



Informatik I

Prof. Dr. Christian Pape

Kapitel 8

Java Kontrollanweisungen /
Algorithmenbegriff / Felder



Inhalt

- Algorithmenbegriff
 - Euklidischer Algorithmus (while-Schleife)
 - do-while Schleife
- Felder
 - Notenspiegel berechnen
 - Sieb des Eratosthenes
 - Pascalsche Dreieck
- Bedingte Anweisung
 - switch



while Schleife

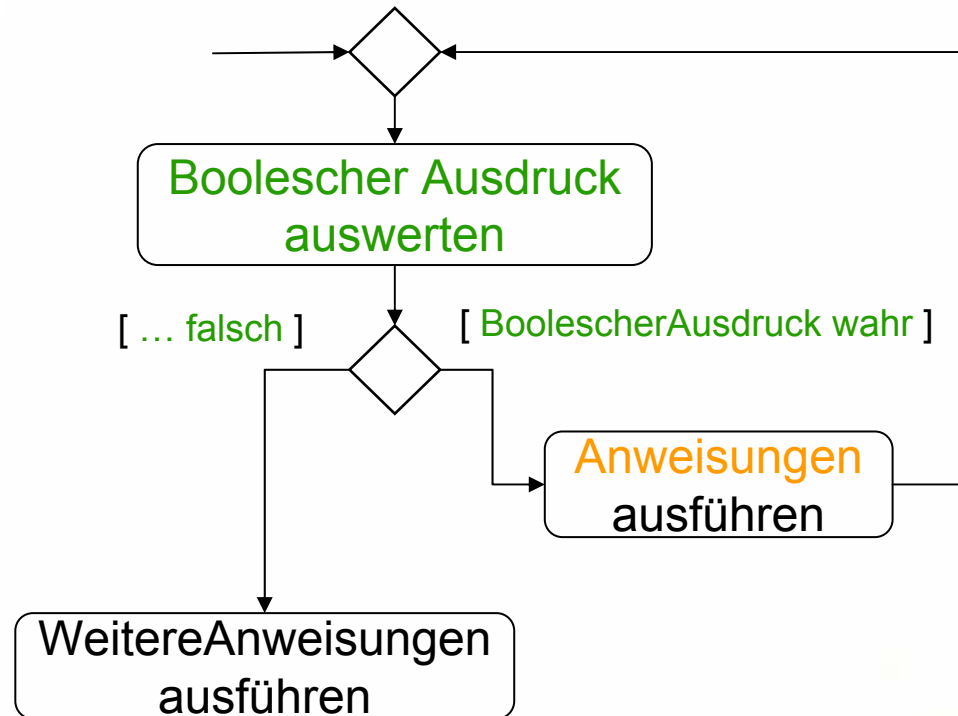
- Bisher Iterationen mit for-Schleife
- Gut geeignet für
 - Kontinuierliche Iteration mit stetigen Auf- oder Abzählen einer Schleifenvariablen
- Schlecht geeignet
 - Keine Schleifenvariablen existieren
 - Abbruchbedingung von mehreren Variablen abhängt, die sich in vielen Anweisungen ändern können

while-Schleife

Java

```
while ( BoolescherAusdruck ) {
  Anweisungen
}
WeitereAnweisungen
```

UML



- Solange BoolescherAusdruck erfüllt (true) ist, führe Anweisungen wiederholt aus.
- Wenn BoolescherAusdruck nicht mehr erfüllt (false) ist, dann weiter mit WeitereAnweisungen.



while-Schleife / Beispiel

```
// Quersumme von a berechnen
```

```
int a = 7463;
```

```
int quersumme = 0;
```

```
while (a > 0) {
```

```
    quersumme = quersumme + (a % 10);
```

```
    a = a / 10;
```

```
}
```

- Bevorzuge for-Schleife
- Verwende while-Schleife, falls Schleifenvariablen nicht lokal bezüglich der Schleife und nicht rein „technisch“ sind
- „Technisch“: Variable hat keine besondere Bedeutung für das zu lösende Problem



Algorithmus

- **Algorithmus**
 - Dient zur Lösung eines Problems (Eingabe des Algorithmus)
 - Verarbeitungsvorschrift, die präzise formuliert ist und von einem Rechner ausgeführt werden kann
 - Ausgabe des Algorithmus ist Lösung des Problems
 - Analoge Alltagsbeispiele: Kochrezept, Bastelanleitung, Spielregeln
 - Algorithmus hängt von einem Berechenbarkeitsmodell ab (mathematisches Modell eines Prozessors): Berechenbarkeitstheorie
- **Korrektheit** eines Algorithmus
 - Wenn Algorithmus terminiert berechnet er wirklich, was er soll und terminiert immer für alle Eingaben, die ein zu lösendes Problem beschreiben
- **Terminierung** Algorithmus
 - Algorithmus endet nach endlicher Anzahl Berechnungsschritte
- **Verifikation** eines Algorithmus
 - Überprüfen, ob Algorithmus korrekt ist (und terminiert)
- **Invariante** eines Teils eines Algorithmus
 - Bedingung oder Eigenschaft, die immer erfüllt ist

Algorithmus

- Problembeschreibung in Form
 - „Gegeben“ (Eingabe)
 - „Gesucht“ (Ausgabe)
- Algorithmus für *Lösung* des Problems gesucht
 - Verarbeitung der Eingabe und Erzeugen der Ausgabe

Beispiel:

- Problem Quersumme berechnen
 - Gegeben: Eine positive ganze Zahl a
 - Gesucht: Summe q aller Dezimalziffern der Zahl
- Eine Lösungsmöglichkeit (**Pseudocode**)
 - $q = 0$
 - Solange $a > 0$ ist
 - $q = q + a \% 10$
 - $a = a / 10$
- Ein Algorithmus ist unabhängig von einer konkreten Implementierung



Terminierung / Quersumme

- Terminiert Lösung immer für die erlaubten Eingaben?
 - Ja
 - a wird durch fortlaufende Division mit 10 immer kleiner
 - Da a immer positiv bleibt, wird a nach endlicher Anzahl von Schritten 0 und die Schleife bricht ab



Korrektheit / Quersumme

- Ist q nach Terminierung die Quersumme von a ?
 - Ja
 - **Invariante** der Schleife:
 - Nach $n > 0$ Durchläufen ist q die Summe der letzten n Dezimalziffern von der Eingabe a
 - Diese Bedingung gilt *vor* und *nach* jedem Schleifendurchlauf
 - Beweis durch vollständige Induktion
 - Schleife wird so oft durchlaufen, wie a Dezimalstellen hat, da mit $a = a / 10$ immer bei jedem Durchlauf die letzte Dezimalstelle gestrichen wird
 - Sei n die Anzahl Dezimalstellen von a , dann ist wegen der Invariante q die Summe aller Dezimalstellen von a



Beispiel grösste gemeinsame Teiler zweier Zahlen

- Definition

- a heißt *Teiler* von b , wenn es eine Zahl c mit $a \cdot c = b$ gibt. Umgekehrt heißt b dann auch ein *Vielfaches* von a .
- Zwei Zahlen x und y haben dann einen grössten gemeinsamen Teiler.

- Beispiele

- $\text{ggT}(2,204) = 2$ (2 teilt 2 und 204, gibt keine grössere Zahl, die 2 teilt)
- $\text{ggT}(17,17) = 17$ (17 teilt 17 und 17, gibt keine grösseren Teiler)
- $\text{ggT}(7,41) = 1$ (beides Primzahlen, 1 einzige gemeinsame Teiler)

Beispiel grösste gemeinsame Teiler zweier Zahlen

- Problem
 - Geben: zwei positive, ganzen Zahlen a und b .
 - Gesucht: der größte gemeinsame Teiler von a und b , $ggT(a,b)$

- Euklidischer Algorithmus

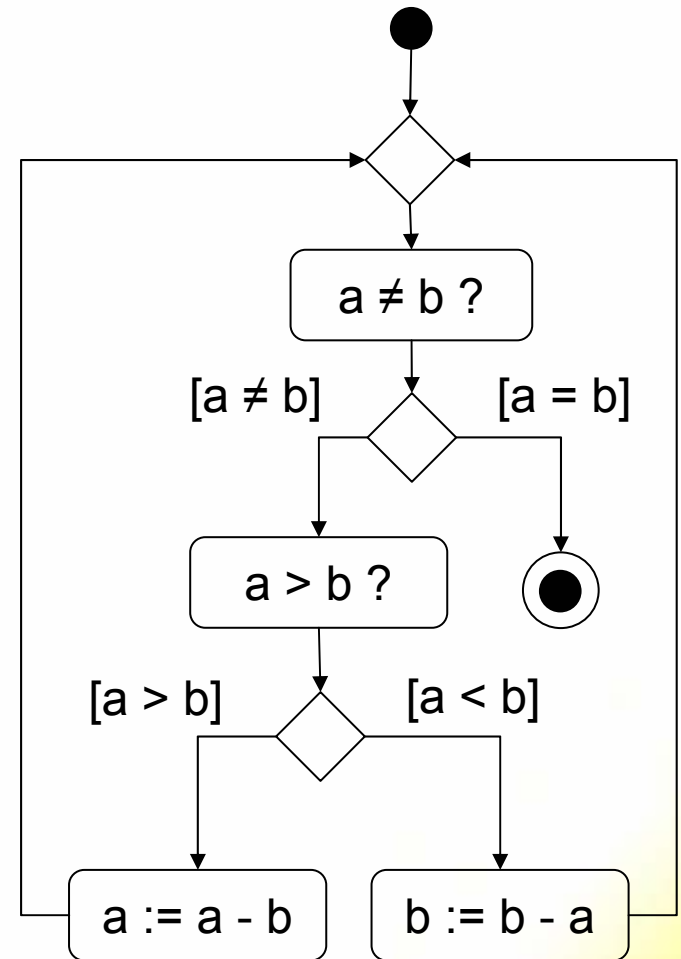
Solange a ungleich b ist:

Wenn $a > b$ dann sei $a := a - b$

Ansonsten sei $b := b - a$

Danach ist a der ggT vom ursprünglichen a und b

($:=$ soll Zuweisung sein,
= Gleichheit)





Euklidischer Algorithmus

- $a = 6, b = 15$
- 1. Durchlauf
 - $b > a$: neues b ist $b - a = 15 - 6 = 9$
- 2. Durchlauf
 - $b > a$: neues b ist $9 - 6 = 3$
- 3. Durchlauf
 - $a > b$: neues a ist $6 - 3 = 3$
- Abbruchbedingung erreicht
- Berechneter Wert ist $3 = \text{ggT}(6, 15)$
 - 3 teilt 6 und 15
 - Nächst größerer Teiler von 6 ist 6, 6 teilt aber nicht 15, 3 ist also größter, gemeinsamer Teiler von 6 und 15



Terminierung Euklidischer Algorithmus

- Berechnung (Euklidischer Algorithmus)
Solange a ungleich b ist:
Wenn $a > b$ dann sei $a = a - b$
Ansonsten sei $b = b - a$
- Terminierung
 - a oder b werden in jedem Schleifendurchlauf kleiner (da a und b positive Zahlen)
 - a und b bleiben immer positiv (**Schleifeninvariante**):
 - Entweder $a > b$, dann $a - b > 0$, also neues $a > 0$
 - Oder $b > a$, dann $b - a > 0$, also neues $b > 0$
 - Da a oder b immer positiv bleiben und in jeden Durchlauf a oder b kleiner wird, müssen Sie irgendwann gleich werden



Korrektheit Euklidischer Algorithmus

- Berechnung (Euklidischer Algorithmus)
 - Solange a ungleich b
 - Wenn $a > b$ dann sei $a = a - b$ (berechne $\text{ggT}(a-b, b)$)
 - Ansonsten sei $b = b - a$ (berechne $\text{ggT}(a, b-a)$)
- Korrektheitsbeweis
 - Es reicht zu zeigen, dass $\text{ggT}(a,b) = \text{ggT}(a-b, b)$ gilt, falls $a > b$ (andere Fall analog, steckt vollst. Induktion dahinter)
 - Zu beweisen sind dann folgenden Aussagen
 - Falls $a > b$ ist, dann ist $\text{ggT}(a, a - b)$ ein Teiler von a und b
 - Falls $a > b$ ist, dann ist $\text{ggT}(a, a - b)$ grösster Teiler von a und b



Korrektheit Euklidischer Algorithmus

- **Behauptung**

Falls $a > b$ ist, dann ist $\text{ggT}(a-b, b)$ ein Teiler von a und b

- **Beweis**

$t = \text{ggT}(a-b, b)$ ist Teiler von $a-b$ und b

Also: $a - b = t \cdot u$ und $b = t \cdot v$ für $u, v > 0$

$$a - b + b = \underline{a} = t \cdot u + t \cdot v = \underline{t \cdot (u + v)}$$

t teilt also auch a (und b sowieso)

- **Behauptung**

Falls $a > b$ ist, dann ist $\text{ggT}(a, a-b)$ grösster Teiler von a und b

- **Beweis**

Annahme es gibt größeren Teiler t von a und b

Es ex. $u, v > 0$ mit $t \cdot v = a$ und $t \cdot u = b$

$$\text{Dann gilt } a - b = t \cdot v - t \cdot u = t \cdot (v - u)$$

Also teilt t auch $a - b$ im Widerspruch dazu, dass t nicht größter gemeinsamer Teiler von a und b war



Grösste gemeinsame Teiler zweier Zahlen

```
int a = 24752;  
int b = 9192748;
```

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

```
// a ist ggt
```

```
int a = 24752;  
int b = 9192748;
```

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

```
// a ist ggt
```

 while verwenden,
da keine technische Schleifenvariable existiert



Testen vs. Verifikation

- **Verifikation eines Algorithmus**
 - *Mathematischer* Beweis der Korrektheit und der Terminierung für alle möglichen (unendlich viele) Eingaben des Algorithmus
- **Testen eines Algorithmus**
 - Ausführen des Algorithmus mit endliche vielen, verschiedenen Eingabe und Vergleich der Ausgabe mit erwarteten Ergebnissen
 - Rechnerübung: Automatisierte Einzeltests mit JUnit
- **Verifikation „besser“ als Testen, aber meist unpraktikabel**
 - Problembeschreibung, muss mathematisch beschrieben sein
 - Korrektheitsbeweise sehr umfangreich
- **Review**
 - Verifikation von Programmen / Entwurf durch Menschen
 - Mensch überprüft fremdes Programm / Entwurf daraufhin, ob das beschriebenen Problem gelöst wird (Anforderungen erfüllt sind)
 - Entwurfs- und Programmierrichtlinien für Info 1 (siehe Website)
- **Praxis: Reviews und Tests**



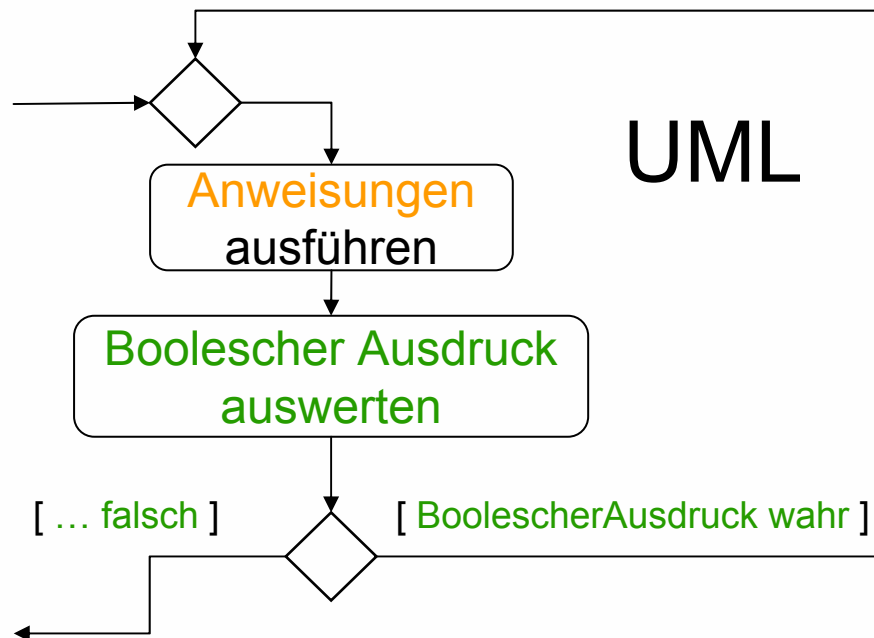
Inhalt

- Algorithmen
 - Euklidischer Algorithmus (while Schleife)
 - do-while
- Felder
 - Notenspiegel berechnen
 - Sieb des Eratosthenes
 - Pascalsche Dreieck
- Bedingte Anweisung
 - switch
 - String
 - switch

do-while Schleife / Syntax

Java

```
do {
    Anweisungen
} while ( BoolescherAusdruck );
```



- Bei initialer Ausführung von do-while, werden Anweisungen einmal ausgeführt
- Wenn BoolescherAusdruck erfüllt (true) ist, führe Anweisungen wiederholt aus.
- Wenn BoolescherAusdruck nicht mehr erfüllt (false) ist, dann weiter mit Anweisungen nach do-while-Block.
- Verwenden, wenn etwas wiederholt und mindestens einmal ausgeführt werden soll



do-while Schleife / Syntax

- Gegeben: Ein Intervall $[0, n)$
- Gesucht: Eine durch 7 teilbare zufällig ausgewählte Zahl aus $[0, n)$
- Beispiel: Intervall: $[0, 10000)$
Zahlen: 5999, 27426, 9282, 16541, ...

```
int n = 100000;  
int zahl = 7;  
do {  
    zahl = (int)           // Nachkommaanteil verwerfen  
           ( Math.random() // zufällige Zahl aus [0, 1)  
             * n ); // [0, n)  
} while (zahl % 7 != 0);
```

- Unter der Annahme, dass `Math.random()` wirklich zufällig Zahlen erzeugt:
Wie groß ist die Wahrscheinlichkeit, dass die Schleife 100 mal durchlaufen wird, bevor sie abbricht?



Felder

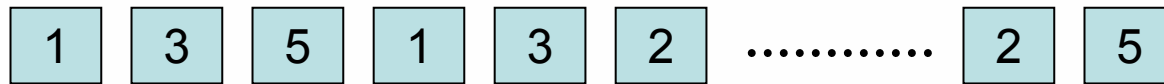
- Bisher
 - Variablen besitzen immer nur einen primitiven Wert oder weisen auf ein einzelnes Objekt
- Realität
 - Person kann *mehrere* Adressen haben
 - Versicherung hat *viele* Versicherte
 - Dozent unterrichtet *viele* Studenten
 - Mathematik: Wie können Vektoren, Matrizen in Java abgebildet werden?



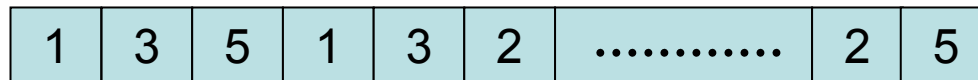
Felder

- Feld (*Array*)

- Daten als *Folge* einzelner Werte speichern
- Bisher:
 - einzelne Datenobjekt, z.B. Typ **int**
 - Jede einzeln als lokale Variablen oder Eigenschaft



- Ein Feld fasst diese als Folge zusammen



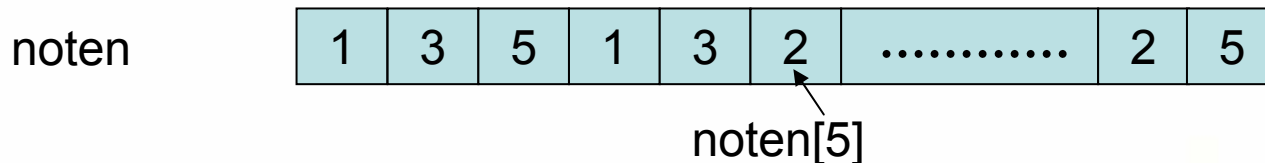
```

int [] noten; // Java-Stil, bevorzugt verwenden
int noten []; // C/C++ Stil, vermeiden!
int [] noten = new int[120]; // Initialisiere Feld mit
                                // Platz für 120 int Werte
                                // alle mit 0 Initialisiert
  
```



Felder

- **Felder in Java**
 - Ein Feld ist ein Objekt (kein Zeiger wie in C/C++): Felder haben Methoden und Eigenschaften
 - Attribute, Variablen und Parameter können als Feld deklariert werden
 - Ein Feld besteht aus endlicher Anzahl Variablen, kann auch 0 sein (dann ist Feld leer)
 - Variablen eines Felds werden nicht über einen Namen identifiziert, sondern über einen Index (byte, short, int, kein long) Variablen haben alle den gleichen Typ: Feldtyp
 - Zugriff auf Element außerhalb gültigen Bereichs (negativ oder über rechten Bereich) erzeugt eine Ausnahme (IndexOutOfBoundsException)
 - Vor Zugriff auf Elemente immer Bereich der Indexvariablen prüfen





Feld

```
// Initialisierung mit angegebener Folge von Werten
int [] noten = {1, 3, 2, 5, 6, 1, 2, 3, 3, 2, 3};
// Initialisierung mit Folge der Länge 0
Person [] personen1 = new Person[0];

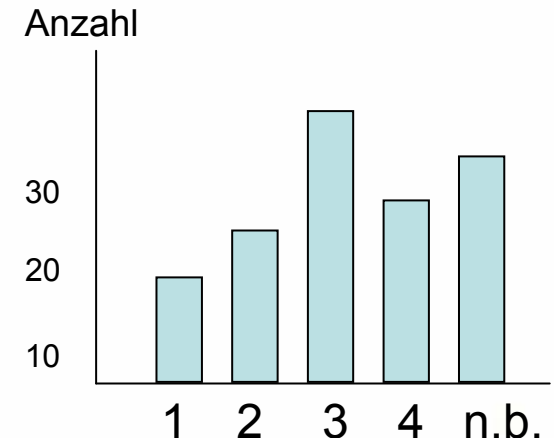
Person [] personen2 = { new Person("Müller"),
                       new Person("Meier")
                       };
```

- Felder sind Objekte: Eigenschaft „length“
 - noten.length repräsentiert Länge des Felds: noten.length ist 11
 - personen1.length ist 0
 - personen2.length ist 2
- Unterschied Referenztyp – elementarer Typ analog lokale Variablen
 - noten[2] beinhaltet Wert (int ist elementarer Typ)
 - personen2[1] ist Referenz auf Objekt (Person ist ein Referenztyp)
 - noten[2] verhält sich wie eine normale Variable
 - noten ist Referenz auf das Feldobjekt



Feld

- Problem
 - Notenspiegel (Noten 1-6) bestimmen.
165 Noten liegen vor.
 - 5,6 gilt als nicht bestanden „n.b.“
 - Die Häufigkeiten sollen gezählt werden.
- Gesucht
 - Ausgabe des Notenspiegels (textuell)
 - Berechnung des Notenspiegels





Ausgabe Notenspiegel

```
int [] notenspiegel = new int[6];
int [] noten = {1, 3, 2, 5, 6, 1, 2, 3, 3, 2, 3};

// notenspiegel[i-1] soll Anzahl Note i enthalten:
// Fehlt: Notenspiegel berechnen

for (int i=0; i < notenspiegel.length; i++) {
    System.out.println("Anzahl Note " + (i + 1)
        + " = " + notenspiegel[i]);
}
```

notenspiegel.length ist 6

Wert an i-ter Stelle (Note i+1) ausgeben



Notenspiegel berechnen 1

```
int [] notenspiegel = new int[6];
int [] oten = {1, 3, 2, 5, 6, 1, 2, 3, 3, 2, 3};

// notenspiegel[i-1] soll Anzahl Note i enthalten:

for (int i = 0; i < noten.length; i++) {
    int note = noten[i];

    notenspiegel[ note - 1 ] = notenspiegel[ note - 1 ] + 1;
}

for (int i=0; i < notenspiegel.length; i++) {
    System.out.println("Anzahl Note " + (i + 1)
        + " = " + notenspiegel[i]);
}
```



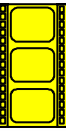
Notenspiegel berechnen 2

```
int notenspiegel [] = new int[6];
int noten [] = {1, 3, 2, 5, 6, 1, 2, 3, 3, 2, 3};

// notenspiegel[i-1] soll Anzahl Note i enthalten:

// Kürzere Variante:
for (int i = 0; i < noten.length; i++) {
    notenspiegel[ noten[i] - 1 ]++;
}

for (int i=0; i < notenspiegel.length; i++) {
    System.out.println("Anzahl Note " + (i + 1)
        + " = " + notenspiegel[i]);
}
```



Notenspiegel

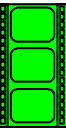
```
int notenspiegel [] = new int[6];  
int noten [] = {1, 3, 2, 5, 6, 1, 2, 3, 3, 2, 3};  
  
for (int i = 0; i < noten.length; i++) {  
    notenspiegel[ noten[i]-1 ]++;  
}
```

1	3	2	5	6	1	2	3	3	2	3
---	---	---	---	---	---	---	---	---	---	---

noten

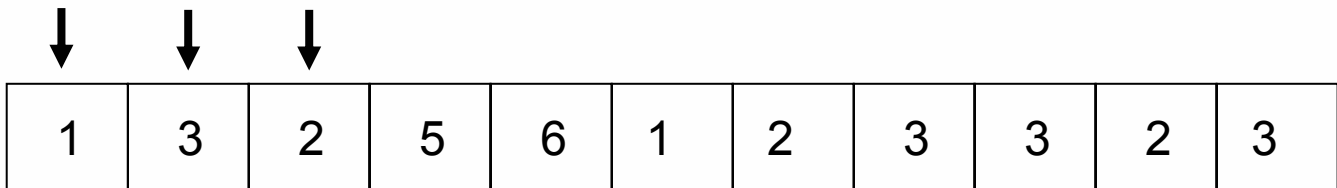
0	0	0	0	0	0
---	---	---	---	---	---

notenspiegel

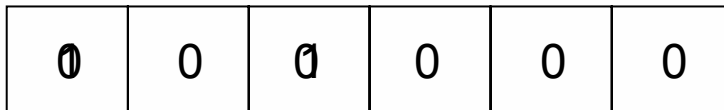


Notenspiegel

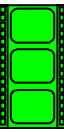
```
int notenspiegel [] = new int[6];  
int noten [] = {1, 3, 2, 5, 6, 1, 2, 3, 3, 2, 3};  
→ for (int i = 0; i < noten.length; i++) {  
→   notenspiegel[ noten[i]-1 ]++;  
}
```



noten



notenspiegel



Notenspiegel

```
int notenspiegel [] = new int[6];
int ergebnisse [] = {1, 3, 2, 5, 6, 1, 2, 3, 3, 2, 3};
```

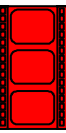
```
→ for (int i = 0; i < noten.length; i++) {
→   notenspiegel[ noten[i]-1 ]++;
→ }
→
```

1	3	2	5	6	1	2	3	3	2	3
---	---	---	---	---	---	---	---	---	---	---

noten

2	3	4	0	1	1
---	---	---	---	---	---

notenspiegel



Notenspiegel

```
int notenspiegel [] = new int[6];  
int ergebnisse [] = {1, 3, 2, 5, 6, 1, 2, 3, 3, 2, 3};  
  
for (int i = 0; i < ergebnisse.length; i++) {  
    notenspiegel[ noten[i]-1 ]++;  
}
```

1	3	2	5	6	1	2	3	3	2	3
---	---	---	---	---	---	---	---	---	---	---

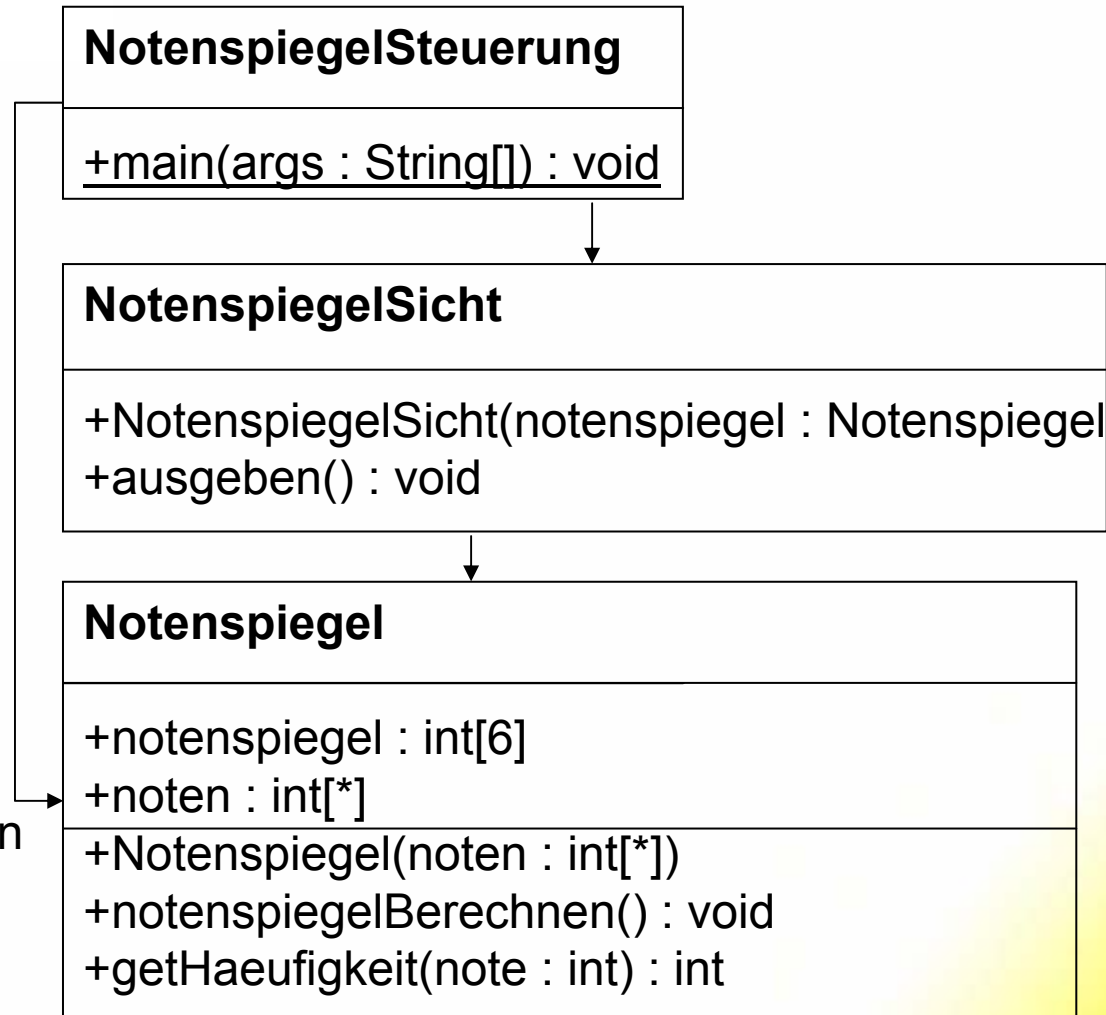
noten

2	3	4	0	1	1
---	---	---	---	---	---

notenspiegel

Notenspiegel Entwurf

- Berechnung/Modell (model), Ausgabe und Steuerung trennen
- Ausgabe/Sicht (view): Darstellung der Daten auf dem Bildschirm (grafisch, textuell)
- Steuerung (control): Benutzereingabe, Auswertung der Eingabe, Steuerung der Anwendung
- Drei Schichten: Steuerung verwendet Modell und Ausgabe, Ausgabe verwendet Modell. Keine umgekehrten Abhängigkeiten
- Modell mit JUnit testen





Notenspiegel

```
public class Notenspiegel {  
  
    private int [] notenspiegel = new int[6];  
    private int [] noten;  
  
    public Notenspiegel(int [] noten) {  
        this.noten = noten;  
    }  
  
    public void berechneNotenspiegel() {  
        for (int i = 0; i < notenspiegel.length; i++) {  
            notenspiegel[i] = 0;  
        }  
        for (int i = 0; i < noten.length; i++) {  
            notenspiegel[noten[i]-1]++;  
        }  
    }  
  
    public int getHaeufigkeit(int note) {  
        return notenspiegel[note - 1];  
    }  
}
```



Notenspiegel

```
public class NotenspiegelSicht {  
  
    private Notenspiegel notenspiegel;  
  
    public NotenspiegelSicht(Notenspiegel notenspiegel) {  
        this.notenspiegel = notenspiegel;  
    }  
  
    public void ausgeben() {  
        for (int note = 1; note <= 6; note++) {  
            System.out.print("Note " + note + " : ");  
            System.out.println(notenspiegel.getHaeufigkeit(note));  
        }  
    }  
}
```



Feld / Notenspiegel

```
// Statt einer main-Methode ist die Steuerung normalerweise
// ein Objekt mit mehreren Methoden
public static void main(String argv[]) {
    int [] noten = {1, 3, 2, 5, 6, 1, 2, 3, 3, 2, 3};
    // die Noten kämen normalerweise vom Benutzer

    Notenspiegel notenspiegel = new Notenspiegel();
    NotenspiegelSicht sicht =
        new NotenspiegelSicht(notenspiegel);

    notenspiegel.berechneNotenspiegel();
    notenspiegel.ausgeben();
}
```



Inhalt

- Algorithmen
 - Euklidischer Algorithmus (while Schleife)
 - do-while
- Felder
 - Notenspiegel berechnen
 - **Sieb des Eratosthenes**
 - Pascalsche Dreieck
- Bedingte Anweisung
 - switch
 - char, String
 - switch



Sieb des Eratosthenes

- Problem:
 - Gegeben: Eine positive Zahl $n > 1$
 - Gesucht: Alle Primzahlen von 2 bis n (möglichst schnell)
- Analyse Primzahlen von 2 bis 7 ...
 - 2 erste Primzahl
Vielfachen von 2 keine Primzahlen (4, 6, 8, 10, ...)
 - 3 nächste Primzahl
Vielfachen von 3 keine Primzahlen (6, 9, 12, 15, ...)
 - 5 nächste Primzahl, (4 ist keine, da Vielfache von 2)
Vielfachen von 5 keine Primzahlen (10, 15, ...)
 - 7 nächste Primzahl (6 keine, da Vielfache von 3)
Vielfachen von 7 keine Primzahlen (14, 21, ...)
- Lösung
 - Beginne mit 2 und „streiche“ alle Vielfachen von 2
 - Nächste nicht gestrichene Zahl ist eine Primzahl
 - Wiederhole dies mit nächster Primzahl bis n (oder besser bis Wurzel n) erreicht ist



Sieb des Eratosthenes

2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17

2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17

2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17

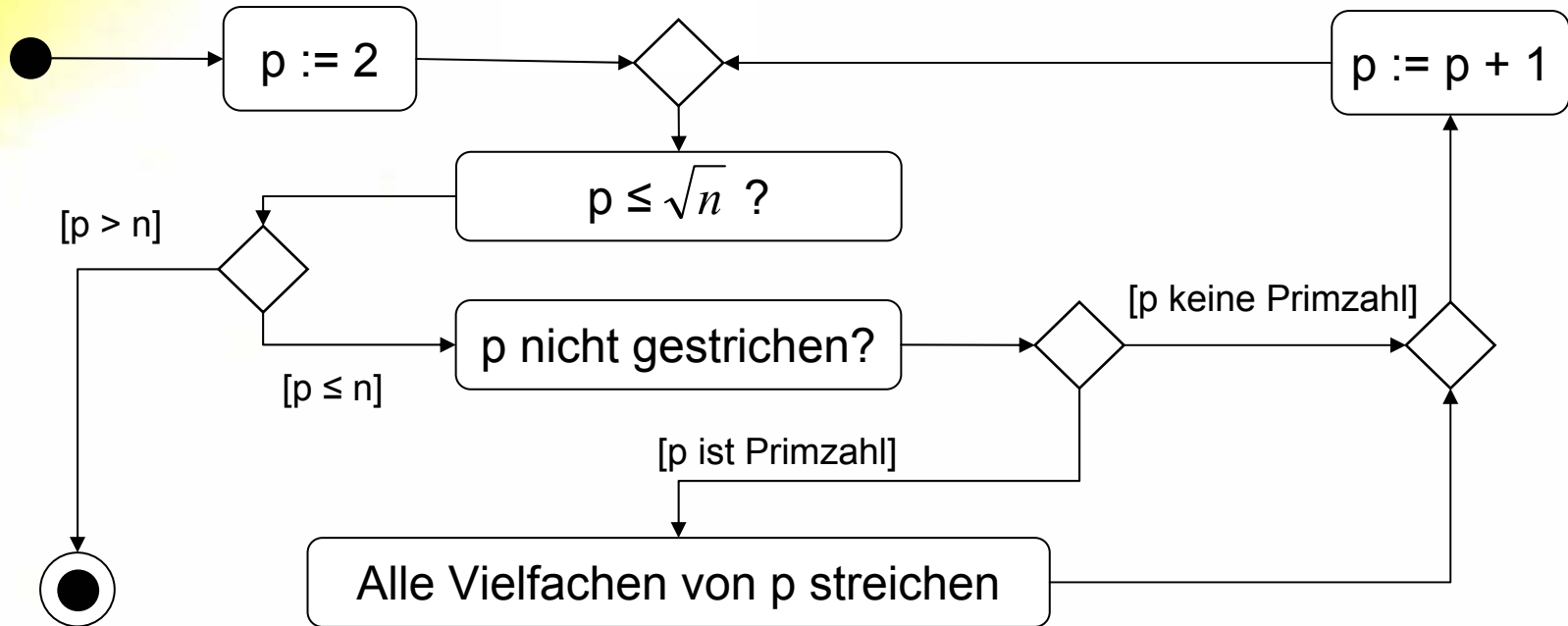
2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17

2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17

2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17

$5 \cdot 5 > 17$: Hier kann aufgehört werden

Sieb des Eratosthenes



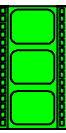
- Implementierung Java

- Alle Zahlen von 2 bis n in ein Feld
- Feldtyp
 - Merken, ob Zahl gestrichen / nicht gestrichen ist
 - *boolean*: false = gestrichen, true = nicht gestrichen
- Wiederholen bis Quadratwurzel von n erreicht (1. for-Schleife)
- Alle Vielfachen von p streichen (2. for-Schleife in 1. for-Schleife)



Sieb des Eratosthenes

- Abbruch bei Quadratwurzel von n
- Wieso ist das korrekt?
- Annahme: Es bliebe eine nicht herausgestrichene Zahl $a \leq n$ übrig, die keine Primzahl ist
 - Dann hat a einen Primfaktor p , der größer sein muss, als alle bisher gefundenen Primzahlen (sonst wäre a ja als Vielfaches dieser Primzahl herausgestrichen worden)
 - p muss größer als die Quadratwurzel von n sein (alle kleineren Primzahlen sind ja gefunden worden)
 - Da a keine Primzahl ist, muss a noch einen anderen Faktor größer 1 besitzen, der ebenfalls einen Primfaktor q enthält
 - Dieser Primfaktor ist ebenfalls (obige Argumentation) größer als die Quadratwurzel von n
 - Beide Primfaktoren multipliziert ergibt $a \geq p \cdot q > n$
 - Widerspruch dazu, daß $a \leq n$ ist



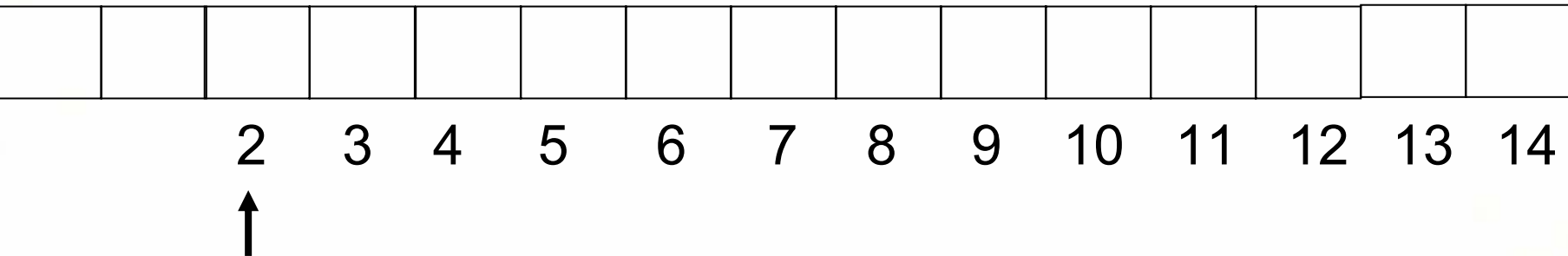
Sieb des Eratosthenes



Zahl nicht gestrichen, Wert true



Zahl gestrichen, Wert false



**Suche die kleinste Primzahl, gestrichene Zahl. Diese Zahl ist eine Primzahl.
 Setze sukzessive jedes Feld zwei Stellen weiter rechts auf false**



Sieb des Eratosthenes

SiebDesEratosthenes

-sieb : boolean []

+SiebDesEratosthenes(groesse : int)

+berechnePrimzahlen() : void

+printPrimzahlen() : void



Sieb des Eratosthenes

```
public class SiebDesEratosthenes {  
  
    private boolean [] sieb;  
  
    public SiebDesEratosthenes(int maximaleZahl) {  
        sieb = new boolean[maximaleZahl];  
    }  
  
    public void berechnePrimzahlen() {  
        // Berechne die Primzahlen mit dem Sieb des Eratosthenes  
    }  
  
    public void printPrimzahlen() {  
        for (int i = 2; i < sieb.length; i++) {  
            if (sieb[i]) {  
                System.out.println(i);  
            }  
        }  
    }  
}
```



Sieb des Eratosthenes

```
public void berechnePrimzahlen() {  
  
    // sieb initialisieren (Initialwerte sind false)  
for (int i=0; i < sieb.length; i++) {  
        sieb[i] = true;  
    }  
for (int p=2; p*p < sieb.length; p++) {  
        if (sieb[p]) { // nächste Primzahl gefunden  
            // streiche jedes weitere j-te Element  
        }  
    }  
}
```

Sieb des Eratosthenes

```
public void berechnePrimzahlen() {
```

```
    // sieb initialisieren (Initialwerte sind false)
```

```
    for (int i=0; i < sieb.length ; i++) {
```

```
        sieb[i] = true;
```

```
    }
```

```
    for (int p = 2; p * p <= sieb.length ; p++) {
```

```
        if (sieb[p]) { // nächste Primzahl gefunden
```

```
            for (int j = p; j < sieb.length - p; j = j + p) {
```

```
                sieb[j+p] = false;
```

```
            }
```

```
        }
```

```
    }
```

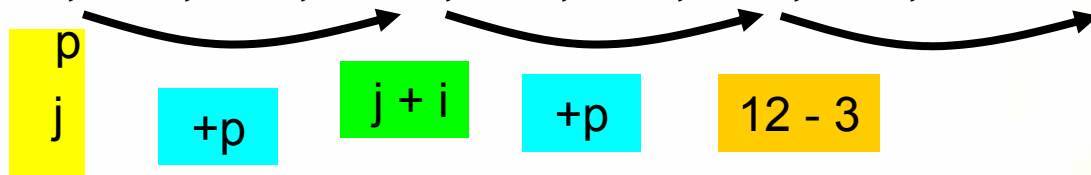
```
}
```

Die nächste p-te Zahl streichen.

Abbruchbedingung vorher erreichen,

Sonst wird auf `sieb[sieb.length+p]` zugegriffen.

0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11





Sieb des Eratosthenes

```
public static void main(String argv[]) {  
    SiebDesEratosthenes sieb = new SiebDesEratosthenes(100);  
  
    sieb.berechnePrimzahlen();  
    sieb.printPrimzahlen();  
}
```

Ersten hundert Primzahlen	2	19	47	79
	3	23	53	83
	5	29	59	89
	7	31	61	97
	11	37	67	
	13	41	71	
	17	43	73	



Sieb des Eratosthenes

- Benötigt Bruchteile von Sekunden für 100 000 Primzahlen (P4 Mobile mit 1,8 GHz)
- Ca. 1-2 s für 1 000 000 (inklusive Ausgabe)
- Vergleich zur Lösung mit Teiler
 - ca. 100 mal schneller bei 100 000 Primzahlen
- Je grösser die Grenze n , desto schneller im Vergleich zur Lösung mit Teiler
- Aber
 - Zur Grenze proportional viel Speicherverbrauch
 - Java: Index kann nur int Werte annehmen, bis $2^{32} - 1$
 - Grenze Primzahlsieb in Java von 2 bis etwas über 2 Mrd.



Sieb des Eratosthenes

- Verfahren ist als „Sieb des Eratosthenes“ bekannt
- Eratosthenes
 - Geb. ca. 284 v. Chr. in Kyrene
 - † 202 v. Chr. in Alexandria
 - Mathematiker, Geograf, Historiker, Philologe, Dichter
 - Direktor der Bibliothek von Alexandria
- Zwei bekannte Werke
 - Primzahlsieb
 - Berechnung Erdumfangs





Inhalt

- Algorithmen
 - Euklidischer Algorithmus (while Schleife)
 - do-while
- Felder
 - Notenspiegel berechnen
 - Sieb des Eratosthenes
 - **Pascalsche Dreieck**
- Bedingte Anweisung
 - switch
 - char, String
 - switch

Mehrdimensionale Felder

- Feld
 - Bisher: eindimensionale Felder
 - Typ des Feldes kann selbst wieder ein Feldtyp sein
 - Mehrdimensionale Felder möglich
 - Zum Beispiel für Matrizen, Tabellen, ...
- Multidimensional: Deklariere Felder mit Feldern als Elemente

Typ Instanz dieses Typs



`int [] a = {1, 5, -1};`

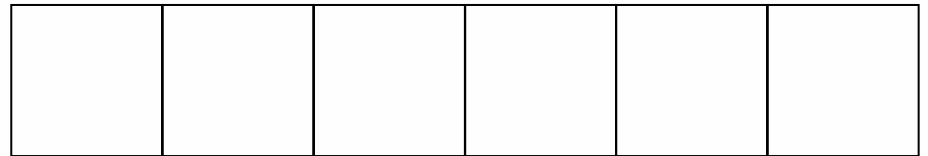
- Deklariere ein Feld mit Elemente von Typ `int []`

`int [][] b;`

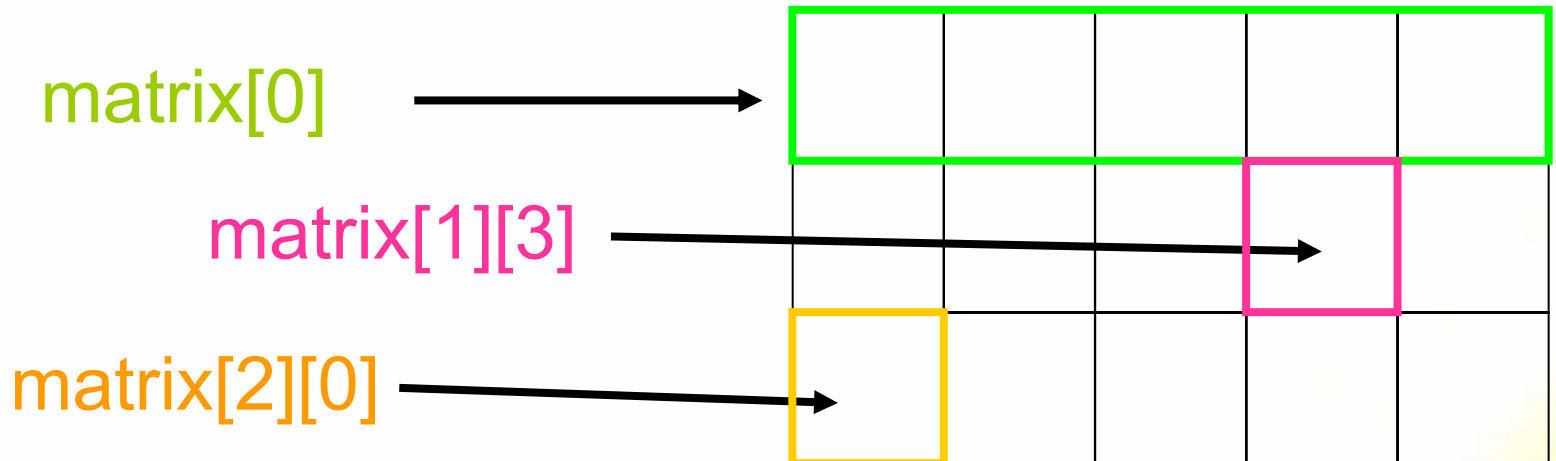
- `b[i]` ist dann ein(e Referenz auf ein) Feld mit int Werten
- `b[i] = a` möglich
- `(b[i])[j]` ist ein int Wert
- `int [][] a, b; // a und b sind jeweils 2-dimensionale Felder`
- Mischen C / Java-Stil vermeiden:
`int [] a, b []; // a ist 1-dimensional, b ist 2-dimensional`

Mehrdimensionale Felder

```
int [] zeile = new int[6];
```



```
int [] [] matrix = new int[3][5];
```





Mehrdimensionale Felder

```
int zeile [] = {5, 1, 6, 3, 5, -5};
```

5	1	6	3	5	-5
---	---	---	---	---	----

```
int matrix [] [] = {  
  { 5, 3, 8, 2, 1},  
  { 2, 1, -5, 4, 3},  
  {-5, 7, 4, 8, -6} };
```

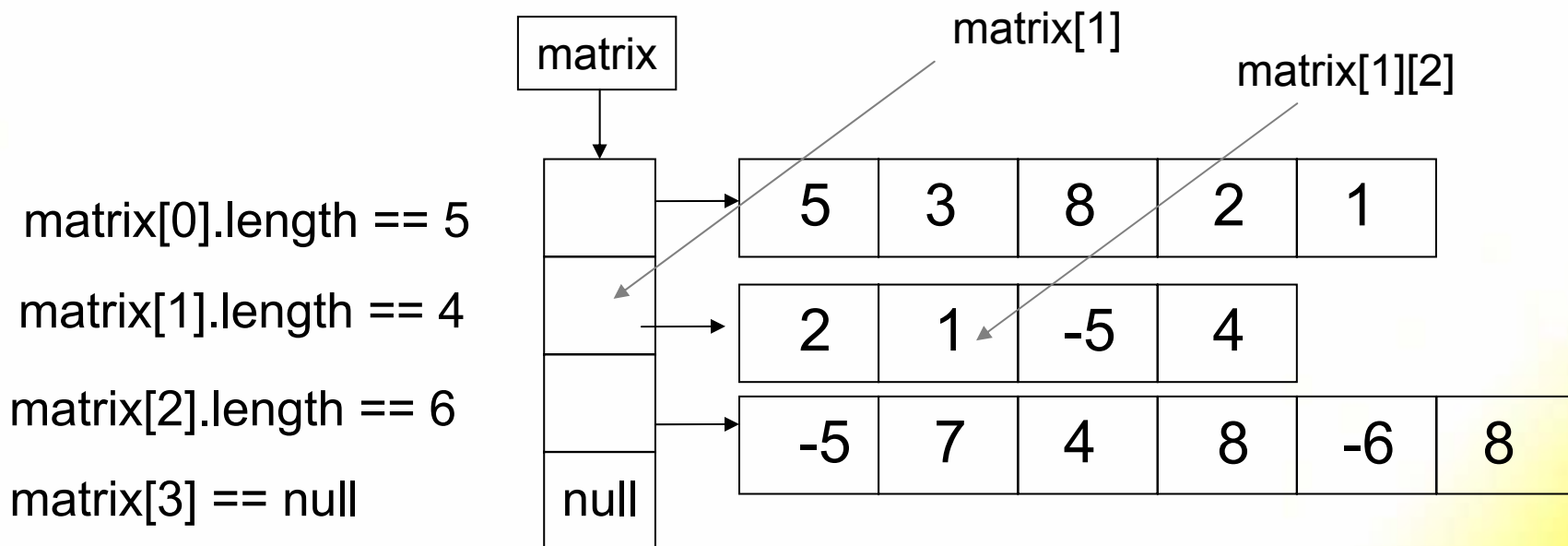
5	3	8	2	1
2	1	-5	4	3
-5	7	4	8	-6

Mehrdimensionale Felder

- Feldelemente eines Felds

- können unterschiedliche Länge haben (Im Ggs. zu Pascal)
- können null sein

```
int [] [] matrix = {
    { 5, 3, 8, 2, 1},
    { 2, 1, -5, 4},
    {-5, 7, 4, 8, -6, 8},
    null
};
```



Mehrdimensionale Felder

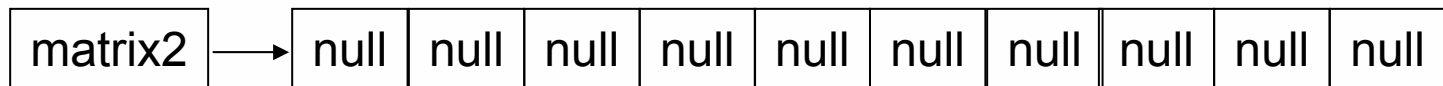
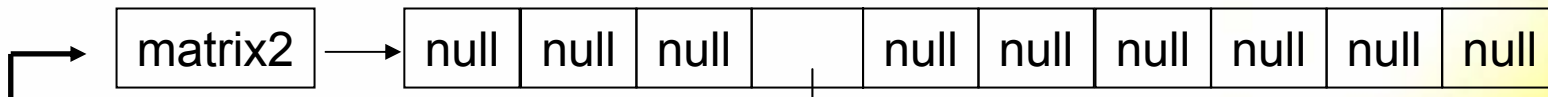
- Mindestens eine Dimension muss bei new angegeben werden
- Die letzten Dimensionen bei new können leer sein

Korrekt

```
int [] [] matrix1 = new int[10][12];
int [] [] matrix2 = new int[10][];
int [] [] [] [] matrix3 = new
int[10][50][][];
```

Compile-
fehler

```
int [] [] matrix = new int[][];
int [] [] matrix = new int[][10];
```

Speicher für
lokale VariableSpeicher für Feldobjekt mit Platz für 10 Objekten
vom Typ int []. Inhalte mit null initialisiert (leere Felder)

matrix[3] = new int[3];



Algorithmen, Kontrollanweisungen



Mehrdimensionale Felder

- Ausgabe eine zwei-dimensionalen Felds

```
int [] [] matrix = {  
    { 5, 3, 8, 2, 1},  
    { 2, 1, -5, 4, 3},  
    {-5, 7, 4, 8, -6} };
```

5	3	8	2	1
2	1	-5	4	3
-5	7	4	8	-6

Mehrdimensionale Felder

```
// Für jede Zeile i
// Inhalt der Zeile matrix[i] ausgeben
// Zeilenumbruch
```

```
for (int i = 0; i < matrix.length; i++) {

    // Inhalt der Zeile matrix[i] ausgeben

    // Zeilenumbruch
}
```

```
for (int i = 0; i < matrix.length; i++) {
    if (matrix[i] != null) {
        for (int j = 0; j < matrix[i].length; j++) {
            System.out.print( matrix[i][j] + " ");
        }
        System.out.println();
    }
}
```

Pascalsche Dreieck

$$(a + b)^2 = 1a^2 + 2ab + 1b^2$$

$$(a + b)^3 = 1a^3 + 3a^2b + 3ab^2 + 1b^3$$

Binomialkoeffizienten

$$(a + b)^n = \sum_{k=0}^n \binom{n}{k} a^{n-k} b^k = \sum_{k=0}^n \frac{n!}{k!(n-k)!} a^{n-k} b^k$$

$$\binom{n}{k} = \binom{n}{n-k}$$

$$\binom{n}{k} = \binom{n-1}{k-1} + \binom{n-1}{k}$$

(Pascalsche Dreieck)



Pascalsche Dreieck

				1						
				1		1				
			1		2		1			
		1		3		3		1		
	1		4		6		4		1	
1		5		10		10		5		1



Pascalsche Dreieck

- Berechnung Binomialkoeffizienten
 - Variante 1

$$\binom{n}{k} = \frac{n!}{k!(n-k)!}$$

- Variante 2

$$\binom{n}{k} = \binom{n}{n-k}$$

$$\binom{n}{k} = \binom{n-1}{k-1} + \binom{n-1}{k}$$



Pascalsche Dreieck

- Berechnung Binomialkoeffizienten
 - Variante 1: schlecht, da bei Berechnung mit long
$$20! = 1*2*3*4* \dots * 19*20 = 2432902008176640000$$
$$21! = -4249290049419214848$$
Koeffizienten zu 20 sind wesentlich kleiner als 20!
 - Variante 2: besser, da Koeffizienten schrittweise aus vorherigen durch Addition berechnet werden



Entwurfstechniken für Algorithmen

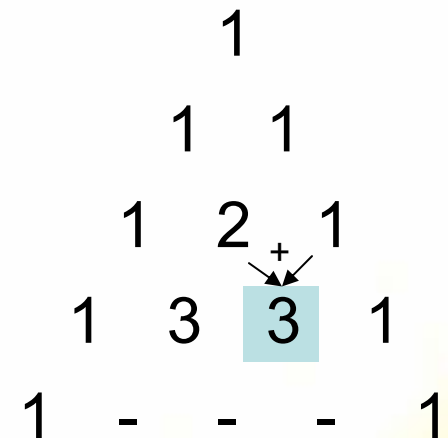
- **Dynamisches Programmieren**
- **Schrittweise Verfeinerung**
- Rekursion
- Backtracking
- Teile-und-Beherrsche
- Branch and bound
- Greedy-Algorithmen (gierige Algorithmen)
- Entwurf durch vollständige Induktion
- Zufallsgesteuerte Optimierung
- Genetische Algorithmen
- Problem auf bekanntes lösbares Problem reduzieren
- ...

Pascalsche Dreieck

- **Dynamisches Programmieren**
- Entwurfstechnik für Algorithmen
- Gegebenes Problem wird gelöst, indem es auf *alle* „kleineren“ Teilprobleme reduziert, diese gelöst und *zwischengespeichert* werden

- **Problem:** Binomialkoeffizient für großes n und k berechnen
- **Lösung:** Dynamisches Programmieren
 - *Speicher* Binomialkoeffizienten in einer Matrix $\text{binom}[][]$ für *alle* Werte kleiner n und k
 - Berechne **nächsten Binomialkoeffizient** mit Pascalschem Dreieck aus den vorherigen:

$$\text{binom}[n][k] = \text{binom}[n-1][k-1] + \text{binom}[n-1][k]$$





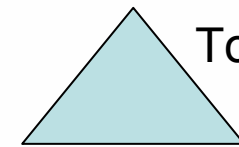
Pascalsche Dreieck

- **Schrittweise Verfeinerung (Top-down-Entwurf)**
- Entwurfs- und *Implementierungstechnik* für Algorithmen
- Problem (Algorithmus) wird schrittweise in sequentielle oder verzweigende Teilaufgaben zerlegt

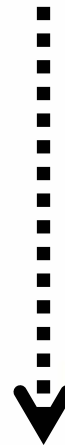
Überlege WAS gemacht werden soll

```
// 1) Berechne n über k mit Hilfe  
// des Pascalschen Dreiecks
```

WAS



Top (Entwurf)



WIE

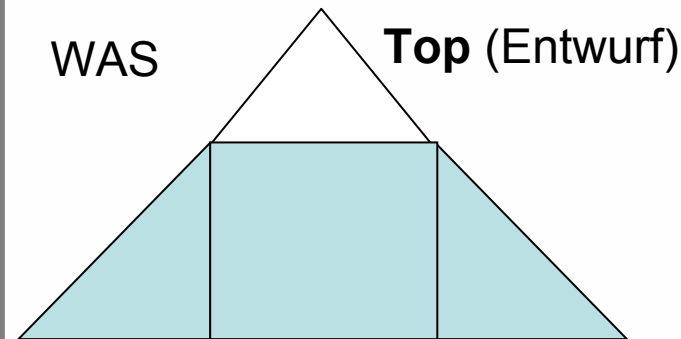
Down (Programm)

Pascalsche Dreieck

- **Schrittweise Verfeinerung (Top-down)**
- Entwurfs- und Implementierungstechnik für Algorithmen
- Problem (Algorithmus) wird schrittweise in sequentielle, iterative oder verzweigende Teile zerlegt

Überlege **WAS** gemacht werden soll

```
// 1 a) erzeuge n+1 x n+1 Matrix
// 1 b) initialisiere Diagonale und
//      1. Spalte mit Eins
// 1 c) Berechne n über k mit
//      Hilfe des Pascalschen Dreiecks
//      aus vorherigen Werten
```



WIE



Down (Implementierung)



Pascalsche Dreieck

Überlegen WIE es gemacht werden soll

```
int n = 5;
long [][] binom;
binom = new int[n+1][n+1]
```

```
// 1 b) initialisiere Diagonale und
//      1. Spalte mit Eins
```

```
// 1 c) Berechne n über k mit
//      Hilfe des Pascalschen Dreiecks
//      aus vorherigen Werten
```

	0	1	→	n
0					
1					
⋮					
⋮					
⋮					
⋮					
n					

Pascalsche Dreieck

Überlegen WIE es gemacht werden soll

```
int n = 5;
long [][] binom;
binom = new int[n+1][n+1]
```

```
// 1 b) initialisiere Diagonale und
//      1. Spalte mit Eins
```

```
// 1 c) Berechne n über k mit
//      Hilfe des Pascalschen Dreiecks
//      aus vorherigen Werten
```

1					
1	1				
1		1			
1			1		
1				1	
1					1

$i = j$

Spaltenindex $j = 0$
 Zeilenindex $i = 0, 1, 2, 3$

Pascalsche Dreieck

Überlegen WIE es gemacht werden soll

```
int n = 5;
long [][] binom;
binom = new int[n+1][n+1]
```

```
for (int i = 0; i <= n; i++) {
    binom[i][0] = 1;
    binom[i][i] = 1;
}
```

```
// 1 c) Berechne n über k mit
//     Hilfe des Pascalschen Dreiecks
//     aus vorherigen werten
```

1					
1	1				
1		1			
1			1		
1				1	
1					1

$i = j$

Spaltenindex $j = 0$
 Zeilenindex $i = 0, 1, 2, 3, \dots, n$

Pascalsche Dreieck

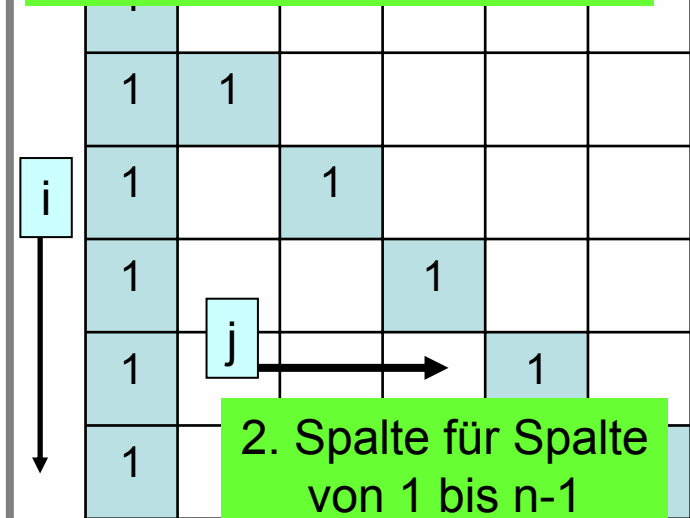
Überlegen WIE es gemacht werden soll

```
int n = 5;
long [][] binom;
binom = new int[n+1][n+1]

for (int i = 0; i <= n; i++) {
    binom[i][0] = 1;
    binom[i][i] = 1;
}
```

```
for (int i = 2; i < n; i++) {
    for (int j = 1; j < i; j++) {
        binom[i][j] = binom[i-1][j-1]
                    + binom[i-1][j];
    }
}
```

1. Zeile für Zeile von 2 bis n



2. Spalte für Spalte von 1 bis n-1

Pascalsche Dreieck

Überlegen WIE es gemacht werden soll

```
int n = 5;
long [][] binom;
binom = new int[n+1][n+1]

for (int i = 0; i <= n; i++) {
    binom[i][0] = 1;
    binom[i][i] = 1;
}
```

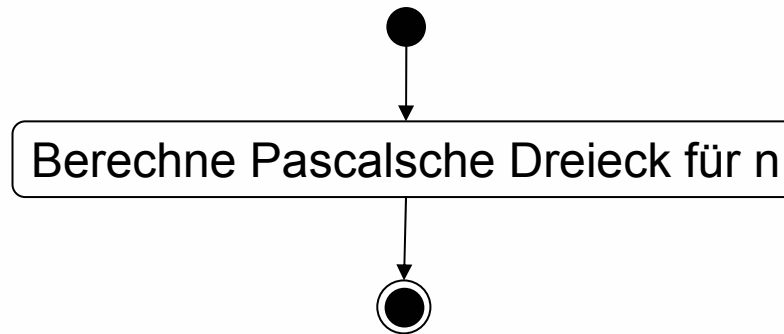
```
for (int i = 2; i < n; i++) {
    for (int j = 1; j < i; j++) {
        binom[i][j] = binom[i-1][j-1]
                    + binom[i-1][j];
    }
}
```

1					
1	1				
1	2	1			
1	3	3	1		
1	4	6	4	1	
1	5	10	10	5	1



Pascalsche Dreieck

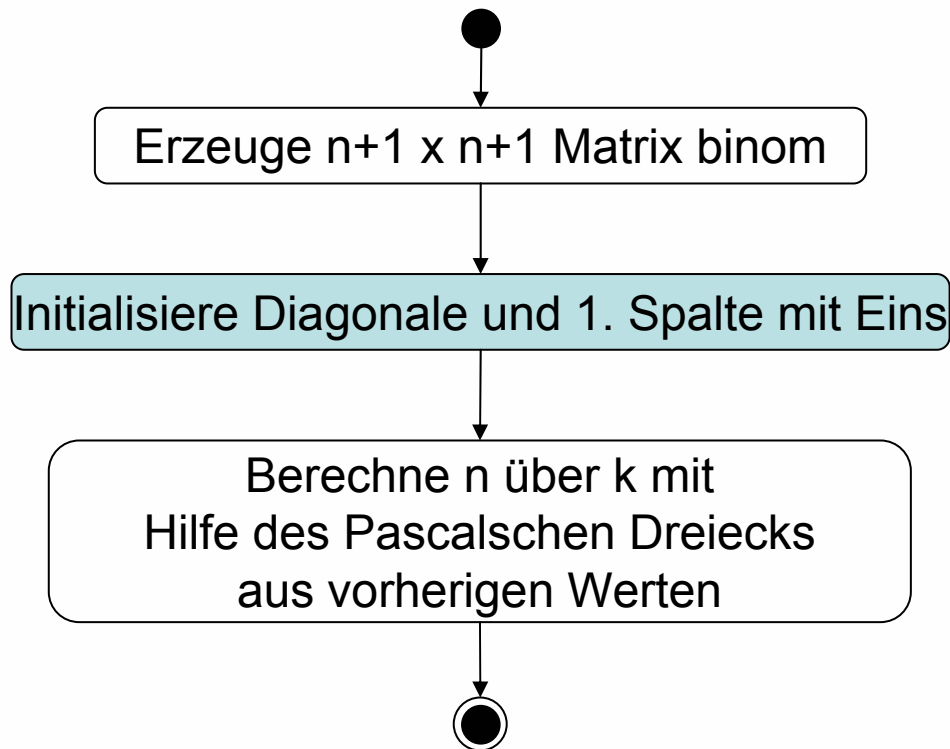
- **Schrittweise Verfeinerung (Top-down-Entwurf)**
 - Auch anwendbar auf Aktivitätsdiagramme
-





Pascalsche Dreieck

- **Schrittweise Verfeinerung (Top-down-Entwurf)**
 - Auch anwendbar auf Aktivitätsdiagramme
-



Pascalsche Dreieck

Erzeuge $n+1 \times n+1$ Matrix binom

$i = 0$

$i \leq n?$

[true]

$\text{binom}[i][0] = \text{binom}[i][i] = 1$
 $i = i + 1$

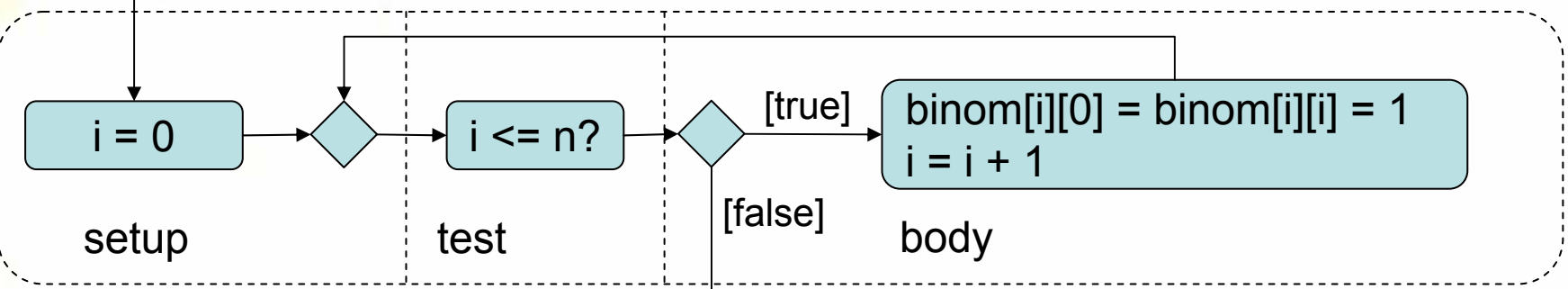
[false]

Berechne n über k mit
Hilfe des Pascalschen Dreiecks
aus vorherigen Werten



Pascalsche Dreieck

Erzeuge $n+1 \times n+1$ Matrix binom



Berechne n über k mit Hilfe des Pascalschen Dreiecks aus vorherigen Werten

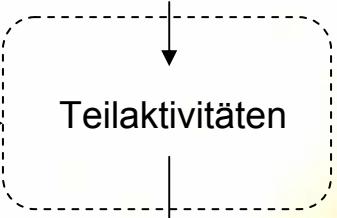


UML 2.0

Structured Activity Node

Enthält Teilaktivitäten

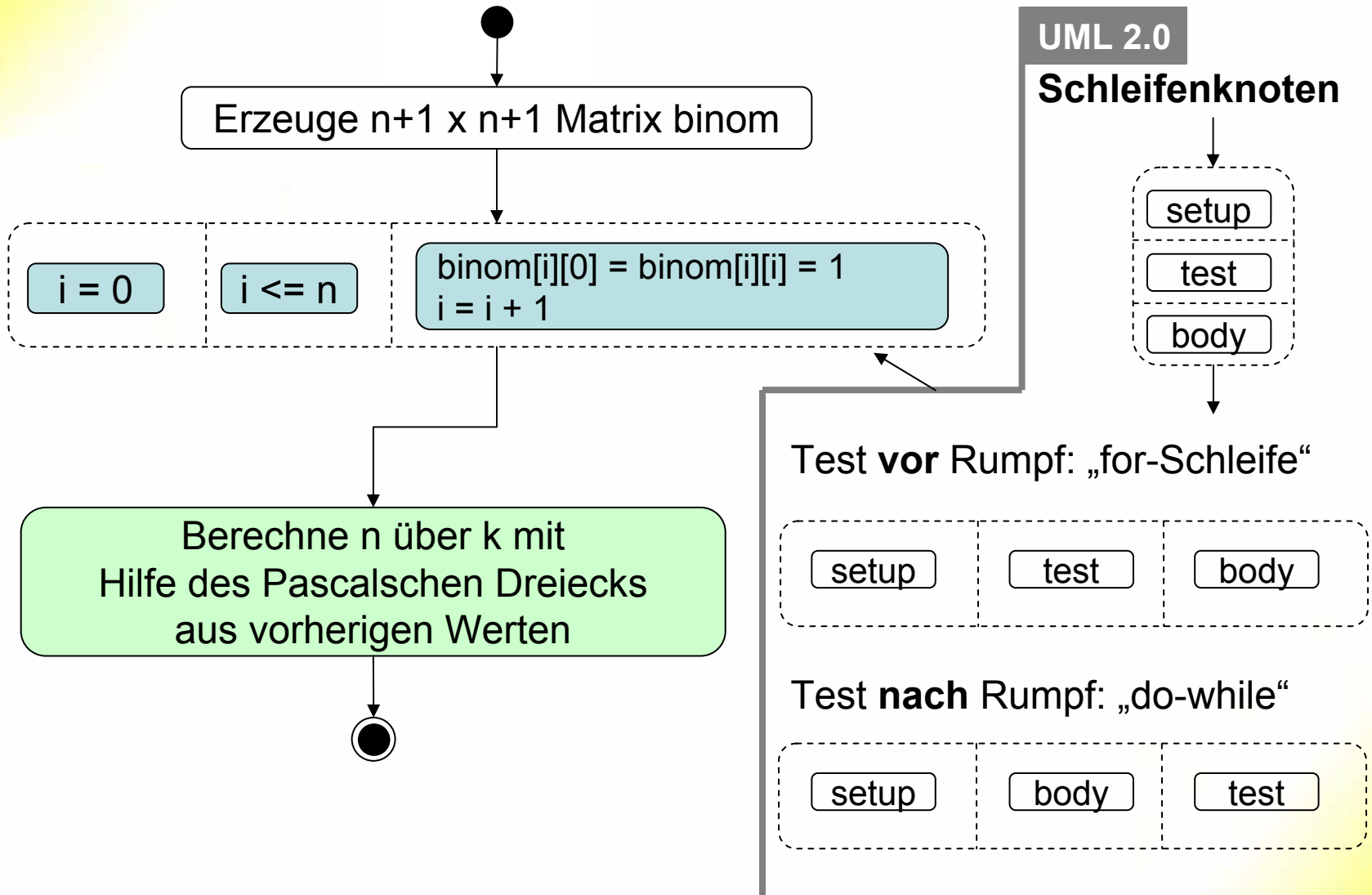
Expansionsraum



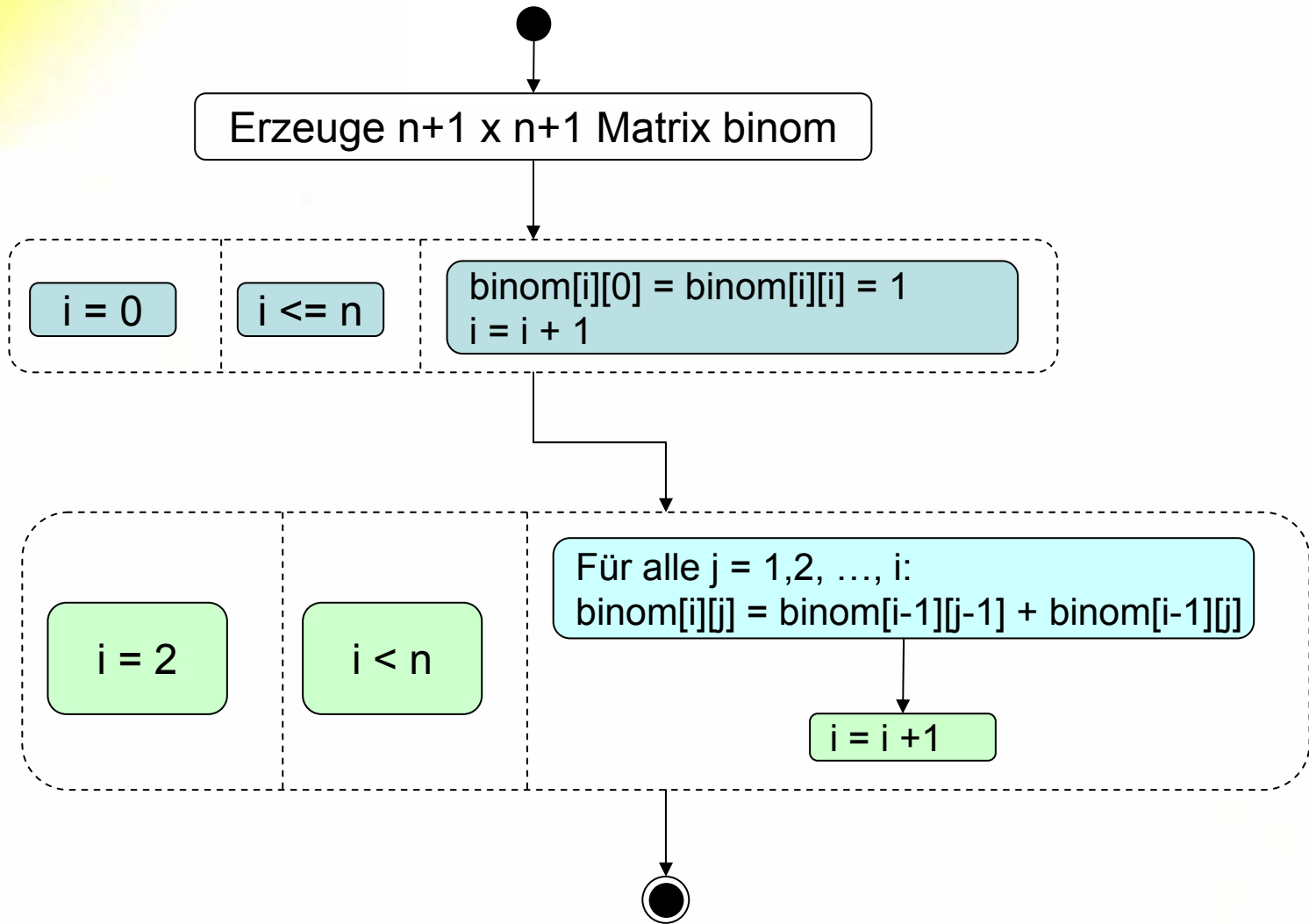
Mehre Ein-/Ausgabekontrollflüsse möglich, aber vermeiden, besser Merge, Join, Fork vor- und nachher verwenden



Pascalsche Dreieck



Pascalsche Dreieck



Pascalsche Dreieck

Erzeuge $n+1 \times n+1$ Matrix binom

$i = 0$

$i \leq n$

$\text{binom}[i][0] = \text{binom}[i][i] = 1$
 $i = i + 1$

$i = 2$

$i < n$

$j = 2$

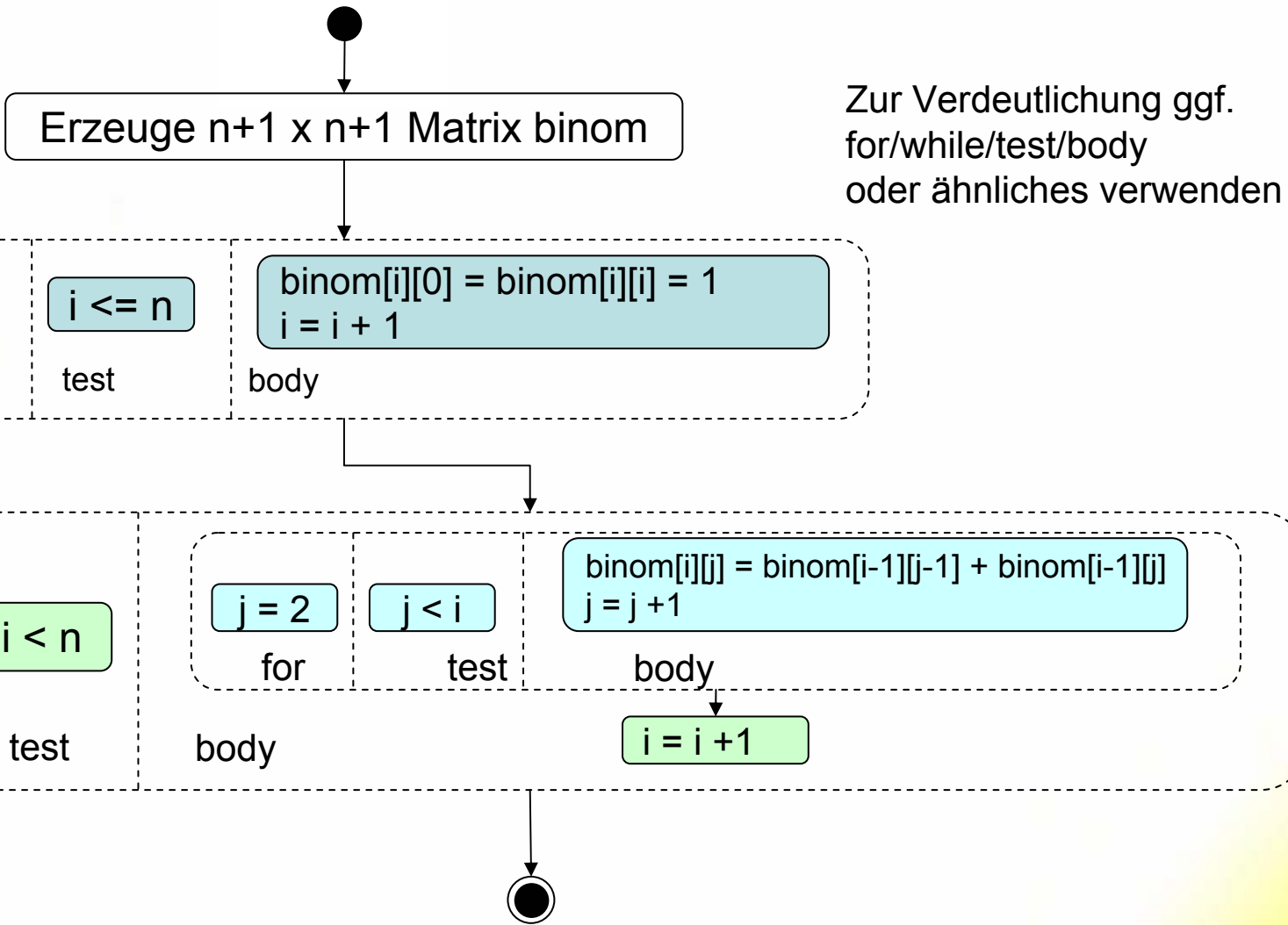
$j < i$

$\text{binom}[i][j] = \text{binom}[i-1][j-1] + \text{binom}[i-1][j]$
 $j = j + 1$

$i = i + 1$



Pascalsche Dreieck



Zur Verdeutlichung ggf.
for/while/test/body
oder ähnliches verwenden

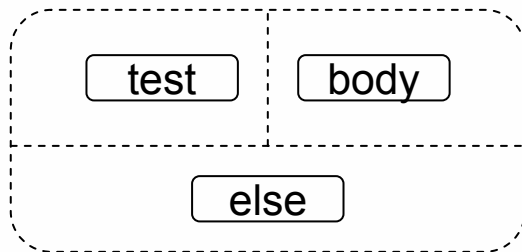
UML 2.0 Conditional Node

UML 2.0

“A **conditional node** is a structured activity node that represents an exclusive choice among **some number** of alternatives.”

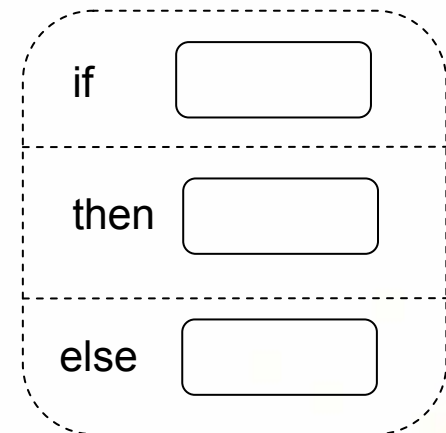
“A conditional node consists of one or more clauses. Each **clause** consists of a **test section** and a **body section**.”

“An “**else**” clause is a clause that is a successor to all other clauses in the conditional and whose test part always returns true”



if-then-else

Schwimmbahnen mit
if / then / else
markieren





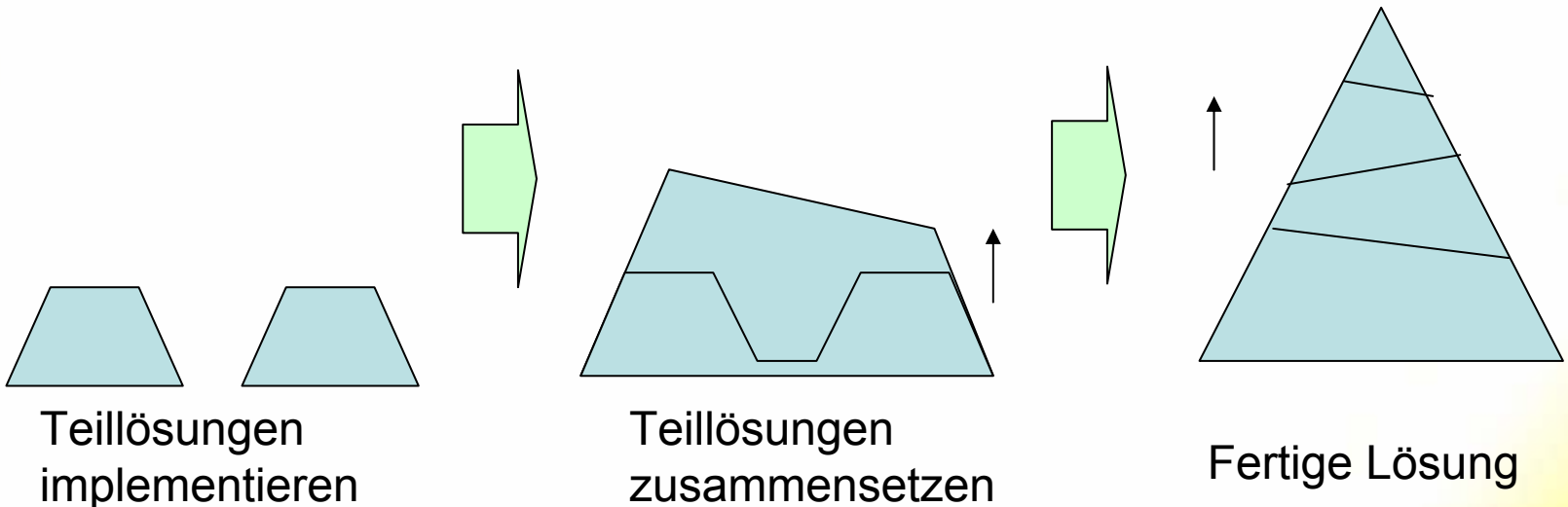
UML 2.0

- **loop / conditional nodes**
 - In UML 2.0 aufgenommen, da in Flußdiagramm / Nassie-Shneiderman Diagrammen enthalten
 - Sehr nah an Programmierung und fern von Analyse/Design
 - Beschreiben WIE etwas gemacht wird
- **Analyse**
 - Analyse beschreibt WAS gemacht werden soll
 - loop / conditional nodes in Analyse **vermeiden**
- **Design (Entwurf)**
 - Design beschreibt WIE etwas *strukturell* aufgebaut werden soll
 - loop / conditionals beim Entwurf **sparsam** verwenden
 - Programmiersprachen bieten oft andere / spezialisierte Schleifen als for / do-while von UML 2.0
- Bei Top-down-Entwurf eines Methodenrumpfs hilfreich



Bottom-Up

- **Bottom-Up**
 - Kleine Codefragmente mit klarem Ziel programmieren
 - Codefragmente zu größeren zusammensetzen
- Manchmal sinnvoll,
 - um Problem genauer „kennen zu lernen“ oder
 - wenn man anders nicht vorankommt

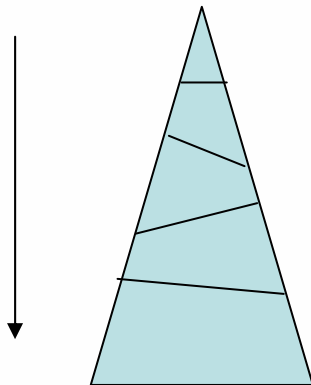




Top-Down / Objekt-orientiert

- **Top-down Entwurf**

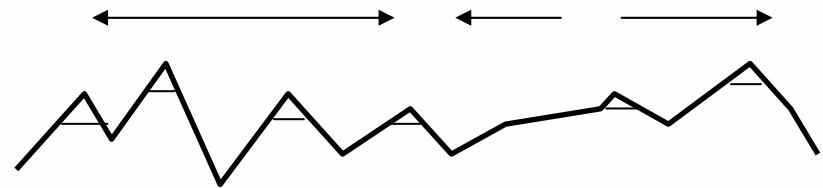
- Für Abläufe / Methoden
- In die Tiefe gehen
- Klar definierter Anfang
- Klar definiertes Ende



Problemstruktur: **Berg**

- **Objekt-orientierter Entwurf**

- Kein „Oben“ vorhanden
- In die Breite gehen
- Berggipfel finden (Methoden)
- Dann Top-down anwenden



Problemstruktur: **Gebirge**



Aufgabe

- Erstellen Sie mit Hilfe der schrittweise Verfeinerung (top-down) oder bottom-up ein detailliertes Aktivitätsdiagramm für das Sieb des Eratosthenes



Pascalsche Dreieck

BinomischeZahlen

-binom : long [] [];

+BinomischeZahlen(n : int)

+berechneBinomischeZahlen(): void

+printPascalscheDreieck(): void

```
BinomischeZahlen bz = new BinomischeZahlen(5);
bz.berechneBinomischeZahlen();
```

```
private class BinomischeZahlen {

    private long [][] binom = new long[0][0];

    public BinomischeZahlen(int n) {
        binom = new long[n+1][n+1];
    }

    public void berechneBinomischeZahlen() {
        for (int i=0; i < binom.length; i++) {
            binom[i][0] = 1;
            binom[i][i] = 1;
        }
        for (int i=2; i < binom.length; i++) {
            for (int j=1; j < i; j++) {
                binom[i][j] = binom[i-1][j-1]
                    + binom[i-1][j];
            }
        }
    }

    // printPascalscheDreieck() nächste Folie
}
```

bz

0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0

1	0	0	0	0
1	1	0	0	0
1	0	1	0	0
1	0	0	1	0
1	0	0	0	1

1	0	0	0	0
1	1	0	0	0
1	2	1	0	0
1	3	3	1	0
1	4	6	4	1

Pascalsche Dreieck

```

public void printPascalscheDreieck() {
    for (int i=0; i < binom.length; i++) {
        for (int j=0; j <= i; j++) {
            System.out.print(binom[i][j] + " ");
        }
        System.out.println();
    }
}

```

```

1
1 1
1 2 1
1 3 3 1
1 4 6 4 1
1 5 10 10 5 1
1 6 15 20 15 6 1
1 7 21 35 35 21 7 1
1 8 28 56 70 56 28 8 1
1 9 36 84 126 126 84 36 9 1

```

Pascalsche Dreieck

- Implementierung mit asymmetrischen Feld
 - Spart nicht wirklich viel Speicher
 - Nicht nachmachen, da fehleranfälliger

```
private class BinomischeZahlen {
    private long [][] binom= new long[0][0];

    public BinomischeZahlen(int n) {
        binom = new long[n][];
        for (int i=0; i < binom.length; i++) {
            binom[i] = new long[i+1];
        }
        // wie zuvor
    }

    // wie zuvor
}
```

Feld mit n Zeilen.
binom[i] enthält null

Analog zu (für n = 3)
binom = {null, null, null};

Für n = 4

0				
0	0			
0	0	0		
0	0	0	0	



for (ab JDK 1.5)

- Aufzählen aller Elemente eines Feldes

```
int [] a = new int[100];  
for (int i = 0; i < a.length; i++) {  
    int x = a[i];  
    System.out.println(x);  
}
```

- Spezielle for-Schleife

```
int [] a = new int[100];  
for (int x : a) {  
    System.out.println(x);  
}
```

Lokale Deklaration von x
In jedem Durchlauf wird x
Wert der nächsten Speicherzelle
von a zugewiesen

- Keine Zählvariable, kein a.length
 - Kürzere Schleifen
 - Übersichtlicher



Inhalt

- **Algorithmen**
 - Euklidischer Algorithmus
 - do-while
- **Felder**
 - Notenspiegel berechnen
 - Sieb des Eratosthenes
 - Pascalsche Dreieck
- **Bedingte Anweisung**
 - switch
 - char, String
 - Beispiel Implementierung endlicher Automat



Java / switch-Anweisung

- Kaskadierende if-else if-else if -:
 - Bedingungen jedes if werden berechnet
 - Nicht effizient bei vielen if-else-if-... und wenn Berechnung der Bedingung nur jeweils in einem Wert unterscheidet
 - Teilweise unübersichtlich
- switch-Anweisung
 - Manchmal übersichtlicher
 - Ausdruck (elementarer Typ) wird einmal ausgewertet
 - Pro Wert einen Fall (case)
 - Wird eher selten verwendet

```
long einheit; // 1 = 1000, k;  
              // 2 = 1000000, M,  
  
...  
String metrisch = "";  
if (einheit == 1) {  
    metrisch = "k";  
} else if (einheit == 2) {  
    metrisch = "M";  
} else if (einheit == 3) {  
    metrisch = "G";  
}
```



Java / switch-Anweisung

```
int faktor;  
...  
String metrisch = "";  
switch ( faktor ) {  
    case 1:  
        metrisch = "k";  
        break;  
    case 2:  
        metrisch = "M";  
        break;  
    case 3:  
        metrisch = "G";  
        break;  
}
```

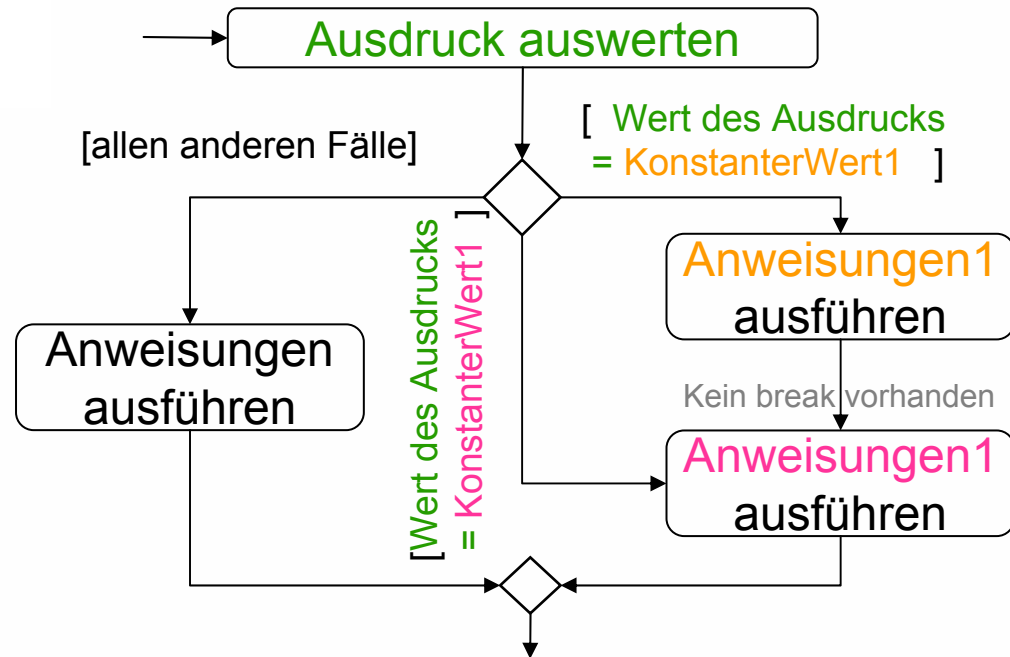
```
int faktor;  
...  
String metrisch = "";  
if (faktor == 1) {  
    metrisch = "k";  
} else if (faktor == 2) {  
    metrisch = "M";  
} else if (faktor == 3) {  
    metrisch = "G";  
}
```

Java / switch-Anweisung

```

switch ( Ausdruck ) {
  case KonstanterWert1_:
    Anweisungen1;
  case KonstanterWert2_:
    Anweisungen2;
  break;
  default:
    Anweisungen
}

```



- **break** (optional): Umgebender Block wird direkt verlassen (switch-Block)
- **default** (optional): wird ausgeführt, wenn kein „case“ zutrifft, muss nicht am Ende sein
- Ausdruck muss Wert mit Ergebnistyp char, byte, short oder int sein.
- Anweisungen kann auch leer sei
- KonstanterWert muss Ausdruck mit konstantem Wert und zuweisungskompatible zu Typ von Ausdruck sein. Jeder der konstanten Werte muss innerhalb eines switch verschieden sein.
- Block { ... } kann leer sein.



Java / switch-Anweisung

Korrekt

```
int x = 12;
switch (7) {
}

switch (x) {
  default:
  case 7: print(7);
}

switch (x - 8) {
  case 7+5: print(12);
}
```

Falsch (Compilerfehler)

```
int x = 12;
switch (x == 12) {
}

switch (x) {
  default:
  case 7: print(7);
  default: print(0);
}

switch (x - 8) {
  case 12+x: print(7+5);
  case 7+5: print(12);
}
```



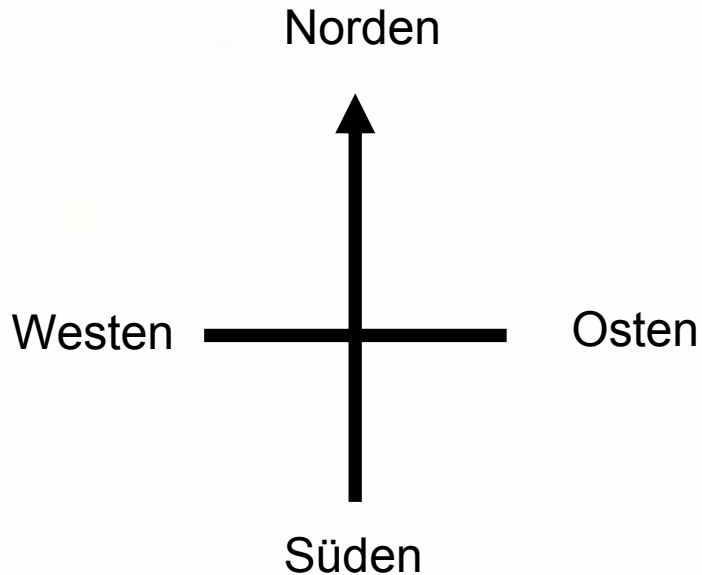
Java / switch-Anweisung

```
int wert;  
  
switch ( wert ) {  
    case 1:  
        System.out.println("1");  
    case 2:  
        System.out.println("2");  
        break;  
    case 3:  
        System.out.println("3");  
    default:  
        System.out.println("?");  
}
```

wert	Ausgabe
1	1
2	2
3	3
4	?
5	?



Java / switch-Anweisung



Himmelsrichtung

-NORDEN : byte = 1;

-WESTEN : byte = 2

-SUEDEN : byte = 3;

-OSTEN : byte = 4;

-himmelsrichtung : byte

+getName() : String



Java / switch-Anweisung

```

public class Himmelsrichtung {

    // Eigenschaften und Konstanten

    public String getName() {
        String name = "??";

        switch (himmelsrichtung) {
            case NORDEN: name = "Norden";
                        break;
            case WESTEN: name = "Westen";
                        break;
            case SUEDEN: name = "Süden";
                        break;
            case OSTEN: name = "Osten";
                        break;
        }

        return name;
    }
}

```

Himmelsrichtung

-NORDEN : byte = 1;

-WESTEN : byte = 2

-SUEDEN : byte = 3;

-OSTEN : byte = 4;

-himmelsrichtung : byte

+getName() : String

default braucht es nicht:
name = "??" hat selben Effekt



Java / switch-Anweisung

```

public class Himmelsrichtung {

    // Eigenschaften und Konstanten

    public String getName() {
        switch (himmelsrichtung) {
            case NORDEN: return "Norden";
            case WESTEN: return "Westen";
            case SUEDEN: return "Süden";
            case OSTEN:  return "Osten";
            default:    return "??";
        }
    }
}

```

Himmelsrichtung

-NORDEN : byte = 1;

-WESTEN : byte = 2

-SUEDEN : byte = 3;

-OSTEN : byte = 4;

-himmelsrichtung : byte

+getName() : String

Java / switch-Anweisung

- Switch vermeiden
 - Kann oft durch eine Tabelle ersetzt werden, wenn die betrachteten Wert aufeinander folgen

```
public class Himmelsrichtung {  
  
    private static String himmelsrichtungsNamen =  
        {"Norden", "Westen", "Süden", "Osten"};  
  
    public String getName() {  
        if (1 <= himmelsrichtung && himmelsrichtung <= 4) {  
            return himmelsrichtungsNamen[himmelsrichtung - 1];  
        } else {  
            return "??";  
        }  
    }  
}
```

- Vorteile einer Tabelle (in diesem Beispiel):
 - Sprachtexte könnten mehrsprachig sein (2-dim Feld)
 - Sprachtexte können in Datei ausgelagert sein
 - Kann schneller sein

Java / switch-Anweisung

```
public class Datum {
    int tag,
        monat,
        jahr;
    private boolean korrekteAnzahlTage() {
        switch (monat) {
            case 1:
            case 3:
            case 5:
            case 7:
            case 8:
            case 10:
            case 12: return (1 <= tag && tag <= 31);
            case 2: return (isSchaltjahr() && 1 <= tag && tag <= 29)
                || (! isSchaltjahr() && 1 <= tag && tag <= 28);
            case 4:
            case 6:
            case 9:
            case 11: return (1 <= tag && tag <= 30);
        }
    }
}
```



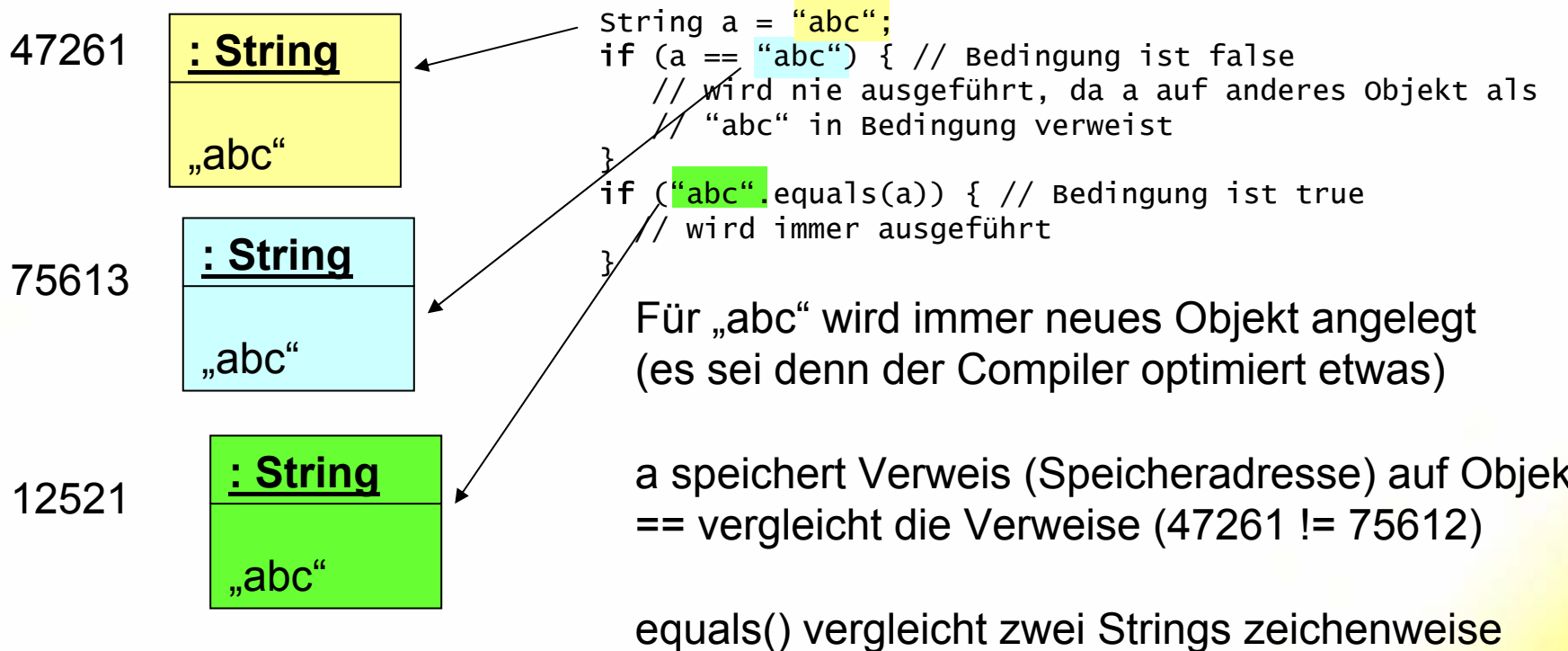
Inhalt

- Algorithmen
 - Euklidischer Algorithmus
 - do-while
- Bedingte Anweisung
 - switch
 - **String**
 - Beispiel Implementierung endlicher Automat

Strings

- Vergleich der Inhalte zweier String (oder anderer Objekt) immer mit „boolean equals(Object o)“-Methode, nie mit Identität ==
- equals() kann von jeder Klasse implementiert werden

Anfangsadresse im Speicher





Strings

```
String listeVonNamen = "Tom, Franzi, Michael,Barbara";

// split(String regexp) spaltet String in Felder von
// Teilstrings auf, die von regexp separiert werden
// regexp ist ein regulärer Ausdruck

String namen [] = listeVonNamen.split(",");
// namen[0] ist „Tom“, namen[1] ist „ Franzi“,
// namen[2] ist „ Michael“, namen[3] ist „Barbara“

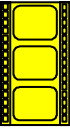
// trim() entfernt führende und anhängende Leerzeichen
String zweiterName = namen[1].trim();
// zweiterName ist „Franzi“ (ohne führendes Leerzeichen)

// spaltet bei , mit beliebig vielen Leerzeichen auf
namen = listeVonNamen.split(", *");
// namen[0] ist „Tom“, namen[1] ist „Franzi“,
// namen[2] ist „Michael“, namen[3] ist „Barbara“
char c = zweiterName.charAt(1);
// c hat wert ´r´
```



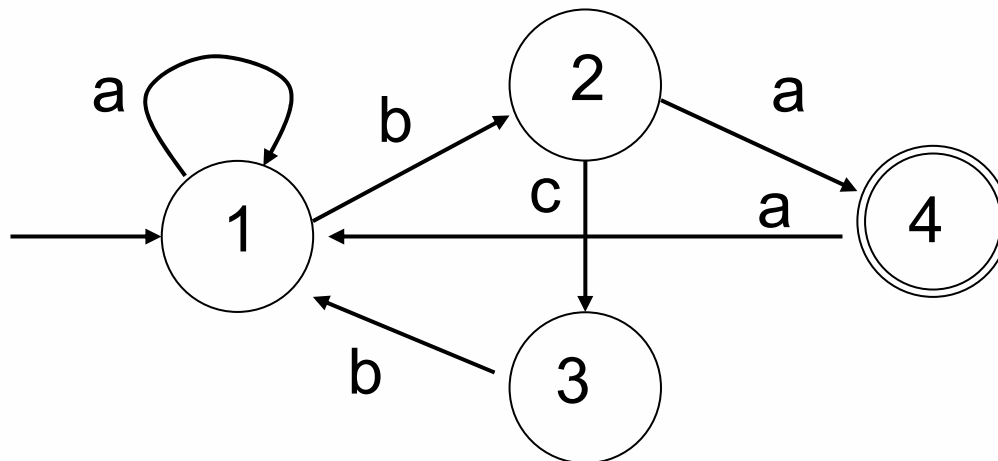
Inhalt

- Algorithmen
 - Euklidischer Algorithmus
 - do-while
- Bedingte Anweisung
 - switch
 - char, String
 - Beispiel Implementierung endlicher Automat
 - [Vor- /Nachteile if-switch](#)



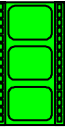
Java / switch-Anweisung

- Gegeben:
 - Deterministischer endlicher Automat mit Zuständen (1, ..., n), Anfangszustand, Endzuständen und Zustandsübergängen
- Gesucht:
 - Ein Programm, das diesen endlichen Automat implementiert
 - Das heisst, eine gegebene Zeichenkette überprüft, ob es von diesen Automaten akzeptiert wird oder nicht



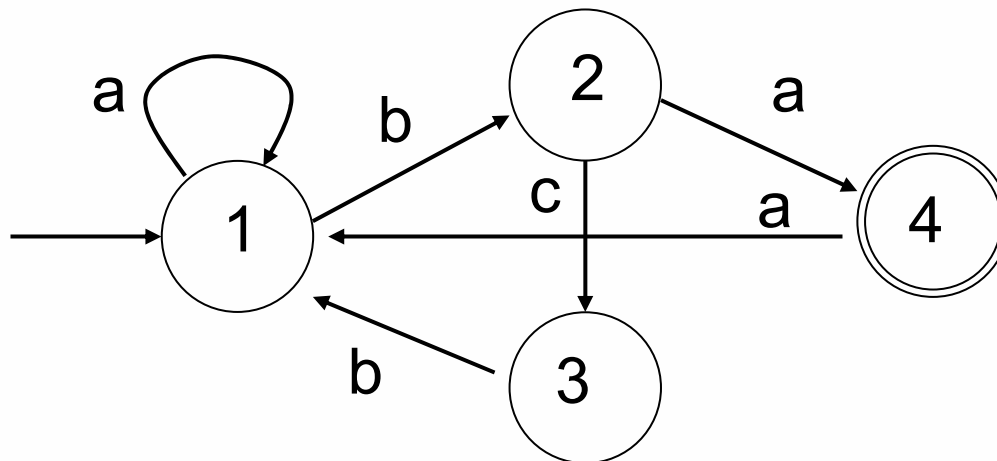
aabcbaba

aaabbbcbba



Java / switch-Anweisung

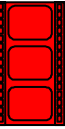
- Gegeben:
 - Deterministischer endlicher Automat mit Zuständen (1, ..., n), Anfangszustand, Endzuständen und Zustandsübergängen
- Gesucht:
 - Ein Programm, das diesen endlichen Automat implementiert
 - Das heisst, eine gegebene Zeichenkette überprüft, ob es von diesen Automaten akzeptiert wird oder nicht



aabc**ba**

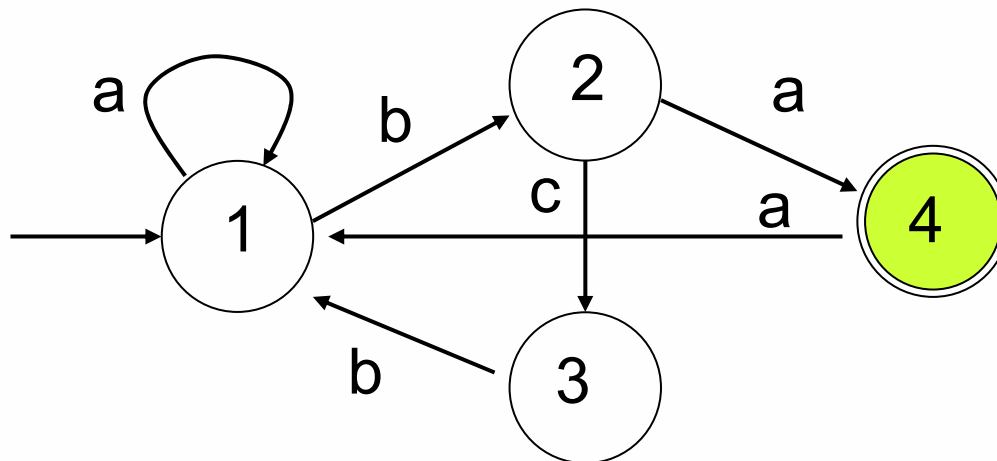
↑↑↑↑↑↑↑↑

aaabbbcbba



Java / switch-Anweisung

- Gegeben:
 - Deterministischer endlicher Automat mit Zuständen $(1, \dots, n)$, Anfangszustand, Endzuständen und Zustandsübergängen
- Gesucht:
 - Ein Programm, das diesen endlichen Automat implementiert
 - Das heisst, eine gegebene Zeichenkette überprüft, ob es von diesen Automaten akzeptiert wird oder nicht



aabcbaba



aaabbbcbba



Java / switch-Anweisung

- Lösung:
 - Merke aktuell zu prüfendes (einzulesendes) Zeichen
 - Merke aktuellen Zustand (Zahl)
 - Implementiere Zustandübergang (Einlesen) als Bedingte Anweisung
 - Wenn Zeichen gleich „a“ und Zustand gleich 2, dann ist Folgezustand 1
 - Falls kein Übergang existiert, dann wird Zeichenkette nicht vom Automaten akzeptiert
 - Wiederhole solange, bis alle Zeichen eingelesen sind
 - Wenn Endzustand erreicht ist, wird das Zeichen akzeptiert

Java / switch-Anweisung

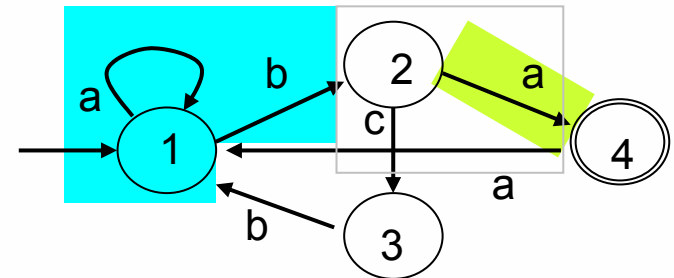
```

boolean akzeptor(String eingabe)
{
    int zustand = 1;

    for (int i = 0; i < eingabe.length(); i++) {
        char c = eingabe.charAt(i);
        if (zustand == 1) {
            if (c == 'b') zustand = 2;
            else if (c != 'a') return false;
        } else if (zustand == 2) {
            if (c == 'a') zustand = 4;
            else if (c == 'c') zustand = 3;
            else return false;
        } else if (zustand == 3) {
            if (c == 'b') zustand = 1;
            else return false;
        } else if (zustand == 4) {
            if (c == 'a') zustand = 1;
            else return false;
        }
    }

    return zustand == 4;
}

```

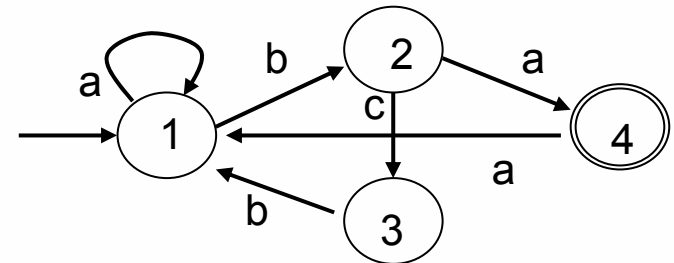


Java / switch-Anweisung

```

for (int i = 0 ; i < eingabe.length(); i++ ) {
  char c = eingabe.charAt(i);
  switch (zustand) {
    case 1: // if( zustand == 1)
      if (c == 'b') zustand = 2;
      else if (c != 'a') return false;
      break;
    case 2: // if (zustand == 2)
      if (c == 'a') zustand = 4;
      else if (c == 'c') zustand = 3;
      else return false;
      break;
    case 3: // if (zustand == 3)
      if (c == 'b') zustand = 1;
      else return false;
      break;
    case 4: // if (zustand == 4)
      if (c == 'a') zustand = 1;
      else return false;
      break;
    default:
      return false;
  }
}

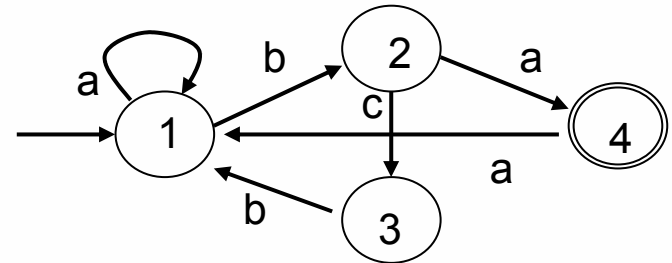
```

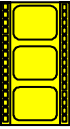


```

switch (zustand) {
  case 1:
    switch (c) {
      case 'b': zustand = 2; break; // if ( c == 'b' )
      case 'a': break;
      default: return false;
    }; break;
  case 2:
    switch (c) {
      case 'a': zustand = 4; break;
      case 'c': zustand = 3; break;
      default: return false;
    }; break;
  case 3:
    switch (c) {
      case 'b': zustand = 1; break;
      default: return false;
    }; break;
  case 4:
    switch (c) {
      case 'a': zustand = 1;
      default: return false;
    }; break;
  default:
    return false;
}

```

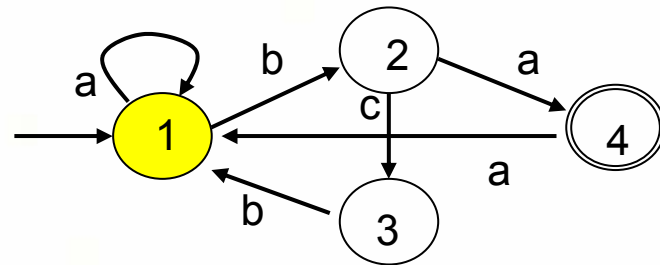




```

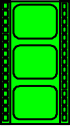
for (int i=0; i<eingabe.length(); i++) {
char c = eingabe.charAt(i);
switch (zustand) {
case 1:
switch (c) {
case 'b': zustand = 2; break;
case 'a': break;
default: return false;
}; break;
case 2:
switch (c) {
case 'a': zustand = 4; break;
case 'c': zustand = 3; break;
default: return false;
}; break;
case 3:
switch (c) {
case 'b': zustand = 1; break;
default: return false;
}; break;
case 4:
switch (c) {
case 'a': zustand = 1;
default: return false;
}; break;
default:
return false;
}
}

```



aabcbaba

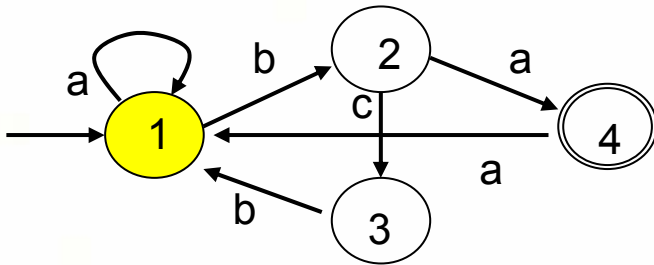
↑
c



```

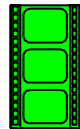
→ for (int i=0; i<eingabe.length(); i++) {
→   char c = eingabe.charAt(i);
→   switch (zustand) {
→     case 1:
      switch (c) {
        case 'b': zustand = 2; break;
        case 'a': break;
        default: return false;
      }; break;
    case 2:
      switch (c) {
        case 'a': zustand = 4; break;
        case 'c': zustand = 3; break;
        default: return false;
      }; break;
    case 3:
      switch (c) {
        case 'b': zustand = 1; break;
        default: return false;
      }; break;
    case 4:
      switch (c) {
        case 'a': zustand = 1;
        default: return false;
      }; break;
    default:
      return false;
  }
}

```



aabcbaba

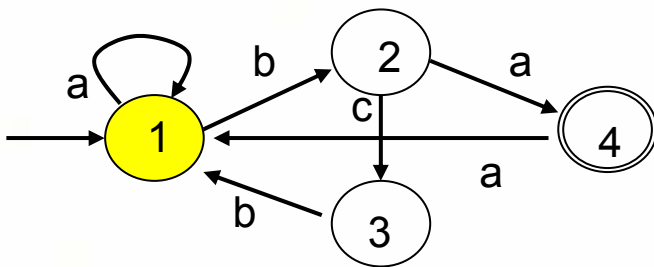
↑
c



```

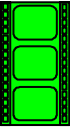
→ for (int i=0; i<eingabe.length(); i++) {
→   char c = eingabe.charAt(i);
→   switch (zustand) {
→     case 1:
      switch (c) {
        case 'b': zustand = 2; break;
        case 'a': break;
        default: return false;
      }; break;
    case 2:
      switch (c) {
        case 'a': zustand = 4; break;
        case 'c': zustand = 3; break;
        default: return false;
      }; break;
    case 3:
      switch (c) {
        case 'b': zustand = 1; break;
        default: return false;
      }; break;
    case 4:
      switch (c) {
        case 'a': zustand = 1;
        default: return false;
      }; break;
    default:
      return false;
  }
}

```



aabcbaba

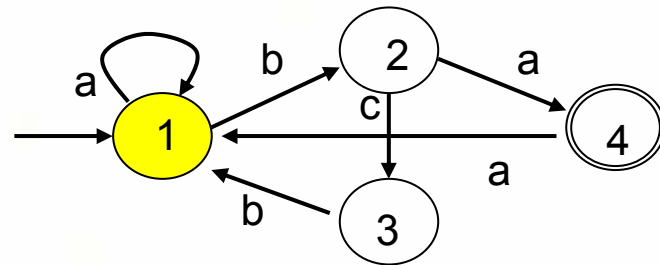




```

→ for (int i=0; i<eingabe.length(); i++) {
→   char c = eingabe.charAt(i);
→   switch (zustand) {
→     case 1:
      switch (c) {
→       case 'b': zustand = 2; break;
        case 'a': break;
        default: return false;
      }; break;
      case 2:
      switch (c) {
        case 'a': zustand = 4; break;
        case 'c': zustand = 3; break;
        default: return false;
      }; break;
      case 3:
      switch (c) {
        case 'b': zustand = 1; break;
        default: return false;
      }; break;
      case 4:
      switch (c) {
        case 'a': zustand = 1;
        default: return false;
      }; break;
      default:
        return false;
    }
  }
}

```



aabcbaba





Vor- / Nachteile if - switch

- Viele if-else-if-else
 - Es müssen im schlimmsten Fall alle Bedingungen ausgewertet werden (sequentielle Suche)
 - Unübersichtlich bei vielen *gleichförmigen* Bedingungen
 - Bedingungen, die am häufigsten erfüllt sind, nach vorne
- switch (je nach Compiler)
 - Bedingung wird nur einmal ausgewertet
 - Manchmal übersichtlicher
 - Direkte Ausführung zugehöriger „case“-Anweisung (durch „Sprungtabelle“)
 - Sprungtabelle kostet Speicher (z.B. short, 16 KB mal Wortgröße)
 - Reduzierte Sprungtabellen mit effizienter Suche des Sprungziels (Binärsuche)
- switch wird eher selten verwendet
 - bei Konvertierung von Codes in andere Codes (könnte aber durch Wertetabelle ersetzt werden)