Große Übung
Praktische Informatik I

Robert Schiele

10. November 2005

Inhaltsverzeichnis

1 Mehrdimensionale Felder

- Manche Javabücher: „Es gibt keine mehrdimendionalen Felder in Java.“ $\Rightarrow$ Halbwahrheit
- Java kann mehrdimensionale Felder wie jede andere praktisch nutzbare (allgemeine) Programmiersprache.
- Richtig: Es gibt keine *spezielle Syntax* für mehrdimensionale Felder.

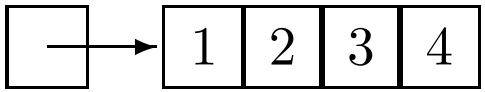
1.1 Wiederholung: Eindimensionale Felder

- Deklaration: `int var[];`
- Initialisierer: `int var[] = { 1, 2, 3, 4 };`
- dynamische Speicherallokation: `var = new int[42];`
- Zugriff: `var[17]`

1.2 Variablen und Felder im Speicher

- Variable besitzt Position im Arbeitsspeicher

- Feld ist referenzierter Speicherblock

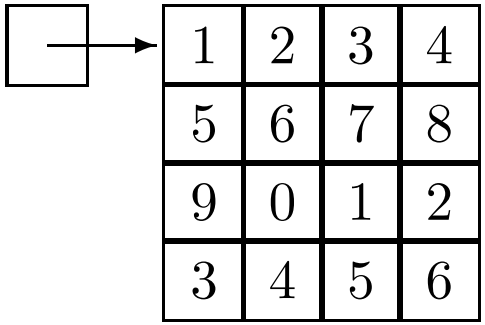


- mehrdimensionales Feld?

1.3 Mehrdimensionale Felder im Speicher

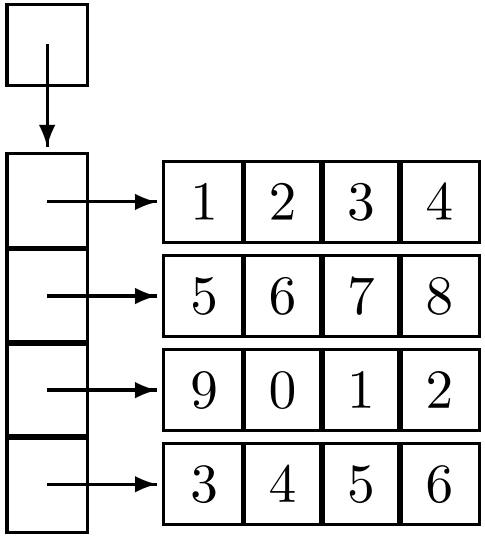
Zwei denkbare Möglichkeiten (am Beispiel einer 4×4-Matrix):

- Speichern als Block



  $\Rightarrow$ effizienter Ansatz

- Speichern als „Feld von Feldern“



$\Rightarrow$ flexibler Ansatz — Warum?

1.4 Verwendung von mehrdimensionalen Feldern

Da Java die Variante „Feld von Feldern (von Feldern...)“ verwendet, ergibt sich:

- Deklaration: `int var[][];`
- Initialisierer:

```
int var[][] = {{ 1, 2, 3, 4 },
               { 5, 6, 7, 8 },
               { 9, 0, 1, 2 },
               { 3, 4, 5, 6 }};
```

- dynamische Speicherallokation:

```
var = new int[42][];
for (int i = 0; i < var.length; ++i)
    var[i] = new int[7];
```

  oder einfacher: `var = new int[42][7];` ;-)

- Zugriff: `var[23][5]`

- Beispiele dreidimensionaler, vierdimensionaler, ... Felder?
- Fragen?

2 Objektorientiertes Programmieren

2.1 Andere Programmierparadigmen

- prozedural (C, Pascal, BASIC, ...)
- funktional (Lisp, Scheme, ML, ...)
- logisch (Prolog)
- objektorientiert (SmallTalk, C++, Java, ...)
- ...

2.2 Prinzipien

- Abstraktion

  Objekte eines Programms abstrahieren Personen oder Objekte der Realwelt.

- Kapselung

  Jedes Objekt besitzt eine Aussenschnittstelle (Methoden). Kommunikation erfolgt nur über diese Außenschnittstelle (Methodenaufruf). Interne Strukturen und Daten sind von außen nicht sichtbar.

- Polymorphie

  Jedes Objekt kann auf eine Nachricht (Methodenaufruf) von außen unterschiedlich reagieren.

- Vererbung

  Eine Art von Objekten (Klasse) kann von einer anderen Art abgeleitet sein. Die neue Art übernimmt dann die Eigenschaften der alten und spezialisiert diese.

2.3 Klassen

- Klasse ist eine bestimmte Art von Objekten (z.B. Fahrzeug)
- Klassendefinition legt sowohl die Außenschnittstelle, als auch das Verhalten einer Klasse fest.
- in Java: `class Klassenname { ...}`

2.4 Methoden

- Methode spezifiziert Verhalten eines Objekts der entsprechenden Klasse.
- Es kann mehrere Varianten einer Methode in einer Klasse geben.
- in Java:

```
class Klassenname {
    int methodenname(int a) { ... }
    int methodenname(String s) { ... }
}
```

- Aufruf in Java: `ergebnis = objektname.methodenname(42);`

2.5 Objektvariablen

- Variablen, die den internen Zustand eines Objektes repraesentieren
- in Java:

```
class Klassenname {
    int i;
}
```

- spezielle Objektvariable in Java: `this` — enthält das Objekt selbst

2.6 Kapselung

- Für jede Methode (und Klasse) kann spezifiziert werden, inwiefern sie von außerhalb der Klasse (oder Package) nutzbar ist.
- in Java gibt es folgende (Erklärung vereinfacht):
  - `public`: Jeder darf zugreifen.
  - `protected`: Die Klasse selbst, ihre Unterklassen und alle Klassen im selben Package dürfen zugreifen.
  - keine Angabe: Die Klasse selbst und alle Klassen im selben Package dürfen zugreifen.
  - `private`: Nur die Klasse selbst darf zugreifen.
- Programmiert man streng nach dem objektorientierten Paradigma, so sind *alle* Objektvariablen `private`.

2.7 Vererbung

- Spezialisierung einer Klasse
- in Java: `class Square extends Rectangle { ... }`
- Neue Methoden können hinzugefügt werden.
- Geerbte Methoden können durch spezielle Varianten ersetzt werden.

3 Dokumentation und Code-Wartbarkeit

3.1 Grundregeln

- Dokumentation *aller* Komponenten (Klassen, Methoden, Packages. . . .)
- lesbare und einheitliche Formatierung des Codes (Einrückung, konsistente Umbrüche, maximal 80 Zeichen pro Zeile)
- maximal eine Anweisung pro Zeile
- kompakte Schreibweise (nicht zusammenquetschen!)
- sinnvolle Funktionsgrößen (Faustregel: nicht mehr als ca. 30 Zeilen pro Funktion)

3.2 Wozu?

```
#include <stdio.h>
main(t,_,a)
char *a;
{
return!0<t?t<3?main(-79,-13,a+main(-87,1-_,main(-86,0,a+1)+a)):
1,t<_?main(t+1,_,a):3,main(-94,-27+t,a)&&t==2?_<13?
main(2,_+1,"%s %d %d\n"):9:16:t<0?t<-72?main(_,t,
"@n'+,#'/*{}w+/w#cdnr/+,{}r/*de}+,/*{*+,/w{%+,/w#q#n+,/#{l+,/n{n+,/+#
;#q#n+,/+k#;*+,/'r :'d*'3,}{w+K w'K:'+}e#';dq#'l \
q#'+d'K#!/+k#;q#'r}eKK#}w'r}eKK{nl]'/#;#q#n')()#}w'){){nl]'/+#n';d}rw' i;#
){nl]!/ n{n#'; r{#w'r nc{nl]'/#{l,+'K {rw' iK{;[{nl]'/w#q#n'wk nw' \
iwk{KK{nl]!/w{%'l##w#' i; :{nl]'/*{q#'ld;r'}{nlwb!/*de}'c \
;;{ nl'-{}rw]'/+,}##'*}#nc,',#nw]'/+kd'+e}+;#'rdq#w! nr'/ ') }+}{rl#'{n' ')#\
```

```
}'+}##(!!/")
   :t<-50?_==*a?putchar(31[a]):main(-65,_,a+1):main((*a=='/')+t,_,a+1)
16     :0<t?main(2,2,"%s"):*a=='/'||main(0,main(-61,*a,
"!ek;dc i@bK'(q)-[w]*%n+r3#l,{}:\nuwloca-O;m .vpbks,fxntdCeghiry"),a+1);
}
```

3.3 Sinnvolle Kommentare

Schlecht:

```
    // Methode factorial mit Parameter a
2   public static long factorial(long a) {
        // Wenn a < 0 gib -1 zurueck.
        if (a < 0)
            return -1;
```

```
    // Setze r auf 1.
    long r = 1;
    // Solange a > 1 multipliziere r mit a und dekrementiere a.
    while (a > 1)
        r *= a--;
    // Gib r zurueck.
    return r;
}
```

Gut:

```
/**
 * Gibt die Fakultaet zu einer gegebenen Zahl a zurueck.
 *
 * Dies geschieht durch interatives Aufmultiplizieren aller Zahlen
 * ausgehend von a bis hinunter zu 2 auf den Initialwert 1. Wird ein
 * ungueltiger Eingabewert fuer a uebergeben, wird als Ergebnis -1
 * zurueckgegeben.
 *
 * @param a Zahl, zu der die Fakultaet berechnet werden soll.
 * @return Fakultaet zur Zahl a.
 */
public static long factorial(long a) {
```

```
        if (a < 0)
            return -1;
3       long r = 1;
        while (a > 1)
            r *= a--;
        return r;
    }
```

Informationen zu Javadoc:

`http://java.sun.com/j2se/1.5.0/docs/guide/javadoc/`

3.4 Sinnvolle Formatierung

- Ziel ist es *nicht* schönen Code zu schreiben, sondern *lesbaren*! (Wobei sich das natürlich nicht unbedingt ausschließt.)
- Am Anfang die Grundregeln beachten + Erfahrung sammeln (Erfahrung sammelt man durch Lesen fremden Codes nicht durch persönliche Engstirnigkeit!)

3.5 API-Dokumentation

- Mit Hilfe von Javadoc kann API-Dokumentation generiert werden.
- Für die beim JDK mitgelieferten Klassen gibt es diese unter `http://java.sun.com/j2se/1.5.0/docs/api/`.