) source)

Constructs a new `Scanner` that produces values scanned from the specified string.

Scanner Methoden aus dem JAVA-API 6)

Method Summary

void	close() Closes this scanner.
Pattern	delimiter() Returns the <code>Pattern</code> this scanner is currently using to match delimiters.
String	findInLine(Pattern pattern) Attempts to find the next occurrence of the specified pattern ignoring delimiters.
String	findInLine(String pattern) Attempts to find the next occurrence of a pattern constructed from the specified string, ignoring delimiters.
String	findWithinHorizon(Pattern pattern, int horizon) Attempts to find the next occurrence of the specified pattern.
String	findWithinHorizon(String pattern, int horizon) Attempts to find the next occurrence of a pattern constructed from the specified string, ignoring delimiters.
boolean	hasNext() Returns true if this scanner has another token in its input.
boolean	hasNext(Pattern pattern) Returns true if the next complete token matches the specified pattern.
boolean	hasNext(String pattern) Returns true if the next token matches the pattern constructed from the specified string.
boolean	hasNextBigDecimal() Returns true if the next token in this scanner's input can be interpreted as a <code>BigDecimal</code> using the nextBigDecimal() method.



Scanner Methoden aus dem JAVA-API 6)

boolean	<code>hasNextBigInteger()</code> Returns true if the next token in this scanner's input can be interpreted as a <code>BigInteger</code> in the default radix using the <code>nextBigInteger()</code> method.
boolean	<code>hasNextBigInteger(int radix)</code> Returns true if the next token in this scanner's input can be interpreted as a <code>BigInteger</code> in the specified radix using the <code>nextBigInteger()</code> method.
boolean	<code>hasNextBoolean()</code> Returns true if the next token in this scanner's input can be interpreted as a boolean value using a case insensitive pattern created from the string "true false".
boolean	<code>hasNextByte()</code> Returns true if the next token in this scanner's input can be interpreted as a byte value in the default radix using the <code>nextByte()</code> method.
boolean	<code>hasNextByte(int radix)</code> Returns true if the next token in this scanner's input can be interpreted as a byte value in the specified radix using the <code>nextByte()</code> method.
boolean	<code>hasNextDouble()</code> Returns true if the next token in this scanner's input can be interpreted as a double value using the <code>nextDouble()</code> method.
boolean	<code>hasNextFloat()</code> Returns true if the next token in this scanner's input can be interpreted as a float value using the <code>nextFloat()</code> method.

Scanner Methoden aus dem JAVA-API 6)

boolean	<code>hasNextInt()</code> Returns true if the next token in this scanner's input can be interpreted as an int value in the default radix using the <code>nextInt()</code> method.
boolean	<code>hasNextInt(int radix)</code> Returns true if the next token in this scanner's input can be interpreted as an int value in the specified radix using the <code>nextInt()</code> method.
boolean	<code>hasNextLine()</code> Returns true if there is another line in the input of this scanner.
boolean	<code>hasNextLong()</code> Returns true if the next token in this scanner's input can be interpreted as a long value in the default radix using the <code>nextLong()</code> method.
boolean	<code>hasNextLong(int radix)</code> Returns true if the next token in this scanner's input can be interpreted as a long value in the specified radix using the <code>nextLong()</code> method.
boolean	<code>hasNextShort()</code> Returns true if the next token in this scanner's input can be interpreted as a short value in the default radix using the <code>nextShort()</code> method.
boolean	<code>hasNextShort(int radix)</code> Returns true if the next token in this scanner's input can be interpreted as a short value in the specified radix using the <code>nextShort()</code> method.
<code>IOException</code>	<code>ioException()</code> Returns the <code>IOException</code> last thrown by this scanner's underlying <code>Readable</code> .

Scanner Methoden aus dem JAVA-API 6)

MatchResult	match() Returns the match result of the last scanning operation performed by this scanner.
String	next() Finds and returns the next complete token from this scanner.
String	next(Pattern pattern) Returns the next token if it matches the specified pattern.
String	next(String pattern) Returns the next token if it matches the pattern constructed from the specified string.
BigDecimal	nextBigDecimal() Scans the next token of the input as a BigDecimal .
BigInteger	nextBigInteger() Scans the next token of the input as a BigInteger .
BigInteger	nextBigInteger(int radix) Scans the next token of the input as a BigInteger .
boolean	nextBoolean() Scans the next token of the input into a boolean value and returns that value.
byte	nextByte() Scans the next token of the input as a byte .
byte	nextByte(int radix) Scans the next token of the input as a byte .
double	nextDouble() Scans the next token of the input as a double .

Scanner Methoden aus dem JAVA-API 6)

float	nextFloat() Scans the next token of the input as a <code>float</code> .
int	nextInt() Scans the next token of the input as an <code>int</code> .
int	nextInt(int radix) Scans the next token of the input as an <code>int</code> .
String	nextLine() Advances this scanner past the current line and returns the input that was skipped.
long	nextLong() Scans the next token of the input as a <code>long</code> .
long	nextLong(int radix) Scans the next token of the input as a <code>long</code> .
short	nextShort() Scans the next token of the input as a <code>short</code> .
short	nextShort(int radix) Scans the next token of the input as a <code>short</code> .
int	radix() Returns this scanner's default radix.
void	remove() The remove operation is not supported by this implementation of <code>Iterator</code> .
Scanner	reset() Resets this scanner.
Scanner	skip(Pattern pattern)

Scanner Methoden aus dem JAVA-API 6)

<code>void</code>	<code>remove()</code> The remove operation is not supported by this implementation of <code>Iterator</code> .
<code>Scanner</code>	<code>reset()</code> Resets this scanner.
<code>Scanner</code>	<code>skip(Pattern pattern)</code> Skips input that matches the specified pattern, ignoring delimiters.
<code>Scanner</code>	<code>skip(String pattern)</code> Skips input that matches a pattern constructed from the specified string.
<code>String</code>	<code>toString()</code> Returns the string representation of this <code>Scanner</code> .
<code>Scanner</code>	<code>useDelimiter(Pattern pattern)</code> Sets this scanner's delimiting pattern to the specified pattern.
<code>Scanner</code>	<code>useDelimiter(String pattern)</code> Sets this scanner's delimiting pattern to a pattern constructed from the specified <code>String</code> .
<code>Scanner</code>	<code>useLocale(Locale locale)</code> Sets this scanner's locale to the specified locale.
<code>Scanner</code>	<code>useRadix(int radix)</code> Sets this scanner's default radix to the specified radix.

Aufgabe: Objekte in Textformat speichern

- Programmieren Sie die Klasse Student
 - Student hat einen Name, studyId, und department.
 - Student hat Konstruktoren und Setter/Getter Methoden.
- Speichern Sie ein Objekt/Array von Objekten in Text-Format
 - `writeData(BufferedWriter out)`: speichert ein Objekt in Text-Format.
 - `writeData(BufferedWriter out, Student[] s)`: speichert ein Array in Text-Format
 - `readData(BufferedReader in)`: liest ein Objekt in Text-Format aus einer Datei.
 - `readData(BufferedReader, boolean all)`: liest ein Array in Text-Format.

Interfaces zum Schreiben/Lesen in binärem Format

- Interfaces zum Schreiben/Lesen in binärem Format
 - **DataOutput Interface** definiert Methoden zum Schreiben in binärem Format
 - **DataInput Interface** definiert Methoden zum Lesen in binärem Format
- Implementation von DataOutput und DataInput Interfaces
 - **DataOutputStream Klasse** implementiert das DataOutput Interface
 - **DataInputStream Klasse** implementiert das DataInput Interface
- Die wichtigsten DataOutput/DataInput Methoden in Überblick

DataInput Methoden	DataOutput Methoden
readBoolean	writeBoolean
readByte	writeByte
readChar	writeChar
readShort	writeShort
readInt	writeInt
readLong	writeLong
readFloat	writeFloat
readDouble	writeDouble
readUTF	writeUTF

Interfaces zum Schreiben/Lesen in binärem Format

- Interfaces zum Schreiben/Lesen in binärem Format
 - **DataOutput Interface** definiert Methoden zum Schreiben in binärem Format
 - **DataInput Interface** definiert Methoden zum Lesen in binärem Format
- Implementation von DataOutput und DataInput Interfaces
 - **DataOutputStream Klasse** implementiert das DataOutput Interface
 - **DataInputStream Klasse** implementiert das DataInput Interface
- Die wichtigsten DataOutput/DataInput Methoden in Überblick

DataInput Methoden	DataOutput Methoden
readBoolean	writeBoolean
readByte	writeByte
readChar	writeChar
readShort	writeShort
readInt	writeInt
readLong	writeLong
readFloat	writeFloat
readDouble	writeDouble
readUTF	writeUTF

Interfaces zum Schreiben/Lesen in binärem Format

- Interfaces zum Schreiben/Lesen in binärem Format
 - **DataOutput Interface** definiert Methoden zum Schreiben in binärem Format
 - **DataInput Interface** definiert Methoden zum Lesen in binärem Format
- Implementation von DataOutput und DataInput Interfaces
 - **DataOutputStream Klasse** implementiert das DataOutput Interface
 - **DataInputStream Klasse** implementiert das DataInput Interface
- Die wichtigsten DataOutput/DataInput Methoden in Überblick

DataInput Methoden	DataOutput Methoden
readBoolean	writeBoolean
readByte	writeByte
readChar	writeChar
readShort	writeShort
readInt	writeInt
readLong	writeLong
readFloat	writeFloat
readDouble	writeDouble
readUTF	writeUTF

Interfaces zum Schreiben/Lesen in binärem Format

- Interfaces zum Schreiben/Lesen in binärem Format
 - **DataOutput Interface** definiert Methoden zum Schreiben in binärem Format
 - **DataInput Interface** definiert Methoden zum Lesen in binärem Format
- Implementation von DataOutput und DataInput Interfaces
 - **DataOutputStream Klasse** implementiert das DataOutput Interface
 - **DataInputStream Klasse** implementiert das DataInput Interface
- Die wichtigsten DataOutput/DataInput Methoden in Überblick

DataInput Methoden	DataOutput Methoden
readBoolean	writeBoolean
readByte	writeByte
readChar	writeChar
readShort	writeShort
readInt	writeInt
readLong	writeLong
readFloat	writeFloat
readDouble	writeDouble
readUTF	writeUTF

Interfaces zum Schreiben/Lesen in binärem Format

- Interfaces zum Schreiben/Lesen in binärem Format
 - **DataOutput Interface** definiert Methoden zum Schreiben in binärem Format
 - **DataInput Interface** definiert Methoden zum Lesen in binärem Format
- Implementation von DataOutput und DataInput Interfaces
 - **DataOutputStream Klasse** implementiert das DataOutput Interface
 - **DataInputStream Klasse** implementiert das DataInput Interface
- Die wichtigsten DataOutput/DataInput Methoden in Überblick

DataInput Methoden	DataOutput Methoden
readBoolean	writeBoolean
readByte	writeByte
readChar	writeChar
readShort	writeShort
readInt	writeInt
readLong	writeLong
readFloat	writeFloat
readDouble	writeDouble
readUTF	writeUTF

Interfaces zum Schreiben/Lesen in binärem Format

- Interfaces zum Schreiben/Lesen in binärem Format
 - **DataOutput Interface** definiert Methoden zum Schreiben in binärem Format
 - **DataInput Interface** definiert Methoden zum Lesen in binärem Format
- Implementation von DataOutput und DataInput Interfaces
 - **DataOutputStream Klasse** implementiert das DataOutput Interface
 - **DataInputStream Klasse** implementiert das DataInput Interface
- Die wichtigsten DataOutput/DataInput Methoden in Überblick

DataInput Methoden	DataOutput Methoden
readBoolean	writeBoolean
readByte	writeByte
readChar	writeChar
readShort	writeShort
readInt	writeInt
readLong	writeLong
readFloat	writeFloat
readDouble	writeDouble
readUTF	writeUTF

Regeln zum Schreiben/Lesen in binärem Format

- Anzahl geschriebener Bytes
 - `writeln(int i)` schreibt immer 4 bytes.
 - `writeln(double d)` schreibt immer 8 bytes.
 - Ergebnis ist für Menschen nicht lesbar.
- Die `writeUTF` Methode
 - schreibt String-Daten mit modifizierten 8-bit Unicode Version.
 - Strings werden zuerst in UTF-16 kodiert dann in UTF8 konvertiert.
 - die modifizierte UTF8 Version ist ab 0xFFFF anders als Standard UTF8.
 - modifizierte UTF8 ist zu alten VM ohne UTF-16 abwärts Kompatibel.
 - Programme, die bytecodes generieren sollten `writeChars` benutzen.

Regeln zum Schreiben/Lesen von binären Daten

- **Big-Endian und Little-Endian Speicher-Methoden**
 - Big-Endian: speichert zuerst das Byte mit der höchsten Signifikant.
 - Little-Endian: speichert zuerst das Byte mit der niedrigsten Signifikant.
 - Big-Endian für int 1234: 00 00 04 D2 ($1234=4*256+13*16+2 =04D2$)
 - Little-Endian für int 1234: D2 04 00 00 ($1234=4*256+13*16+2 =04D2$)
 - Java benutzt Big-Endian, C++ little Endian Methode.
- **DataOutputStream/DataInputStream anwenden**
 - `DataOutputStream out = new FileOutputStream(new FileOutputStream("test.dat"));`
 - `DataInputStream in = new DataInputStream(new FileInputStream("test.dat"));`

DataOutput Interface aus dem JAVA API 6.0

Method Summary

void	write (byte[] b)	Writes to the output stream all the bytes in array <code>b</code> .
void	write (byte[] b, int off, int len)	Writes <code>len</code> bytes from array <code>b</code> , in order, to the output stream.
void	write (int b)	Writes to the output stream the eight low-order bits of the argument <code>b</code> .
void	writeBoolean (boolean v)	Writes a <code>boolean</code> value to this output stream.
void	writeByte (int v)	Writes to the output stream the eight low-order bits of the argument <code>v</code> .
void	writeBytes (String s)	Writes a string to the output stream.
void	writeChar (int v)	Writes a <code>char</code> value, which is comprised of two bytes, to the output stream.
void	writeChars (String s)	Writes every character in the string <code>s</code> , to the output stream, in order, two bytes per character.

DataOutput Interface aus dem JAVA API 6.0

void	<code>writeChar</code> (int v) Writes a <code>char</code> value, which is comprised of two bytes, to the output stream.
void	<code>writeChars</code> (String s) Writes every character in the string <code>s</code> , to the output stream, in order, two bytes per character.
void	<code>writeDouble</code> (double v) Writes a <code>double</code> value, which is comprised of eight bytes, to the output stream.
void	<code>writeFloat</code> (float v) Writes a <code>float</code> value, which is comprised of four bytes, to the output stream.
void	<code>writeInt</code> (int v) Writes an <code>int</code> value, which is comprised of four bytes, to the output stream.
void	<code>writeLong</code> (long v) Writes a <code>long</code> value, which is comprised of eight bytes, to the output stream.
void	<code>writeShort</code> (int v) Writes two bytes to the output stream to represent the value of the argument.
void	<code>writeUTF</code> (String s) Writes two bytes of length information to the output stream, followed by the modified UTF-8 representation of every character in the string <code>s</code> .

DataInput Interface aus dem JAVA API 6.0

Method Summary

boolean	<u>readBoolean()</u> Reads one input byte and returns <code>true</code> if that byte is nonzero, <code>false</code> if that byte is zero.
byte	<u>readByte()</u> Reads and returns one input byte.
char	<u>readChar()</u> Reads two input bytes and returns a <code>char</code> value.
double	<u>readDouble()</u> Reads eight input bytes and returns a <code>double</code> value.
float	<u>readFloat()</u> Reads four input bytes and returns a <code>float</code> value.
void	<u>readFully(byte[] b)</u> Reads some bytes from an input stream and stores them into the buffer array <code>b</code> .
void	<u>readFully(byte[] b, int off, int len)</u> Reads <code>len</code> bytes from an input stream.
int	<u>readInt()</u> Reads four input bytes and returns an <code>int</code> value.

DataInput Interace aus dem JAVA API 6.0

<code>int</code>	<p><code>readInt()</code> Reads four input bytes and returns an <code>int</code> value.</p>
<code>String</code>	<p><code>readLine()</code> Reads the next line of text from the input stream.</p>
<code>long</code>	<p><code>readLong()</code> Reads eight input bytes and returns a <code>long</code> value.</p>
<code>short</code>	<p><code>readShort()</code> Reads two input bytes and returns a <code>short</code> value.</p>
<code>int</code>	<p><code>readUnsignedByte()</code> Reads one input byte, zero-extends it to type <code>int</code>, and returns the result, which is therefore in the range 0 through 255.</p>
<code>int</code>	<p><code>readUnsignedShort()</code> Reads two input bytes and returns an <code>int</code> value in the range 0 through 65535.</p>
<code>String</code>	<p><code>readUTF()</code> Reads in a string that has been encoded using a modified UTF-8 format.</p>
<code>int</code>	<p><code>skipBytes(int n)</code> Makes an attempt to skip over <code>n</code> bytes of data from the input stream, discarding the skipped bytes.</p>

Aufgabe: Speichern in binärem Format

- **Programmieren Sie die Klasse Student**
 - Student hat einen Name, studyId, und department.
 - Student hat Konstruktoren und Setter/Getter Methoden.
- **Speichern Sie ein Objekt/Array von Objekten in Text-Format**
 - `writeData(PrintOutput out)`: speichert ein Objekt in Text-Format.
 - `writeData(PrintOutput out, Student[] s)`: speichert ein Array in Text-Format
 - `readData(Scanner in)`: liest ein Objekt in Text-Format.
 - `readData(Scanner in, boolean all)`: liest ein Array in Text-Format.
- **Speichern Sie ein Objekt/Array von Objekten in binärem Format**
 - `writeData(DataOutputStream out)`: speichert ein Objekt in binärem-Format.
 - `writeData(DataOutputStream out, Student[] s)`: speichert ein Array(Binär)
 - `readData(DataInputStream in)`: liest ein Objekt in binärem-Format.
 - `readData(DataInputStream in, boolean all)`: liest ein Array(Binär)

Random-Access

- Random Access Prinzip

- Random Access = Lesen/Schreiben aus einer/in eine beliebiger Position.
- Random Access ermöglicht einen schnellen Zugriff auf die Einträge.
- Für Random Access bietet Java die Klasse `RandomAccessFile` an.

- Die `RandomAccessFile` Klasse

- realisiert das `RandomAccess` Verhalten.
- implementiert `DataOutput` und `DataInput` Interfaces.
- verfügt daher über die `write` und `read` Methoden.

- `RandomAccessFile` Lese/Schreibzugriff Objekte definieren

- Lesezugriff: `RandomAccessFile f = new RandomAccessFile("x.dat", "r");`
- Schreibzugriff: `RandomAccessFile f = new RandomAccessFile("x.dat", "w");`
- Lese/Schreibzugriff: `f = new RandomAccessFile("x.dat", "rw");`
- Mit "r" wird eine Lesezugriff "Read" im Konstruktor definiert.
- Mit "w" wird ein Schreibzugriff und mit "rw" beides definiert.
- "x.dat" ist eine beliebige Dateiname

Random-Access

- **Random Access Prinzip**

- Random Access = Lesen/Schreiben aus einer/in eine beliebiger Position.
- Random Access ermöglicht einen schnellen Zugriff auf die Einträge.
- Für Random Access bietet Java die Klasse `RandomAccessFile` an.

- **Die `RandomAccessFile` Klasse**

- realisiert das `RandomAccess` Verhalten.
- implementiert `DataOutput` und `DataInput` Interfaces.
- verfügt daher über die `write` und `read` Methoden.

- **`RandomAccessFile` Lese/Schreibzugriff Objekte definieren**

- Lesezugriff: `RandomAccessFile f = new RandomAccessFile("x.dat", "r");`
- Schreibzugriff: `RandomAccessFile f = new RandomAccessFile("x.dat", "w");`
- Lese/Schreibzugriff: `f = new RandomAccessFile("x.dat", "rw");`
- Mit "r" wird eine Lesezugriff "Read" im Konstruktor definiert.
- Mit "w" wird ein Schreibzugriff und mit "rw" beides definiert.
- "x.dat" ist eine beliebige Dateiname

Random-Access

- Random Access Prinzip

- Random Access = Lesen/Schreiben aus einer/in eine beliebiger Position.
- Random Access ermöglicht einen schnellen Zugriff auf die Einträge.
- Für Random Access bietet Java die Klasse `RandomAccessFile` an.

- Die `RandomAccessFile` Klasse

- realisiert das `RandomAccess` Verhalten.
- implementiert `DataOutput` und `DataInput` Interfaces.
- verfügt daher über die `write` und `read` Methoden.

- `RandomAccessFile` Lese/Schreibzugriff Objekte definieren

- Lesezugriff: `RandomAccessFile f = new RandomAccessFile("x.dat", "r");`
- Schreibzugriff: `RandomAccessFile f = new RandomAccessFile("x.dat", "w");`
- Lese/Schreibzugriff: `f = new RandomAccessFile("x.dat", "rw");`
- Mit "r" wird eine Lesezugriff "Read" im Konstruktor definiert.
- Mit "w" wird ein Schreibzugriff und mit "rw" beides definiert.
- "x.dat" ist eine beliebige Dateiname

Random-Access

- **Random Access Prinzip**

- Random Access = Lesen/Schreiben aus einer/in eine beliebiger Position.
- Random Access ermöglicht einen schnellen Zugriff auf die Einträge.
- Für Random Access bietet Java die Klasse `RandomAccessFile` an.

- **Die `RandomAccessFile` Klasse**

- realisiert das `RandomAccess` Verhalten.
- implementiert `DataOutput` und `DataInput` Interfaces.
- verfügt daher über die `write` und `read` Methoden.

- **`RandomAccessFile` Lese/Schreibzugriff Objekte definieren**

- Lesezugriff: `RandomAccessFile f = new RandomAccessFile("x.dat", "r");`
- Schreibzugriff: `RandomAccessFile f = new RandomAccessFile("x.dat", "w");`
- Lese/Schreibzugriff: `f = new RandomAccessFile("x.dat", "rw");`
- Mit "r" wird eine Lesezugriff "Read" im Konstruktor definiert.
- Mit "w" wird ein Schreibzugriff und mit "rw" beides definiert.
- "x.dat" ist eine beliebige Dateiname

Random-Access

- **Random Access Prinzip**

- Random Access = Lesen/Schreiben aus einer/in eine beliebiger Position.
- Random Access ermöglicht einen schnellen Zugriff auf die Einträge.
- Für Random Access bietet Java die Klasse `RandomAccessFile` an.

- **Die `RandomAccessFile` Klasse**

- realisiert das `RandomAccess` Verhalten.
- implementiert `DataOutput` und `DataInput` Interfaces.
- verfügt daher über die `write` und `read` Methoden.

- **`RandomAccessFile` Lese/Schreibzugriff Objekte definieren**

- Lesezugriff: `RandomAccessFile f = new RandomAccessFile("x.dat", "r");`
- Schreibzugriff: `RandomAccessFile f = new RandomAccessFile("x.dat", "w");`
- Lese/Schreibzugriff: `f = new RandomAccessFile("x.dat", "rw");`
- Mit "r" wird eine Lesezugriff "Read" im Konstruktor definiert.
- Mit "w" wird ein Schreibzugriff und mit "rw" beides definiert.
- "x.dat" ist eine beliebige Dateiname

Random-Access

- **Random Access Prinzip**

- Random Access = Lesen/Schreiben aus einer/in eine beliebiger Position.
- Random Access ermöglicht einen schnellen Zugriff auf die Einträge.
- Für Random Access bietet Java die Klasse `RandomAccessFile` an.

- **Die `RandomAccessFile` Klasse**

- realisiert das `RandomAccess` Verhalten.
- implementiert `DataOutput` und `DataInput` Interfaces.
- verfügt daher über die `write` und `read` Methoden.

- **`RandomAccessFile` Lese/Schreibzugriff Objekte definieren**

- **Lesezugriff:** `RandomAccessFile f = new RandomAccessFile("x.dat", "r");`
- **Schreibzugriff:** `RandomAccessFile f = new RandomAccessFile("x.dat", "w");`
- **Lese/Schreibzugriff:** `f = new RandomAccessFile("x.dat", "rw");`
- Mit "r" wird eine Lesezugriff "Read" im Konstruktor definiert.
- Mit "w" wird ein Schreibzugriff und mit "rw" beides definiert.
- "x.dat" ist eine beliebige Dateiname

Grundlagen zum Arbeiten mit RandomAccessFile

- **RandomAccessFile hat einen File Pointer**

- File Pointer zeigt auf die Position “des nächsten Bytes” im Stream.
- “Das nächste Byte” wird dann gelesen bzw. geschrieben.
- seek(long pos) setzt die File Pointer auf beliebige Position “pos”
- long getFilePointer() liefert die Aktuelle Position, auf die FilePointer zeigt.
- long length() liefert die Länge der Datei in Bytes.

- **Lesen und Schreiben in random-access Datei**

- RandomAccessFile implementiert DataOutput/DataInput, hat ihre Methoden.
- Zum Lesen: DataInput read-Methoden: readInt(), readChar(),..
- Zum Schreiben: DataOutput write-Methoden: writeInt(); writeChar(),..

Grundlagen zum Arbeiten mit RandomAccessFile

- **RandomAccessFile hat einen File Pointer**

- File Pointer zeigt auf die Position “des nächsten Bytes” im Stream.
- “Das nächste Byte” wird dann gelesen bzw. geschrieben.
- `seek(long pos)` setzt die File Pointer auf beliebige Position “pos”
- `long getFilePointer()` liefert die Aktuelle Position, auf die FilePointer zeigt.
- `long length()` liefert die Länge der Datei in Bytes.

- **Lesen und Schreiben in random-access Datei**

- `RandomAccessFile` implementiert `DataOutput/DataInput`, hat ihre Methoden.
- Zum Lesen: `DataInput` read-Methoden: `readInt()`, `readChar()`,..
- Zum Schreiben: `DataOutput` write-Methoden: `writeInt()`; `writeChar()`,..

Grundlagen zum Arbeiten mit RandomAccessFile

- **RandomAccessFile hat einen File Pointer**

- File Pointer zeigt auf die Position “des nächsten Bytes” im Stream.
- “Das nächste Byte” wird dann gelesen bzw. geschrieben.
- `seek(long pos)` setzt die File Pointer auf beliebige Position “pos”
- `long getFilePointer()` liefert die Aktuelle Position, auf die FilePointer zeigt.
- `long length()` liefert die Länge der Datei in Bytes.

- **Lesen und Schreiben in random-access Datei**

- `RandomAccessFile` implementiert `DataOutput/DataInput`, hat ihre Methoden.
- Zum Lesen: `DataInput` read-Methoden: `readInt()`, `readChar()`,..
- Zum Schreiben: `DataOutput` write-Methoden: `writeInt()`; `writeChar()`,..

Grundlagen zum Arbeiten mit RandomAccessFile

- **RandomAccessFile hat einen File Pointer**

- File Pointer zeigt auf die Position “des nächsten Bytes” im Stream.
- “Das nächste Byte” wird dann gelesen bzw. geschrieben.
- `seek(long pos)` setzt die File Pointer auf beliebige Position “pos”
- `long getFilePointer()` liefert die Aktuelle Position, auf die FilePointer zeigt.
- `long length()` liefert die Länge der Datei in Bytes.

- **Lesen und Schreiben in random-access Datei**

- `RandomAccessFile` implementiert `DataOutput/DataInput`, hat ihre Methoden.
- Zum Lesen: `DataInput` read-Methoden: `readInt()`, `readChar()`,..
- Zum Schreiben: `DataOutput` write-Methoden: `writeInt()`; `writeChar()`,..

Elegante Speicherung der Daten in binärem Format

- **Speicherungs-Problem: unterschiedliche Bytelänge bei jedem Eintrag**
 - Integer und Fließkommazahlen haben feste Bytelänge.
 - Strings haben natürlich unterschiedliche Länge.
 - Ergebnis: Einträge haben unterschiedliche Byte-Anzahl.
 - `seek(long pos)` springt zu einem Byte nicht zum Anfang eines Eintrages.
 - Problem: sehr schwer, den gesuchten Eintrag schnell zu finden!
 - Beispiel: "Lara:20:Informatik" und "Marina:20:informatik"
- **Lösung: gleiche Bytelänge bei jedem Eintrag (binäres Format)**
 - Strings in fester Länge speichern z.B. Länge n .
 - hat ein String Länge $L < n$ dann wird $(n-L)$ Sonderzeichen an S angefügt
 - Ergebnis: Alle Einträge haben gleiche Länge.
 - `seek((x-1) * Eintragleänge)` Springt zum Anfang von Eintrag x .
 - Lesen/Überschreiben von Eintrag x ist jetzt sehr einfach.
 - navigiere zum Eintrag X durch `seek((x-1)* Eintragleänge);`
 - lese bzw. (über)schreibe dann den Eintrag X .

Elegante Speicherung der Daten in binärem Format

- **Speicherungs-Problem: unterschiedliche Bytelänge bei jedem Eintrag**
 - Integer und Fließkommazahlen haben feste Bytelänge.
 - Strings haben natürlich unterschiedliche Länge.
 - Ergebnis: Einträge haben unterschiedliche Byte-Anzahl.
 - `seek(long pos)` springt zu einem Byte nicht zum Anfang eines Eintrages.
 - Problem: sehr schwer, den gesuchten Eintrag schnell zu finden!
 - Beispiel: "Lara:20:Informatik" und "Marina:20:informatik"
- **Lösung: gleiche Bytelänge bei jedem Eintrag (binäres Format)**
 - Strings in fester Länge speichern z.B. Länge n .
 - hat ein String Länge $L < n$ dann wird $(n-L)$ Sonderzeichen an S angefügt
 - Ergebnis: Alle Einträge haben gleiche Länge.
 - `seek((x-1) * Eintragleänge)` Springt zum Anfang von Eintrag x .
 - Lesen/Überschreiben von Eintrag x ist jetzt sehr einfach.
 - navigiere zum Eintrag X durch `seek((x-1)* Eintragleänge);`
 - lese bzw. (über)schreibe dann den Eintrag X .

Elegante Speicherung der Daten in binärem Format

- **Speicherungs-Problem: unterschiedliche Bytelänge bei jedem Eintrag**
 - Integer und Fließkommazahlen haben feste Bytelänge.
 - Strings haben natürlich unterschiedliche Länge.
 - Ergebnis: Einträge haben unterschiedliche Byte-Anzahl.
 - `seek(long pos)` springt zu einem Byte nicht zum Anfang eines Eintrages.
 - Problem: sehr schwer, den gesuchten Eintrag schnell zu finden!
 - Beispiel: "Lara:20:Informatik" und "Marina:20:informatik"
- **Lösung: gleiche Bytelänge bei jedem Eintrag (binäres Format)**
 - Strings in fester Länge speichern z.B. Länge n .
 - hat ein String Länge $L < n$ dann wird $(n-L)$ Sonderzeichen an S angefügt
 - Ergebnis: Alle Einträge haben gleiche Länge.
 - `seek((x-1) * Eintragleänge)` Springt zum Anfang von Eintrag x .
 - Lesen/Überschreiben von Eintrag x ist jetzt sehr einfach.
 - navigiere zum Eintrag X durch `seek((x-1)* Eintragleänge);`
 - lese bzw. (über)schreibe dann den Eintrag X .

Elegante Speicherung der Daten in binärem Format

- **Speicherungs-Problem: unterschiedliche Bytelänge bei jedem Eintrag**
 - Integer und Fließkommazahlen haben feste Bytelänge.
 - Strings haben natürlich unterschiedliche Länge.
 - Ergebnis: Einträge haben unterschiedliche Byte-Anzahl.
 - `seek(long pos)` springt zu einem Byte nicht zum Anfang eines Eintrages.
 - Problem: sehr schwer, den gesuchten Eintrag schnell zu finden!
 - Beispiel: "Lara:20:Informatik" und "Marina:20:informatik"
- **Lösung: gleiche Bytelänge bei jedem Eintrag (binäres Format)**
 - Strings in fester Länge speichern z.B. Länge n .
 - hat ein String Länge $L < n$ dann wird $(n-L)$ Sonderzeichen an S angefügt
 - Ergebnis: Alle Einträge haben gleiche Länge.
 - `seek((x-1) * Eintragleänge)` Springt zum Anfang von Eintrag x .
 - Lesen/Überschreiben von Eintrag x ist jetzt sehr einfach.
 - navigiere zum Eintrag X durch `seek((x-1)* Eintragleänge);`
 - lese bzw. (über)schreibe dann den Eintrag X .

X-te Eintrag in random-access Datei lesen/schreiben

- Lesen des x-ten Eintrages (Alle Einträge sind gleicher Länge)

```
RandomAccessFile in =  
new RandomAccessFile("x.dat", "r");  
in.seek((x-1) * ENTRY_SIZE);  
Student s = new Student();  
s.readData(in);
```

- Schreiben des x-ten Eintrages (Alle Einträge sind gleicher Länge)

```
RandomAccessFile out =  
new RandomAccessFile("x.dat", "w");  
in.seek((x-1) * ENTRY_SIZE);  
s.writeData(out);
```

- Anzahl der Einträge berechnen

```
int n = in.length();  
int entrys = (int)(n/ENTRY_SIZE);
```

X-te Eintrag in random-access Datei lesen/schreiben

- Lesen des x-ten Eintrages (Alle Einträge sind gleicher Länge)

```
RandomAccessFile in =  
new RandomAccessFile("x.dat", "r");  
in.seek((x-1) * ENTRY_SIZE);  
Student s = new Student();  
s.readData(in);
```

- Schreiben des x-ten Eintrages (Alle Einträge sind gleicher Länge)

```
RandomAccessFile out =  
new RandomAccessFile("x.dat", "w");  
in.seek((x-1) * ENTRY_SIZE);  
s.writeData(out);
```

- Anzahl der Einträge berechnen

```
int n = in.length();  
int entrys = (int)(n/ENTRY_SIZE);
```

X-te Eintrag in random-access Datei lesen/schreiben

- Lesen des x-ten Eintrages (Alle Einträge sind gleicher Länge)

```
RandomAccessFile in =  
new RandomAccessFile("x.dat", "r");  
in.seek((x-1) * ENTRY_SIZE);  
Student s = new Student();  
s.readData(in);
```

- Schreiben des x-ten Eintrages (Alle Einträge sind gleicher Länge)

```
RandomAccessFile out =  
new RandomAccessFile("x.dat", "w");  
in.seek((x-1) * ENTRY_SIZE);  
s.writeData(out);
```

- Anzahl der Einträge berechnen

```
int n = in.length();  
int entrys = (int)(n/ENTRY_SIZE);
```

X-te Eintrag in random-access Datei lesen/schreiben

- Lesen des x-ten Eintrages (Alle Einträge sind gleicher Länge)

```
RandomAccessFile in =  
new RandomAccessFile("x.dat", "r");  
in.seek((x-1) * ENTRY_SIZE);  
Student s = new Student();  
s.readData(in);
```

- Schreiben des x-ten Eintrages (Alle Einträge sind gleicher Länge)

```
RandomAccessFile out =  
new RandomAccessFile("x.dat", "w");  
in.seek((x-1) * ENTRY_SIZE);  
s.writeData(out);
```

- Anzahl der Einträge berechnen

```
int n = in.length();  
int entrys = (int)(n/ENTRY_SIZE);
```

X-te Eintrag in random-access Datei lesen/schreiben

- Lesen des x-ten Eintrages (Alle Einträge sind gleicher Länge)

```
RandomAccessFile in =  
new RandomAccessFile("x.dat", "r");  
in.seek((x-1) * ENTRY_SIZE);  
Student s = new Student();  
s.readData(in);
```

- Schreiben des x-ten Eintrages (Alle Einträge sind gleicher Länge)

```
RandomAccessFile out =  
new RandomAccessFile("x.dat", "w");  
in.seek((x-1) * ENTRY_SIZE);  
s.writeData(out);
```

- Anzahl der Einträge berechnen

```
int n = in.length;  
int entrys = (int)(n/ENTRY_SIZE);
```

X-te Eintrag in random-access Datei lesen/schreiben

- Lesen des x-ten Eintrages (Alle Einträge sind gleicher Länge)

```
RandomAccessFile in =  
new RandomAccessFile("x.dat", "r");  
in.seek((x-1) * ENTRY_SIZE);  
Student s = new Student();  
s.readData(in);
```

- Schreiben des x-ten Eintrages (Alle Einträge sind gleicher Länge)

```
RandomAccessFile out =  
new RandomAccessFile("x.dat", "w");  
in.seek((x-1) * ENTRY_SIZE);  
s.writeData(out);
```

- Anzahl der Einträge berechnen

```
int n = in.length;  
int entrys = (int)(n/ENTRY_SIZE);
```

RandomAccessFile aus dem JAVA API 6.0

Constructor Summary

[RandomAccessFile](#)([File](#) file, [String](#) mode)

Creates a random access file stream to read from, and optionally to write to, the file specified by the [File](#) argument.

[RandomAccessFile](#)([String](#) name, [String](#) mode)

Creates a random access file stream to read from, and optionally to write to, a file with the specified name.

Method Summary

void	close ()	Closes this random access file stream and releases any system resources associated with the stream.
FileChannel	getChannel ()	Returns the unique FileChannel object associated with this file.
FileDescriptor	getFD ()	Returns the opaque file descriptor object associated with this stream.
long	getFilePointer ()	Returns the current offset in this file.
long	length ()	Returns the length of this file.
int	read ()	Reads a byte of data from this file.

RandomAccessFile aus dem JAVA API 6.0

int	<u>read</u> (byte[] b) Reads up to <code>b.length</code> bytes of data from this file into an array of bytes.
int	<u>read</u> (byte[] b, int off, int len) Reads up to <code>len</code> bytes of data from this file into an array of bytes.
boolean	<u>readBoolean</u> () Reads a <code>boolean</code> from this file.
byte	<u>readByte</u> () Reads a signed eight-bit value from this file.
char	<u>readChar</u> () Reads a character from this file.
double	<u>readDouble</u> () Reads a <code>double</code> from this file.
float	<u>readFloat</u> () Reads a <code>float</code> from this file.
void	<u>readFully</u> (byte[] b) Reads <code>b.length</code> bytes from this file into the byte array, starting at the current file pointer.
void	<u>readFully</u> (byte[] b, int off, int len) Reads exactly <code>len</code> bytes from this file into the byte array, starting at the current file pointer.
int	<u>readInt</u> () Reads a signed 32-bit integer from this file.

RandomAccessFile aus dem JAVA API 6.0

<code>String</code>	<code>readLine()</code> Reads the next line of text from this file.
<code>long</code>	<code>readLong()</code> Reads a signed 64-bit integer from this file.
<code>short</code>	<code>readShort()</code> Reads a signed 16-bit number from this file.
<code>int</code>	<code>readUnsignedByte()</code> Reads an unsigned eight-bit number from this file.
<code>int</code>	<code>readUnsignedShort()</code> Reads an unsigned 16-bit number from this file.
<code>String</code>	<code>readUTF()</code> Reads in a string from this file.
<code>void</code>	<code>seek(long pos)</code> Sets the file-pointer offset, measured from the beginning of this file, at which the next read or write occurs.
<code>void</code>	<code>setLength(long newLength)</code> Sets the length of this file.
<code>int</code>	<code>skipBytes(int n)</code> Attempts to skip over <i>n</i> bytes of input discarding the skipped bytes.

RandomAccessFile aus dem JAVA API 6.0

void	write (byte[] b) Writes <code>b.length</code> bytes from the specified byte array to this file, starting at the current file pointer.
void	write (byte[] b, int off, int len) Writes <code>len</code> bytes from the specified byte array starting at offset <code>off</code> to this file.
void	write (int b) Writes the specified byte to this file.
void	writeBoolean (boolean v) Writes a <code>boolean</code> to the file as a one-byte value.
void	writeByte (int v) Writes a <code>byte</code> to the file as a one-byte value.
void	writeBytes (String s) Writes the string to the file as a sequence of bytes.
void	writeChar (int v) Writes a <code>char</code> to the file as a two-byte value, high byte first.
void	writeChars (String s) Writes a string to the file as a sequence of characters.
void	writeDouble (double v) Converts the double argument to a <code>long</code> using the <code>doubleToLongBits</code> method in class <code>Double</code> , and then writes that <code>long</code> value to the file as an eight-byte quantity, high byte first.

RandomAccessFile aus dem JAVA API 6.0

void	writeChar (int v) Writes a <code>char</code> to the file as a two-byte value, high byte first.
void	writeChars (String s) Writes a string to the file as a sequence of characters.
void	writeDouble (double v) Converts the double argument to a <code>long</code> using the <code>doubleToLongBits</code> method in class <code>Double</code> , and then writes that <code>long</code> value to the file as an eight-byte quantity, high byte first.
void	writeFloat (float v) Converts the float argument to an <code>int</code> using the <code>floatToIntBits</code> method in class <code>Float</code> , and then writes that <code>int</code> value to the file as a four-byte quantity, high byte first.
void	writeInt (int v) Writes an <code>int</code> to the file as four bytes, high byte first.
void	writeLong (long v) Writes a <code>long</code> to the file as eight bytes, high byte first.
void	writeShort (int v) Writes a <code>short</code> to the file as two bytes, high byte first.
void	writeUTF (String str) Writes a string to the file using modified UTF-8 encoding in a machine-independent manner.

Random Access Programmieraufgabe

- **Programmieren Sie die Klasse Person**
 - Person hat einen Name, id, und eine Adresse.
 - Person hat Konstruktoren und Setter/Getter Methoden.
- **Programmieren Sie die Hilfsmethoden**
 - `writeString(DataOutput out, String s, int size)`: speichert s in Länge size.
 - `readString(DataInput in, int size)`: liest ein String s in Länge size.
- **Speichern Sie ein Objekt/Array von Objekten in binärem Format**
 - `writeData/readData` schreiben/lesen Einträge gleicher Länge.
 - `writeData(DataOutputStream out)`: speichert ein Objekt in binärem-Format.
 - `writeData(DataOutputStream out, Student[] s)`: speichert ein Array(Binär)
 - `readData(DataInputStream in)`: liest ein Objekt in binärem-Format.
 - `readData(DataInputStream in, boolean all)`: liest ein Array(Binär)
 - Schreiben Sie ein Array der Länge 5 in eine Datei mit `writeData(out,true)`;
 - Überschreiben Sie den 3-ten Eintrag mit neuen Werten und lesen Sie ihn.