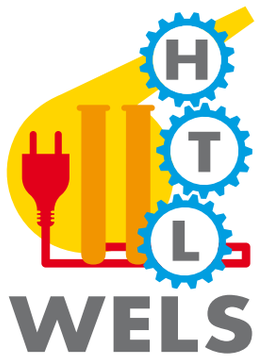
CSV DATEIEN PARSEN MIT JAVA BOARDMITTEL

SEW 3

DI Thomas Helml



INHALT

- CSV Dateien
- CSV Dateien parsen
  - Methode split()
  - Lesen von Token aus Textdatei
  - Scanner

CSV DATEIEN

- **CSV** = **C**omma **S**eparated **V**alues
- „durch Komma(Trennzeichen) getrennte Werte“
- pro Zeile wird ein Datensatz gespeichert
  - zwischen den einzelnen Werten wird ein Trennzeichen verwendet: Comma, Semikolon, Tab, Space, …
- CSV Dateien können in Excel importiert und exportiert werden
  - daher werden sie gerne zum Datenaustausch verwendet

CSV DATEIEN

- Bsp: CSV Datei mit Überschrift!

```
Vorname;Name;Abteilung;Klasse;
Thomas;Helml;IT;5AHIT;
Gams;Erich;IT;4AHIT;
...
```

- Selbe CSV Datei in Excel:

Vorname	Name	Abteilung	Klasse
Thomas	Helml	IT	5AHIT
Gams	Erich	IT	4AHIT

CSV DATEIEN PARSEN

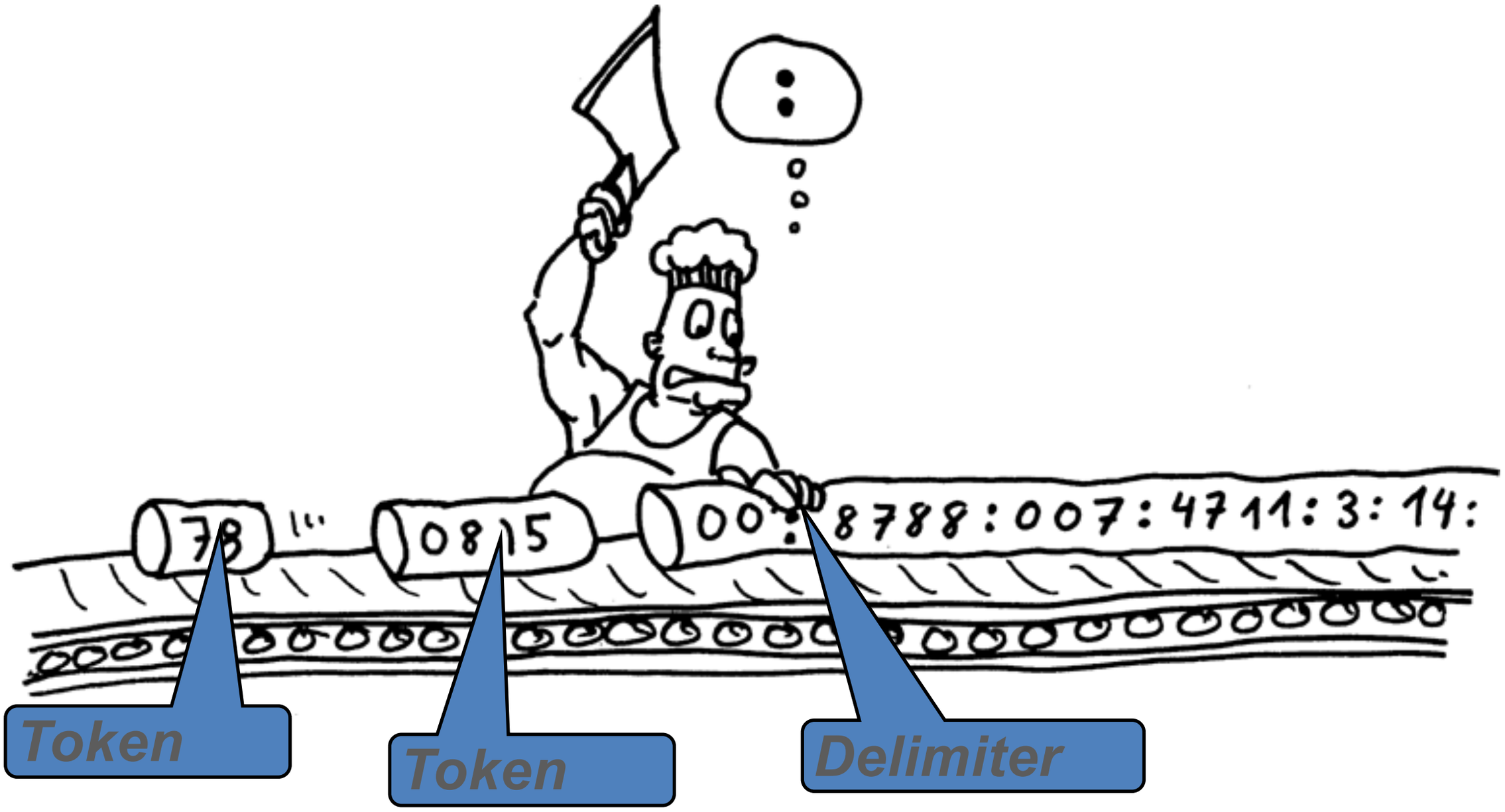
- In Java gibt es viele Möglichkeiten, um CSV Dateien zu zerlegen
- Grundsätzlich gibt es 2 Möglichkeiten:
  - Zeilenweises Einlesen und dann zerlegen
  - Direkt beim Einlesen zerlegen
- Auf den folgenden Folien werden beide Methoden vorgestellt

STRING.SPLIT()

- Annahme: Zeile wurde als String eingelesen
- in Klasse String gibt es Methode:
  - `public String[] split(String regex)`
  - zerlegt String in Array von Teilstrings anhand regulären Ausdruck
- Bsp:

```
String s="GAME;3AHIT;INSY;FR;10"
String[] zeile = s.split(";");
String s1=zeile[0]; // => GAME
int i=Integer.valueOf(zeile[4]); // => 10
```

LESEN VON TOKEN AUS TEXTDATEI



LESEN VON TOKEN AUS TEXTDATEI

- Token: Teil eines Stream, der von Delimitern begrenzt wird
- Beispiel:
  - Stream: "abc de,f 66,z-2"
  - Delimiter: {" "} //Leerzeichen
    - Token: abc de,f 66,z-2
  - Delimiter: {" ","","} //Leerzeichen und Komma
    - Token: abc de f 66 z-2
- Ziel: Extraktion der Token aus Stream, die bestimmte Bedeutung haben (Parsen)
  - Abhängig von Menge der Delimiter

EXTRAKTION VON TOKEN: KLASSE STREAMTOKENIZER

- Standard-Delimiter: Leerzeichen, Zeilenende, Dateiende, ... (können ergänzt werden)
- Methoden:
  - `StreamTokenizer(Reader)`: Konstruktor
  - `int nextToken()`: schaltet auf nächstes Token um
- Attribute:
  - int ttype: Typ des Token
    - `TT_EOF:` Dateiende
    - `TT_EOL:` Zeilenende
    - `TT_NUMBER:` Zahl
    - `TT_WORD :` String
  - `double nval:` Token als double
  - `String sval:` Token als String

BEISPIEL: LESEN VON TOKEN AUS TEXTDATEI

```
try{
    FileReader fr = new FileReader(“Beispiel.txt");
    StreamTokenizer st = new StreamTokenizer(fr);
    do{
        st.nextToken();
        switch (st.ttype){
            case StreamTokenizer.TT_NUMBER:
                double d = st.nval; break;
            case StreamTokenizer.TT_WORD:
                String s = st.sval; break;
            default: break;
        }
    } while(st.ttype != StreamTokenizer.TT_EOF);
    fr.close();
}catch(Exception e) {...}
```

SCANNER

- Scanner eignet sich für beide Varianten
  - Zeile als String einlesen und dann parsen
  - Direkt vom Stream parsen
- Hat mehrere Konstruktoren:
  - Scanner(File source)
  - Scanner(InputStream source)
  - Scanner(String source)
  - ...

SCANNER

- Beispiel:

```
Scanner sc = new Scanner(...);
sc.useDelimiter(";");


if (sc.hasNext())
    String s1 = sc.next();
if (sc.hasNextInt())
    int i = sc.nextInt();
...
```

3RD PARTY BIBLIOTHEKEN

- Common CSV:
  - https://commons.apache.org/proper/commons-csv/
- Super CSV:
  - http://super-csv.github.io/super-csv/index.html