Input

Konsole:

```
import java.util.*;
Scanner name = new Scanner(System.in);
int alter = name.nextInt();
```

Import

Scanner erstellen

Benutzen

Method	Description
nextInt()	reads an int from the user and returns it
nextDouble()	reads a double from the user
next()	reads a one-word String from the user
nextLine()	reads a one-*line* String from the user

File:

```
import java.io.*;

File file = new File("example.txt");
if (file.exists() && file.length() > 1000) {
  file.delete();
}
```

exists()	Gibt **true** zurück, falls diese Datei existiert, sonst **false**
canRead()	Gibt **true** zurück, falls diese Datei gelesen werden kann, sonst **false**
getName()	Gibt den Namen dieser Datei zurück
length()	Gibt die Dateigrösse, in Bytes, zurück
delete()	Löscht die Datei!
renameTo(file)	Benennt die Datei um!

Scanner und File:

```
import java.io.*;     // für File
import java.util.*;   // für Scanner

File file = new File("input.txt");
Scanner scanner = new Scanner(file);
int zahl = scanner.nextInt();
```

kürzer:

```
Scanner scanner = new Scanner(new File("input.txt"));
```

Method	Description
`nextInt()`	reads an `int` from the user and returns it
`nextDouble()`	reads a `double` from the user
`next()`	reads a one-word `String` from the user
`nextLine()`	reads a one-*line* `String` from the user

Input Mismatch Verhindern:

Method	Description
`hasNext()`	returns `true` if there is a next token
`hasNextInt()`	returns `true` if there is a next token and it can be read as an `int`
`hasNextDouble()`	returns `true` if there is a next token and it can be read as a `double`

Bsp:

```
import java.io.*;                 // for File
import java.util.*;               // for Scanner

public class HoursWorked {
    public static void main(String[] args)
            throws FileNotFoundException {
       Scanner input = new Scanner(new File("hours.txt"));
```

Mit ganzen Zeilen:

- Wir wollen die Tokens verarbeiten aber wir wollen auch wissen, wo eine Zeile endete (\n sagt an wann die Daten für eine Person abgeschlossen sind).

Eine bessere Lösung basiert auf einem *hybriden* Ansatz:

- Zuerst: zerlege den Input in Zeilen.
- Dann zerlege jede Zeile in Tokens.

Method	Description
`nextLine()`	returns next entire line of input (from cursor to **\n**)
`hasNextLine()`	returns `true` if there are any more lines of input to read (always true for console input)

→ Nie gleichen Scanner für Tokens und Zeilen verwenden

```
Scanner input = new Scanner(new File("fileName"));
while (input.hasNextLine()) {
    String line = input.nextLine();
    // bearbeiten dieser Zeile
}
```

Bsp

```
while (input.hasNextLine()) {
    String line = input.nextLine();
    Scanner lineScan = new Scanner(line);
    int id = lineScan.nextInt();          // e.g. 456
    String name = lineScan.next();        // e.g. "Erich"
    double sum = 0.0;
    int count = 0;
    while (lineScan.hasNextDouble()) {
        sum = sum + lineScan.nextDouble();
        count++;
    }
```