Resümee der Vorlesung

Resümee – Ziele der Vorlesung

- *Primär:* Informatik-Grundbegriffe
  - Konzepte, Modelle, Problemlösungstechniken
  - Algorithmen und Datenstrukturen
- *Sekundär:* Programmieren
  - Java
- *Auch:* Techniken für qualitativ hochwertige Software
  - Strukturen (Objektorientierung...)
  - Qualitätsmerkmale
  - Korrektheit

Methode

- Verzahnte und verwobene Einführung programmiersprachlicher Konstrukte und konzeptioneller Aspekte
- *Die gleichzeitige Einführung einer Programmiersprache und von Informatik-Grundkonzepten gelingt nicht immer ganz glatt - z.B. muss gelegentlich die Darstellung eines konzeptionellen Aspektes unterbrochen werden, um ein programmiersprachliches Konstrukt einzuführen - doch überwiegen wohl die Vorteile gegenüber einer völlig getrennten Behandlung deutlich.*
- Programmbeispiele sollen gleichzeitig Informatikkonzepte illustrieren / implementieren und programmiersprachliche Konstrukte einführen

3

Konzepte — Java

Korrektheitsnachweis (Invarianten und vollst. Indukt.)
Robustes Programmieren

Java: C-Level
Java-Klassen als Datenstrukturen
Dynamische Klassen und Referenzen

Bäume
Syntaxdiagramme
Rekursiver Abstieg
Infix, Postfix, Operatorbaum, Stack
Codegenerierung, Compiler, Interpreter

Java-VM als Bytecode-Interpreter
Pakete
Klassenhierarchie

Verzahnte und verwobene Einführung konzeptioneller Aspekte und programmiersprachlicher Konstrukte

Polymorphie

Abstrakte Klassen
Exceptions

Suchbäume, Sortieren
Backtracking
Spieltheorie, Minimax, AlphaBeta
Rekursives Problemlösen
Effizienz, O-Notation
Simulation (zeitgesteuert, ereignisgesteuert)
Heap, Heapsort
Pseudoparallelität

Programmbeispiele dienen gleichzeitig der Einführung programmiersprachlicher Konstrukte und der Illustration von Informatikkonzepten

Threads in Java

4

Resümee (1)

- Algorithmus „altägyptische Multiplikation"
  - Verdoppeln und Halbieren ohne Rest
- Rekursion: Reduktion auf eine einfachere Instanz des gleichen Problems
  - Algorithmus als rekursives Java-Programm

$$f(a,b) = \begin{cases} a & ,falls\ b = 1 \\ f(2a, b/2) & ,falls\ b\ gerade \\ a + f\left(2a, \frac{b-1}{2}\right) & ,sonst \end{cases}$$

- Korrektheitsnachweis $\forall\, a,b \in \mathbb{N}^+$: **f(a,b) = a × b** mit vollst. Induktion (über b)
- Fehlerhaftes Verhalten
  - Z.B. Stack-Überlauf bei Eingabe b = 0
  - Robustes Programmieren durch Ausnahmebehandlung (Exceptions) mittels try und catch
- Multiplikationsalgorithmus iterativ (while-Schleife)
  - Verifikation mittels Invarianten

a	b
6	9
~~12~~	~~4~~
~~24~~	~~2~~
48	1
54	

5

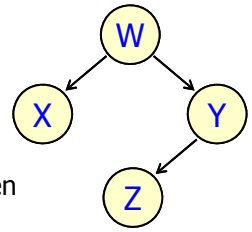
Resümee (2)

- Effizienz des (rekursiven) Multiplikationsalgorithmus
  - Zählen von Elementaroperationen: insgesamt weniger als $5 \log_2 b$
  - Vergleich der „altägyptischen Methode" mit der Schulmethode
- Elementare Berechnungsoperationen
  - *gerade*, *halbiere*, *verdopple*,... rekursiv implementierbar
  - sogar *decrement* mittels *increment*
- Funktionales Programmieren
  - Keine Variablen und Zuweisungen
- Java
  - Bytecode, Applets
  - Programmstruktur
  - Einfache Datentypen
  - Arrays
  - Typkonversion
  - Hüllenklassen
  - Ein- und Ausgabe
  - Strings

6

Resümee (3)

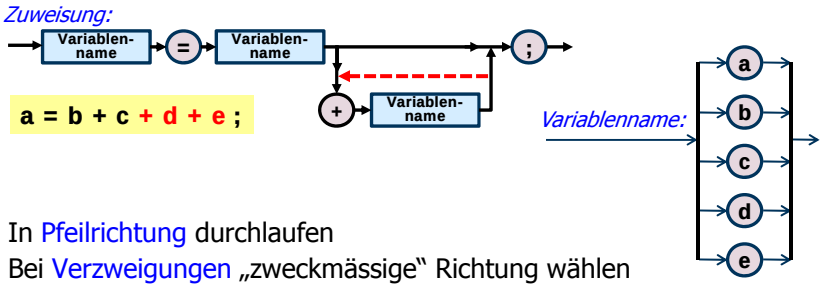
- Java: Klassen als Datenstrukturen
  - Beispiel-Klasse „Datum" (mit Methoden *„frueher_als", „gleich"...)*
  - this
  - Instanzen- / klassenbezogene („static") Variablen und Methoden
  - Zugriffseinschränkungen, information hiding („public", „private")
- Java: Dynamische Klassen („Instanzen") und Referenzen
  - Erzeugen von Objekten: „new"; überladene Konstruktoren
  - Zuweisung und Vergleich („==") von Referenzen
  - Alias-Effekt
- Bäume
  - Definition, Charakterisierung
  - Wurzelbaum
  - Verschiedene Darstellungen von Wurzelbäumen
  - Binärbaum in Array-Repräsentation



7

Resümee (3b)

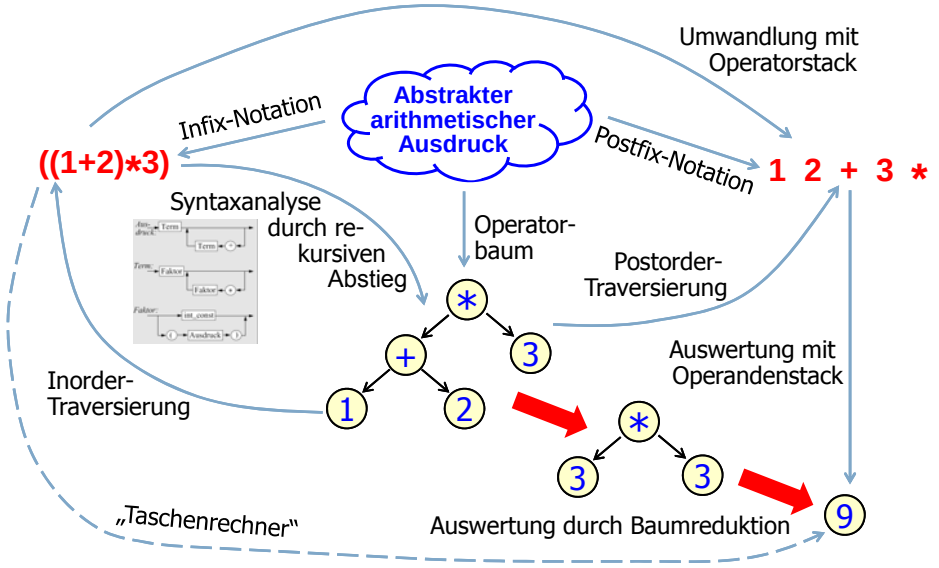
- Syntaxdiagramme
  - Generierung ausschliesslich syntaktisch korrekter Programme



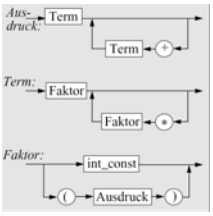
  - In Pfeilrichtung durchlaufen
  - Bei Verzweigungen „zweckmässige" Richtung wählen
  - Durchlaufene Terminalsymbole aufschreiben
  - Wenn ein Nicht-Terminal getroffen wird, in das zugehörige Teildiagramm „abtauchen" und nach dem „Auftauchen" weitermachen

8

Arithmetische Ausdrücke – Überblicksgemälde



Resümee (4)



- Syntaxanalyse durch rekursiven Abstieg
  - Parser als Implementierung von Syntaxdiagrammen
  - Erzeugen von Syntaxbäumen
- Auswertung von Syntaxbäumen (Operatorbäumen)
- Traversieren von Bäumen in inorder und postorder
  - Liefert bei Operatorbäumen den Ausdruck in Infix- bzw. Postfixnotation
- Stacks: Implementierung und Anwendungen
  - Transformation infix → postfix
  - Auswertung von Postfix-Ausdrücken

Resümee (5)

- Klammerchecker: Blockstruktur-Analyse von Programmen mittels eines Stack

```
1 { …
2  …
3   { …
4     { …
5     }
6     { …
7     }
8   }
9 }
```

- Arithmetische Infix-Ausdrücke
  - Codegenerierung für eine Stackmaschine
  - Interpreter (→ „Taschenrechner“)

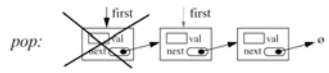
```
(1+2)*3
push(1)
push(2)
plus
push(3)
mult
```

```
while (c == '+') { …
  stk.push(stk.pop()
  + stk.pop());   }
```

- Übersetzung von Java nach Bytecode
  - Java-VM als Bytecode-Interpreter

```
0 iconst_5
1 istore_1
2 bipush 7
4 istore_2
5 iload_2
6 iload_1
7 iadd
```

- Pakete in Java
  - Beispiel verkettete Liste / Stack

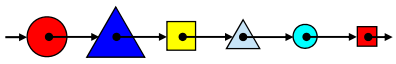


11

Resümee (6)

```
import bruchPak; …
a = b.plus
   (new Bruch(1,8));
```

- Pakete in Java: Beispiel Bruchrechnung
- Klassenhierarchie / objektorientiertes Programmieren
  - Klassifizierung, abgeleiteter Klassen, Vererbung
  - Zuweisungskompatibilität von Variablen verschiedener Hierarchiestufe
  - Zugriff auf Attribute und Methoden abgeleiteter Klassen
- Abstrakte Klassen und abstrakte Methoden
- Polymorphie
  - Beispiel: polymorphe Listen

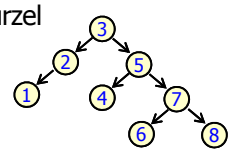


- Generische (d.h. typunabhängige) Algorithmen
  - Sortierverfahren für beliebige total geordnete Objekte, z.B.:
  - 1) Sortieren von Zahlen
  - 2) Sortieren von „Studi“-Objekten
  - 3) Sortieren geometrischer Objekte nach Flächenwert

12

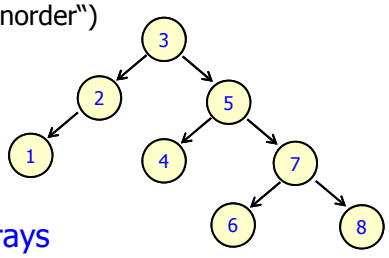
Resümee (7)

- Java: Mehrfachvererbung und Interfaces
- Exceptions
  - try, catch, throw
  - Definieren eigener Ausnahmen
- Bäume als Zeigergeflechte („Referenzstrukturen")
- Binäre Suchbäume
  - Definition rekursiv: Werte im linken Unterbaum kleiner, im rechten grösser als Wurzel
  - Einfügen
  - Suchen
  - Löschen



Resümee (8a)

- Binäre Suchbäume
  (rekursiv: Werte im linken Unterbaum kleiner, im rechten grösser als Wurzel)
  - Symmetrisches Traversieren („inorder")
  - Sortieren mit Suchbäumen
    (n log n Schritte im Normalfall)



- Binärsuche auf sortierten Arrays
  - Iterativer bzw. rekursiver Algorithmus
  - Fortgesetzte Intervallhalbierung
  - Suchaufwand: log n
  - Korrektheit: Terminierung sichergestellt?

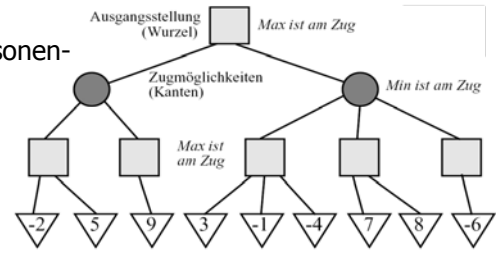
Resümee (8b)

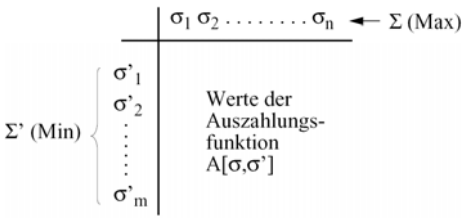
- Backtracking
  - Systematisches Durchmustern eines Problembereichs
  - Beispiel: n-Damen-Problem
- Computerschach und andere Computerspiele
  - kurzer historischer Rückblick
- Spieltheorie

15

Resümee (9a)

- Spieltheorie
  - Endliche strategische 2-Personen-Nullsummenspiele mit vollständiger Information
  - Spielbäume
  - Strategien
  - Auszahlungsmatrix





16

Resümee (9b)

- Optimale Strategie, Gewinnstrategie, Gewinnstellung
- Minimax-Prinzip
  - Garantierter Mindestgewinn
  - Analogie zur Auswertung von Operatorbäumen von Ausdrücken
- Auswertung von (partiellen) Spielbäumen
  - Tiefensuche, Breitensuche, Bestensuche
- α-β-Algorithmus
  - Schnitte (Verallgemeinerung der Shortcut-Operatoren '&&', '||')
  - α-β-Schranken
  - Effizienz
- Optimierte Spielbaumanalyse
  - Spekulative Suchfenster, last move improvement, Nullfenster

17

Resümee (10)

- Rekursives Problemlösen
  - „Türme von Hanoi"-Spiel
  - „Divide et impera"-Paradigma
- Mergesort (als Anwendung von „divide et impera")
  - Bottom-up / top-down
  - Mit array bzw. verketteter Liste
  - Zeitaufwand proportional zu *n log n*
- Aufwand von Algorithmen
  - O-Notation, asymptotische Zeitkomplexität
  - O(…)-Beispiele aus Algorithmen der Vorlesung
  - Komplexitätsklassen O(f(n))
  - Beispiel: Beweis für Zeitkomplexität *O(n log n)* bei mergesort

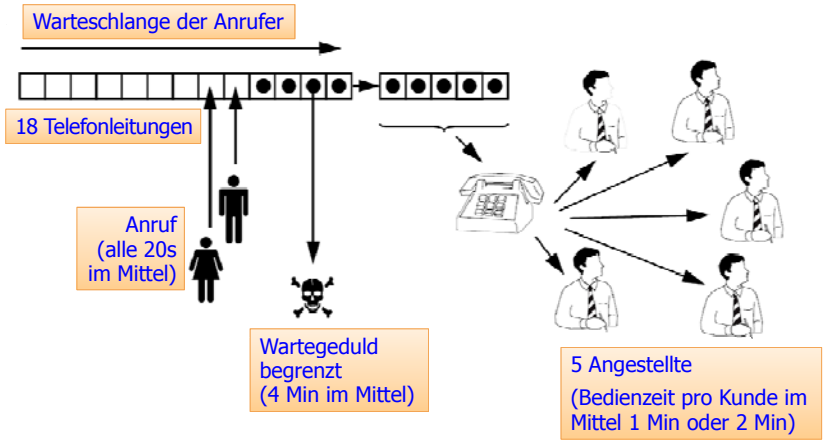
18

Resümee (11)

- Simulation
  - Definition, Zweck, Anwendungsgebiete
  - Modell und Modellierung
- Zeitgesteuerte Simulation – Bsp. „Weizen, Mäuse, Katzen"
  - „Korrekte" Umsetzung der Spezifikation in ein Simulationsmodell
  - Periodische Neuberechnung der 3 relevanten Zustandsgrössen
  - Grösse von Δt
- Ereignisgesteuerte Simulation
  - Vorantreiben der Simulation (und Simulationszeit) durch Ereignisse
  - Beispiel: Call-Center eines Reisebüros

19

Resümee (11) Unser Reisebüro



20

Resümee (12)

- Ereignisgesteuerte Simulation
  - Modellierung quasi-paralleler Abläufe durch verzahnte Ereignisfolgen
  - Ereigniseinplanung; Ereignisroutinen; Simulatorzyklus
- Ereignislisten (*insert*, *get_min*) als abstrakte Datentype
  - Als sortierte Liste; als unsortierte Liste; als Heap („partiell sortiert")
- Heap-Datenstruktur
  - Definition (Binärbaum...)
  - Niveauweise Speicherung als Array
  - Java-Implementierung von *insert* und *get_min*
- Heapsort
  - O(n log n) Zeitaufwand auch im worst case, „in place"

21

Resümee (13)

- Prozesse, Multitasking
  - Prozesszustände
  - Kontextwechsel
- Threads in Java
  - Erzeugen
  - Beispiel für parallele Threads („Hin-Her")
  - Methoden zur Thread-Steuerung (start, stop, suspend, resume, join,...)
- Scheduling von Threads
  - Prioritäten
  - „Simulation" eines Zeitscheiben-Schedulers

22

Resümee (14)

- Multithreading: race conditions
- Atomarität
  - Inkonsistenzen bei Nicht-Atomarität
  - Unterbrechungssperren mittels Thread-Prioritäten?
- Kritische Abschnitte
  - Safety, Liveness, Fairness
  - Realisierung mit dem Synchronized-Konstrukt von Java
- Deadlocks

Ende