MODULJOURNAL

Modul 411

Datenstrukturen und Algorithmen

Klasse IFZ 724-004-A

Lehrperson: Didier Buclin



Abbildung 1 // WISS Logo

**Ausgabe**

Wörter und Zahlen kann man beide ganz einfach mit dem gleichen Befehl über die Konsole ausgeben und dabei kann man sogar noch rechnen:

int x = 4;

int y = 6;

System.out.println("Die Summe von "+ x +" und "+ y +" ist "+ (x+y));

**Eingabe**

Um hingegen benutzereingaben einzulesen muss man ein Objekt der Klasse Scanner, welche eine Klasse der Standard Java-Library ist erzeugen.

Scanner scan = new Scanner(System.in);

int a = scan.nextInt();

String s = scan.next(); // scannt bis zum ersten Leerzeichen

String s2 = scan.nextLine(); // scannt die gesamte Eingabe

**BufferedReader**

Die Klasse BufferedReader aus der Java Library wird benutzt, um Textfiles einzulesen.

Erstmal um ein BufferedReader-Objekt zu Instanziieren muss man im Konstruktor angeben von welcher Quelle gelesen werden soll. In diesem Beispiel sollen Text-Dateien gelesen werden. Entsprechend ist die Quelle ein FileReader-Objekt. Der Konstruktor der FileReader-Klasse erfordert den Pfad zum zu lesenden File als String. Mit der Instanz des FileReader als Parameter erstellen wir zunächst eine BufferReader Instanz.

Mit dem folgenden Code kann man den Text (in diesem Fall ist es keine Text Datei, sondern ein String welcher die Struktur einer CSV Datei hat) bei jedem Semikolon unterteilt und dann ein wert pro Zeile ausgegeben.

String line = "Huber;Peter;1955";

String[] lineParts = line.split(";");

for (String p : lineParts) {

System.out.println(p);

}

Da man dem Benutzer eine Fehlermeldung geben möchte, wenn es ein Problem beim öffnen der Datei gab muss man dies mit try { } und catch { } machen.

Man kann für das einlesen von Files auch die Scanner Klasse verwenden was sich jedoch nicht empfiehlt da diese nicht "thred safe" ist, was bedeutet das man verhindern muss das ein gleichzeitiger Dateizugriff von Aussen stattfindet.

Als nächstes haben wir eine Aufgabe erledigt, bei welcher es darum ging eine CSV (Comma Separated Values) Datei die folgendermassen aus:

Huber;Peter;1955  
Meier;Irene;1960  
Muster;Anna;1974

Bei dieser Aufgabe ging es nicht mehr bei einem Zeilenumbruch den String zu teilen, sondern nach jedem Semikolon. Ausserdem mussten wir den Jahrgang in Jahre umwandeln.

**Codebeispiel BufferedReader**

public class main {

public main() {

/\* create reader variable \*/

BufferedReader bfr = null;

try {

String line;

/\* instantiate reader variable \*/

bfr = new BufferedReader(new FileReader(fileName));

/\* read as long as possible \*/

while ((line = bfr.readLine()) != null) {

String[ ] parts = line.split(";");

String name = parts[0];

String vorname = parts[1];

int age = LocalDate.now().getYear() - Integer.parseInt(parts[2]);  
 System.out.println(name+", "+vorname+", Alter: "+age+"Jahre");

}

bfr.close();

}

/\* in case something goes wrong... \*/

catch(IOException ex){

System.out.println(ex.toString());

ex.printStackTrace();

}

}

public static void main(String[ ] args) {

new main();

}

**Datenstrukturen**

Eine Datenstruktur ist ein Objekt zur Speicherung und Organisation von Daten. Die Daten werden in einer bestimmten Art und Weise angeordnet und verknüpft, um den Zugriff auf sie und ihre Verwaltung effizient zu ermöglichen.

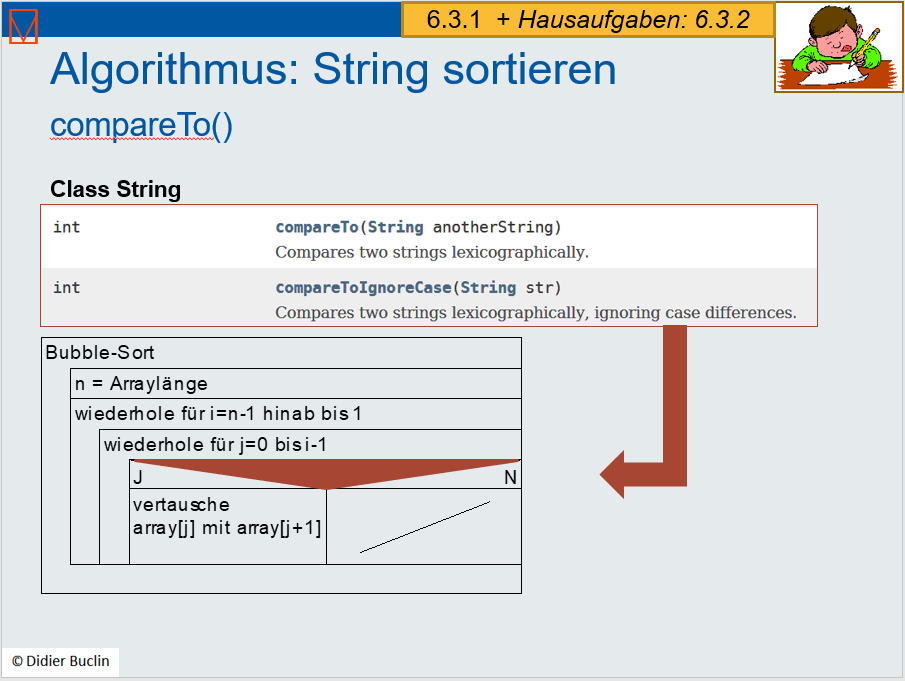
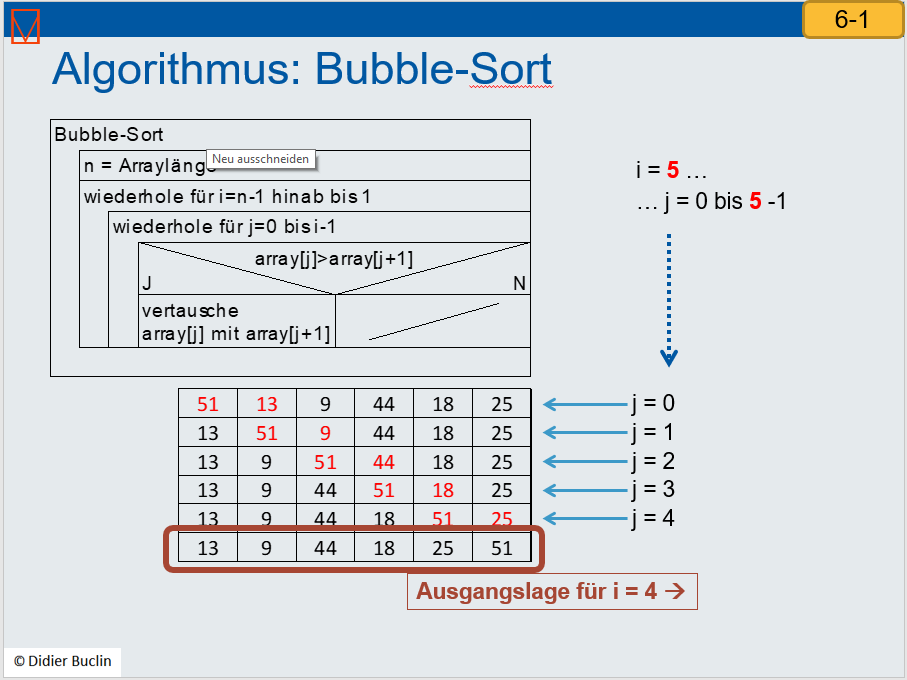
* Array
* ArrayList
* LinkedList
* Stack

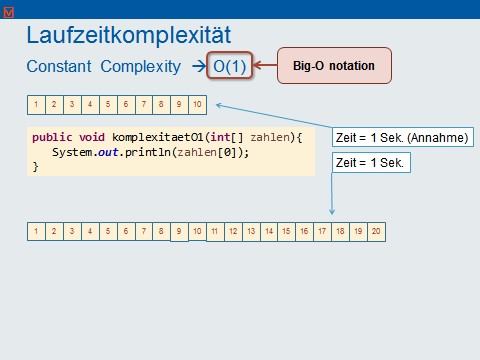
**Algorithmen**

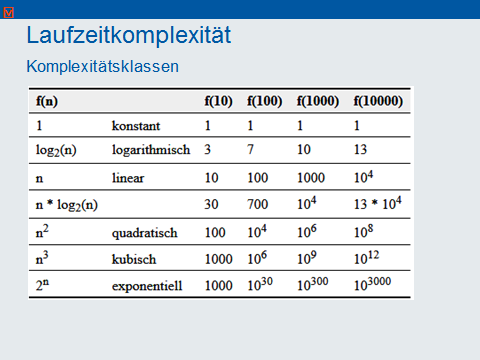
Verfahren zur Lösung eines Problems.

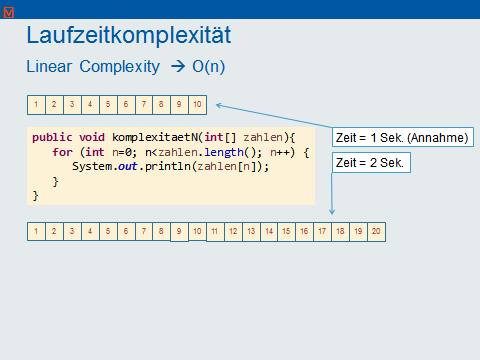
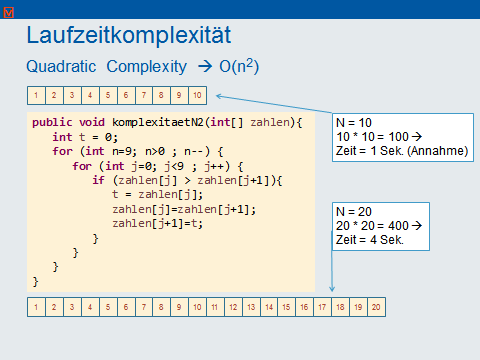
* Sortieralgorithmen (Bubble-Sort, Quick-Sort, Insert-Sort)
* Suchalgorithmen
* Kalenderrechnung
* Kryptographie

**Bubble-Sort**

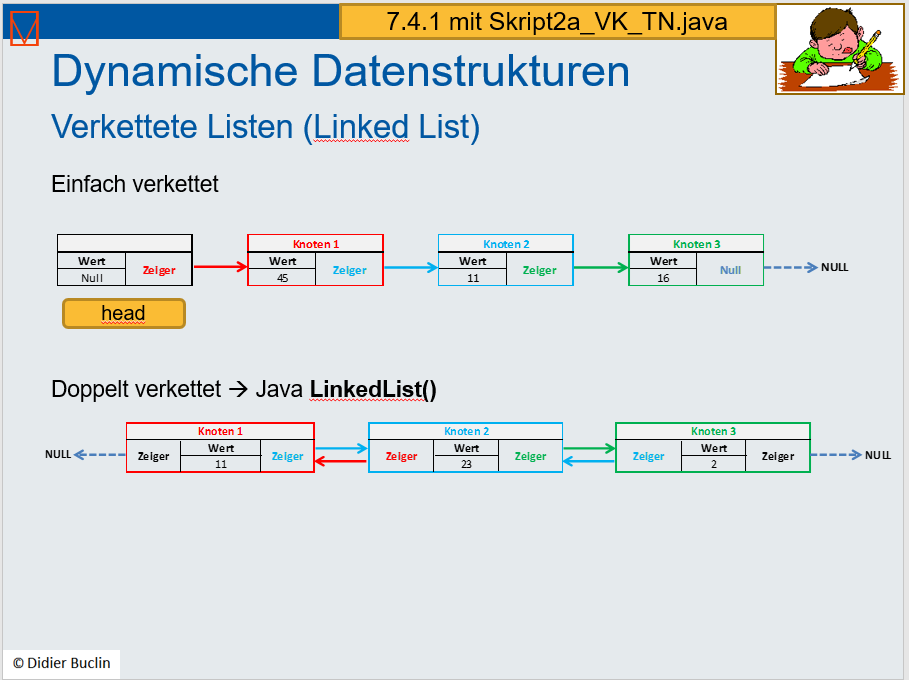
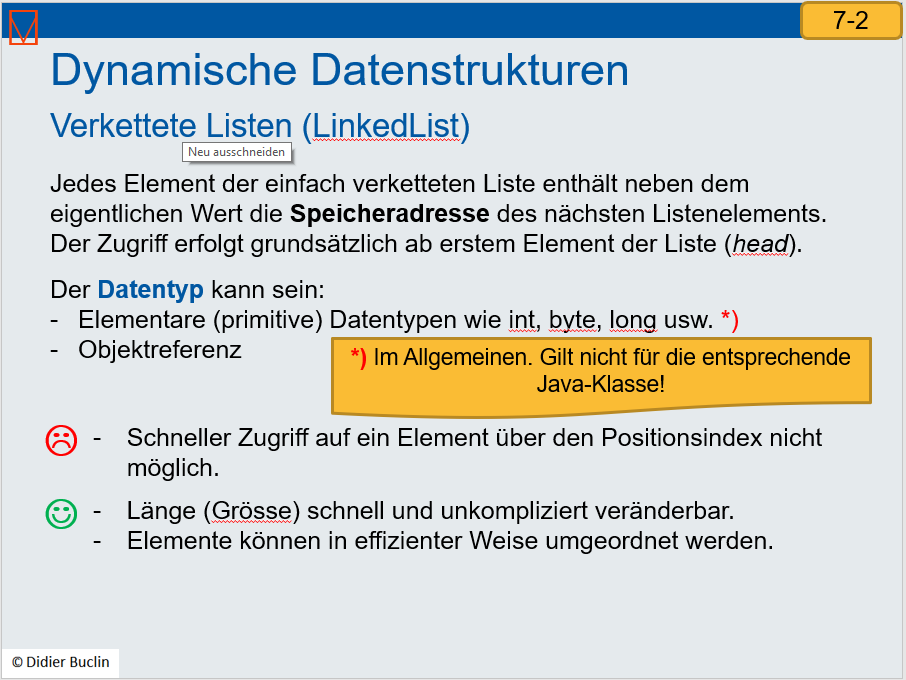


**Laufzeitkomplexität**





**Verkette Listen**

****

**Eigenschaften, Funktionsweise, Vor- und Nachteile von**

**ArrayList**

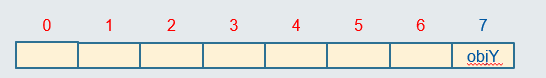
* Kann nur Objektreferenzen speichern (keine primitiven Datentypen)
* Geordnete Liste (Sequenz)
* Dynamisch erweiterbar (irgendwo), z.B am Schluss

*Pro*

* Schneller Zugriff auf ein Element über den Positionsindex

*Kontra*

* Elemente hinzufügen oder löschen ist rechenintensiv (langsam)
* Nicht Thread-Sicher

****

**Vector**

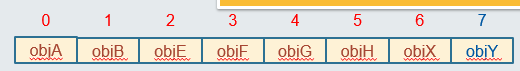
* Vector gilt für manche Entwickler als deprecated. Für Oracle aber nicht.
* Kann nur Objektreferenzen speichern (keine primitiven Datentypen)
* Geordnete Liste (Sequenz)
* Dynamisch erweiterbar (irgendwo), z.B am Schluss

*Pro*

* Synchronisiert (Thread-Sicher)

*Kontra*

* Langsamer als Arraylist (Synchronisation)



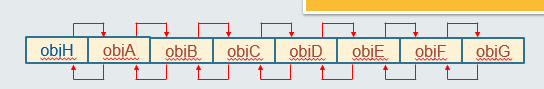
**LinkedList**

* In Java ist LinkedList eine doppelt verkettete Liste
* Kann nur Objektreferenzen speichern (keine primitiven Datentypen)
* Geordnete Liste (Sequenz)
* Dynamisch erweiterbar (irgendwo), z.B am Anfang

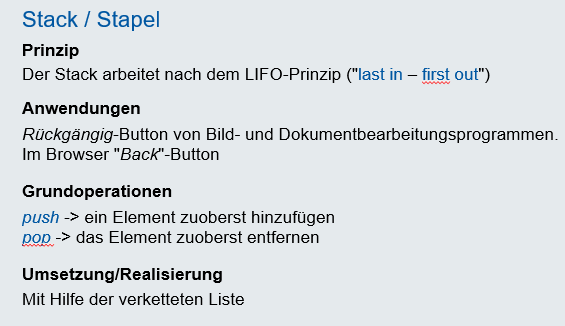
*Pro*

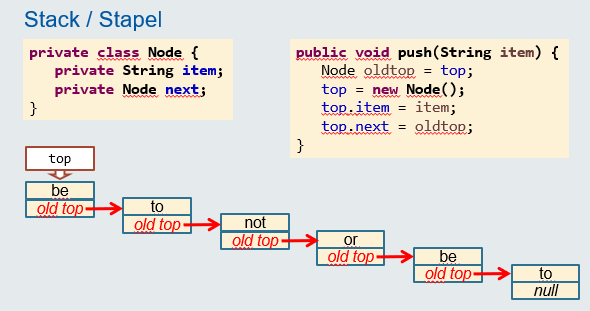
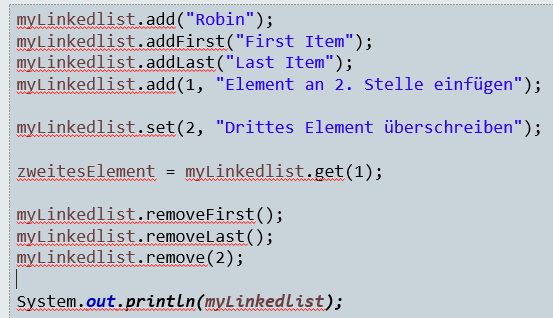
* Unterstütz das Durchschreiten von vorne nach hinten.
* Elemente hinzufügen oder löschen ist schneller als ArrayList

*Kontra*

* Nicht Thread-Sicher
* ****Element auslesen ist langsamer als ArrayList.

**Verkette Liste LinkedList() Methode**

****

****