Ausnahmen und IO

Fehler, Ausnahmen, Java-Exception, throw, catch, Ströme, Puffer, Dateien lesen, schreiben, Tastatur, Terminal, HTTP

Fehler - Ausnahmen

- Fehler : Unreparierbares Ereignis
  - Partielle Operationen:
    - Division durch 0
  - Hardwareausfälle
- Ausnahme : Abweichung vom Normalfall
  - Methoden, die meist, aber nicht immer ein Resultat liefern
    - int finde(Object [ ] menge, Object x)
  - Normalerweise
    - finde(menge,x) : Index, an dem x zum ersten Mal auftritt
    - Was, wenn x nicht vorhanden ist ?
  - Kein Fehler, sondern eine Ausnahme - exception
  - Der Programmierer sollte die Ausnahme abfangen (to catch) und gesondert behandeln

Lösung

- Quick fix:
  - offensichtlich ungültige Werte als Ergebnis für Ausnahme
  - **`finde(menge,x) = -1`**, falls **`x`** nicht in menge
- Nachteil
  - der Benutzer von *finde* muss informiert sein
  - er darf nicht vergessen, diesen Fall zu berücksichtigen
- Kann man garantieren, dass der Benutzer nicht vergisst, die Ausnahme zu berücksichtigen ?
  - Ja, wenn die Ausnahme zum Bestandteil der Signatur wird:

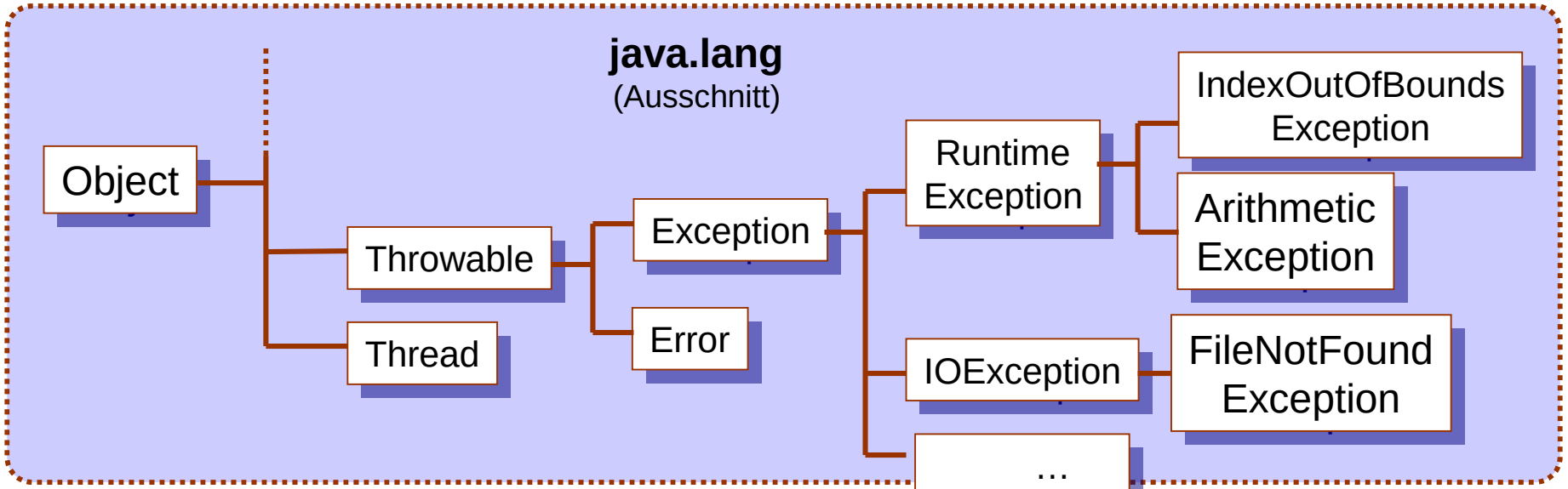
Signatur (Kopf)

Rumpf

```
Object finde(Object[]menge, Object x)throws NixDaException
{
  ...
}
```

Verschiedene Arten von Ausnahmen

- Ausnahmen können weitere Informationen enthalten
  - einen Text
  - Zwischenwerte
  - ...
  - beliebige Daten.
- Ausnahmen sind gekapselte Daten
  - ... ja, genau: Objekte
  - Java hat zugehörige Klassen
- Klassen für Exceptions können selbst definiert werden



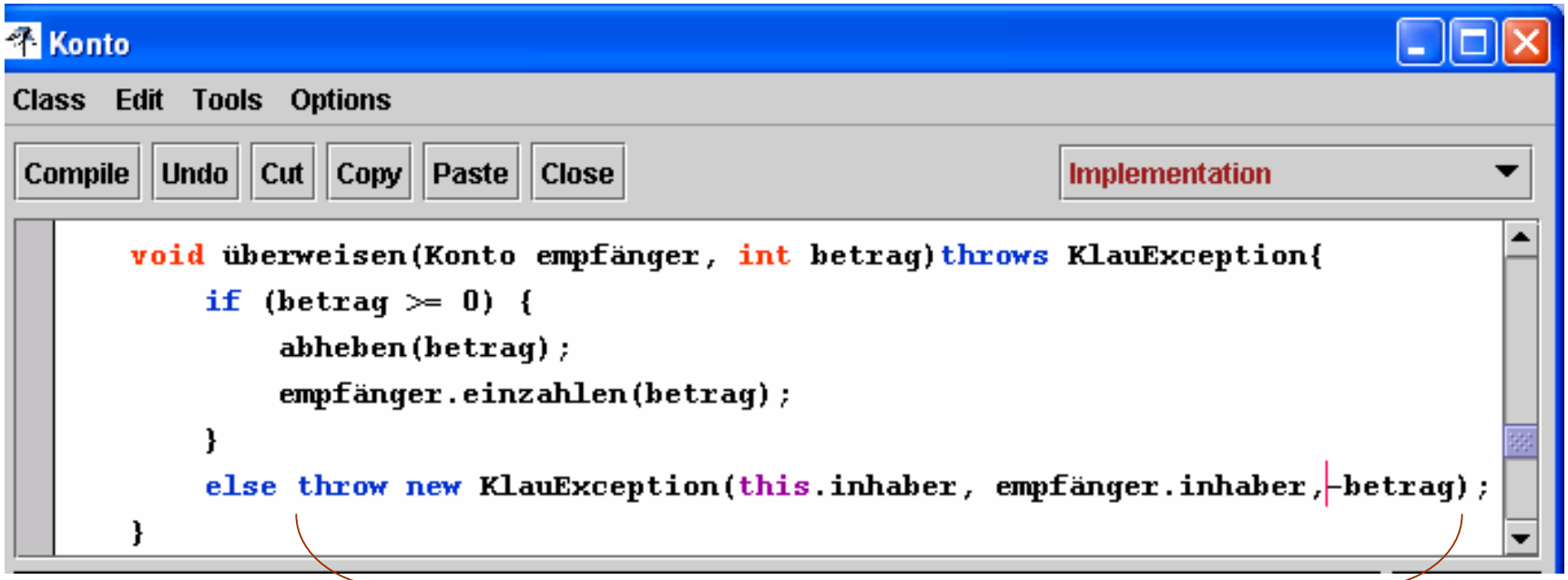
Ein Bankraub als Ausnahme

- Meist geht in der Bank alles mit rechten Dingen zu
- Manchmal wird aber geklaut
- Wir definieren eine Ausnahme **`KlauException`**
- Als Information wünschen wir:
-
  - den Namen des Diebs
  - den Namen des Bestohlenen
  - den Betrag

```
public class KlauException extends Exception{
    int betrag;
    String dieb;
    String opfer;

    KlauException(String dieb, String opfer, int betrag)
    {
        this.dieb=dieb;
        this.opfer = opfer;
        this.betrag = betrag;
    }
}
```

Wir stellen eine Falle

- Ein Dieb könnte versuchen einen negativen Betrag zu überweisen
- Wir stellen ihm eine Falle



eine Ausnahme wird erzeugt (new) und geworfen (throw)

Behandeln - oder delegieren

- Wenn man eine Methode benutzt, die eine Exception auswirft, gibt es zwei Möglichkeiten
  - die Ausnahme zu behandeln
    - `try{ ... } catch ( ... ){ ... }`
    - „Haltet den Dieb“
  - die Ausnahme explizit zu delegieren
    - Schlüsselwort: **`throws`**
    - „Pass auf, es kann geklaut werden ... „
  - Delegieren ist am einfachsten
    - Wenn jeder delegiert kann es zum Laufzeitfehler kommen
  - Auf jeden Fall kann keiner sagen
    - „Ich habe nichts von der Gefahr gewusst“

Exception delegieren

- Soll sich doch der Benutzer meiner Methode drum kümmern

```
public class MeineFinanzen
{
    static String name = "K.R. Ösus";
    static Konto giro = new Konto(name);
    static Konto aktien = new Konto(name);

    public static void verkaufen(int betrag) throws KlauException{
            giro.überweisen(aktien,betrag);
   }
}
```

Ich hab ja gewarnt

Wenn sich niemand um die Exception kümmert, entsteht ein Laufzeitfehler.

```
C:\WINDOWS\System32\cmd.exe
C:\Program Files\BlueJ\examples>java Test
Exception in thread "main" java.lang.NoClassDefFoundError: Test

C:\Program Files\BlueJ\examples>_
```

Exception behandeln

try: Versuche, die Methode regulär zu Ende zu bringen.

catch: Falls eine Exception ausgelöst wird, fange sie.

```
try
  { falls alles gut geht }
catch(ExceptionDeklaration)
    { ...falls Exception
            auftritt ... }
```

```
public class MeineFinanzen
{
    static String name = "K.R. Ösus";
    static Konto giro = new Konto(name);
    static Konto aktien = new Konto(name);

    public static void verkaufen(int betrag){
        try {
              giro.überweisen(aktien,betrag);
            }
        catch (KlauException k){
            System.out.println("Haltet "+k.dieb+". Er hat");
            System.out.println(k.opfer+" "+k.betrag+" € gestohlen");
        }
   }
}
```

Mehrere Exceptions

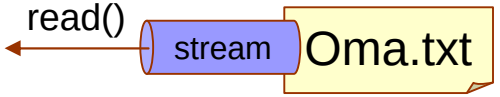
- Eine Methode kann mehrere Exceptions
  - werfen
    - `myMethod() throws KlauException, IOException{ … }`
  - oder behandeln

```
try    { … Normalfall …}
catch ( Klauexception k ) { … behandeln … }
catch { IOException e   ) { … behandeln … }
…
finally{ … räume auf … }
```

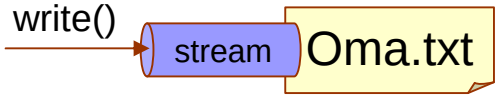
- Eine optionale **`finally`**-Klausel wird gerne für abschließende Aufräumarbeiten benutzt

Ströme

- Unidirektionale Datenverbindung
  - Englisch: **stream**
  - Wie eine Röhre, durch die Daten fließen
- Alle externen Verbindungen nutzen streams
  - Drucker, Dateien, Modem, Tastatur, Terminal, CD-ROM, …
- Zum Lesen: Input-Strom
  - Typische Methoden :
    - read(), readln(),



- Zum Schreiben: Output-Strom
  - Typische Methoden:
    - write(char c), writeln(String s), …

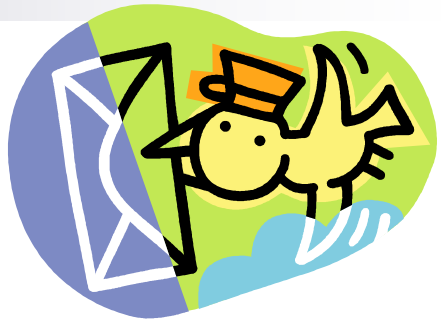


- Notwendige Klassen im Paket **`java.io`**

Schreiben in eine Datei

- SchreibStrom auf eine Datei aufmachen
  - `FileWriter brief = newFileWriter("C:/Beispiele/Oma.txt");`
  - Die Datei wird
    - automatisch erstellt, oder ggf.
    - überschrieben
- Schreiben
  - `brief.write("Hallo Oma,\nbrauche Kohle\n");`
  - `brief.write("Dein Enkel\nMax");`
- Datei schließen
  - `brief.close();`
- Allerdings muss eine Ausnahme berücksichtigt werden:
  - `IOException`

BriefAnOma



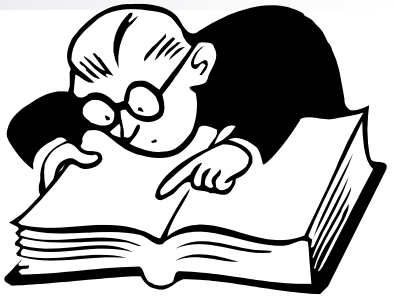
- Das komplette Programm

```
import java.io.*;

public class BriefAnOma
{
  static void schreibe(){
    try{ FileWriter brief = new FileWriter("C:\\Beispiel\\Oma.txt");
         brief.write("Hallo Oma\nich brauche Kohle\n");
         brief.write("Immer Dein\nMax");
         brief.close();
    }
    catch(IOException e){ }
  }
}
```

DOS-Pfadname benutzt „\\“ slash

backslash „\“ ist „Escape-Zeichen“
\n : newline
\\ : backslash selber

Aus einer Datei lesen

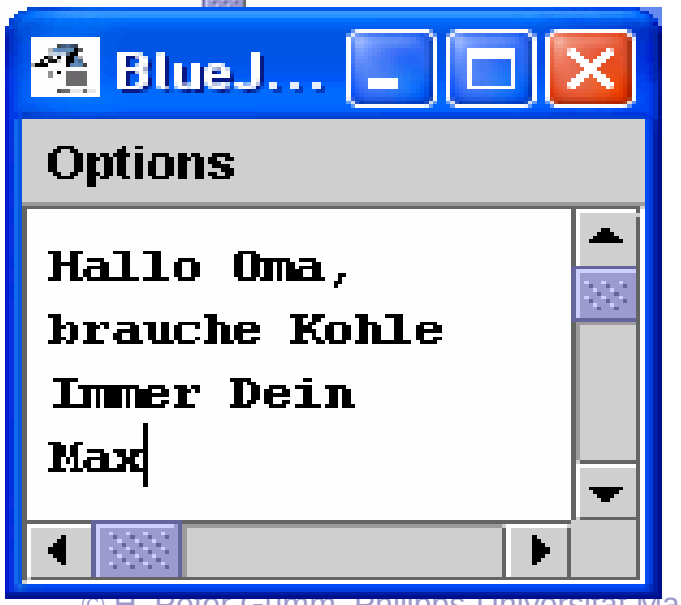


- LeseStrom auf eine Datei aufmachen
  - `FileReader datei = new FileReader("Oma.txt");`
    - Der FileReader liest per default die Datei als int-Folge
- Nächstes Byte lesen und als **int** interpretieren
  - `int zahl;`
  - `zahl = datei.read();`
- Datei am Ende, falls gelesener Wert negativ:
  - `while( zahl != -1) { tuWas(); zahl=datei.read(); }`
- Alternativ: nächstes Bytes als char-Wert lesen
  - `char next;`
  - `next =(char)datei.read();`
- Datei schließen
  - `datei.close();`
- Hier muss neben **`IOException`** eine weitere Exception berücksichtigt werden:
  - `FileNotFoundException`

OmaLiest

- Oma liest den Brief und schreibt ihn in das Terminal Fenster
  - das mit den großen Buchstaben …

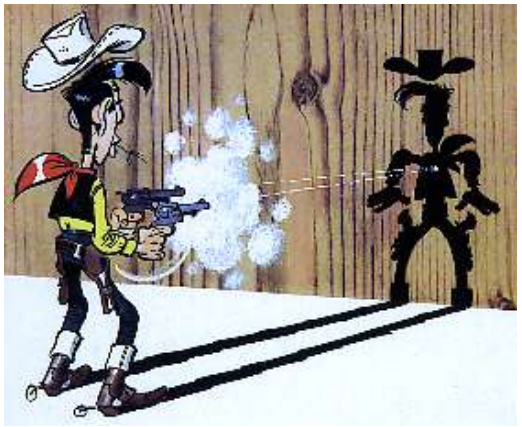
```
import java.io.*;

public class OmaLiest{

    static void liesVor(){
        try{
            FileReader brief = new FileReader("Oma.txt");
            int c = brief.read();
            while( c!= -1){
                System.out.print((char)c);
                c=brief.read();
            }
            brief.close();
        }
        catch(FileNotFoundException f){ }
        catch(IOException e){ }
    }
}
```



Schmutzige Tricks

- Statt der klaren Schleife

```
int c = datei.read();
  while( c!= -1){
     System.out.print((char)c);
      c = datei.read();
  }
```



- schreiben C- und Java-Cowboys gerne

```
int c;
while( (c = datei.read()) != -1 ){
     System.out.print((char)c);
}
```

- Dabei nutzen sie aus:
  - Jede Zuweisung ist auch ein Ausdruck
  - `x = 5` ist ein Ausdruck mit Wert **5**
  - `c = datei.read()` ist auch ein Ausdruck
  - Das Einlesen der nächsten Zahl ist ein Seiteneffekt

Gepuffertes Schreiben und Lesen

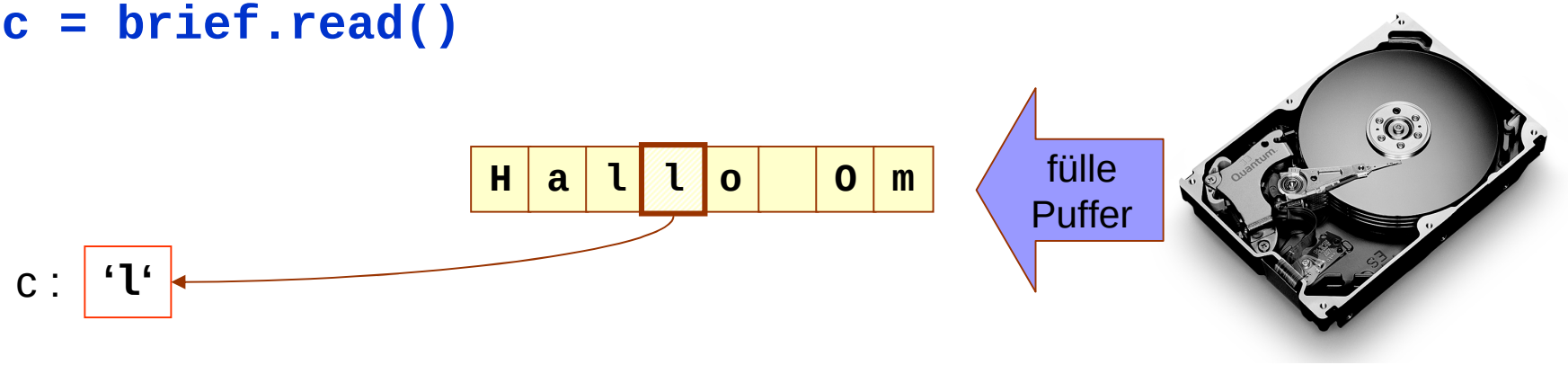
- Lesen und Schreiben einzelner Zeichen ist unökonomisch
  - Festplatte muss sich drehen
  - Schreib-Lesekopf muss sich bewegen

- Ganze Blöcke werden auf einmal gelesen oder geschreiben
  - ein Block kann z.B. ein Sektor auf der Platte sein
- Resultat wird gepuffert
  - Puffer ist ein Behälter im Hauptspeicher
  - Schreiben: Zeichen in Puffer schreiben
    - wenn Puffer voll ist, in einem Rutsch auf die Platte schreiben
    - ebenso wenn der `close()` Befehl kommt.
  - Lesen: einen ganzen Puffer voll von der Platte lesen
    - read()-Befehl liest aus dem Puffer, solange dort noch Zeichen vorhanden sind.

Gepuffertes Lesen

- Der Puffer kann z.B. ein Array im Hauptspeicher sein
- Ein Java-Befehl **`read(), readLine()`** o.ä. bewirkt
  - lies das nächste Zeichen, bzw. die nächste Zeile aus dem Puffer
  - falls der Puffer leer wird, fülle ihn wieder mit einem Sektor der Platte

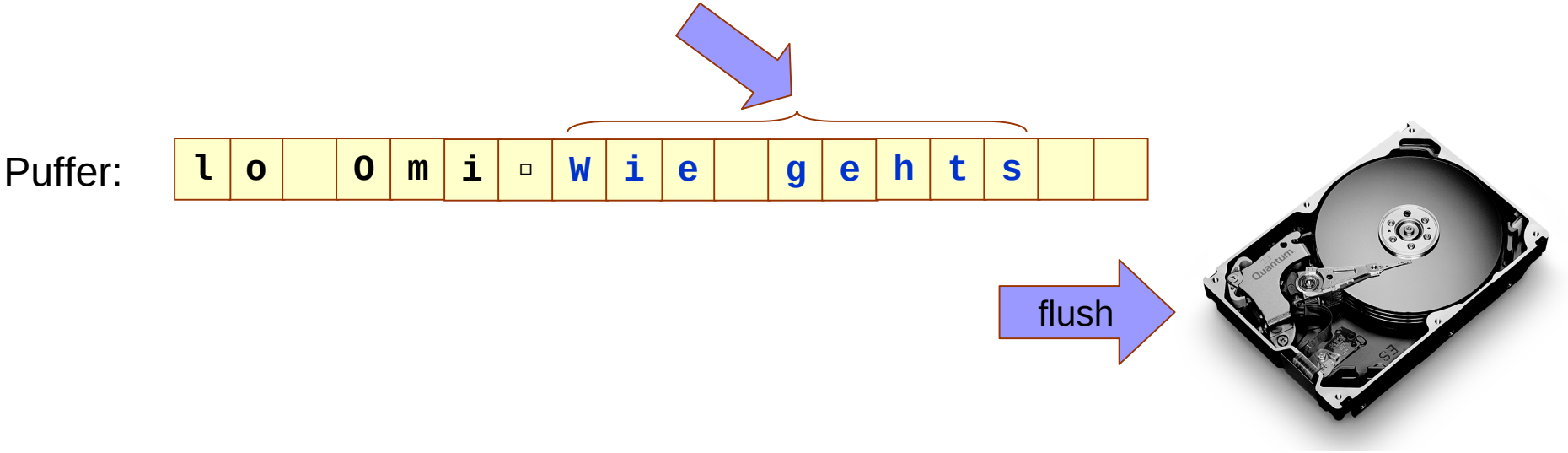
```
c = brief.read()
```



Gepuffertes Schreiben

- Geschrieben wird in den Puffer
  - Wenn der Puffer voll ist, wird er in einem Vorgang auf die Platte geschrieben
  - **`close()`** : entleere Puffer auf die Platte, schließe Puffer.

```
brief.write("Wie gehts");
```



Java setzt streams zusammen

- `FileReader rohBrief = new FileReader(„Oma.txt“);`
  `BufferedReader brief = new BufferedReader(rohBrief);`
- oder - auf einen Schlag:
- `BufferedReader brief =`
  `new BufferedReader(new FileReader(„Oma.txt“));`

- `new BufferedWriter(new FileWriter(„Oma.txt“))`

Gemütliches Schreiben und Lesen

```
import java.io.*;

public class BriefAnOma
{
    static void schreibe(){
        try{ BufferedWriter brief =
            new BufferedWriter(new FileWriter("Oma.txt"));
             brief.write("Hallo Omi,\nwie gehts\n");
             brief.write("Immer Dein\nMax");
             brief.close();
        }
        catch(FileNotFoundException f){ }
        catch(IOException e){ }
    }
}
```

- Ein **`FileWriter`** wird in einen **`BufferedWriter`** „verpackt“
- alles andere - wie vorher

Gemütliches Lesen

```
import java.io.*;

public class OmaLiest{

    static void liesVor(){
        try{
            BufferedReader brief =
              new BufferedReader(new FileReader("Oma.txt"));
            String zeile = brief.readLine();
            while( zeile != null){
                System.out.println(zeile);
                zeile=brief.readLine();
            }
            brief.close();
        }
        catch(FileNotFoundException f){ }
        catch(IOException e){ }
    }
}
```

- FileReader eingepackt in BufferedReader
- BufferedReader liest einen String
- und zwar zeilenweise

FileNotFoundException ist Unterklasse von IOException

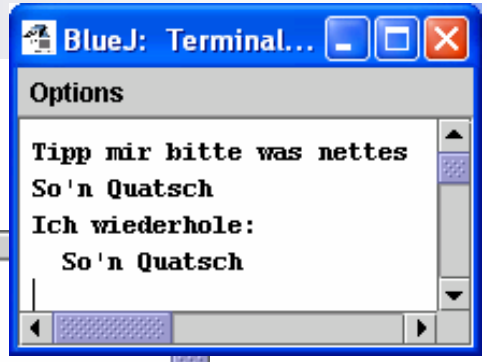
Tastatur, Terminal

- Tastatur und Terminal wie Ströme behandelt
- Tastatur ist ein **`InputStream`**
  - In der Klasse **`System`** ist **`in`** als statische Variable definiert
  - **`public class System{ static InputStream in; ... }`**
- Terminal ist ein **`PrintStream`**
  - In der Klasse **`System`** als statisches Feld **`out`** definiert
  - **`public class System{ static PrintStream out; ... }`**
- Für Fehlermeldungen gibt es einen eigenen **`PrintStream err`**.

InputStream

PrintStream

Terminaleingabe



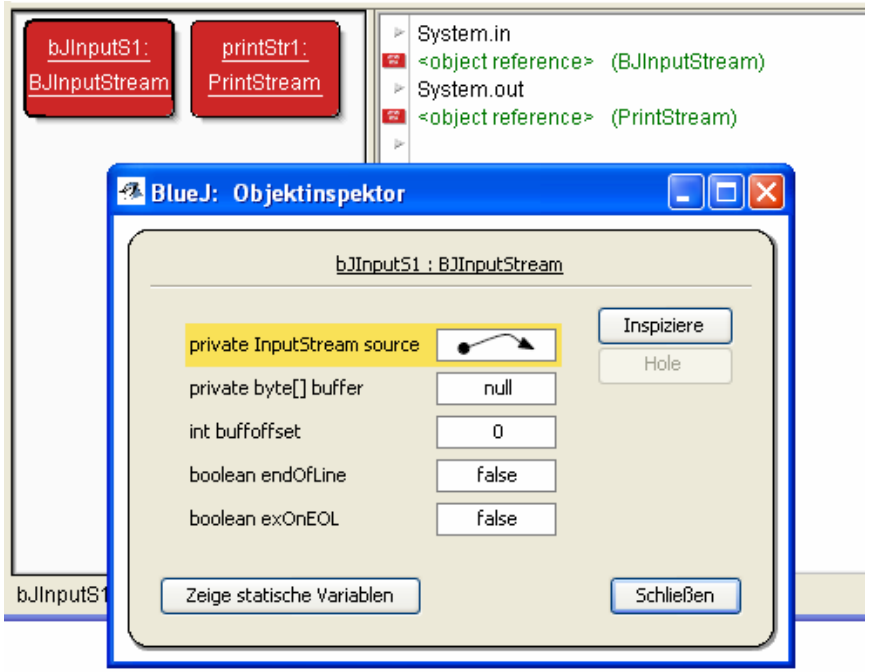

```
import java.io.*;
public class TastaturLeser
{
    static String readln(){
        BufferedReader eingabe =
           new BufferedReader(new InputStreamReader(System.in));
        String zeile ="";
        try{ zeile = eingabe.readLine(); }
        catch (IOException e){ zeile = e.toString(); }
        finally{ return zeile; }
    }

    static void test(){
        System.out.println("Tipp mir bitte was nettes");
        String gelesen = readln();
        System.out.println("Ich wiederhole:\n  "+gelesen);
    }
}
```

die Tastatur

das Terminal

Standard i/o umlegen

- Normalerweise
  - **`System.in`** : die Tastatur
    - Klasse: **`BJInputStream`**
    - ein **`InputStream`** mit zusätzlichen Feldern
  - **`System.out`**: das Terminal
    - ein **`PrintStream`**
- Das kann man ändern
  - **`System.setOut(`**
    `new FileStream(„meineDatei.txt“)`
    `);`
  - Jedes **`System.out.println`** geht ab jetzt in die Datei



Lesen im Internet

- Internet Adresse: URL
  - Stelle Verbindung her
  - Lese InputStream der Verbindung

```
>java ReadURL "http://www.mathematik.uni-ma
```

```
public static void main (String[] args) throws IOException{

   URL url = new URL(args[0]);    // Aufruf mit URL als Argument
   HttpURLConnection huc = (HttpURLConnection)url.openConnection();

   InputStream inStream = huc.getInputStream();
   Reader reader = new InputStreamReader(inStream,"ISO8859_1");
   BufferedReader input = new BufferedReader(reader);

   String line;
   do {
      line = input.readLine();
      System.out.println(line);
   } while (line != null);

   input.close();
  }
}
```

ISO-LATIN-1 Zeichensatz

BlueJ: Konsole - mail

Optionen

```
<h2><font color="#990000">Lehre</font></h2>
<ul>
  <li>
    <h3>Veranstaltungen im Wintersemester 04/05</h3>
    <ul>
      <li><b><font color="red">Praktische Informatik I</font></b>: <
      <li><b><font color="red">Zustandsbasierte Systeme - Coalgebren
    </ul>
  </li>
</ul>
<h2><font color="#990000">Kontakt</font></h2>
<ul>
  <li>Tel.: (06421) 28-<b>21516</b> </li>
  <li>Fax: (06421) 28-<b>25419</b> </li>
  <li><a href="post.jpg">E-Mail</a></li>
  <li>Sekretariat (Ebene D5, Zi 5426)
    <ul>
      <li>Frau Heinb&auml;cher,  Tel.: -<b>21514</b> </li>
      <li>Frau Dinklage, Tel: -<b>21513</b></li>
    </ul>
  </li>
```

Benötigt:

java.net.*, java.io.*, java.util.*