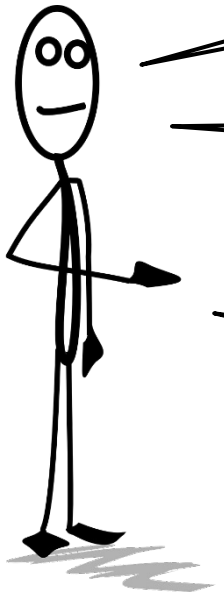
Technische Universität Braunschweig

Programmieren 1 – Vorlesung #11

Arne Schmidt

Wiederholung

Letzte Woche



Assertions zum Testen, z.B. für testgetriebene Programmierung

C_0- und C_1-Tests

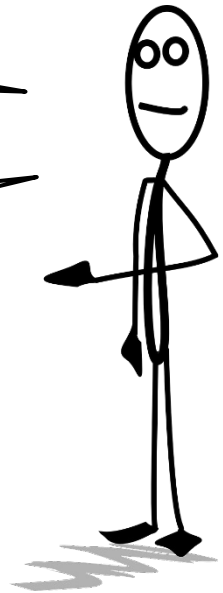
JUnit für automatisiertes Testen.

Diese Woche

Wie können Daten während der Laufzeit eingegeben werden?

Wie können Daten aus Dateien ausgelesen oder in Dateien geschrieben werden?

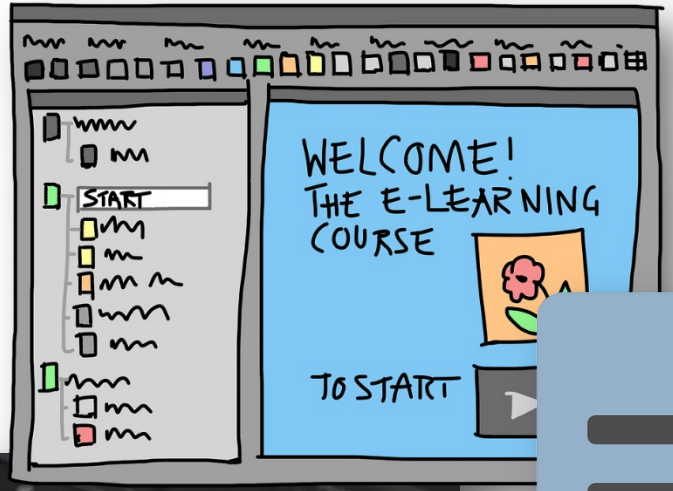
Wie kann die Ausgabe auf der Konsole möglich gut lesbar angezeigt werden?



Kapitel 7 – Ein- und Ausgabe

Ein- und Ausgabe

Ein- / Ausgabe
über Konsole

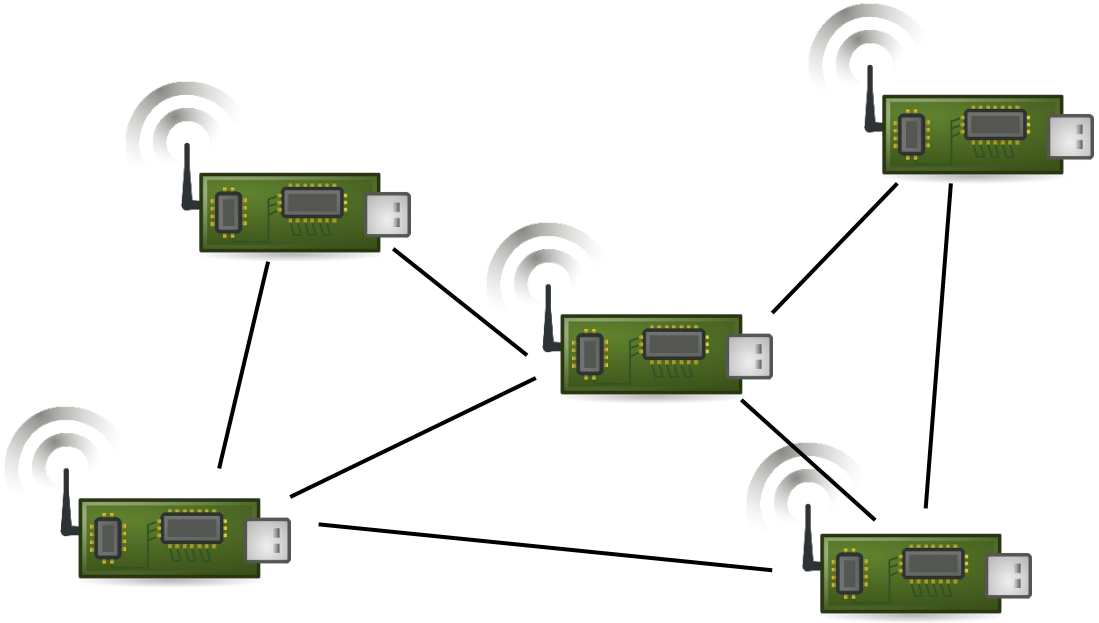


Ein- / Ausgabe
über GUI

Ein- / Ausgabe
über Dateien

Eingabe über Tastatur

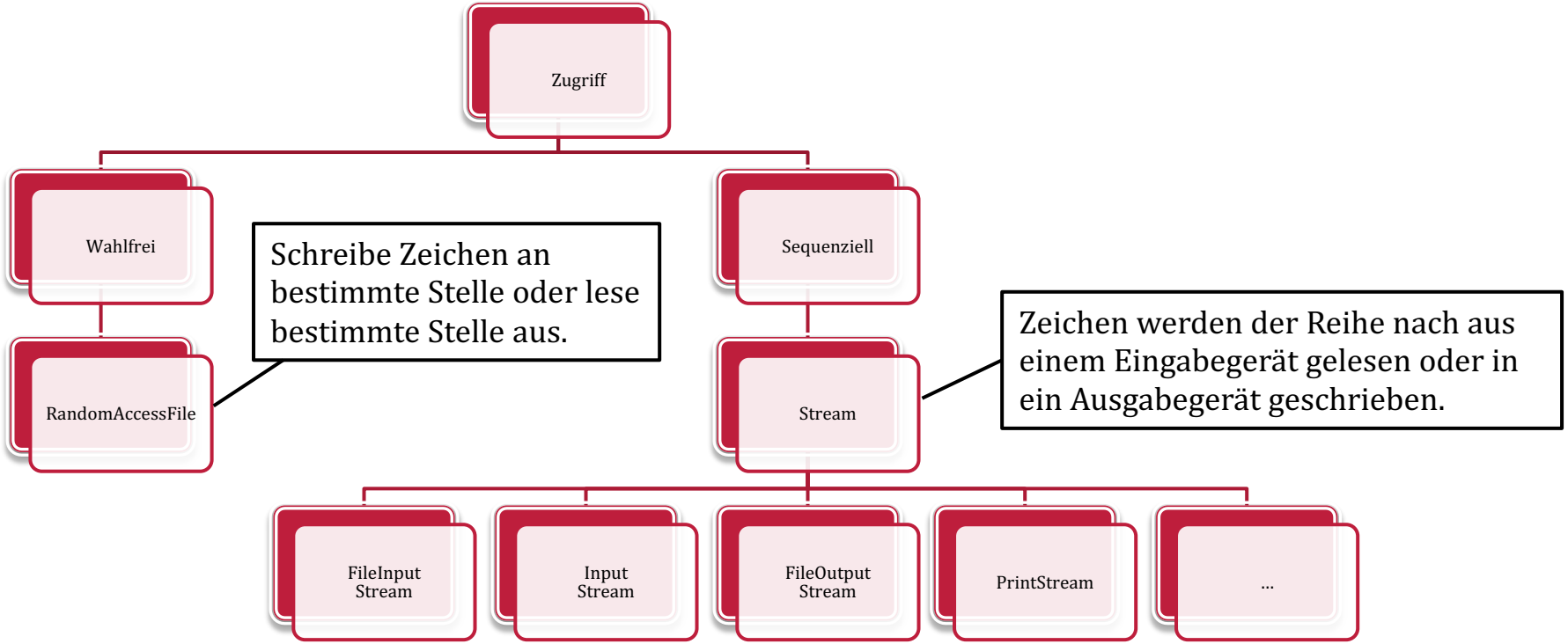
Ein- und Ausgabe: Prozesse / Verteilte Systeme



Prozesse / Systeme kommunizieren und tauschen so Daten untereinander.

Parallele Prozesse und graphische Oberflächen werden in Programmieren 2 behandelt.

Grundlagen


Zugriff
Wahlfrei
Sequenziell
Schreibe Zeichen an bestimmte Stelle oder lese bestimmte Stelle aus.
RandomAccessFile
Stream
Zeichen werden der Reihe nach aus einem Eingabegerät gelesen oder in ein Ausgabegerät geschrieben.
FileInput Stream
Input Stream
FileOutput Stream
PrintStream
...

Kapitel 7.1 – Streams

Streams

Streams können verkettet oder verschachtelt werden.

Durch Verkettung („chaining“) werden mehrere Streams zu einem zusammengefasst, der die Streams nacheinander verarbeitet.

Schachtelung von Streams erlaubt bspw., Filter zu konstruieren.

Beide Konzepte sind durch Klassen realisiert und können vom Benutzer modifiziert und erweitert werden.

Viele Klassen befinden sich in den Paketen java.io und java.util

Stream-Beispielklassen

Byte-Streams
    Jede Transporteinheit ist 8 Bit lang.
Character-Streams
    Character-Streams verwenden Unicode-Zeichen der Länge 16 Bit.
Brückenklassen ermöglichen die Umwandlung beider Stream-Arten ineinander.
    Beispiele: OutputStreamWriter, InputStreamWriter.

Generell:
OutputStream bzw. InputStream schreiben / lesen Bytes.
Writer bzw. Reader schreiben / lesen Character.

Die Klasse PrintStream

Die Klasse PrintStream bietet Methoden zum Schreiben der Werte vieler Datentypen:

```
class PrintStream {
   ...
   print( ... );
   println( ... );
   printf( ... );
   ...
}
```

Das kennen wir schon!

```
System.out.println( ... );
java.lang.System.out.println( ... );
```

Wagen wir einen genaueren Blick in die Klasse System.

Kapitel 7.2 – Die Klasse System

Die Klasse System

Die Klasse System ist nicht instanziierbar und stellt eine Reihe systemnaher Attribute und Methoden bereit.

Attribute:
- PrintStream out
  (Standard-Ausgabe-Stream)
- InputStream in
  (Standard-Eingabe-Stream)
- PrintStream err
  (Standard-Fehler-Stream)

Methoden (Auswahl):
- static void exit(int status)
  (Beendet das Java Programm. Parameter ist Statuscode. Wert 0 heißt fehlerfreies Beenden)
- static Properties getProperties()
  (siehe nächste Slide)
- static String getProperty(String key)

System Properties

Version:	java.version
Firma:	java.vendor
Home:	java.home
Betriebssystem:	os.name
Architektur:	os.arch
Version:	os.version
Benutzername:	user.name
Home-Verzeichnis:	user.home
Verzeichnis:	user.dir
Zeilentrennzeichen:	line.separator
Dateitrennzeichen:	file.separator
Pfadtrennzeichen:	path.separator

Beispiele:

```
System.getProperty ("line.separator");
```

```
System.getProperty (“os.name");
```

```
class JavaVersion {
   public static void main (String [] args) {
      String s = System.getProperty("java.version");
      System.out.println(s);
      s = System.getProperty("java.vendor");
      System.out.println(s);
   }
}
```

Mit System können wir die ersten Ein- und Ausgaben starten!

Ausgabe in eine Datei

```
import java.io.*;
class HelloFile {
   public static void main (String [] args) {
      String ls = System.getProperty("line.separator");
      String hello = "Hallo Java!";
      FileWriter f1;

      try {
         f1 = new FileWriter ("hallo.txt");
         f1.write ( hello + ls + hello + ls);
         f1.close();
      } catch (IOException e) {
         System.out.println("Fehler der Dateierstellung!");
      }
   }
}
```

Eingabe über Tastatur

```
public static void readAndAddInts () throws IOException {
   int a, b, c;
   BufferedReader din = new BufferedReader (
         new InputStreamReader(System.in)
   );

   System.out.println("Bitte a eingeben: ");
   a = Integer.parseInt(din.readLine());
   System.out.println("Bitte b eingeben: ");
   b = Integer.parseInt(din.readLine());
   c = a + b;
   System.out.println("a + b = " + c);
}
```

BufferedReader nimmt den Standard-Input vom System.

Wartet an dieser Stelle auf Eingabe im Stadard-Input des Systems.

BufferedReader

Um nicht jedes Zeichen einzelnd zu lesen oder zu schreiben, was durch eine hohe Anzahl an Zugriffen auf externe Quellen zu Effizienverlust führt, können die Daten gepuffert werden.

Die Methode readLine() der Klasse BufferedReader liest eine ganze Zeile.

```
String readWithBufferedReader(String pathFile) throws IOException {
    try {
        FileReader fis = new FileReader(pathFile);
        BufferedReader reader = new BufferedReader(fis);
        String actualContent = "";
        String line;
        while ((line = reader.readLine()) != null) {
            actualContent += line;
        }
        return actualContent.toString();
    } catch (IOException | FileNotFoundException e){...}
}
```

Liest jede Zeile einer Datei, bis keine weitere existiert.

Lesen aus einer Datei und Schließen

```
FileReader f;
int c;

try {
    f = new FileReader ("Hallo.txt");
    while ((c = f.read()) != -1) {
        System.out.print((char) c);
    }
} catch (IOException | FileNotFoundException e) {
    System.out.println("Fehler beim Lesen der Datei!");
} finally {
    try {
        if (f != null) {
            f.close();
        }
    } catch (IOException e) {
        System.out.println("Fehler beim Schließen der Datei: " + e.getMessage());
    }
}
```

Kapitel 7.3 – Arbeiten mit Dateien

Die Klasse File

Die Klasse File ist Teil des Pakets java.io. Objekte repräsentieren in der Regel Dateien oder Verzeichnisse.

Konstruktoren:

- File(String pathname)
- File(String parent, String child)
- File(File parent, String child)

Das Anlegen einer Datei besteht aus zwei Schritten:

1. Überprüfen, ob die Datei bereits existiert.
2. Falls nein, Datei anlegen.

Methoden der Klasse File:

boolean createNewFile() throws IOException

boolean delete() oder void deleteOnExit()

File – Get- und Abfrage-Methoden

- String getName(): Gibt den Namen der Datei zurück
- String getPath(): Gibt den Pfad zur Datei zurück
- String getAbsolutePath(): Gibt den absoluten Pfad zurück.
- String getParent(): Gibt den Pfad zum übergeordneten Ordner zurück, falls existent.

- boolean exists(): Prüft, ob die Datei / das Verzeichnis existiert.
- boolean isFile(): Prüft, ob Variable auf eine Datei zeigt.
- boolean isDirectory(): Prüft, ob Variable auf ein Verzeichnis zeigt.
- boolean canWrite(): Prüft, ob in die Datei geschrieben werden kann.
- boolean canRead(): Prüft, ob die Datei gelesen werden kann

- String[] list(): Listet Pfade von Dateien und Verzeichnisse.
- File[] listFiles(): Listet Files von Dateien und Verzeichnissen auf.

File – Manipulation

boolean mkdir(): Erstellt Datei / Verzeichnis unter angegebenen Pfad
boolean mkdirs(): Wie mkdir() für jedes aller nicht existenten Elternverzeichnis unter angegebenen Pfad
boolean renameTo(File dest): Bennent Datei / Verzeichnis zu einem Zielverzeichnis um.
boolean delete(): Löscht eine Datei / ein Verzeichnis

Kapitel 7.4 – Formatierte Ausgabe

Unformatierte Ausgabe

```
for (double x = 0.5; x <= 1.4; x += 0.1) {
   System.out.println(x + ":    " + Math.sin(x));
}
```

Ausgabe

```
0.5:    0.479425538604203
0.6:    0.5646424733950354
0.7:    0.644217687237691
0.7999999999999999:    0.717356090899523
0.8999999999999999:    0.7833269096274833
0.9999999999999999:    0.8414709848078964
1.0999999999999999:    0.8912073600614353
1.2:    0.9320390859672263
1.3:    0.963558185417193
```

Formatierte Ausgaben mit printf

Die Methode printf ermöglicht eine formatierte Ausgabe:
printf(*AusgabeText* [, *Wert1*, *Wert2*, ...])

Der AusgabeText kann mit **Formatstrings** gesteuert werden. Diese beginnen immer mit einem Prozentzeichen (*%*) und enden mit einem **Konversionszeichen** (bspw. *f*). Dazwischen liegen **Steuerzeichen**, die die Formatierung steuern (bspw. *9.6*, um 9 Zeichen und 6 Nachkommastellen zu erhalten).

Für jedes Steuerzeichen, welches einen konkreten Wert erfordert, muss ein solcher als Parameter der Funktion übergeben werden.

Die Klassenmethode format(*format*, [, *Wert1*, *Wert2*, ...]) der Klasse String gibt einen String in dem gewünschten Format zurück.

Formatstring

%b	boolescher Wert
%c	Zeichen
%d	ganzzahliger Wert in Dezimaldarstellung
%o	ganzzahliger Wert in Oktaldarstellung
%x	ganzzahliger Wert in Hexadezimaldarstellung (Kleinbuchstaben)
%X	ganzzahliger Wert in Hexadezimaldarstellung (Großbuchstaben)
%f	Fließkommazahl
%e	Fließkommazahl, Exponentenschreibweise
%g	Fließkommazahl, gemischte Schreibweise
%s	Strings, Objekte
%n	neue Zeile
%%	das Prozentzeichen

Steuerzeichen

-	linksbündige Ausgabe
+	immer mit Vorzeichen ausgeben
0	führende Nullen ausgeben
,	Zahlen mit Tausendermarkierung ausgeben (negative Zahlen in Klammern, kein Vorzeichen)
m	Mindestbreite
.p	bei %g, %e oder %f die Anzahl der Nachkommastellen

Beispiele:
%9.6f Fließkommazahl mit mindestens 9 Stellen, davon 6 Nachkommastellen

%+05d ganzzahliger Wert in Dezimaldarstellung, mindestens 5 Stellen, führende Nullen und Vorzeichen werden ausgegeben

Besser formartierte Ausgabe

```
for ( double x = 0.5; x <= 1.4; x += 0.1) {
   System.out.printf("%6.2f:   %9.6f%n", x, Math.sin(x));
}
```

```
 0,50:   0,479426
 0,60:   0,564642
 0,70:   0,644218
 0,80:   0,717356
 0,90:   0,783327
 1,00:   0,841471
 1,10:   0,891207
 1,20:   0,932039
 1,30:   0,963558
```

Zwei Stellen sind frei, da nur 4 Zeichen benötigt.

Eine Stelle ist frei, da nur 8 Zeichen benötigt.

Kapitel 7.5 – Parsen von Eingaben

Die Klasse Scanner

Die Klasse Scanner stellt einen einfachen Text-Scanner zur Verfügung, mit dessen Hilfe primitive Datentypen und Strings analysiert werden können.

Diese Klasse befindet sich im Paket java.util: import java.util.Scanner;

Wir erläutern die Anwendung dieser Klasse an drei Beispielen:
1. Eingabe von der Tastatur,
2. Eingabe durch einen String,
3. Eingabe aus einer Datei.

Weitere Einzelheiten, insbesondere die Verwendung regulärer Ausdrücke als Trennzeichen, können der Java-Dokumentation entnommen werden.

Scanner – Eingabe per Tastatur

```
Scanner sc = new Scanner(System.in);
int i, j;
System.out.print("Bitte geben Sie die erste Zahl ein: ");
i = sc.nextInt();
System.out.print("Bitte geben Sie die zweite Zahl ein: ");
j = sc.nextInt();
System.out.printf("Die Summe der Zahlen beträgt %d.%n", i+j);
```

Scanner – Eingabe über einen String

```
int i, j;
Scanner sc = new Scanner("34  45");
i = sc.nextInt();
j = sc.nextInt();
System.out.printf ("Die Summe der Zahlen beträgt %d.%n", i+j);
```

Scanner – Eingabe über eine Datei

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

    Scanner sc = new Scanner(new File("zahlen.txt"));
    double k = 0.0;
    while (sc.hasNextDouble()) {
        k += sc.nextDouble();
    }
    System.out.printf("Die Summe der Zahlen ist %.4f.%n", k);
}
```

Weiteres

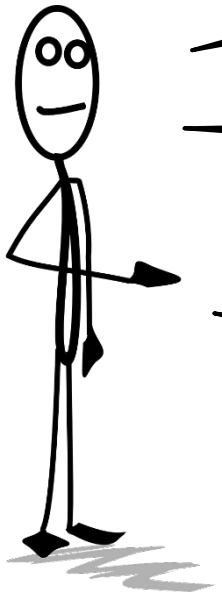
Java bietet sehr viele weitere Klassen und Methoden für die Ein- und Ausgabe von Daten an.

Beispiel: Betrachte die Beschreibung der Klassen

- Console,
- Path,
- Paths und
- Files

in der Java-Api an.

Zusammenfassung



Verschiedene Möglichkeiten zur Ein- und Ausgabe

Eine Grundlage bietet die Klasse Streams.

Klassen System, File und Scanner

Nächste Woche