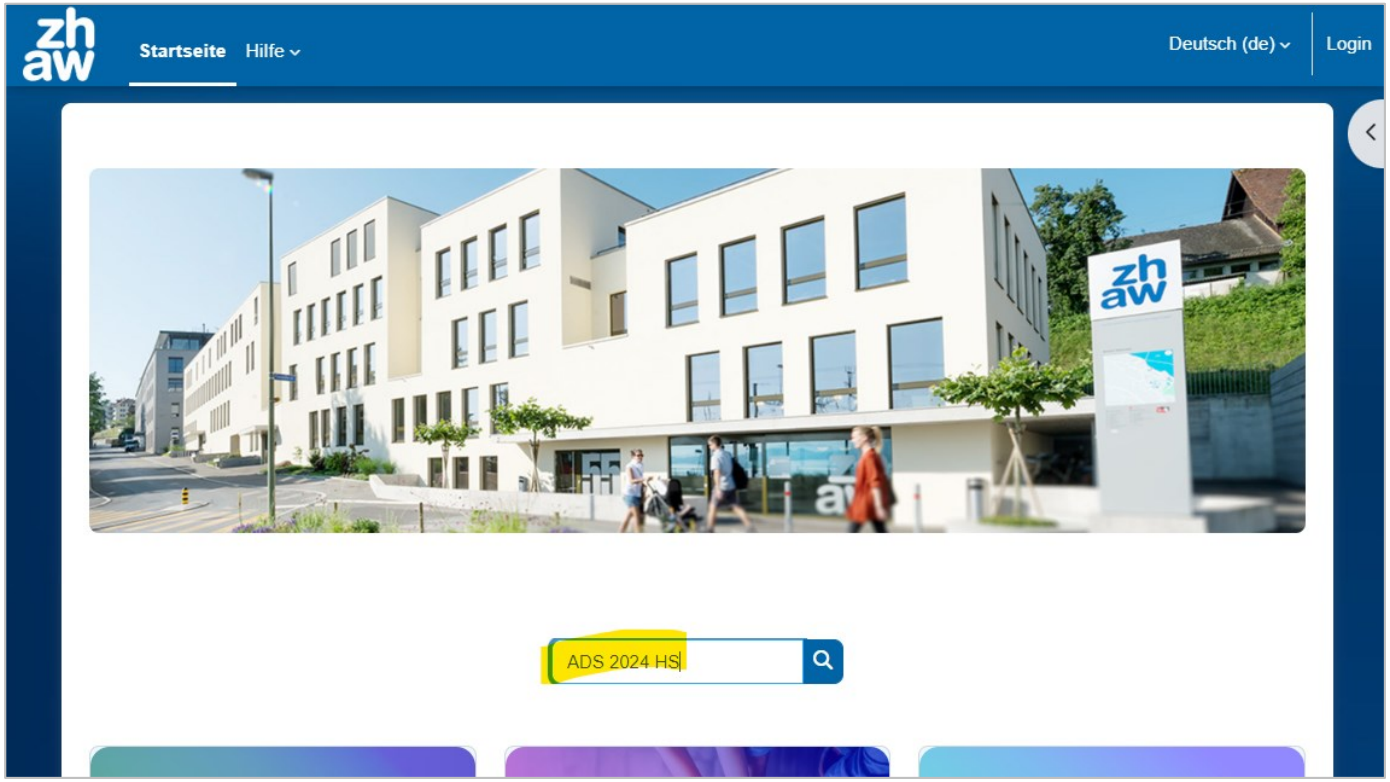
Einschreibung Moodle

Sie wurden mittels Evento in den Kurs eingetragen.

So suchen Sie den Kurs «ADS 2024 HS» (moodle.zhaw.ch):

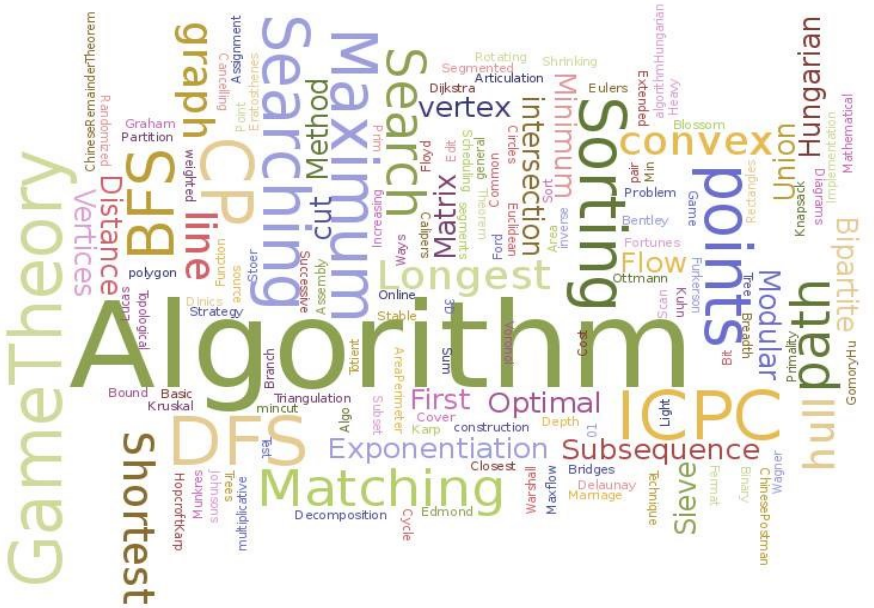


Falls Sie den Kurs nicht finden, sind Sie allenfalls nicht eingeschrieben → wenden Sie Sich an Ihre Dozent:in.

ADS: Algorithmen und Datenstrukturen

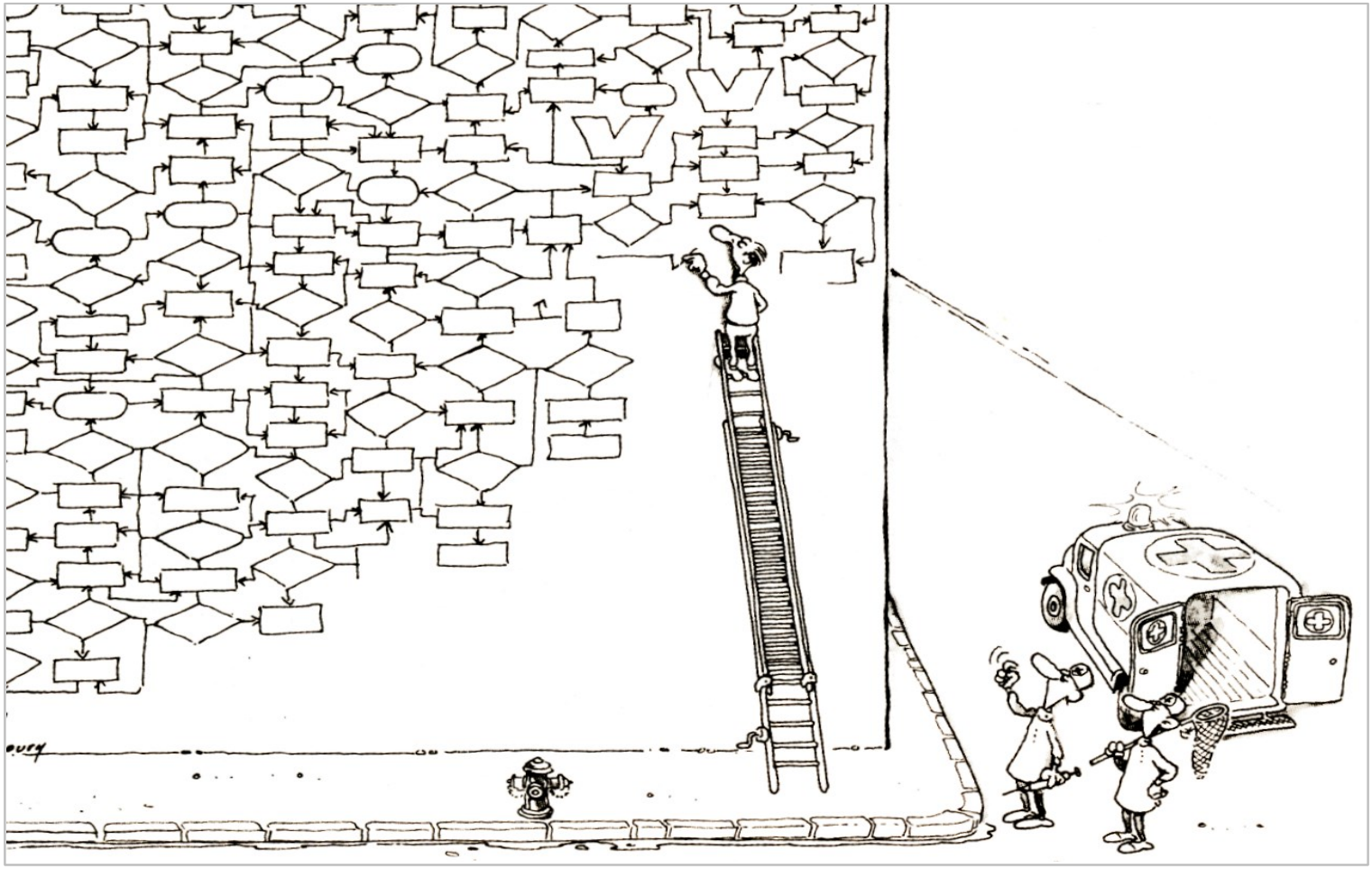
Vorlesung basiert auf Material von:

Stephan Metzler
Karl Rege
Marcela Ruiz
Jürgen Spielberger
Giovanni Toffetti
?



Einführung in das Modul ADS

Einführung: Motivation



Einführung: Motivation

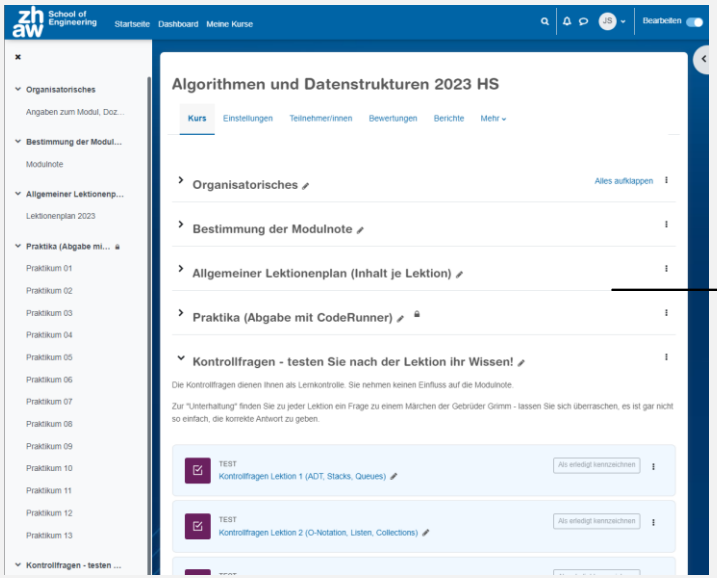
- Algorithmen und Datentypen/Strukturen sind zentral in jedem IT-System.
- Früher musste man noch vieles selber ausprogrammieren, z.B. Sortierung, heute meist Bestandteil der Programmiersprache oder Teil der Bibliotheken.

Zielsetzung dieser Vorlesung:

- Nicht alle Aufgaben sind gelöst → persönlichen Werkzeugkasten füllen. Sie lernen einige grundlegende Algorithmen-Entwurfsmuster anwenden.
- Sie kennen und üben die Vorgehensweise, die bei algorithmisch komplexen Problemen angewendet wird.
- Sie kennen die zentralen Datenstrukturen der Informatik und wie sie in Java umgesetzt werden → Einsatz von Java-Bibliotheken setzt ein Grund-Verständnis voraus (wie sortiert Java z.B. ein int[]-Array der Grösse 1'000'000?).
- Sie können den Rechenaufwand abschätzen, um auch zukünftig eine befriedigende Performance (mit grösserem Volumen) einer Anwendung sicherzustellen.

Einführung: Semesterplan

Auf Moodle finden Sie:



Moodle gilt!

- Lektionen- und Stundenplan
- Folien
- Praktika und Präsentationen
- Kontrollfragen und Lösungen
- Organisatorisches / Modulnote

Einführung: Lektionenplan

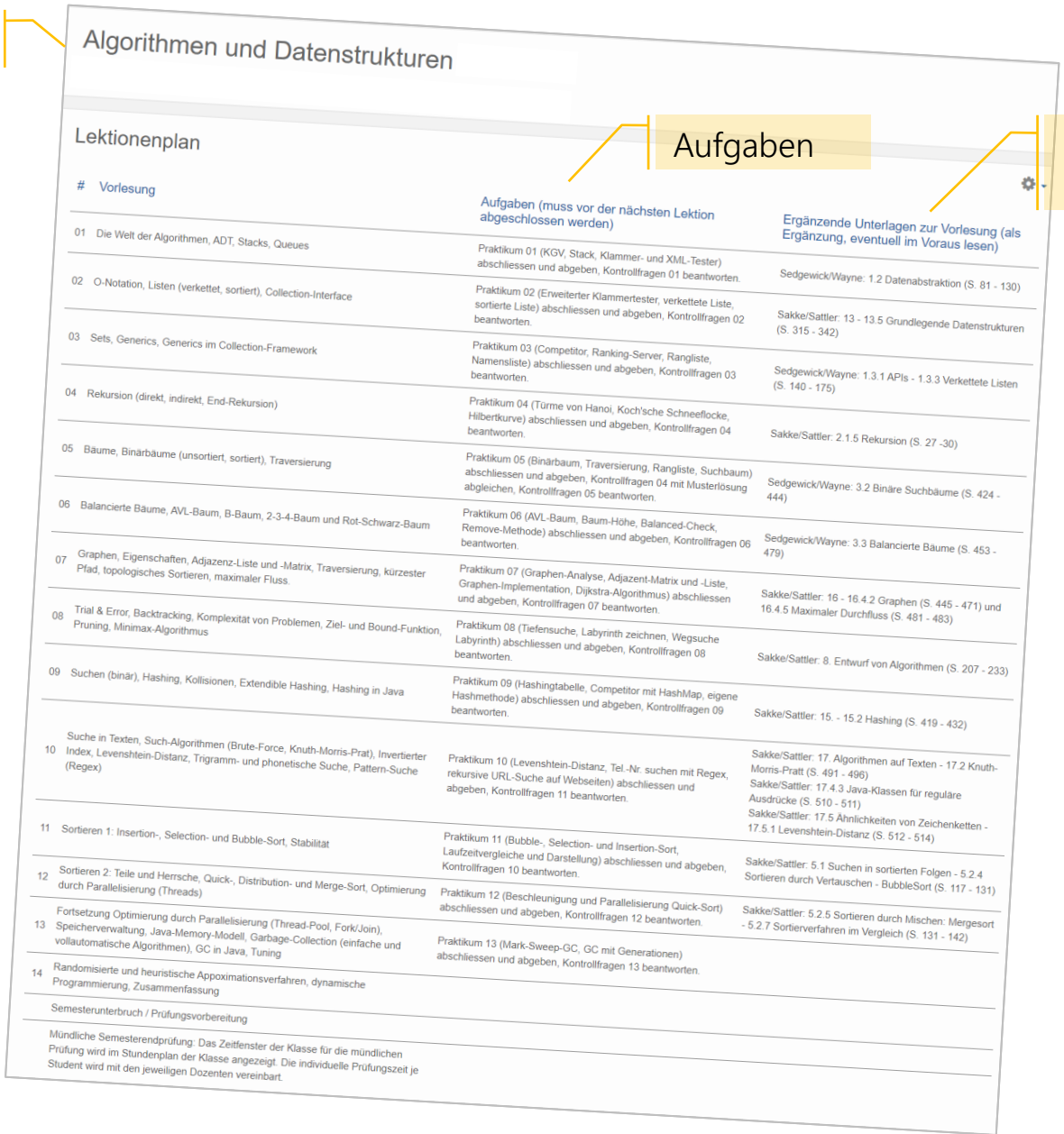
Zürcher Hochschule für Angewandte Wissenschaften
School of Engineering

Lektionenplan (siehe Moodle):

1. Einführung, ADT, Stacks, Queues
2. O-Notation, Listen (verkettet, sortiert), Collection-Interface
3. Sets, Generics, Generics im Collection-Interface
4. Rekursion (direkt, indirekt), End-Rekursion
5. Bäume, Binärbäume (unsortiert, sortiert), Traversierung
6. Balancierte Bäume, AVL-Baum, B-Baum, 2-3-4-Baum, Rot-Schwarz-Baum
7. Graphen, Eigenschaften, Adjazenz-Liste und -Matrix, Traversierung, kürzester Pfad, topologisches Sortieren, maximaler Fluss
8. Trial & Error, Backtracking, Komplexität von Problemen, Ziel- und Bound-Funktion, Pruning, Minimax-Algorithmus
9. Suchen (binär), Hashing, Kollisionen, Extendible Hashing, Hashing in Java
10. Suche in Texten, Such-Algorithmen (Brute-Force, Knuth-Morris-Prat), Invertierter Index, Levenshtein-Distanz, Trigramm- und phonetische Suche, Pattern-Suche (Regex)
11. Sortieren 1: Insertion-, Selection- und Bubble-Sort, Stabilität
12. Sortieren 2: Teile und Herrsche, Quick-, Distribution- und Merge-Sort, Optimierung durch Parallelisierung (Threads)
13. Fortsetzung Optimierung durch Parallelisierung (Thread-Pool, Fork/Join), Speicherverwaltung, Java-Memory-Modell, Garbage-Collection (einfache und vollautomatische Algorithmen), GC in Java, Tuning
14. Randomisierte und heuristische Approximationsverfahren, dynamische Programmierung, Zusammenfassung

Einführung: Lektionenplan

Auf Moodle



Algorithmen und Datenstrukturen

Lektionenplan

Aufgaben

Ergänzende Literatur

#	Vorlesung	Aufgaben (muss vor der nächsten Lektion abgeschlossen werden)	Ergänzende Unterlagen zur Vorlesung (als Ergänzung, eventuell im Voraus lesen)
01	Die Welt der Algorithmen, ADT, Stacks, Queues	Praktikum 01 (KGV, Stack, Klammer- und XML-Tester) abschliessen und abgeben, Kontrollfragen 01 beantworten.	Sedgewick/Wayne: 1.2 Datenabstraktion (S. 81 - 130)
02	O-Notation, Listen (verkettet, sortiert), Collection-Interface	Praktikum 02 (Erweiterter Klammertester, verkettete Liste, sortierte Liste) abschliessen und abgeben, Kontrollfragen 02 beantworten.	Sakke/Sattler: 13 - 13.5 Grundlegende Datenstrukturen (S. 315 - 342)
03	Sets, Generics, Generics im Collection-Framework	Praktikum 03 (Competitor, Ranking-Server, Rangliste, Namensliste) abschliessen und abgeben, Kontrollfragen 03 beantworten.	Sedgewick/Wayne: 1.3.1 APIs - 1.3.3 Verkettete Listen (S. 140 - 175)
04	Rekursion (direkt, indirekt, End-Rekursion)	Praktikum 04 (Türme von Hanoi, Koch'sche Schneeflocke, Hilbertkurve) abschliessen und abgeben, Kontrollfragen 04 beantworten.	Sakke/Sattler: 2.1.5 Rekursion (S. 27 -30)
05	Bäume, Binärbäume (unsortiert, sortiert), Traversierung	Praktikum 05 (Binärbaum, Traversierung, Rangliste, Suchbaum) abschliessen und abgeben, Kontrollfragen 04 mit Musterlösung abgleichen, Kontrollfragen 05 beantworten.	Sedgewick/Wayne: 3.2 Binäre Suchbäume (S. 424 - 444)
06	Balancierte Bäume, AVL-Baum, B-Baum, 2-3-4-Baum und Rot-Schwarz-Baum	Praktikum 06 (AVL-Baum, Baum-Höhe, Balanced-Check, Remove-Methode) abschliessen und abgeben, Kontrollfragen 06 beantworten.	Sedgewick/Wayne: 3.3 Balancierte Bäume (S. 453 - 479)
07	Graphen, Eigenschaften, Adjazenz-Liste und -Matrix, Traversierung, kürzester Pfad, topologisches Sortieren, maximaler Fluss.	Praktikum 07 (Graphen-Analyse, Adjazent-Matrix und -Liste, Graphen-Implementation, Dijkstra-Algorithmus) abschliessen und abgeben, Kontrollfragen 07 beantworten.	Sakke/Sattler: 16 - 16.4.2 Graphen (S. 445 - 471) und 16.4.5 Maximaler Durchfluss (S. 481 - 483)
08	Trial & Error, Backtracking, Komplexität von Problemen, Ziel- und Bound-Funktion, Pruning, Minimax-Algorithmus	Praktikum 08 (Tiefensuche, Labyrinth zeichnen, Wegsuche Labyrinth) abschliessen und abgeben, Kontrollfragen 08 beantworten.	Sakke/Sattler: 8. Entwurf von Algorithmen (S. 207 - 233)
09	Suchen (binär), Hashing, Kollisionen, Extendible Hashing, Hashing in Java	Praktikum 09 (Hashingtabelle, Competitor mit HashMap, eigene Hashmethode) abschliessen und abgeben, Kontrollfragen 09 beantworten.	Sakke/Sattler: 15. - 15.2 Hashing (S. 419 - 432)
10	Suche in Texten, Such-Algorithmen (Brute-Force, Knuth-Morris-Prat), Invertierter Index, Levenshtein-Distanz, Trigramm- und phonetische Suche, Pattern-Suche (Regex)	Praktikum 10 (Levenshtein-Distanz, Tel.-Nr. suchen mit Regex, rekursive URL-Suche auf Webseiten) abschliessen und abgeben, Kontrollfragen 11 beantworten.	Sakke/Sattler: 17. Algorithmen auf Texten - 17.2 Knuth-Morris-Pratt (S. 491 - 496) Sakke/Sattler: 17.4.3 Java-Klassen für reguläre Ausdrücke (S. 510 - 511) Sakke/Sattler: 17.5 Ähnlichkeiten von Zeichenketten - 17.5.1 Levenshtein-Distanz (S. 512 - 514)
11	Sortieren 1: Insertion-, Selection- und Bubble-Sort, Stabilität	Praktikum 11 (Bubble-, Selection- und Insertion-Sort, Laufzeitvergleiche und Darstellung) abschliessen und abgeben, Kontrollfragen 10 beantworten.	Sakke/Sattler: 5.1 Suchen in sortierten Folgen - 5.2.4 Sortieren durch Vertauschen - BubbleSort (S. 117 - 131)
12	Sortieren 2: Teile und Herrsche, Quick-, Distribution- und Merge-Sort, Optimierung durch Parallelisierung (Threads)	Praktikum 12 (Beschleunigung und Parallelisierung Quick-Sort) abschliessen und abgeben, Kontrollfragen 12 beantworten.	Sakke/Sattler: 5.2.5 Sortieren durch Mischen: Mergesort - 5.2.7 Sortierverfahren im Vergleich (S. 131 - 142)
13	Fortsetzung Optimierung durch Parallelisierung (Thread-Pool, Fork/Join), Speicherverwaltung, Java-Memory-Modell, Garbage-Collection (einfache und vollautomatische Algorithmen), GC in Java, Tuning	Praktikum 13 (Mark-Sweep-GC, GC mit Generationen) abschliessen und abgeben, Kontrollfragen 13 beantworten.	
14	Randomisierte und heuristische Appoximationsverfahren, dynamische Programmierung, Zusammenfassung		
	Semesterunterbruch / Prüfungsvorbereitung		
	Mündliche Semesterendprüfung: Das Zeitfenster der Klasse für die mündlichen Prüfung wird im Stundenplan der Klasse angezeigt. Die individuelle Prüfungszeit je Student wird mit den jeweiligen Dozenten vereinbart.		

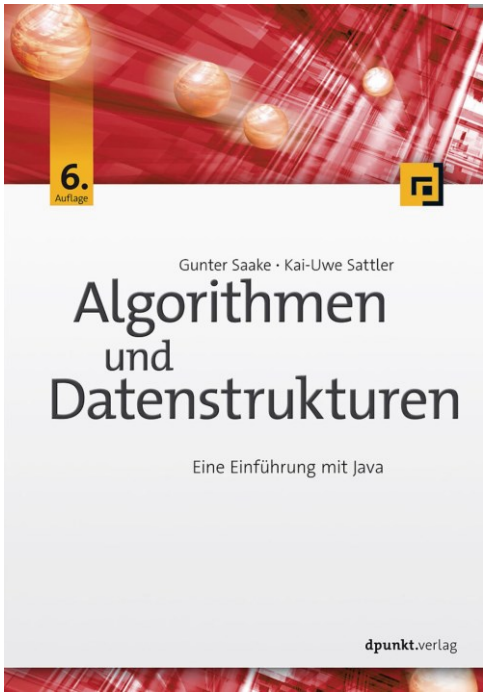
Einführung: Bücher

1. Buchempfehlungen, nicht obligatorisch aber hilfreich.
2. Es gibt viele, der Inhalt ist ähnlich.
3. Verwenden Sie jenes, das SIE mögen.
4. Vorgeschlagene Bücher.

- Relativ einfach zu verfolgen.
- Java-ähnlicher Code.
- Empfehlung gemäss Evento.
- Auch auf Englisch verfügbar.
- Im Lektionenplan finden Sie Referenzen für dieses Buch.
- https://algs4.cs.princeton.edu/home/ (Englisch)

Einführung: Bücher

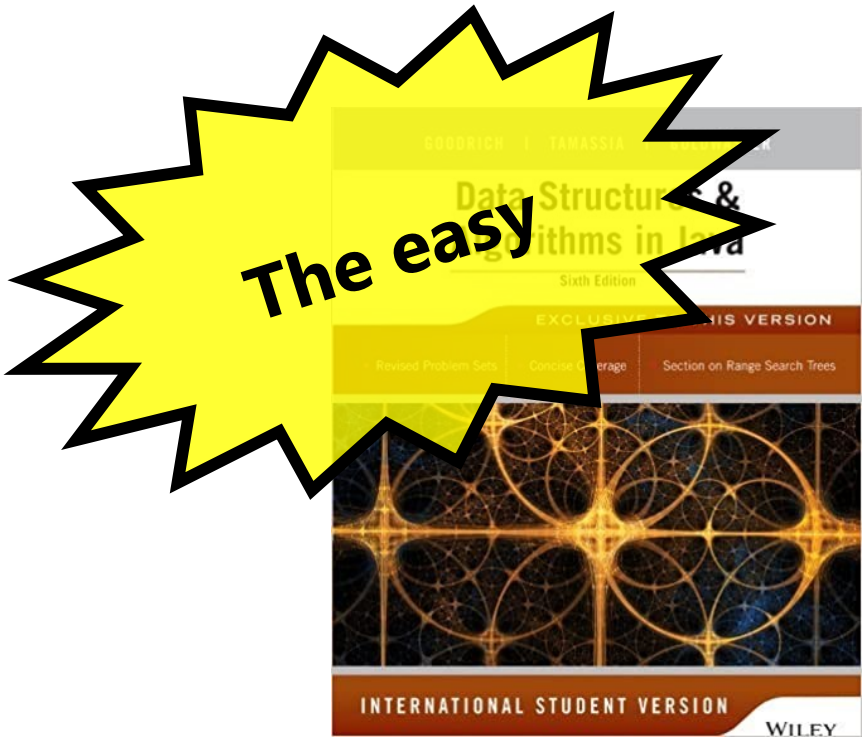
1. Buchempfehlungen, nicht obligatorisch aber hilfreich.
2. Es gibt viele, der Inhalt ist ähnlich.
3. Verwenden Sie jenes, das SIE mögen.
4. Vorgeschlagene Bücher.



- 6. Auflage.
- Nur auf Deutsch.
- Empfehlung gemäss Evento.
- Im Lektionenplan finden Sie Referenzen für dieses Buch.

Einführung: Bücher

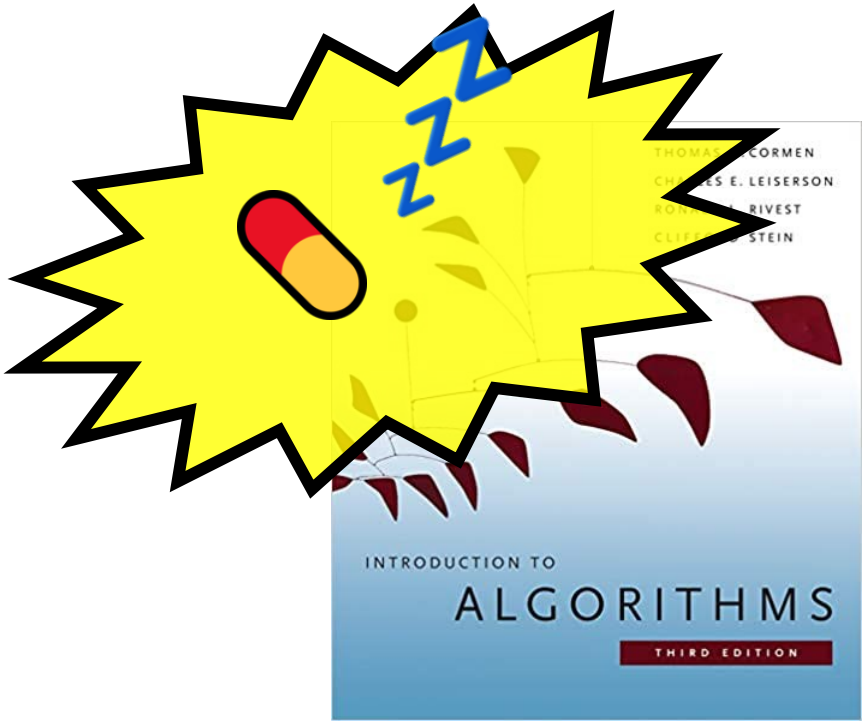
1. Buchempfehlungen, nicht obligatorisch aber hilfreich.
2. Es gibt viele, der Inhalt ist ähnlich.
3. Verwenden Sie jenes, das SIE mögen.
4. Vorgeschlagene Bücher.



- Leicht verständlich
- Nur auf Englisch
- Java-ähnlicher Code
- Fast keine Mathematik

Einführung: Bücher

1. Buchempfehlungen, nicht obligatorisch aber hilfreich.
2. Es gibt viele, der Inhalt ist ähnlich.
3. Verwenden Sie jenes, das SIE mögen.
4. Vorgeschlagene Bücher.



- Nur wenn Sie unter akuter Schlaflosigkeit leiden.
- Oder wenn Ihr Gehirn für Mathe fest verdrahtet ist.
- Nur auf Englisch.

Einführung: Praktika

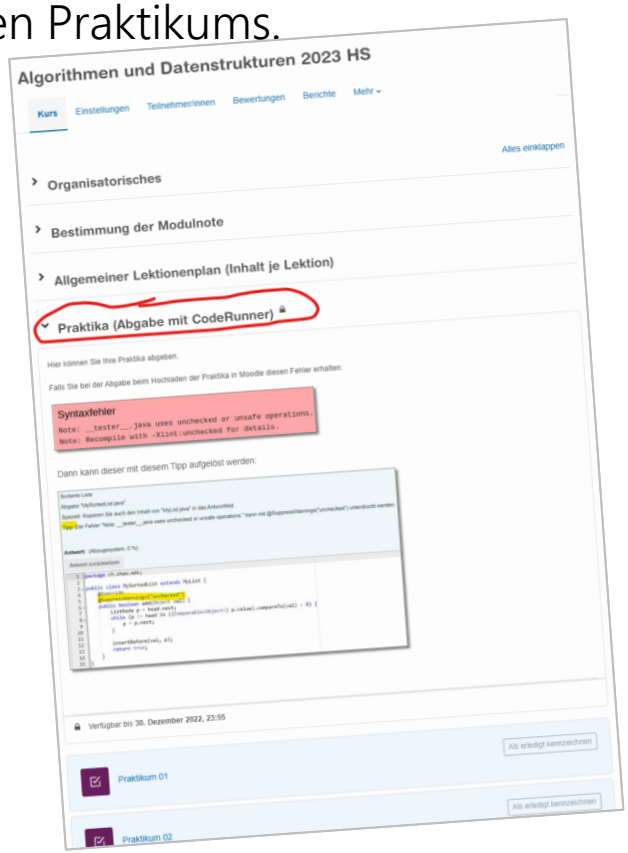
Zürcher Hochschule für Angewandte Wissenschaften
zh aw School of Engineering

Top! 30% der Note

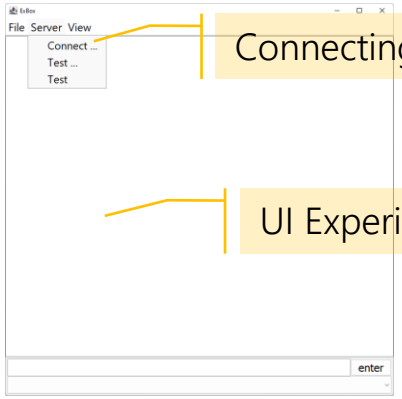
Die Praktika sind SEHR wichtig!

Lösen der Praktikumsaufgaben:

- ✓ Aufgabenstellung und Code in Moodle holen.
- ✓ Reichen Sie Ihre Lösungen über Moodle ein (siehe Bild, Details nächste Folie).
- ✓ Deadline für jedes Praktikum: vor Beginn des nächsten Praktikums.
- ✓ Die Qualität der Lösung wird nicht bewertet.
  Die Lösung muss «nur» funktionieren.
- ✓ ACHTUNG: «**`package ch.zhaw.ads`**» muss verwendet werden.
- ✓ Bewertung Praktika: jede erfolgreich abgegebene Aufgabe gibt einen Punkt
  - Die Praktika können stichprobenweise kontrolliert und kommentiert werden (Feedback).
  - Es müssen alle Praktika abgegeben werden
  - Die Praktikumsnote wird linear zwischen 0 Punkten (Note 1.0) und der maximalen Punktzahl (Note 6.0) berechnet.
  - Sie dürfen die Praktika in Gruppen von 2 bis 3 Studierenden lösen. Es muss aber jeder Studierende die Lösung auf Moodle abgeben.



Einführung: Praktika

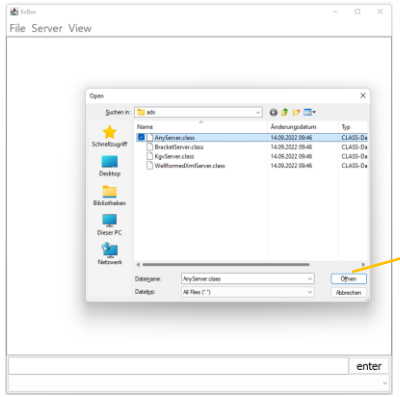


Connecting with Server

UI Experimentierbox

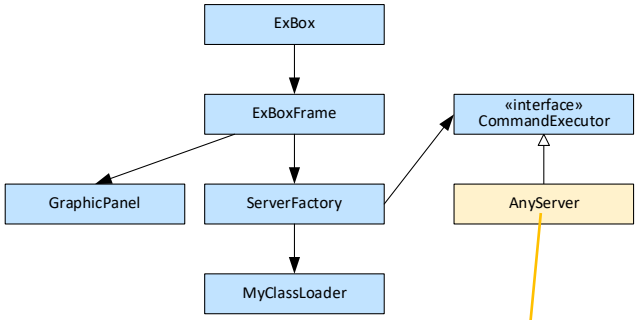
Top!
30% der Note

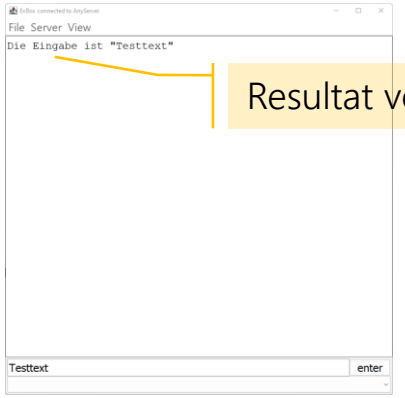
Zürcher Hochschule für Angewandte Wissenschaften
zh aw School of Engineering



Auswahl des Servers

Die Experimentierbox:





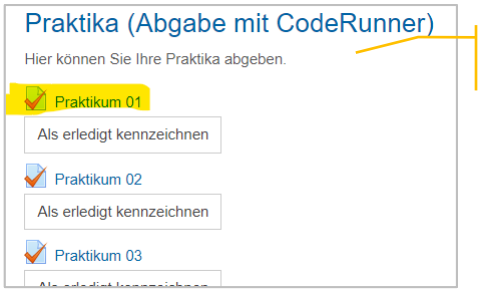
Resultat von AnyServer

Ihre Lösungen zu den Praktika

Einführung: Praktika

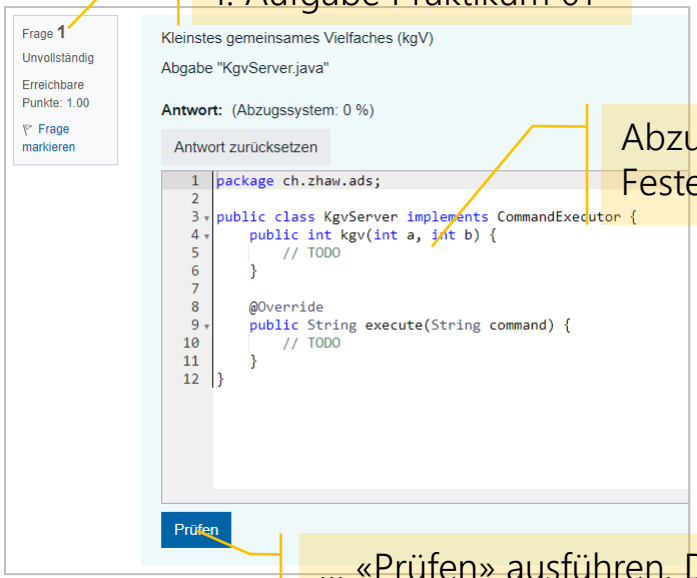
Top! 30% der Note

So geben Sie Ihre Lösungen über Moodle ab:



Link auf Moodle anklicken.

1. Aufgabe Praktikum 01



Abzugebenden Code in Fester einfügen und …

… «Prüfen» ausführen. Das kann beliebig oft wiederholt werden.

Sie erhalten sofort Feedback zu den ausgeführten Unit-Tests.

Einführung: Praktika

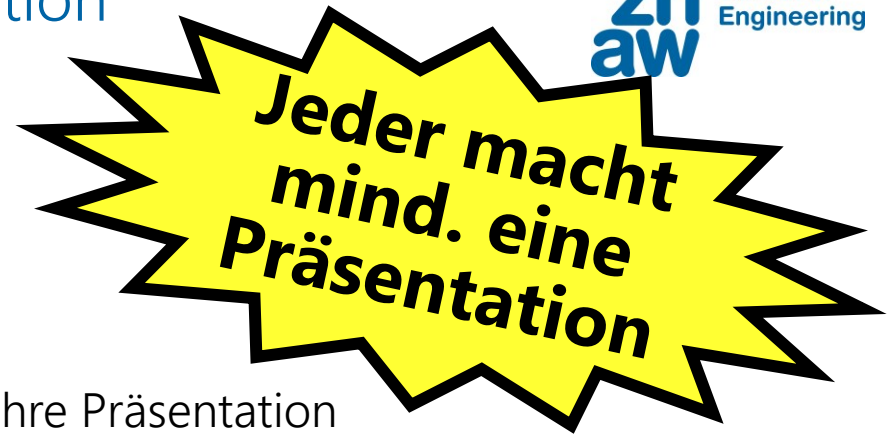
Die Praktika sind SEHR wichtig!

Lösen der Praktikumsaufgaben:

- Die meisten Aufgaben finden Sie irgendwo im Netz.
- Widerstehen Sie der Versuchung, diesen Code zu übernehmen (das Abgabetool hat einen Plagiat-Check).
- Sie lernen nichts dabei → die Prüfung zeigt, was Sie können und was nicht.

➢ Wenn Sie trotz eines ehrlichen Lösungsversuchs nicht weiterkommen, fragen Sie.

Einführung: Praktikumspräsentation



- Wöchentliche Präsentation
  in Gruppen von 2 – 3 Studierenden.
- Auswahl des Themas erfolgt z.B. über
  Moodle («Präsentation Praktikum»).
- Sie haben maximal 10 Minuten Zeit für Ihre Präsentation
  (physisch oder online). PowerPoint- und Live-Demos sind willkommen.
- Besprechen Sie während Ihrer Präsentation die folgenden Punkte:
  1. Was haben Sie über diese Übung gedacht, bevor Sie sie gelöst haben?
  2. Mit welchen Problemen sind Sie konfrontiert worden?
  3. Welche Literatur haben Sie durchgesehen, um die Übung lösen zu können?

Denken Sie daran:
DIE Lösung gibt es nicht,
Ihre Ideen sind wertvoll!

Nerd-Zone

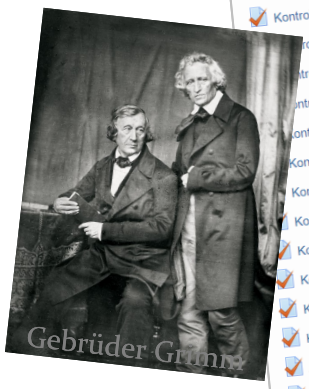
Markiert mit Nerd-Tafel

- In der Nerd-Zone finden Sie ergänzende Folien zu den besprochenen Themen.
- Diese sind nicht Teil der Praktika oder Prüfungen.

Einführung: Kontrollfragen

- Arbeiten Sie nach der Vorlesung die entsprechenden Kontrollfragen auf Moodle durch.
- Wenn Sie etwas nicht verstanden haben, fragen Sie mich in der nächsten Vorlesung oder im Praktikum.
- Fragen Sie im Plenum, so haben alle etwas davon.
- Kontrollfragen werden nicht bewertet.
- Sie erhalten die korrekten Antworten unmittelbar nach der Beantwortung.
- Zur «Unterhaltung» hat jede Lektion eine Frage zu einem Märchen der Gebrüder Grimm:

1. Der gestiefelte Kater
2. König Drosselbart
3. Schneewittchen
4. Sterntaler
5. Hans im Glück
6. Das Rumpelstilzchen
7. Froschkönig
8. Rapunzel
9. Der Wolf und die 7 Geisslein
10. Frau Holle
11. Aschenputtel
12. Die Bremer Stadtmusikanten
13. Hänsel und Gretel
14. Das tapfere Schneiderlein

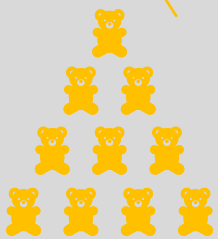


Einführung: Semester-End-Prüfung / SEP

- Mündliche Prüfung 20 Minuten, keine Hilfsmittel.
- Zeitplan wird zeitnah erstellt (Info erfolgt durch jeweiligen Dozierenden).
- Die mündlichen Prüfungen finden vor Ort an der ZHAW statt. Bei Corona-Auflagen bis zum Semesterende wird die Prüfung allenfalls mit Teams (remote) durchgeführt.
- Unterschied Praktika / Prüfung:
  - Praktikumspunkte zu bekommen ist leicht.
  - Im Praktikum wiegt ein (ehrlich gemachter) Fehler nicht schwer.
  - Prüfungspunkte zu bekommen ist schwerer.
  - Sie müssen auf die richtige Antwort kommen, sonst gibt es keine Punkte.

In jeder Vorlesung hat es eine kleine fakultative Knobelaufgabe die Sie von Hand, oder (fast immer) mit einem Algorithmus lösen können.

Wenn man den Algorithmus kennt, dann ist es immer einfach…



Kleine Knobelaufgabe für die Freizeit 😊:

1. Wie kann die Pyramide auf den Kopf gestellt werden, indem 3 Gummibärchen verschoben werden.
2. Wie sieht ein effizienter Algorithmus aus, der die Bewegungen (minimale Anzahl) für eine beliebig hohe Pyramide findet?

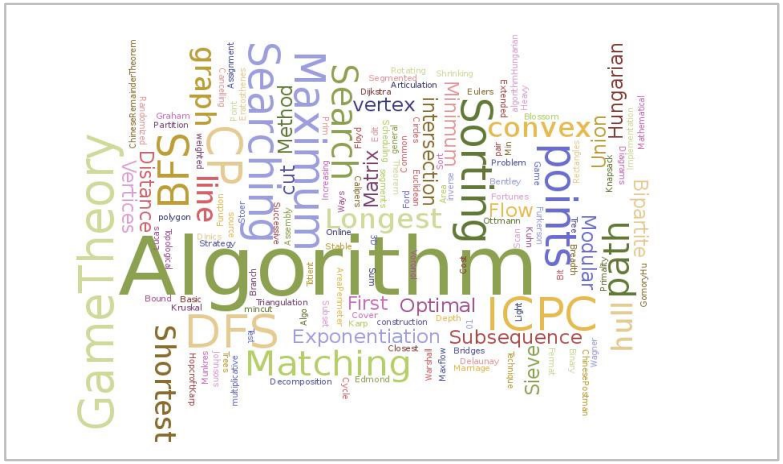
Im Verlauf der Vorlesungen werden wir verschiedene Algorithmen-Muster kennen lernen. Nach der gesamten Vorlesungsreihe sollten Sie dann geeignete Algorithmen kennen, um die Knobelaufgaben effizient zu lösen.

Kleiner Tipp: Sie finden die Lösung auf den Notizenseiten.

Einführung in ADS

Algorithmen und Datenstrukturen

- Sie wissen, was ein Algorithmus ist.
- Sie kennen das Konzept des ADTs.
- Sie wissen, wie die Stack- (Stapel) Datenstruktur funktioniert.
- Sie können diese selber implementieren.
- Sie kennen Anwendungen dieser Datenstruktur.
- Sie wissen, wie die Queue (Warteschlange) Datenstruktur funktioniert.
- Sie können diese implementieren.



Einführung: Geschichte der Algorithmen

Einführung: Geschichte der Algorithmen

- 300: Euklids Algorithmus zur Bestimmung des grössten gemeinsamen Teilers ggT in seinem 7. Buch der Elemente.
- 800: Mohammed ibn Musa abu Djafar al Choresmi الخوارزمي أبو عبد الله محمد بن موسى
  - Übernahm indische Zahlen 1..9 und führte die 0 ein, 10er-System.
  - Verschiedene Lehrbücher: bis 16. Jh. in Europa als Standard verwendet (Bildung des Begriffs Algorithmus aus seinem Namen und dem griechischen «arithmo» für Zahl).
- 1574: das Rechenbuch von Adam Riese.
- 1614: erste Logarithmentafel (in ca. 30 Jahren erstellt).
- 1703: binäres Zahlensystem von Leibniz.
- 1931: Gödels Unvollständigkeitssatz: Bei hinreichend starken widerspruchsfreien Systemen gibt es immer unbeweisbare Aussagen.
- 1936: Church'sche These: stellt die Behauptung auf, dass eine Turingmaschine alle von Menschen berechenbaren mathematischen Funktionen lösen kann (http://de.wikipedia.org/wiki/Turingmaschine).
- Danach umfangreicher Ausbau der Algorithmentheorie.

Einführung: Was ist ein Algorithmus?

- Aufgabenstellung:
  - Gegeben: sind zwei ganze Zahlen a, b.
  - Gesucht: c soll der grösste gemeinsame Teiler (ggT) von a und b sein.
- Einfacher Algorithmus:
  - Setze c zum Minimum der beiden Zahlen.
  - Subtrahiere von c solange 1, bis die Divisionen a/c und b/c keinen Rest mehr ergeben.
- In Java:

```
c = Math.min(a, b);
while ((a % c != 0) || (b % c != 0)) c--;
```

- Lösung wird gefunden, aber?

Einführung: Was ist ein Algorithmus?

- Euklidischer Algorithmus: Der ggT zweier Zahlen a, b ist auch ggT ihrer Differenz, falls a ≠ b.
- Es gilt also:

ggT(a, b) =
1. a = b → a, resp. b, ist der ggT
2. a > b → ggT(a - b, b)
3. a < b → ggT(b - a, a)

```
int ggT(int a, int b) {
   if (a > b) return ggT(a - b, b);
   else if (a < b) return ggT(a, b - a);
   else return a;
}
```

Rekursive Algorithmen haben immer einen gewissen Overhead, daher ist es, wenn man nur die Performance betrachtet, besser einen iterativen Algorithmus zu verwenden.

Einführung: Was ist ein Algorithmus?

- Euklidischer Algorithmus: Iterativ:

ggT(a, b) =

1. a = b → a, resp. b, ist der ggT
2. a > b → a = a - b
3. a < b → b = b – a

Der ggT von a und b ist auch ggT von (a-b).

```
while (a != b) {
   if (a > b) a = a - b;
   else b = b - a;
}
return a;
```

Kann das noch verbessert werden?

Einführung: Was ist ein Algorithmus?

- Ist a viel grösser, wird wiederholt b abgezogen, bis a < b. Der Werte, der dabei entsteht, kann auch als a mod b berechnet werden (Hippasos von Metapont).

ggT(a, b) =

1. b = 0 → a ist der ggT
2. a > b → a = a mod b
3. a < b → b = b mod a

a und b sind beim Start ungleich 0.

```
if (a == 0) return b;
while (b != 0) {
   c = a % b;
   a = b;
   b = c;
}
return a;
```

Ist vor der ersten Ausführung a > b werden im ersten Durchlauf die Werte von a und b ausgetauscht (32 % 92 = 32)

Für die Zahlen 92 und 32 haben wir nach 3 Schritten bereits die Lösung, anstatt 9 Schritten beim Euclid-Algorithmus.

Einführung: Was ist ein Algorithmus?

Ein Algorithmus ist eine Anleitung zur Lösung einer Aufgabenstellung, die so präzise formuliert ist, dass sie “mechanisch” ausgeführt werden kann.

- Mögliche Beschreibungsmethoden sind:
  - Prosa
  - Pseudocode
  - Programmiersprache (hier immer Java)

- Algorithmisches Problem: Problem, das mit einem Algorithmus gelöst werden kann.

Einführung: Was ist ein Algorithmus?

Eigenschaften von Algorithmen:

- Determiniertheit: Identische Eingaben führen stets zu identischen Ergebnissen.
- Determinismus: Ablauf des Verfahrens ist an jedem Punkt fest vorgeschrieben (keine Wahlfreiheit).
- Terminierung: Für jede Eingabe liegt das Ergebnis nach endlich vielen Schritten vor.
- Effizienz: «Wirtschaftlichkeit» des Aufwands relativ zu einem vorgegebenen Massstab (z.B. Laufzeit, Speicherplatzverbrauch).

Aber: Es darf auch nicht-terminierende, ineffiziente, nicht-deterministische und nicht-determinierte Algorithmen geben!

- Programm versus Algorithmus:
  - Jedes Programm repräsentiert einen bestimmten Algorithmus.
  - Ein Algorithmus wird durch viele verschiedene Programme repräsentiert.
  - Programmieren setzt Algorithmen-Entwicklung voraus: Kein Programm ohne Algorithmus !

Abstrakter Datentyp

Abstrakter Datentyp

Ein grundlegendes Konzept in der Informatik ist das Information Hiding:

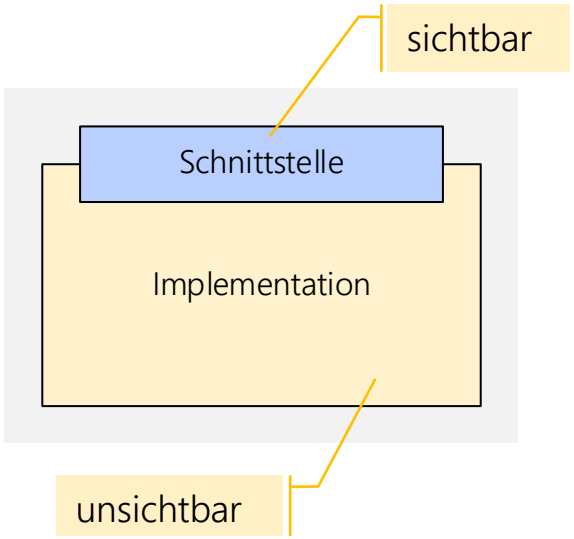
- Nur gerade so viel wie für die Verwendung einer Klasse nötig ist, wird auch sichtbar gemacht.
- Jede Klasse besteht aus einer von aussen sichtbarer Schnittstelle und aus einer ausserhalb des Moduls unsichtbaren Implementation.

Schnittstelle:

- Ein wesentliches Konzept von ADT's ist die Definition einer Schnittstelle in Form von Zugriffsmethoden.
- Nur diese Zugriffsmethoden können die eigentlichen Daten des ADT's lesen oder verändern.
- Dadurch ist sichergestellt, dass die innere Logik der Daten erhalten bleibt.

Implementation:

- Die Implementation eines ADT's kann verändert werden, ohne dass dies das verwendende Programm merkt.
- Man kann auch verschiedene Implementationen in Erwägung ziehen, die sich zum Beispiel bezüglich Speicherbedarfs und Laufzeit unterscheiden.



Abstrakter Datentyp

- ADTs sind ein allgemeines Konzept.
- In verschiedenen Sprachen unterschiedlich realisiert.
- In Java am saubersten durch Interfaces & Klassen:
  - interface: Beschreibung der Schnittstelle (aber ohne Implementation)
  - class: Implementation der Schnittstelle

```
interface Stack {
   Object push(Object obj);
   Object pop();
}
```

sichtbar

```
class MyStack implements Stack {
      Object push(Object obj) {
      // Implementation
   }
}
```

unsichtbar

```
Stack stack = new MyStack();
```

Wir verzichten vorerst auf Typparameter (generische Interfaces/Methoden). Wir werden diese später einführen und detailliert betrachten.

Abstrakter Datentyp

«Abstrakte» Datentypen in Java:

- Einfache Datentypen:
  - byte, short, int, long
  - float, double
  - char
  - boolean
  - …
- Referenz Datentypen:
  - Array
  - String
  - Objects

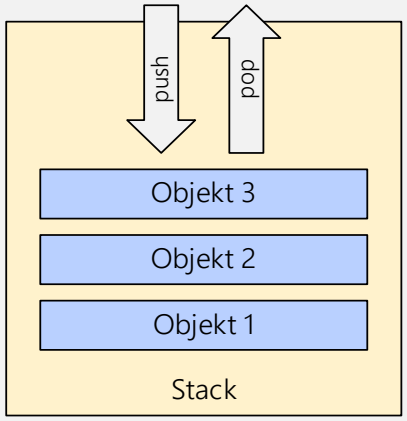
Thema dieser Vorlesung

- Was man z.T. selbst machen muss, oder was das JDK zur Verfügung stellt, man aber sicher verstehen muss:
  - Stacks
  - Collections
  - List
  - Queues
  - Hashtable
  - Trees
  - …

Unser erstes Thema

Stacks

Stack: als ADT



Der Stack (auch Stapel- oder Kellerspeicher) ist eine LIFO-Datenstruktur (Last In First Out), die Objekte als Stapel speichert:

- neue Objekte können nur oben auf den Stapel gelegt werden.
- auch das Entfernen von Objekten vom Stapel ist nur oben möglich.

`Object push(Object x)`	Legt x auf den Stapel.
`Object pop()`	Entfernt das oberste Element und gibt es zurück.
`Boolean isEmpty()`	Gibt true zurück, falls der Stapel leer ist.
`Object peek()`	Gibt das obereste Element zurück, ohne es zu entfernen.
`void removeAllElements()`	Leert den ganzen Stapel.
`Boolean isFull()`	Gibt true zurück, fall der Stapel voll ist.

Stack: als Java-Interface (ADT)

Zürcher Hochschule für Angewandte Wissenschaften
School of Engineering

```
/**
 * Interface für abstrakten Datentyp (ADT) Stack.
 */
public interface Stack {


   /**
    * Legt eine neues Objekt auf den Stack, falls noch nicht voll.
    * @param x ist das Objekt, das dazugelegt wird.
    */
   public Object push(@NotNull Object x) throws StackOverflowError;


   /**
    * Entfernt das oberste und damit das zuletzt eingefügte Objekt.
    * Ist der Stack leer, wird null zurückgegeben.
    * @return Gibt das oberste Objekt zurück oder null, falls leer.
    */
   public Object pop();


}
```

Java hat bereits eine Stack-Klasse. Im Beispiel soll lediglich gezeigt werden, wie ein ADT mittels Java deklariert werden kann.

Stack: als Java-Interface (ADT)

```
/**
 * Testet, ob der Stack leer ist.
 * @return Gibt true zurück, falls der Stack leer ist.
 */
public boolean isEmpty();

/**
 * Gibt das oberste Objekt zurück, ohne es zu entfernen.
 * Ist der Stack leer, wird null zurückgegeben.
 * @return Gibt das oberste Objekt zurück oder null, falls leer.
 */
public Object peek();

/**
 * Entfernt alle Objekte vom Stack. Ein Aufruf von isEmpty()
 * ergibt nachher mit Sicherheit true.
 */
public void removeAllElements();
```

Stack: als Java-Interface (ADT)

```
    /**
     * Testet, ob der Stack voll ist.
     * @return Gibt true zurück, falls der Stack voll ist.
     */
    public boolean isFull();

}
```

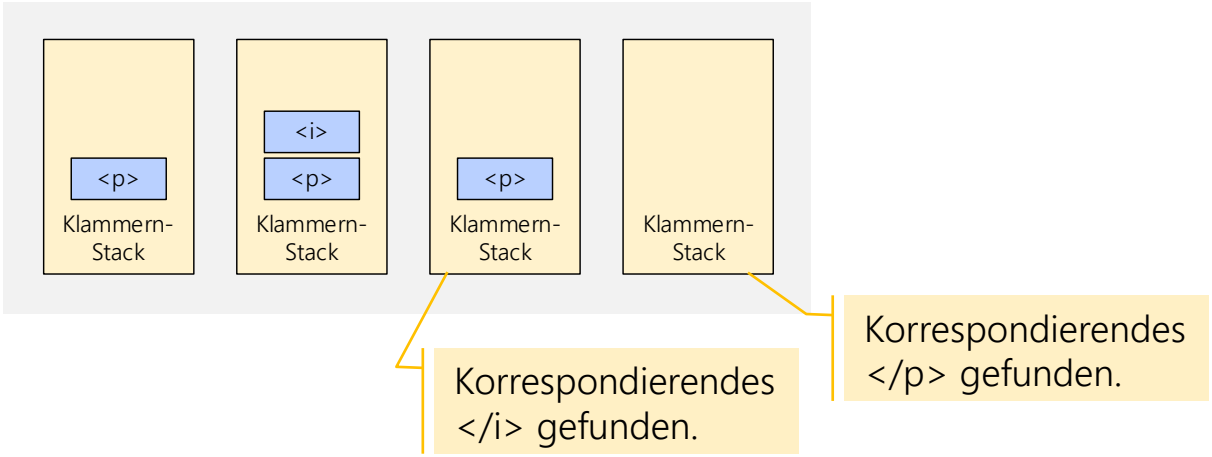
Auf den nachfolgenden Folien dieser Vorlesungsreihe wird in der Regel zugunsten einer besseren Übersichtlichkeit auf Clean-Code und auf Javadoc verzichtet! D.h. nicht, dass Sie in den Praktika darauf verzichten können.

Stack: Anwendungsbeispiel HTML-Syntax-Check

Mit einem Stack lässt sich überprüfen, ob die öffnenden (z.B. `<p>`) und schliessenden (z.B. `</p>`) Tags in einem HTML-Ausdruck korrekt gesetzt sind:

1. Jeder öffnende HTML-Ausdruck wird auf den Stack gelegt.
2. Bei jedem schliessenden HTML-Ausdruck wird überprüft, ob der korrespondierende öffnende Ausdruck zuoberst auf dem Stack liegt, falls nein, dann ist der Ausdruck fehlerhaft.

Nach dem Abarbeiten des gesamten HTML-Ausdrucks muss der Stack wieder leer sein (im Beispiel `<p>ein <i>kleiner</i> Test</p>`).

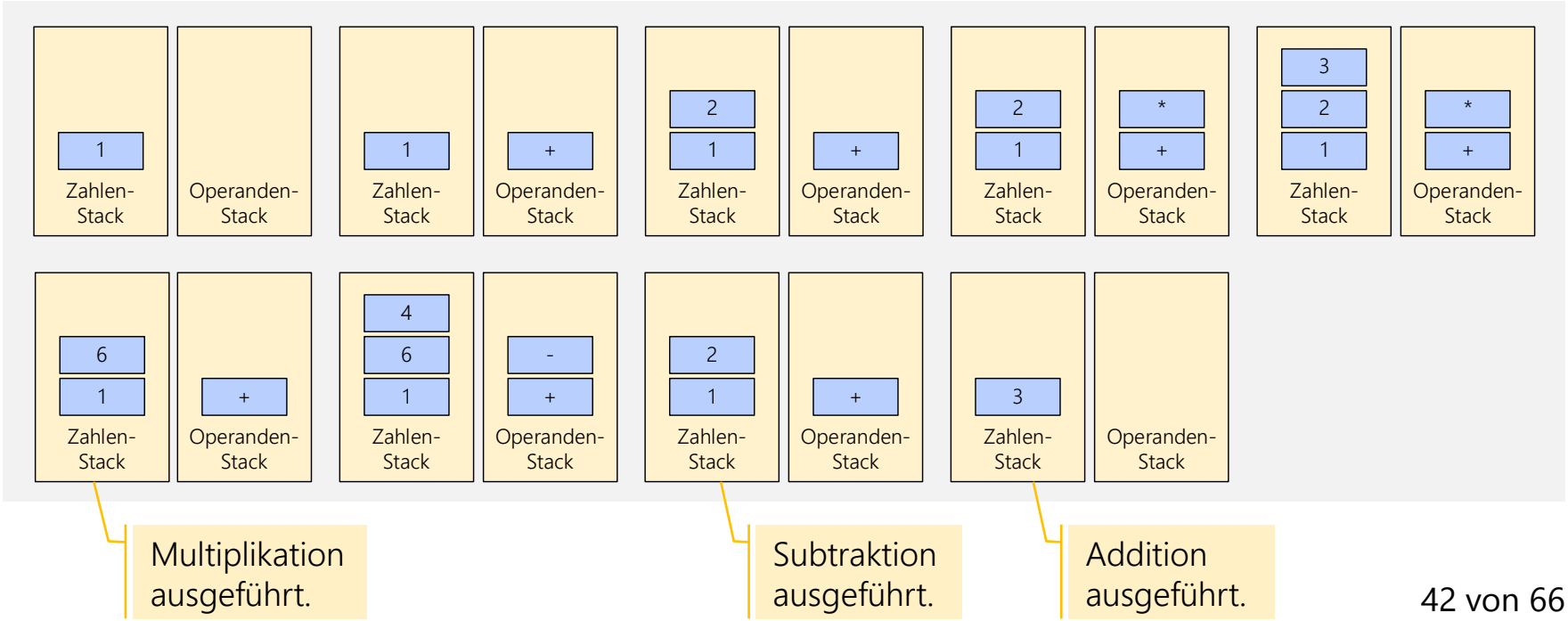


Stack: Anwendungsbeispiel Parser

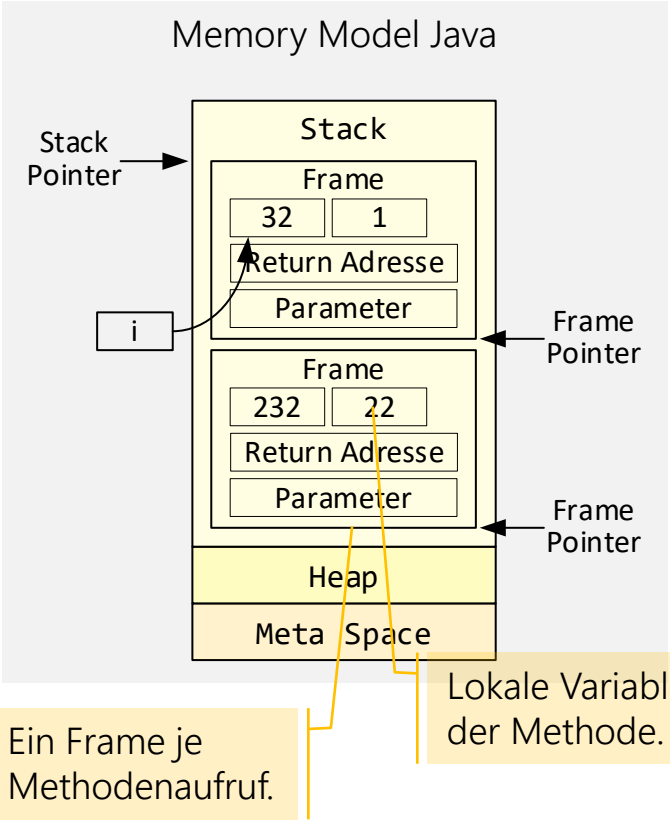
Beim Parsen von Programmiersprachen und Termen kommen Stacks zu Einsatz.

Um den Term 1 + 2 * 3 - 4 auszuwerten, und dabei die Punkt-Vor-Strich-Regel einzuhalten, führen wir zwei Stacks ein. Einen Zahlen-Stack und einen Operanden-Stack (wir vernachlässigen Klammern). Wir benötigen zwei Regeln:

1. Punkt-Operationen dürfen nach dem Lesen der nachfolgenden Zahl ausgeführt werden. Das Resultat wird auf den Zahlen-Stack gelegt.
2. Strich-Operationen werden am Ende ausgeführt.



Stack: Anwendungsbeispiel Memory-Modell-Java



Das Memory-Modell von Java[1)] enthält nebst dem Heap und dem Meta-Space einen Stack. Auf dem Stack werden folgende Informationen abgelegt:

- Beim Sprung in eine Methode werden die Parameter und der Programmzähler (Return Adresse) auf den Stack gelegt.
- Lokalen Variablen der Methode werden auf dem Stack abgelegt.
- Beim Return wird der Programmzähler vom Stack geholt und an der Aufrufstelle weiter gefahren.

Damit können Methoden «reentrant» genutzt werden (später unter Rekursion).

[1)] Wir werden das Memory-Modell von Java in Lektion 13 genauer betrachten.

Stack: Implementation mittels Array

- Die Array-Implementation nutzt einen Array von Objekten als eigentlichen Stapel.
- Der top-Zeiger (zeigt auf den nächsten freien Platz) wird als int realisiert. Ein Wert von 0 bedeutet, dass der Stack leer ist.
- Der Konstruktor erwartet die maximale Anzahl Elemente, die der Stack aufnehmen kann.

```
public class StackArray implements Stack {

   Object[] data;
   private int top;
   private int capacity;

   public StackArray(int capacity){
      this.capacity = capacity;
      removeAll();
   }
```

Stack: Implementation mittels Array

```
public void push(Object x)
   throws StackOverflowException {
   if (isFull()) {
      throw new StackOverflowException();
   }
   data[top] = x;
   top++;
}

public Object pop() {
   if (isEmpty()) {return null;}
   top--;
   Object topItem = data [top];
   data [top] = null;
   return topItem;
}
```

Stack: Implementation mittels Array

```
    public Object peek() {
        if (isEmpty()){return null;}
        return data [top-1];
    }

    public void removeAll() {
        data = new Object[capacity];
        top = 0;
    }

    public boolean isEmpty() {
        return (top == 0);
    }

    public boolean isFull() {
        return top == data.length;
    }
}
```

Was sind die Vor- und Nachteile dieser «Array»-Implementation?

Stack: Implementation im JDK

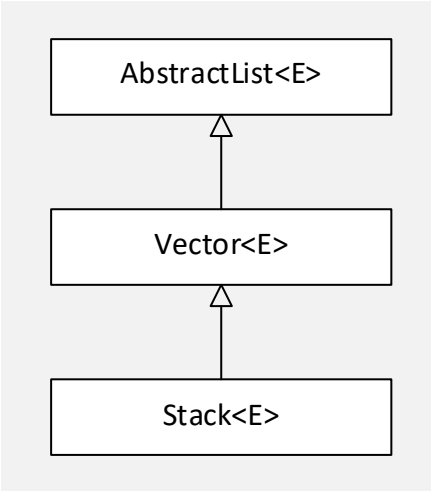
Es existiert eine Stack-Implementation im java.util-Package. Von der Verwendung dieser Klasse wird allerdings von Oracle abgeraten:

«A more complete and consistent set of LIFO stack operations is provided by the Deque[1N)] interface and its implementations, which should be used in preference to this class.»

Siehe Fussnote 1 in den Notizen.

- Verletzung der Namenskonventionen:
  - empty() statt isEmpty() in Klasse Stack.
- Information-Hiding-Prinzip verletzt.
  - Implementation ist immer ein Vector.
- Encapsulation-Prinzip verletzt:
  - Man kann mit Vector-Methoden die Konsistenz des Stacks zerstören.
  - Muss die alte Vector-Klasse verwenden.

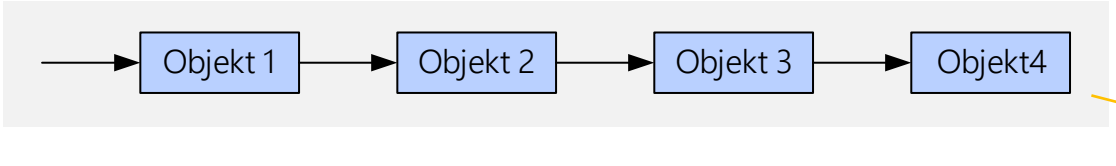
➢ Nur wenn man «ist ein» sagen kann, darf man Vererbung verwenden.



Listen

Listen werden in Lektion 2 detaillierter betrachtet.

Liste: als ADT



Die Liste ist ein abstrakter Datentyp, der eine Liste von Objekten verwaltet:

- Die Liste ist eine grundlegende Datenstruktur der Informatik.
- Mit der Liste kann ein Stack variabler Grösse implementiert werden.
- Speichert Objekt oder Wert durch Typenplatzhalter bestimmt (z.B. generisch).

`boolean add(Object o)`	Fügt o am Ende der Liste an.
`void add(int pos, Object o)`	Fügt o an der Position pos in die Liste ein.
`Object get(int pos)`	Gibt das Element an Position pos zurück.
`Object remove(int pos)`	Entfernt das Element an Position pos und gibt es zurück.
`int size()`	Gibt die Anzahl Elemente zurück.
`boolean isEmpty()`	Gibt true zurück, fall die Liste leer ist.

- In Java gibt es im Package java.util das Interface java.util.List mit diversen Implementationen: LinkedList, ArrayList, etc.

Liste: Stack-Implementation mittels Liste

```
import java.util.*;

public class StackLinkedList implements Stack {
   private List list;

   public StackLinkedList() {
      removeAll()
   }

   public void push(Object x) {
      list.add(x,0);
   }

   public Object pop() {
      if (isEmpty()) { return null; };
      return list.remove(0);
   }
```

Die Elemente werden an der Position 0 der Liste hinzu-gefügt und entnommen.

Liste: Stack-Implementation mittels Liste

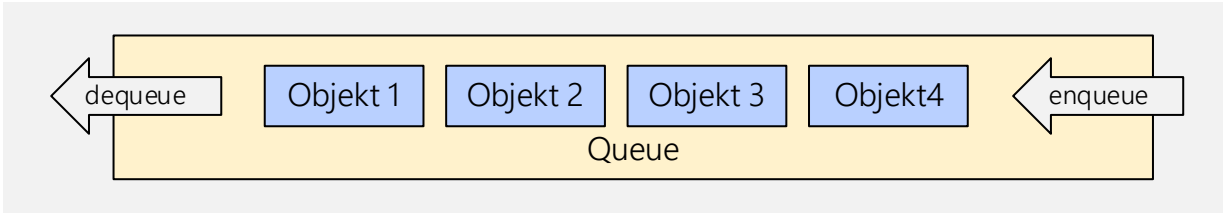
```
   public Object peek() {
      if (isEmpty()) { return null; };
      return list.get(0);
   }

   public void removeAll() {
        list = new LinkedList()
  }

   public boolean isEmpty() {
      return list.isEmpty();
   }

   public boolean isFull() {
      return false;
   }
}
```

Wir werden Listen später noch genauer betrachten.

Ist immer `false`.

Queues

Queue: als ADT



Eine Queue speichert Objekte in einer (Warte-)Schlange. Sie werden in der selben Reihenfolge entfernt, wie sie eingefügt werden (FIFO: First In First Out).

`void enqueue(Object x)`	Fügt x in die Queue ein.
`Object dequeue()`	Entfernt das älteste Element und gibt es als Rückgabewert zurück.
`Objekt peek()`	Gibt das älteste Element zurück, ohne es zu entfernen.
`void removeAll()`	Leert die gesamte Queue.
`Boolean isEmpty()`	Gibt true zurück, fall die Queue leer ist.
`Boolean isFull()`	Gibt true zurück, fall die Queue voll ist.

Queue: als Java-Interface (ADT)

```
/**
 * Interface für abstrakten Datentyp (ADT) Queue.
 */
public interface Queue {


   /**
    * Legt ein neues Objekt in die Queue, falls noch nicht voll.
    * @param x ist das Objekt, das dazugelegt wird.
    */
   public void enqueue(Object x) throws Overflow;


   /**
    * Entfernt das «älteste» und damit das zuerst eingefügte Objekt von der Queue.
    * Ist die Queue leer, wird null zurückgegeben.
    * @return Gibt das oberste Objekt zurück oder null, falls leer.
    */
   public Object dequeue();
```

Java hat bereits eine Queue-Klasse. Im Beispiel soll lediglich gezeigt werden, wie ein ADT mittels Java deklariert werden kann.

Queue: als Java-Interface (ADT)

```
/**
 * Testet, ob die Queue leer ist.
 * @return Gibt true zurück, falls die Queue leer ist.
 */
public boolean isEmpty();

/**
 * Gibt das älteste Objekt zurück, ohne es zu entfernen.
 * Ist die Queue leer, wird null zurückgegeben.
 * @return Gibt das älteste Objekt zurück oder null, falls leer.
 */
public Object peek ();

/**
 * Entfernt alle Objekte von der Queue. Ein Aufruf von isEmpty()
 * ergibt nachher mit Sicherheit true.
 */
public void removeAll ();
```

Queue: als Java-Interface (ADT)

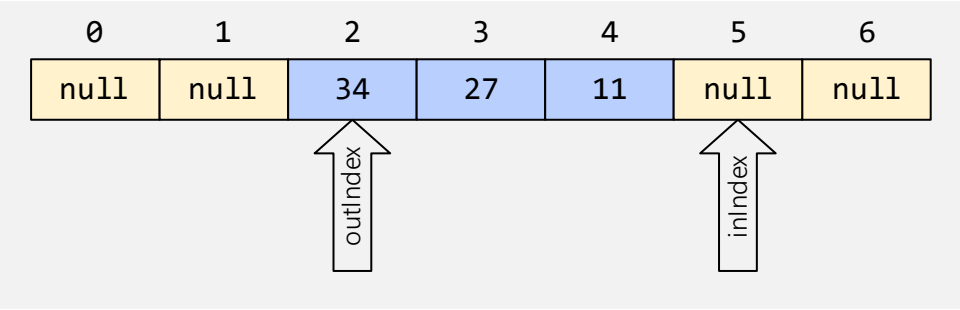
```
    /**
     * Testet, ob die Queue voll ist.
     * @return Gibt true zurück, falls die Queue voll ist.
     */
    public boolean isFull();

}
```

Queue: Anwendungsbeispiel Warteschlangen

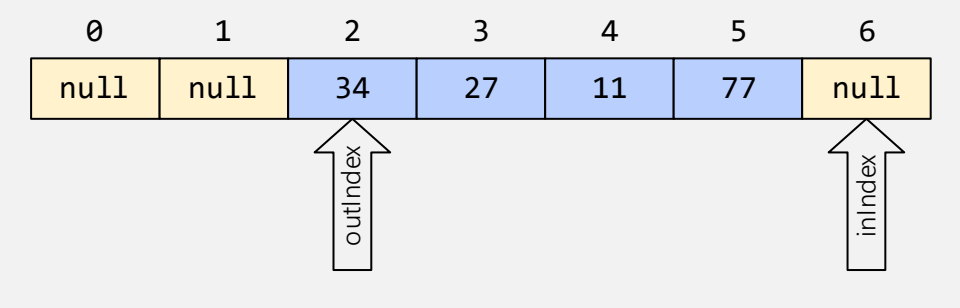
- Warteschlangen werden in der Informatik oft eingesetzt. Z.B. ein Drucker, der von mehreren Anwendern geteilt wird.
- Allgemein dort, wo der Zugriff auf irgendeine Ressource nicht parallel, sondern gestaffelt erfolgen muss.
- Warteschlangen-Theorie:
  - Ein ganzer Zweig der Mathematik beschäftigt sich mit der Theorie von Warteschlangen.
  - Fragestellung 1: Anzahl benötigter Kassen in einem Geschäft:
    - Gewisses Kundenaufkommens (z.B. 10 pro Minute); bestimmte Verteilung.
    - Gewisse Verarbeitungszeit (z.B. 1 Minute).
    - Wie viele Kassen müssen geöffnet werden, damit die mittlere Wartezeit 10 Minuten nicht übersteigt?
  - Fragestellung 2: Auslegung des Mobiltelefonnetzes:
    - Gewisse Anzahl Mobiltelefonbenutzer.
    - Benutzungswahrscheinlichkeit und Dauer.
    - Wie viele Antennen notwendig, damit in 99% der Fälle eine Verbindung zustande kommt.?

Queue: Implementation mittels Array

- Die Array-Implementation nutzt ein Array von Objekten als eigentlichen Queue.
- Zwei int Variablen, outIndex und inIndex bestimmen den momentan gültigen Inhalt der Queue.
- Dabei zeigt outIndex auf das «älteste» Element, während inIndex auf die nächste freie Stelle zeigt. Zusätzlich werden noch die Anzahl Elemente der Queue in noOfItems gespeichert.



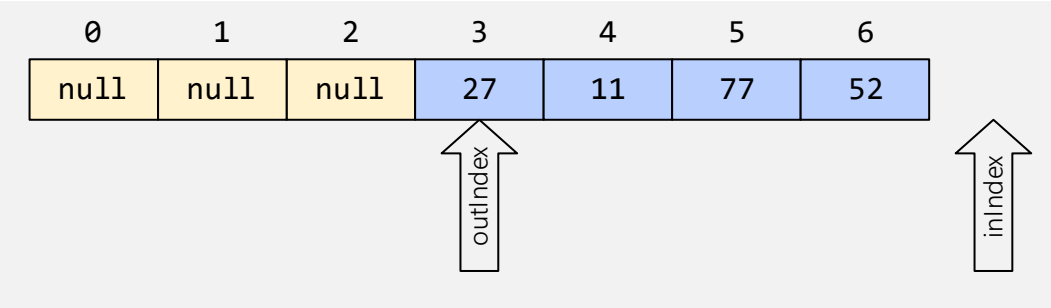
Beim Einfügen wird das Element 77 an der Stelle inIndex eingefügt und noOfItems um 1 erhöht.

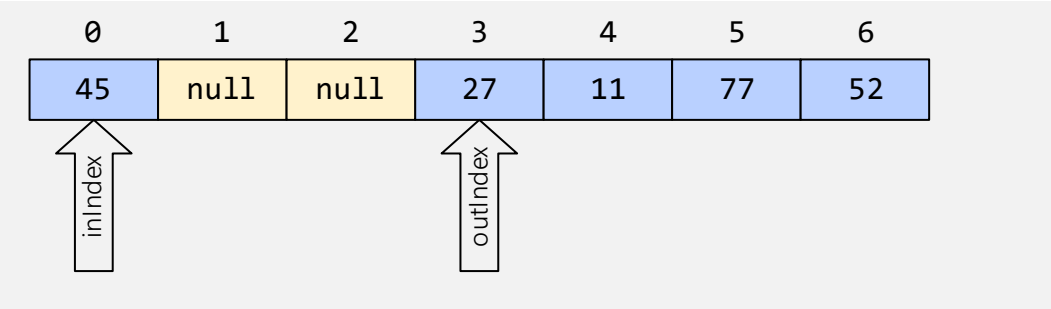


Beim Entfernen wird das Element 34 an der Stelle outIndex entfernt und noOfItems um 1 reduziert.

Queue: Implementation mittels Array

Was passiert, wenn noch zwei Elemente (52 und 45) eingefügt werden?





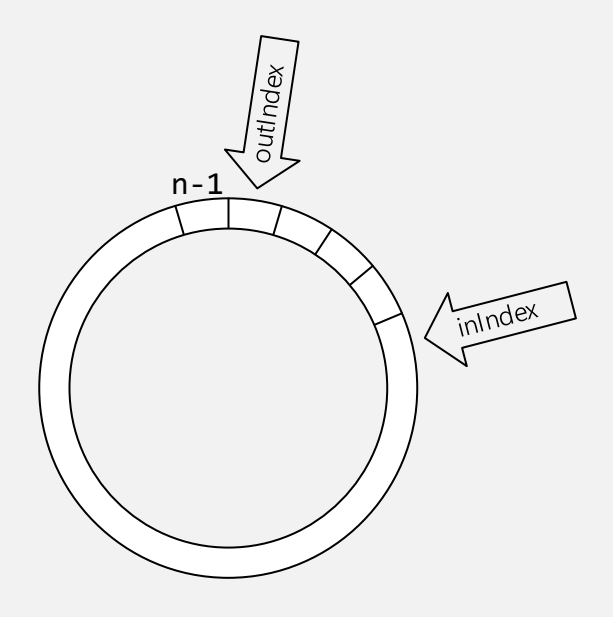
Nach dem Einfügen zeigt der inIndex nicht mehr auf den Array. Lösungen:

1. Wir geben eine «Overflow»-Fehler-meldung aus.
2. Wir haben noch Platz. Der Zeiger inIndex springt um das Ende herum zurück zum Anfang.

Wie kann die Operation «**`erhöhe inIndex um 1 und setze auf 0 falls die Array-Grenze überschritten wird`**» programmiert werden?

Queue: Implementation mittels Array

Die Queue implementiert mittels Array ist ein effizienter Ringpuffer (Engl. Ring Buffer). Dieser wird zum Beispiel bei Tastatureingaben und in Datenbanksystemen (Transaktionslog) eingesetzt.




```
public void enqueue(Object x) {
   if (noOfItems == size) {
      throw new Exception("buffer overflow");
   }
   content[inIndex] = x;
   inIndex = (inIndex + 1) % size;
   noOfItems++;
}

public Object dequeue() {
   if (noOfItems == 0) {
      throw new Exception ("buffer underflow");
   }
   Object x = content[outIndex];
   outIndex = (outIndex + 1) % size;
   noOfItems--;
   return x;
}
```

Implementation des Abstrakten Datentyp (ADT) Queue mit Hilfe des Arrays.

Queue: Implementation mittels Liste

```
import java.util.*;

public class QueueLinkedList implements Queue {
   private List list;

   public QueueLinkedList() {
      removeAll()
   }

   public void enqueue(Object x) {
      list.add(x);
   }

   public Object dequeue() {
      if (isEmpty()) { return null; };
      return list.remove(0);
   }
```

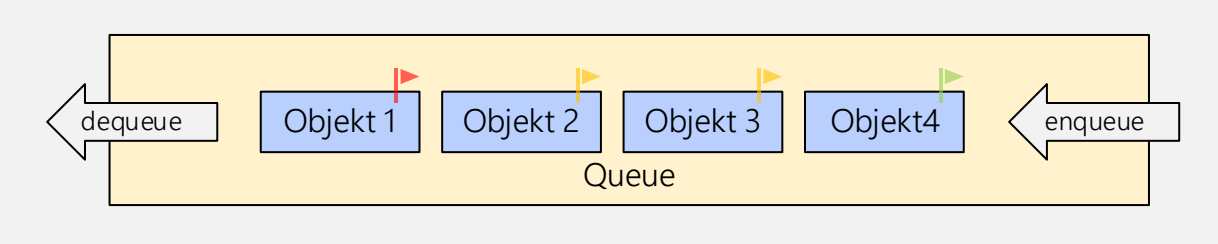
Implementation des Abstrakten Datentyp (ADT) Queue mit Hilfe der vordefinierten Klasse `java.util.LinkedList`.

Queue: Implementation mittels Liste

```
    public boolean isEmpty() {
        return list.isEmpty();
    }

    public Object peek() {
        if (isEmpty()) { return null; };
        return list.get(0);
    }

    public void removeAll() {
        list = new LinkedList();
    }

    public boolean isFull() {
        return false;
    }

}
```

Priority Queues

Priority Queue: als ADT



Objekte hoher Priorität wandern nach vorne. Objekte der gleichen Priorität werden in der selben Reihenfolge entnommen wie sie eingefügt wurden.

Im Vergleich zur «normalen» Queue wird einzig die enqueue()-Methode ergänzt. Dieser muss zusätzlich die Priorität mitgegeben werden.

```
void enqueue(Object x, int priority)
```

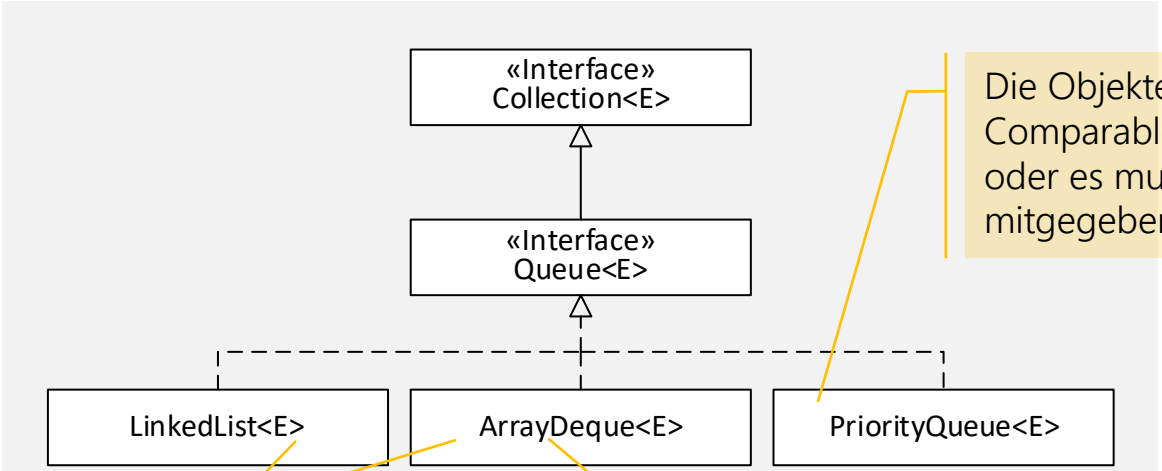
Anwendungen:

- Scheduling von Prozessen in Betriebssystemen
- Taskliste nach Prioritäten geordnet
- Rechnungen bezahlen nach Rechnungssteller: Mafia zuerst
- usw.

Priority Queue: Übersicht zum Package java.util

Queue-Interface im Package java.util:

- Enqueue:
  Einfügen mittels `add()-` oder `offer()-Methode`. Falls die Queue voll ist, wirft **`add`** einen Fehler, **`offer`** gibt **`false`** zurück.
- Dequeue:
  Entnehmen des ersten Elements mittels `poll()-` oder `remove()-Methode`.
  `poll` liefert **`null`** zurück, falls die Queue leer ist, **`remove`** wirft einen Fehler.



Letzte... 05

Zusammenfassung

- Einführung ADS-Vorlesung
- Konzept des ADTs in Java
  - Beschreibung der Schnittstelle durch Interfaces
  - Implementierung durch Klasse
- Der Stack als LIFO-Datenstruktur
  - Anwendungen von Stack: HTML-Syntax-Check, Parser, Java-Memory-Modell
  - Als Array-Implementation
  - Als Listen-Implementation
- Listen zur Implementation von andern Datenstrukturen
- Die Queue als FIFO-Datenstruktur
  - Anwendungen von Queue: Printer-Queue, Warteschlangentheorie
  - Als Array-Implementation: der Ringpuffer
  - Als Listen-Implementation
- Priority-Queue
  - Queue-Interface in Java