

Ein- und Ausgabe

Datei und Verzeichnis

- ❖ Verwendung findet die Klasse File
- ❖ keine Unterstützung für plattformspezifische Details wie Rechtsverwaltung
- ❖ NIO.2 ist hier eine Erweiterung

Die Klasse File

- ❖ repräsentiert einen Datei-oder Verzeichnisnamen
- ❖ muss nicht physikalisch existieren

```
public static void main(String[] args) {  
    //Konstruktor File(String pathname)  
    File f1 = new File("/Users/schulung/workspace/f1.txt");  
    System.out.println(f1);  
  
    //Konstruktor File(String parent, String child)  
    File f2 = new File(System.getProperty("user.dir"), "f2.txt");  
    System.out.println(f2);  
  
    //Konstruktor File(File file, String child)  
    File verzeichnis = new File(System.getProperty("user.dir"));  
    File f3 = new File(verzeichnis, "f3.txt");  
    System.out.println(f3);  
  
    //Konstruktor File(URI uri)  
    try {  
        URI uri = new URI("file:///Users/schulung/workspace/f4.txt");  
        File f4 = new File(uri);  
        System.out.println(f4);  
    } catch (URISyntaxException e) {  
        e.printStackTrace();  
    }  
}
```

Pfadtrenner

- ❖ plattformabhängig
 - ❖ Windows (»\«), Unix (»/«)
- ❖ zwei öffentliche Konstanten zugänglich

```
char separatorChar = File.separatorChar;
System.out.println(separatorChar);
String separatorString = File.separator; //System.getProperty("file.separator")
System.out.println(separatorString);
```

Namen erfragen und auflösen

```
/*
 * File getCanonicalFile() throws IOException
 * Gibt den Pfadnamen des Dateiobjekts zurück,
 * der keine relativen Pfadangaben mehr enthält.
 * Kann im Gegensatz zu den anderen Pfadmethoden eine IOException aufrufen,
 * da mitunter verbotene Dateizugriffe erfolgen.
 */
try
{
    File f5 = new File("./f5.txt")
        .getCanonicalFile();
    System.out.println( f5 );           // /Users/schulung/workspace/f5.txt
}
catch ( IOException e ) { e.printStackTrace(); }

//getName - gibt den Dateinamen zurück
File f6 = new File("/Users/schulung/workspace/f6.txt");
System.out.println(f6.getName()); // f6.txt

//getPath - gibt den Pfadnamen zurück
File f7 = new File("/Users/schulung/workspace/f7.txt");
System.out.println(f7.getPath()); // /Users/schulung/workspace/f7.txt
```

Namen erfragen und auflösen

```
/*
 * getAbsolutePath
 * getAbsoluteFile
 * Liefert den absoluten Pfad.
 * Ist das Objekt kein absoluter Pfadname,
 * so wird ein Objekt aus dem aktuellen Verzeichnis,
 * einem Separator-Zeichen und dem Dateinamen aufgebaut.
 *
 */
File f8 = new File("/Users/schulung/workspace/f8.txt");
System.out.println(f8.getAbsolutePath()); // /Users/schulung/workspace/f8.txt
System.out.println(f8.getAbsoluteFile()); // /Users/schulung/workspace/f8.txt

//getParentFile - gibt den Pfad des Vorgängers zurück als File.
//getParent - gibt den Pfad des Vorgängers zurück als String.
File f9 = new File("/Users/schulung/workspace/f9.txt");
System.out.println(f9.getParentFile());
System.out.println(f9.getParent());

//isAbsolutePath - Liefert true, wenn der Pfad in der systemabhängigen Notation absolut ist.
File f10 = new File("./f10.txt");
File f10a = new File("/Users/schulung/workspace/f10a.txt");
System.out.println(f10.isAbsolute());
System.out.println(f10a.isAbsolute());
```

Existiert es überhaupt

```
//exists() - Liefert true, wenn das File-Objekt eine existierende Datei
//oder einen existierenden Ordner repräsentiert.
File f11 = new File("/Users/schulung/workspace/f11.txt");
File f11a = new File("/Users/schulung/workspace/");
System.out.println(f11.exists());    //false
System.out.println(f11a.exists());   //true

//isDirectory - Gibt true zurück, wenn es sich um ein Verzeichnis handelt.
File f12 = new File("/Users/schulung/workspace/f12.txt");
File f12a = new File("/Users/schulung/workspace/");
System.out.println(f12.isDirectory()); //false
System.out.println(f12a.isDirectory()); //true

//isFile - Gibt true zurück, wenn es sich um ein Verzeichnis handelt.
File f13 = new File("/Users/schulung/workspace/f13.txt"); //muss aber bereits da sein
File f13a = new File("/Users/schulung/workspace/");
System.out.println(f13.isFile());    //true
System.out.println(f13a.isFile());   //false
```

Verzeichnis- und Dateieigenschaften

```
/*
 * boolean canExecute()
 * boolean canRead()
 * boolean canWrite()
 * Liefert true, wenn die Ausführungsrechte/Leserechte/Schreibrechte gesetzt sind.
 */
File f14e = new File("/Users/schulung/workspace/f14e.txt");
File f14r = new File("/Users/schulung/workspace/f14r.txt");
File f14w = new File("/Users/schulung/workspace/f14w.txt");

System.out.println(f14e.canExecute());
System.out.println(f14r.canRead());
System.out.println(f14w.canWrite());

//length
//Gibt die Länge der Datei in Byte zurück oder 0L,
//wenn die Datei nicht existiert oder es sich um ein Verzeichnis handelt.
File f15 = new File("/Users/schulung/workspace/f15.txt");
System.out.println(f15.length());
```

Änderungsdatum einer Datei

```
//lastModified  
File f16 = new File("/Users/schulung/workspace/f16.txt");  
System.out.println(f16.lastModified()); //als long  
  
//lastModified  
File f17 = new File("/Users/schulung/workspace/f17.txt");  
f17.setLastModified(1400000000000L);  
System.out.println(f17.lastModified()); //als long
```

Umbenennen und Verzeichnisse anlegen

```
//mkdir - Legt das Unterverzeichnis an.  
File f18 = new File("/Users/schulung/workspace/f18");  
System.out.println(f18.mkdir());  
  
//mkdirs - Legt das Unterverzeichnis inklusive weiterer Verzeichnisse an.  
File f19 = new File("/Users/schulung/workspace/f19/1");  
System.out.println(f19.mkdirs());  
  
//renameTo - Benennt die Datei in den Namen um, der durch das File-Objekt d gegeben ist.  
File f20 = new File("/Users/schulung/workspace/f20.txt");  
System.out.println(f20.renameTo(new File("/Users/schulung/workspace/f20a.txt")));
```

Verzeichnisse auflisten

```
/*
 * list()
 * listFiles()
 * Gibt eine Liste der Dateien in einem Verzeichnis als File-Array
 * oder String-Array zurück. Das Feld enthält weder »..« noch »...«.
 */
String[] entries = new File( "/Users/schulung/workspace/" ).list();
System.out.println( Arrays.toString(entries) );

File[] files = new File( "/Users/schulung/workspace/" ).listFiles();
System.out.println( Arrays.toString(files) );
```

Dateien und Verzeichnisse löschen

```
/* delete - Löscht die Datei oder das leere Verzeichnis.  
 * Falls die Datei nicht gelöscht werden konnte,  
 * gibt es keine Ausnahme, sondern den Rückgabewert false.  
 */  
File f21 = new File("/Users/schulung/workspace/f21.txt");  
System.out.println(f21.delete());
```

Übung

- ❖ Uebung1.txt

Datei mit wahlfreiem Zugriff

- ❖ zwei Arten Dateien zu lesen und schreiben
 - ❖ Datenstrom(erzwingt strenge Sequenz)
 - ❖ wahlfreier Zugriff(beliebig hin- und herspringen)
- ❖ wahlfreier Zugriff über RandomAccessFile

RandomAccessFile öffnen/lesen/schreiben

```
File f = new File("/Users/schulung/workspace/ranaf1.txt");
try {
    RandomAccessFile ranaf = new RandomAccessFile(f, "r");
    System.out.println(ranaf.readLine());
    ranaf.close();
} catch (FileNotFoundException e) {
    e.printStackTrace();
} catch (IOException e) {
    e.printStackTrace();
}

//öffnet in rw mode mit String als Parameter
String f1 = "/Users/schulung/workspace/ranaf1.txt";
try {
    RandomAccessFile ranaf1 = new RandomAccessFile(f1, "rw");
    ranaf1.write(116);
    System.out.println(ranaf1.readLine());
    ranaf1.close();
} catch (FileNotFoundException e) {
    e.printStackTrace();
} catch (IOException e) {
    e.printStackTrace();
}
```

RandomAccessFile

```
RandomAccessFile file = null;
try{
    file = new RandomAccessFile("/Users/schulung/workspace/randomaccess.txt","rw");

    //Writing to the file
    file.write("Hello".getBytes());

    System.out.println(file.getFilePointer());
    file.seek(0);      //Go to the begining//Reading from the file
    System.out.println(file.getFilePointer());
    System.out.println((char)file.read());
    System.out.println((char)file.read());
    System.out.println((char)file.read());
    System.out.println((char)file.read());
    System.out.println((char)file.read());

    System.out.println("-----");
    file.seek(2);  //Go to the Second Item
    System.out.println((char)file.read());

    //Go to the end and append false to the file
    file.seek(file.length());
    file.write("RandomAccessFile".getBytes());

    System.out.println("Vorher: "+file.getFilePointer());
    file.seek(4);
    System.out.println(file.getFilePointer());
    System.out.println(file.read());
    file.close();
}catch(Exception e) {}
}
```



NIO

Dateisysteme unter NIO.2

- ❖ Wie lässt sich eine Datei einfach und schnell kopieren?
- ❖ Wie lässt sich eine Datei verschieben, wobei die Semantik auf unterschiedlichen Plattformen immer gleich ist.
- ❖ Wie lässt sich auf eine Änderung im Dateisystem reagieren, sodass ein Callback uns informiert, dass sich eine Datei verändert hat?
- ❖ Wie lässt sich einfach ein Verzeichnis rekursiv ablaufen?
- ❖ Wie lässt sich eine symbolische Verknüpfung anlegen und verfolgen?
- ❖ Wie lässt sich realisieren, dass die File-Operationen abstrahiert werden und nicht nur auf dem lokalen Dateisystem basieren? Wünschenswert ist eine Abstraktion, sodass die gleiche API auch ein virtuelles Dateisystem im Hauptspeicher, entfernte Dateisysteme wie FTP oder ein Repository anspricht.

NIO

```
Path path = Paths.get(URI.create("file:/Users/schulung/workspace/sayings.dat"));
System.out.println("1 "+path.getClass());
System.out.println("2 "+path.getFileSystem().getClass());
//zu vergleichen mit
File file = new File("/Users/schulung/workspace/sayings.dat");
//-----
Path path1 = Paths.get(URI.create("file:/Users/schulung/workspace/sayings.dat"));
InputStream inStream = Files.newInputStream(path);
//zu vergleichen mit
File file1 = new File("/Users/schulung/workspace/sayings.dat");
//-----
Path path2 = Paths.get(URI.create("file:/Users/schulung/workspace/sayings.dat"));
ByteArrayOutputStream out = new ByteArrayOutputStream();
Files.copy(path2, out);
byte[] fileBytes = out.toByteArray();
System.out.println("++");
System.out.println(new String(fileBytes));

//zu vergleichen mit
File file2 = new File("/Users/schulung/workspace/sayings.dat");
FileInputStream in = new FileInputStream(file2);
ByteArrayOutputStream out1 = new ByteArrayOutputStream();
for(int b = in.read(); b > -1; b = in.read()) {
    out1.write(b);
}
byte[] fileBytes1 = out1.toByteArray();
System.out.println("--");
System.out.println(new String(fileBytes1));
```

NIO Link

```
Path path = Paths.get(URI.create("file:/Users/schulung/workspace/datenstroeme/test.txt"));

Path slink = Paths.get(URI.create("file:/Users/schulung/workspace/datenstroeme/slink.txt"));
Files.createSymbolicLink(slink, path);

Path hlink = Paths.get(URI.create("file:/Users/schulung/workspace/datenstroeme/hlink.txt"));
Files.createLink(hlink, path);

Path path2 = Paths.get(URI.create("file:/Users/schulung/workspace/datenstroeme/test.txt"));
if(Files.isSymbolicLink(path2)) {
    System.out.println(Files.readSymbolicLink(path2));
} else {
    System.out.println(path2);
}
```

NIO Berechtigungen

```
Path path3 = Paths.get(URI.create("file:/Users/schulung/workspace/datenstroeme/test.txt"));
Set<PosixFilePermission> permissions = Files.getPosixFilePermissions(path3);
boolean groupHasPermission = permissions.contains(PosixFilePermission.GROUP_READ);
permissions.remove(PosixFilePermission.GROUP_READ);
System.out.println(groupHasPermission);
boolean groupHasStillPermission = permissions.contains(PosixFilePermission.GROUP_READ);
System.out.println(groupHasStillPermission);
```

NIO Notification

```
Path path3 = Paths.get(URI.create("file:/Users/schulung/workspace/datenstroeme/test.txt"));
Set<PosixFilePermission> permissions = Files.getPosixFilePermissions(path3);
boolean groupHasPermission = permissions.contains(PosixFilePermission.GROUP_READ);
permissions.remove(PosixFilePermission.GROUP_READ);
System.out.println(groupHasPermission);
boolean groupHasStillPermission = permissions.contains(PosixFilePermission.GROUP_READ);
System.out.println(groupHasStillPermission);
```

Stream-Klassen und Reader/Writer

FileInputStream, FileReader, FileOutputStream, FileWriter

| | Bytes (oder Byte- Arrays) | Zeichen (oder Zeichen-Arrays, Strings) |
|----------------------|--------------------------------------|---|
| Aus Dateien lesen | FileInputStream | FileReader |
| In Dateien schreiben | FileOutputStream | FileWriter |

FileWriter

```
Writer fw = null;
try {
    fw = new FileWriter("fileWriter.txt");
    fw.write("Java macht Spaß...");
    fw.append(System.getProperty("line.separator")); // e.g. "\n"
} catch (IOException e) { // Konstruktor und write,append schmeißen IOException
    System.err.println("Konnte Datei nicht erstellen");
} finally {
    if (fw != null)
        try {
            fw.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
}
```

FileWriter

```
try {
    //true - Erzeugt einen Ausgabestrom und hängt die Daten an eine existierende Datei an
    fw = new FileWriter("fileWriter.txt", true);
    fw.write("...und erfordert Übung");
    fw.append(System.getProperty("line.separator")); // e.g. "\n"
} catch (IOException e) { // Konstruktor und write,append schmeißen IOException
    System.err.println("Konnte Datei nicht erstellen");
} finally {
    if (fw != null)
        try {
            fw.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
}
```

FileWriter mit FileDescriptor

```
try {
    fw = new FileWriter(FileDescriptor.out);
    fw.write("...ab auf die Konsole");
    fw.append(System.getProperty("line.separator")); // e.g. "\n"
} catch (IOException e) { // Konstruktor und write,append schmeißen IOException
    System.err.println("Konnte Datei nicht erstellen");
} finally {
    if (fw != null)
        try {
            fw.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
}
```

FileWriter Methoden beispielhaft

```
fw.write("...ab auf die Konsole");
//weitere Methoden
//einzelnes char anfügen
fw.append('c');
//char array schreiben
char[] c = {'a', 'b'};
fw.write(c);
//oder einen String schreiben
fw.write(" passt scho");
```

FileReader

```
Reader reader = null;
try
{
    reader = new FileReader( "fileWriter.txt" );

    for ( int c; ( c = reader.read() ) != -1; )
        System.out.print( (char) c );
}
catch ( IOException e ) {
    System.err.println( "Fehler beim Lesen der Datei!" );
}
finally {
    try { reader.close(); } catch ( Exception e ) { e.printStackTrace(); }
}
```

FileReader - weitere Konstruktion

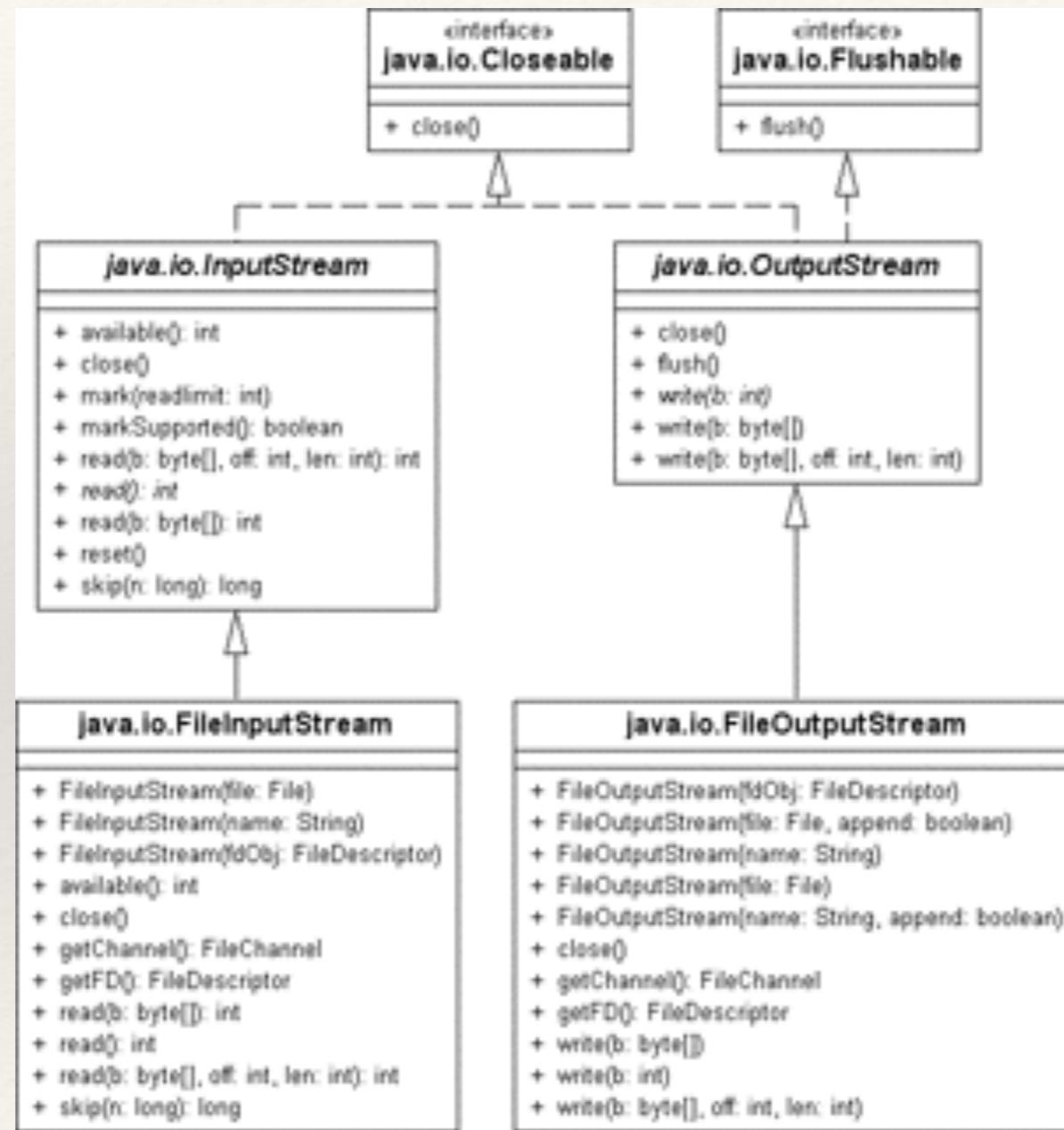
```
try
{
    reader = new FileReader( FileDescriptor.in);

    for ( int c; ( c = reader.read() ) != -1; )
        System.out.print( (char) c );
}
catch ( IOException e ) {
    System.err.println( "Fehler beim Lesen der Datei!" );
}
finally {
    try { reader.close(); } catch ( Exception e ) { e.printStackTrace(); }
}
```

Kopieren mit FileOutputStream und FileInputStream

- ❖ FileOutputStream bietet grundsätzliche Methoden, um in Dateien zu schreiben
- ❖ FileInputStream ist der Gegenspieler und dient zum Lesen der Binärdateien.

Klassendiagramm



Vereinfachung mit NIO

```
-----  
try ( OutputStream out = Files.newOutputStream( Paths.get( "nioDatei.txt" ) ) )  
{  
    out.write( "alles ok".getBytes() );  
}  
catch ( IOException e )  
{  
    e.printStackTrace();  
}
```

OpenOption

| OpenOption | Beschreibung |
|---|---|
| java.nio.file.StandardOpenOption | |
| READ | Öffnen für Lesezugriff |
| WRITE | Öffnen für Schreibzugriff |
| APPEND | Neue Daten kommen an das Ende. Atomar bei parallelen Schreiboperationen. |
| TRUNCATE_EXISTING | Für Schreiber: Existiert die Datei, wird die Länge vorher auf 0 gesetzt. |
| CREATE | Legt Datei an, falls sie noch nicht existiert. |
| CREATE_NEW | Legt Datei nur an, falls sie vorher noch nicht existierte. |
| DELETE_ON_CLOSE | Die Java-Bibliothek versucht, die Datei zu löschen, wenn sie geschlossen wird. |
| SPARSE | Hinweis für das Dateisystem, die Datei kompakt zu speichern, da sie aus vielen Null-Bytes besteht |
| SYNC | Jeder Schreibzugriff und jedes Update der Metadaten soll sofort zum Dateisystem. |
| DSYNC | Jeder Schreibzugriff soll sofort zum Dateisystem. |
| java.nio.file.LinkOption | |
| NOFOLLOW_LINKS | Symbolischen Links wird nicht gefolgt. |

Basisklasse für die Ein- und Ausgabe

- ❖ drei zentrale Prinzipien der Strom-Klassen im java.io
 - ❖ abstrakte Basisklassen, die Operationen für Ein-und Ausgabe vorschreiben
 - ❖ abstrakten Basisklassen gibt es einmal für Unicode-Zeichen und einmal für Bytes
 - ❖ Implementierungen entweder konkrete Ein-/ Ausgabe in eine bestimmte Resource oder sind Filter.

Basisklassen für Ein- und Ausgabe

| Basisklasse für | Bytes (oder Byte-Arrays) | Zeichen (oder Zeichen-Arrays) |
|------------------------|---------------------------------|--------------------------------------|
| Eingabe | InputStream | Reader |
| Ausgabe | OutputStream | Writer |

Wichtige Eingabeklassen

| Byte-Stream-Klasse für die Eingabe | Zeichen-Stream-Klasse für die Eingabe | Beschreibung |
|------------------------------------|---------------------------------------|--|
| InputStream | Reader | Abstrakte Klasse für Zeicheneingabe und Byte-Arrays |
| BufferedInputStream | BufferedReader | Puffert die Eingabe. |
| LineNumberInputStream? | LineNumberReader | Merkt sich Zeilennummern beim Lesen. |
| ByteArrayInputStream | CharArrayReader | Liest Zeichen-Arrays oder Byte-Arrays. |
| (keine Entsprechung) | InputStreamReader | Wandelt einen Byte-Stream in einen Zeichen-Stream um. Diese Klasse ist das Bindeglied zwischen Byte und Zeichen. |
| DataInputStream | (keine Entsprechung) | Liest Primitive und auch UTF-8. |
| FilterInputStream | FilterReader | Abstrakte Klasse für gefilterte Eingabe |
| PushbackInputStream | PushbackReader | Erlaubt, gelesene Zeichen wieder in den Stream zu geben. |
| PipedInputStream | PipedReader | Liest von einem PipedWriter oder PipedOutputStream. |
| StringBufferInputStream? | StringReader | Liest aus Strings. |
| SequenceInputStream | (keine Entsprechung) | Verbindet mehrere InputStreams. |

Wichtige Ausgabeklassen

| Byte-Stream-Klasse für die Ausgabe | Zeichen-Stream-Klasse für die Ausgabe | Beschreibung |
|---|--|---|
| OutputStream | Writer | Abstrakte Klasse für Zeichenausgabe oder Byte-Ausgabe |
| BufferedOutputStream | BufferedWriter | Ausgabe des Puffers. Nutzt passendes Zeilenendezeichen. |
| ByteArrayOutputStream | CharArrayWriter | Schreibt in Arrays. |
| DataOutputStream | (keine Entsprechung) | Schreibt Primitive und auch UTF-8. |
| (keine Entsprechung) | OutputStreamWriter | Übersetzt Zeichen-Streams in Byte-Streams. |
| FileOutputStream | FileWriter | Schreibt in eine Datei. |
| PrintStream | PrintWriter | Konvertiert primitive Datentypen in Strings und schreibt sie in einen Ausgabestrom. |
| PipedOutputStream | PipedWriter | Schreibt in eine Pipe. |
| (keine Entsprechung) | StringWriter | Schreibt in einen String. |

Sortierung nach Resource

| Ressource | Zeilchenorientierte Klasse | Byte-orientierte Klasse |
|-----------|--|--|
| Datei | FileReader FileWriter | FileInputStream FileOutputStream |
| Speicher | CharArrayReader CharArrayWriter StringReader StringWriter | ByteArrayInputStream ByteArrayOutputStream -- |
| Pipe | PipeReader PipeWriter | PipeInputStream PipeOutputStream |

OutputStream

- ❖ nur eine Methode abstract
- ❖ schreibt grundsätzlich byte arrays in den Ausgabestrom
- ❖ implementiert Closeable und Flushable leer
 - ❖ Designentscheidung der Entwickler

Closeable

- ❖ wird von allen lesenden und schreibenden Datenstrom-Klassen implementiert, die geschlossen werden können.
- ❖ IOException sollte behandelt werden, da eventuell auch noch nicht geschriebene Daten geschrieben werden und man informiert werden sollte, wenn dabei etwas schief geht.

Flushable

- ❖ findet sich nur bei schreibenden Klassen und ist insbesondere wichtig bei Klassen die Daten puffern.

InputStream

- ❖ Gegenstück zum OutputStream
- ❖ klassische Methoden wie read(), read(byte[] b), available(), skip(long n)

InputStream

```
String str = "Dies ist ein Bytestrom";
ByteArrayInputStream bais = new ByteArrayInputStream(str.getBytes());
int c = 0;

//read()
//Liest ein Byte als Integer aus dem Datenstrom.
//Ist das Ende des Datenstroms erreicht, wird -1 übergeben.
//Die Methode ist überladen, wie die nächsten Signaturen zeigen.
while((c = bais.read()) != -1) {
    System.out.print((char)c);
}
```

InputStream

```
String str2 = "Dies ist ein anderer Bytestrom";
ByteArrayInputStream bais2 = new ByteArrayInputStream(str2.getBytes());
//available
//Gibt die Anzahl der verfügbaren Zeichen im Datenstrom zurück,
//die sofort ohne Blockierung gelesen werden können.
System.out.println(bais2.available());
```

InputStream

```
byte[] b = new byte[5];
try {
    //skip(long n)
    //Überspringt eine Anzahl von Zeichen.
    System.out.println(bais2.skip(1));

    //read(byte[] b)
    //Liest mehrere Bytes in ein Feld.
    //Die tatsächliche Länge der gelesenen Bytes wird zurückgegeben
    //und muss nicht b.length() sein.
    bais2.read(b);
} catch (IOException e) {
    e.printStackTrace();
}
for(int i = 0; i < b.length; i++) {
    System.out.print((char)b[i]);
}
```

Writer

- ❖ Pendant zu OutputStream für Zeichen

Writer

```
Writer writer = new CharArrayWriter();
char c[] = {'H', 'a', 'l', 'l', 'o'};
try {
    //write(char[] cbuf, int off, int len)
    //Schreibt len Zeichen des Felds cbuf ab der Position off.
    writer.write(c, 0, 3);
    System.out.println(writer.toString());

    //write(String str)
    //Schreibt einen String.
    writer.write("Bonjour");
    System.out.println(writer.toString());

    //Writer append(char c) throws IOException
    //Hängt ein Zeichen an. Verhält sich wie write(c),
    //nur liefert es, wie die Schnittstelle Appendable verlangt,
    //ein Appendable zurück. Writer ist ein passendes Appendable.
    writer.append('1');
    System.out.println(writer.toString());
}

} catch (IOException e) {
    // TODO Auto-generated catch block
    e.printStackTrace();
}
```

Reader

- ❖ Pendant zu InputStream

Reader

```
char c[] = {'H', 'a', 'l', 'l', 'o'};
char result[] = new char[3];
Reader reader = new CharArrayReader(c);
try {
    reader.read(result, 0, 3);
} catch (IOException e) {
    e.printStackTrace();
}
System.out.println(result);
/*
 * Zum Lesen aller Zeichen muss der Datenstrom
 * so lange ausgesaugt werden,
 * bis keine Daten mehr verfügbar sind.
 * Der Endetest kann auf zwei Arten geschehen:
 */
//1. Variante
try {
    while (reader.ready())
        System.out.println((char)reader.read());
} catch (IOException e) {
    e.printStackTrace();
}

//2. Variante
try {
    for (int i; (i = reader.read()) != -1; )
        System.out.println((char) i);
} catch (IOException e) {
    e.printStackTrace();
}
```

Sprünge und Markierungen

Datenströme filtern und verketten

- ❖ Filter können die Daten auf dem Weg verändern
- ❖ Fähigkeiten der Klasse sollen erweitert werden
 - ❖ Puffern(Pufferung befindet sich in separater Klasse)
- ❖ BufferedWriter nimmt einen anderen Writer auf:
 - ❖ nützlich beispielsweise bei

```
Writer fw = new FileWriter( "BufferedWriter.txt" );
Writer bw = new BufferedWriter( fw );
```

bereits existierende Filter

| Eingabe | Ausgabe | Anwendung |
|-----------------------|----------------------|---|
| BufferedInputStream | BufferedOutputStream | Daten puffern |
| BufferedReader | BufferedWriter | |
| CheckedInputStream | CheckedOutputStream | Checksumme berechnen |
| DataInputStream | DataOutputStream | Primitive Datentypen aus dem Strom holen und in den Strom schreiben |
| DigestInputStream | DigestOutputStream | Digest (Checksumme) mitberechnen |
| InflaterInputStream | DeflaterOutputStream | Kompression von Daten |
| LineNumberInputStream | | Mitzählen von Zeilen |
| LineNumberReader | | |
| PushbackInputStream | | Daten in den Lesestrom zurücklegen |
| PushbackReader | | |
| CipherInputStream | CipherOutputStream | Daten verschlüsseln und entschlüsseln |

BufferedWriter

```
Writer bw = null;
try
{
    Writer fw = new FileWriter( "BufferedWriter.txt" );
    bw = new BufferedWriter( fw );

    for ( int i = 1; i < 10000; i++ ) {
        bw.write( "Zeile " + i );
        bw.append( '\n' );
    }

}
catch ( IOException e ) {
    System.err.println( "Error creating file!" );
} finally {
    bw.close();
}
```

BufferedReader

```
try {
    InputStream in = new BufferedInputStream(new FileInputStream(
        "BufferedWriter.txt"));
    try {
        for (int c; (c = in.read()) != -1 /* EOF */)
            System.out.write(c);
    } finally {
        in.close();
    }
} catch (IOException e) {
    System.err
        .println("cat: Fehler beim Verarbeiten von BufferedWriter.txt");
    System.exit(1);
}
```

Vermittler zwischen Byte-Streams und Unicode-Strömen

- ❖ OutputStreamWriter
- ❖ InputStreamReader