7. Methoden

Java-Beispiele:

Fakultaet.java
Zeitplan.java

Schwerpunkte

- Abstraktionen in der SW-Entwicklung
- Wesen der 'Methode':
  algorithmische (prozedurale) Abstraktion
- Methodendeklaration - Methodenaufruf
- Parameterübergabe: als Wert - als Referenz
- Rückgabewerte (+ void)
- Lokale Variablen – globale Variablen
- Lebensdauer - Gültigkeitsbereich
- Sichtbarkeit: private - public
- Bisher umfangreichstes Beispiel:
  Terminkalender (Zeitplan)

Grundproblem der SW-Entwicklung: Komplexität

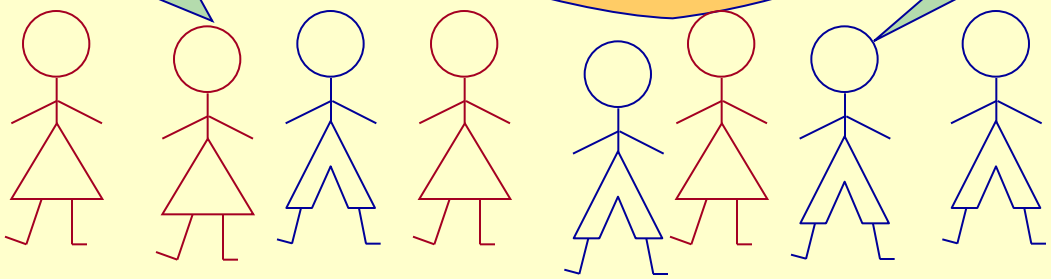
Software kann sehr komplex sein...

Software

Was tun?

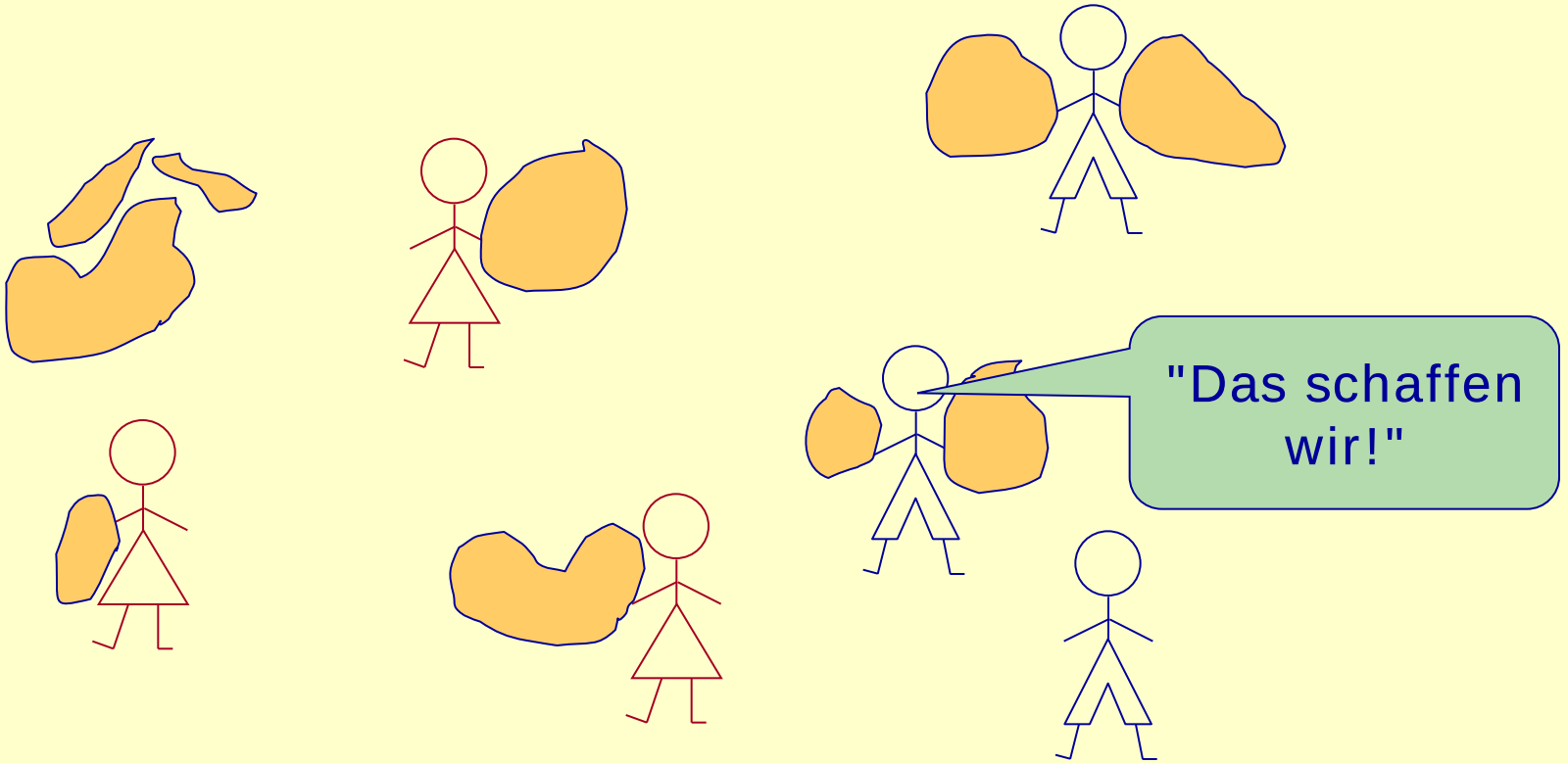
Das verstehen wir nie!
(Wartungsingenieure)

Das schaffen wir nie!
(Entwickler)



Software kann sehr komplex sein...

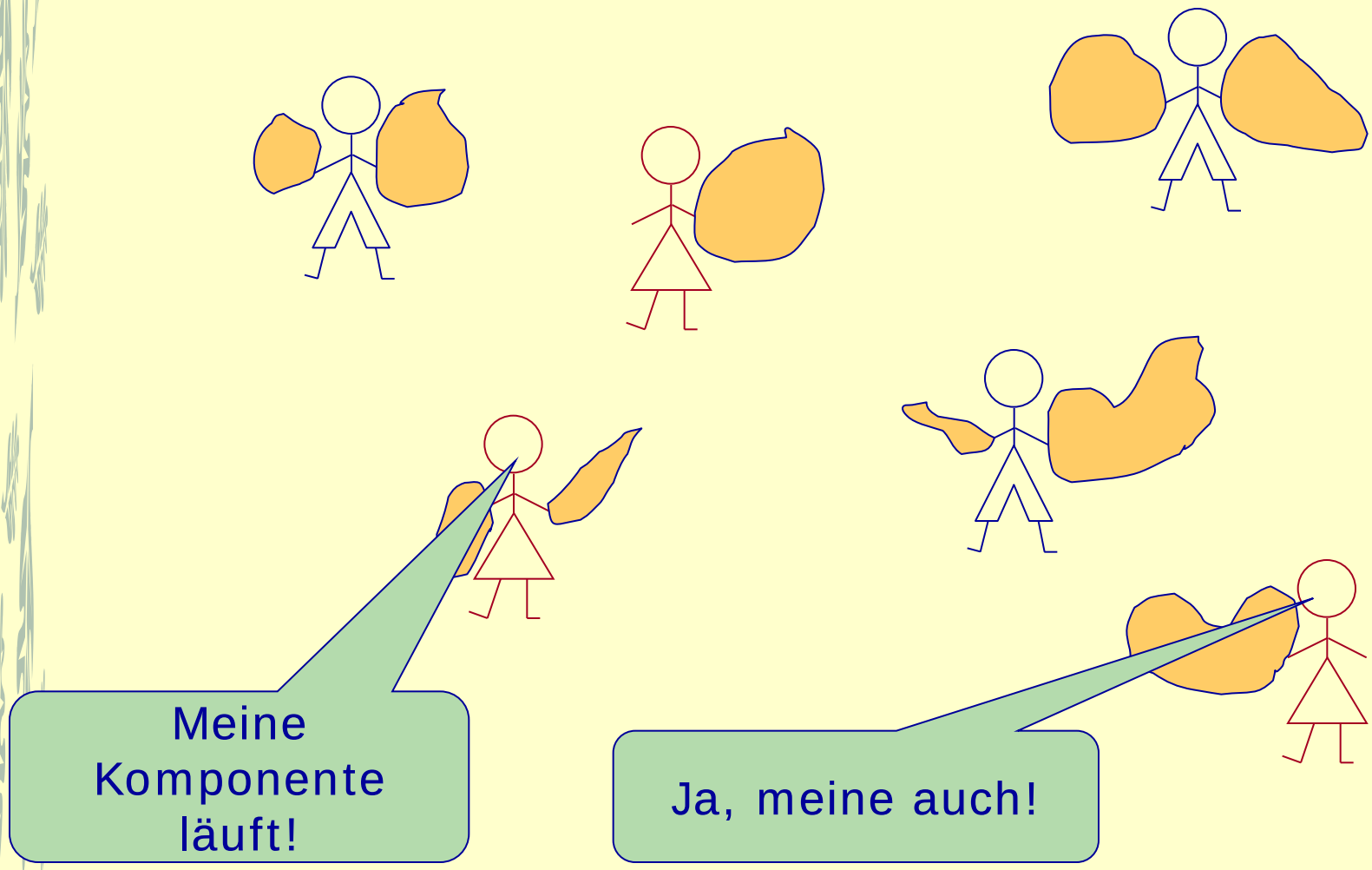
➔ Zerlegung in Komponenten



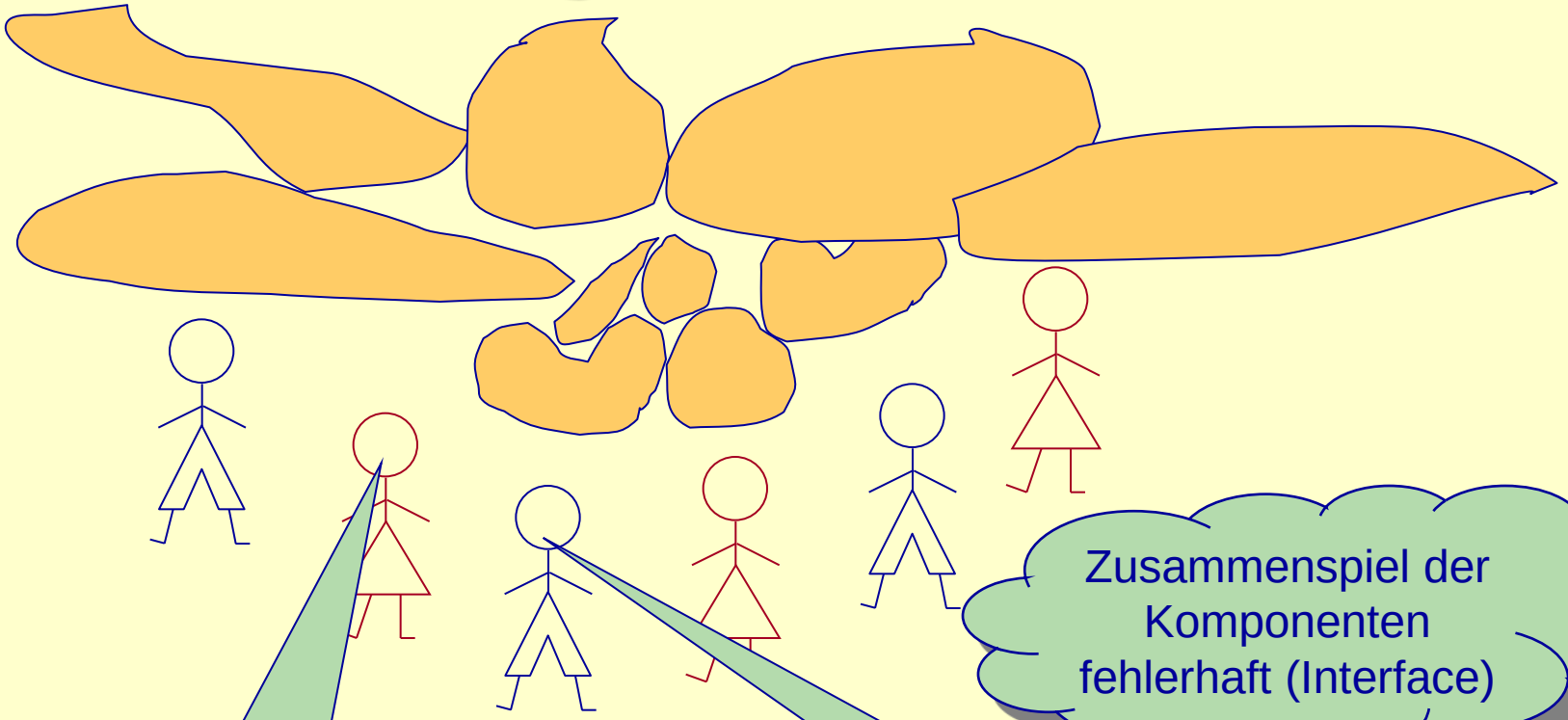
➔ teile und herrsche . . .

Dekomposition

Ein Jahr später ...
→ "Wir sind fertig"



"Irgendwie sieht es nicht so aus wie gedacht"



Zusammenspiel der Komponenten fehlerhaft (Interface)

Ja, aber an mir liegt es nicht!

Meine Komponente ist korrekt!

Abstraktion

Komplexität beherrschen durch:

- **Dekomposition**
- **Abstraktion**

Dekomposition in der SW-Entwicklung

"Softwaresysteme ... Man kann sie weder im vorhinein, beim Entwurf, noch im nachhinein, im Betrieb ... vollständig verstehen" (Denert)

Beherrschung komplexer SW: Dekomposition

→ Zerlegung der SW in Komponenten

→ jede Komponente: einzeln beherrschbar

→ Komponenten:

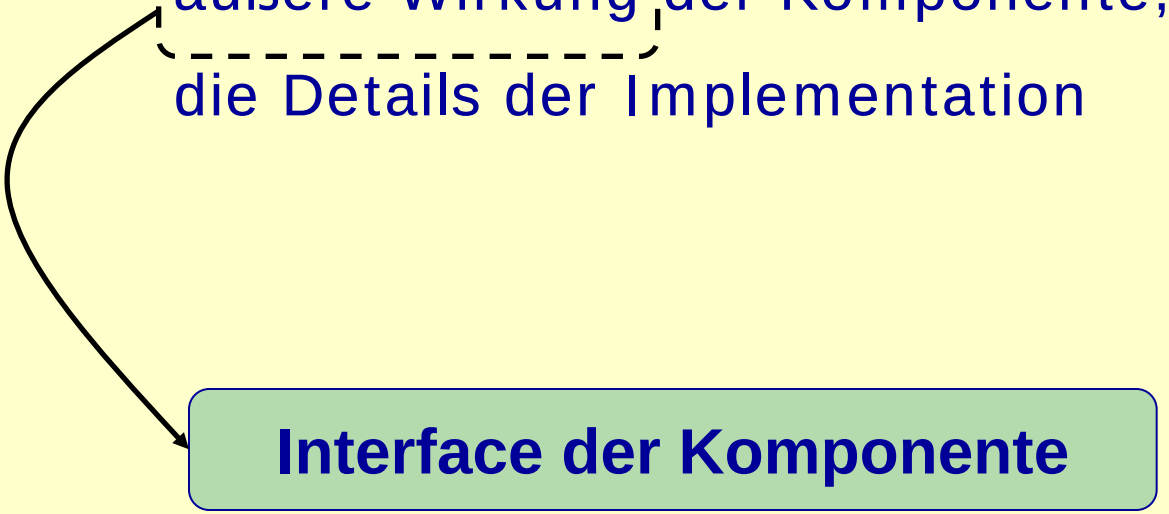
- unabhängig vom Restsystem entwickelt
- werden semantisch entworfen:

  Komponenten entsprechen Teilaufgaben

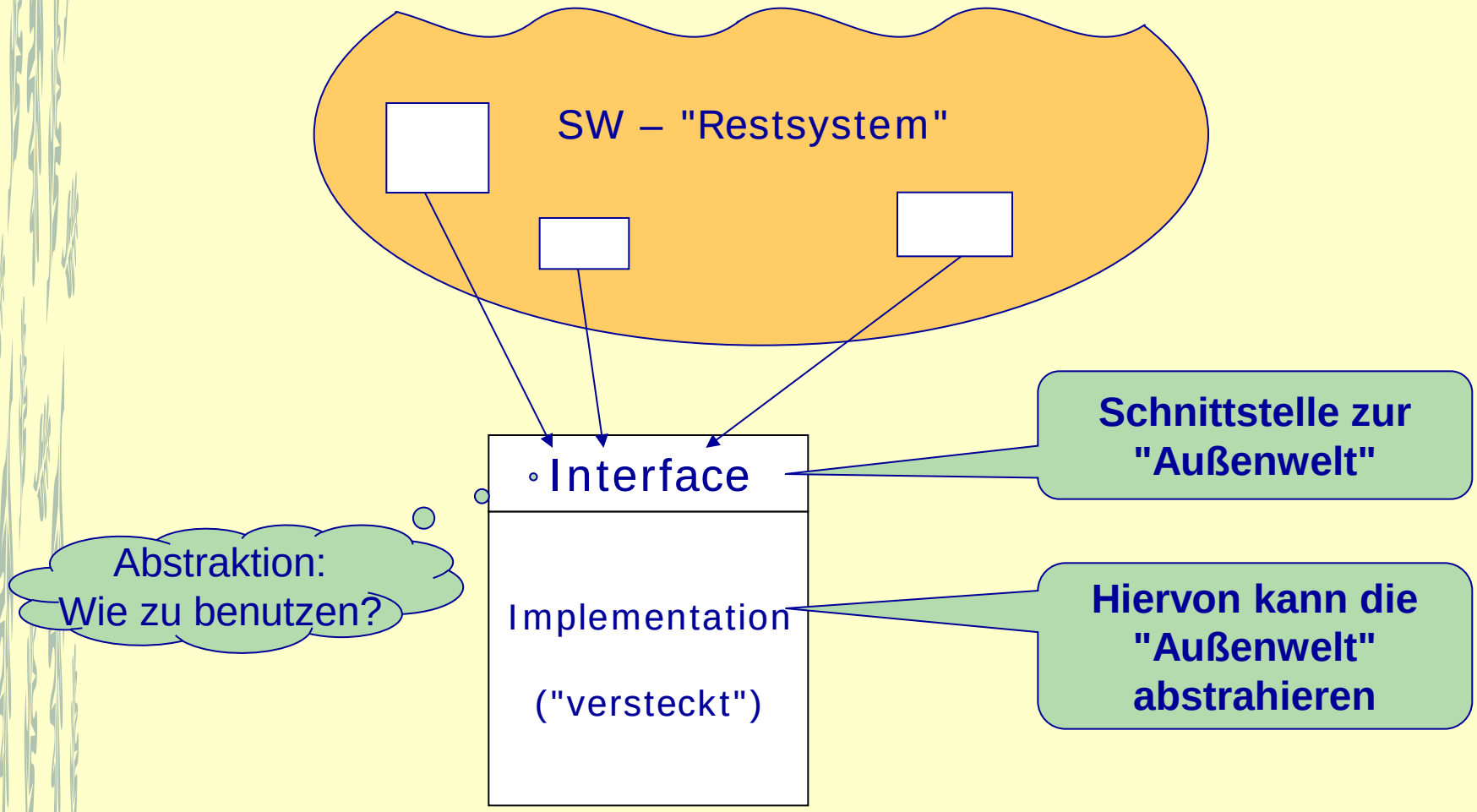
Abstraktionen in der SW-Entwicklung

Komponenten stellen Abstraktionen dar:

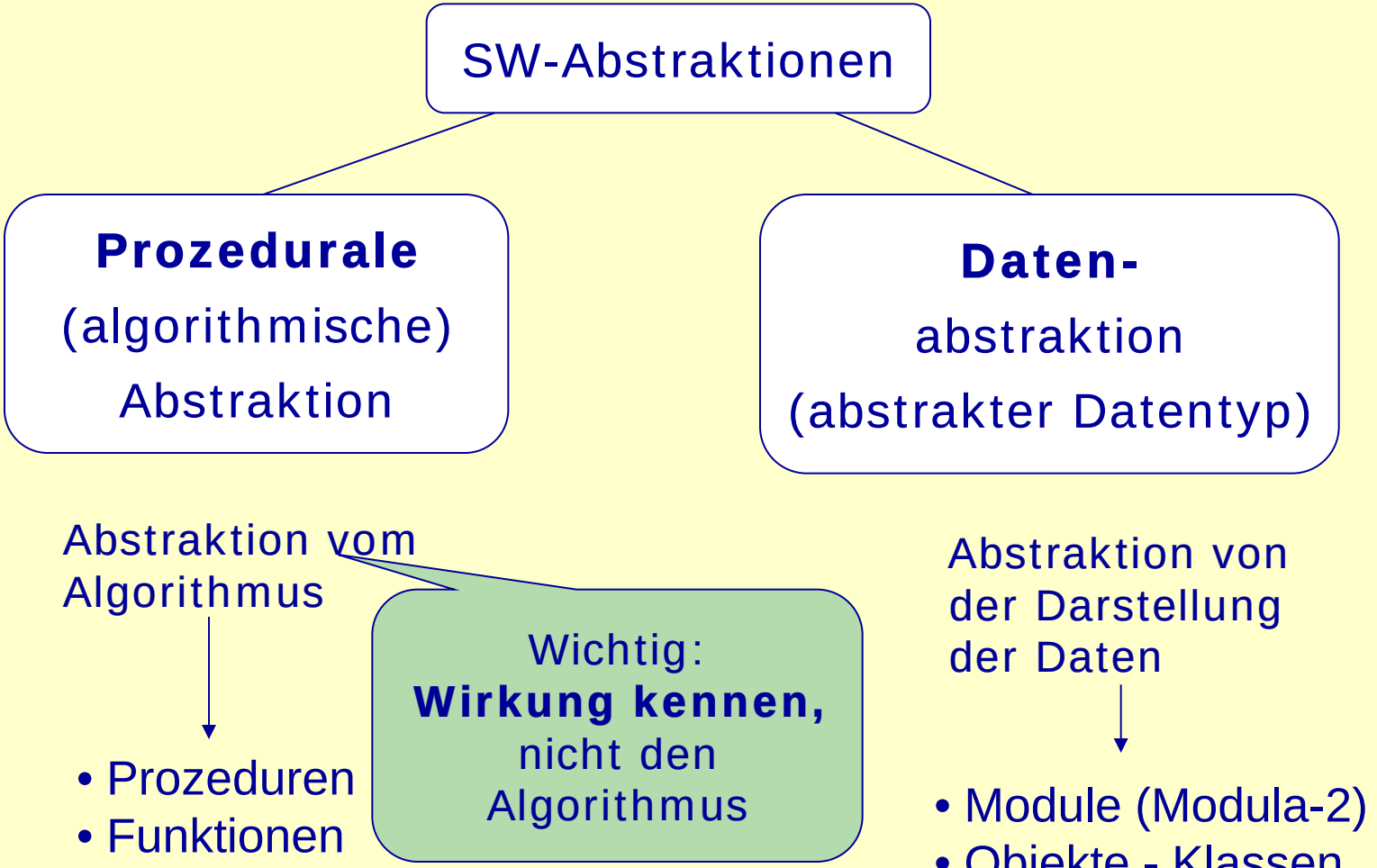
Das restliche SW-System kennt nur die äußere Wirkung der Komponente, nicht aber die Details der Implementation



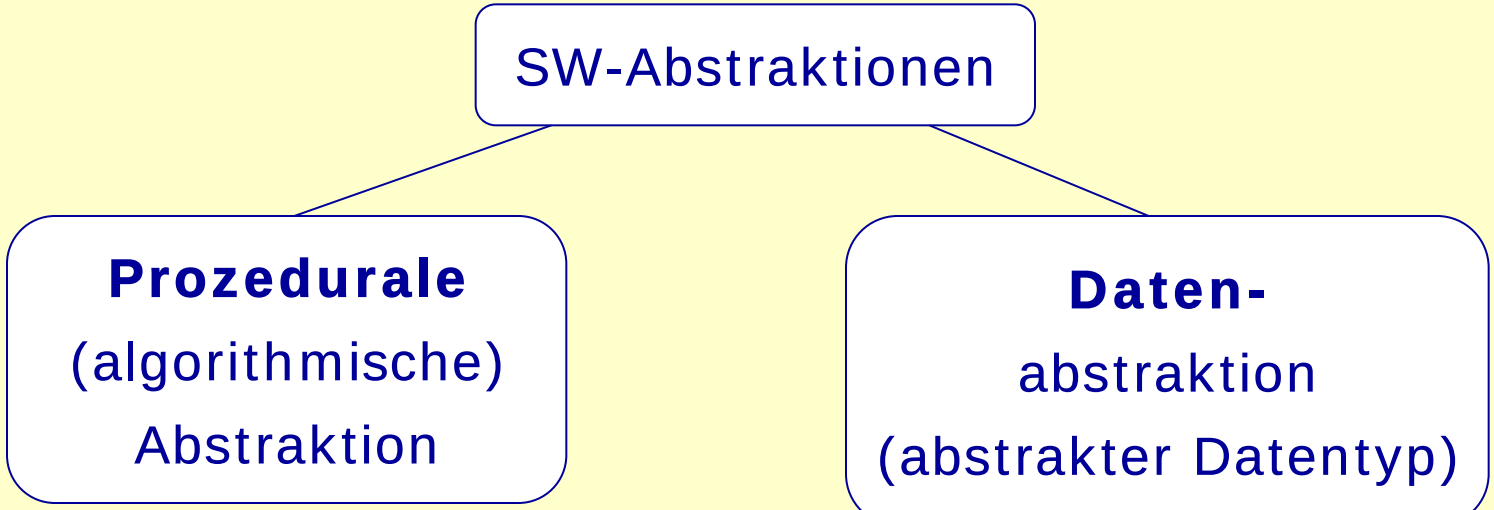
Komponente als Abstraktion: Interface + Implementation



Grundformen von SW-Abstraktionen



Grundformen von SW-Abstraktionen



Abstraktion vom Algorithmus

- Prozeduren
- Funktionen
- Methoden

Wichtig:
Wirkung kennen,
nicht den Algorithmus

Beispiel:
Wirkung: GGT
(größter gemeinsamer Teiler)
Mögliche Algorithmen:
1. Euklidischer Algorithmus
2. Mit Primfaktorenzerlegung
3. Sonstige (z.B. Probieren)

z.B. GGT(9,6) = 3

Imperative Programmierung: orientiert auf Beschreibung von Algorithmen

Pascal:

```
PROGRAM Alles ;
VAR x, y, z: real;
BEGIN
   readln (x);
   writeln (y, z);
END .
```

Eingaben

Zerlegen

Algorithmus:

100 000 Zeilen

Ausgaben

Imperative Programmierung: orientiert auf Beschreibung von Algorithmen

Java:

```
class Alles {
  public static void main (...) {

    x = Keyboard.readDouble();

    Algorithmus:

         100 000 Zeilen

    System.out.print(y);
    System.out.println(z);
  }
}
```

Eingaben

Zerlegen

in Teilalgorithmen: Methoden

Ausgaben

Methoden:

- Methodendeklaration, Methodenaufruf
- aktueller Parameter, formaler Parameter
- Werteparameter, Referenzparameter

Imperative Programmierung:

Zerlegung der Algorithmen in Teil-Algorithmen

(prozedurale / algorithmische Abstraktion)

```
class Alles {
```

Algorithmus → Teil-Algorithmen

```
fakultaet (int n)

ausgabe ()

main ( )

. . .
```

```
}
```

Prozedurale Abstraktion: Methoden / Prozeduren

"Restsystem":

```
x = fakultaet (10) ;
```

Fakultaet.java

```
public static int fakultaet (int n) {
```

Interface

```
    int x , fak = 1 ;

    for (x = 1; x <= n ; x++)
        fak = fak * x;
    return fak ;
}
```

Implementation

Komponente: Methode Fakultät "Restprogramm": main( )

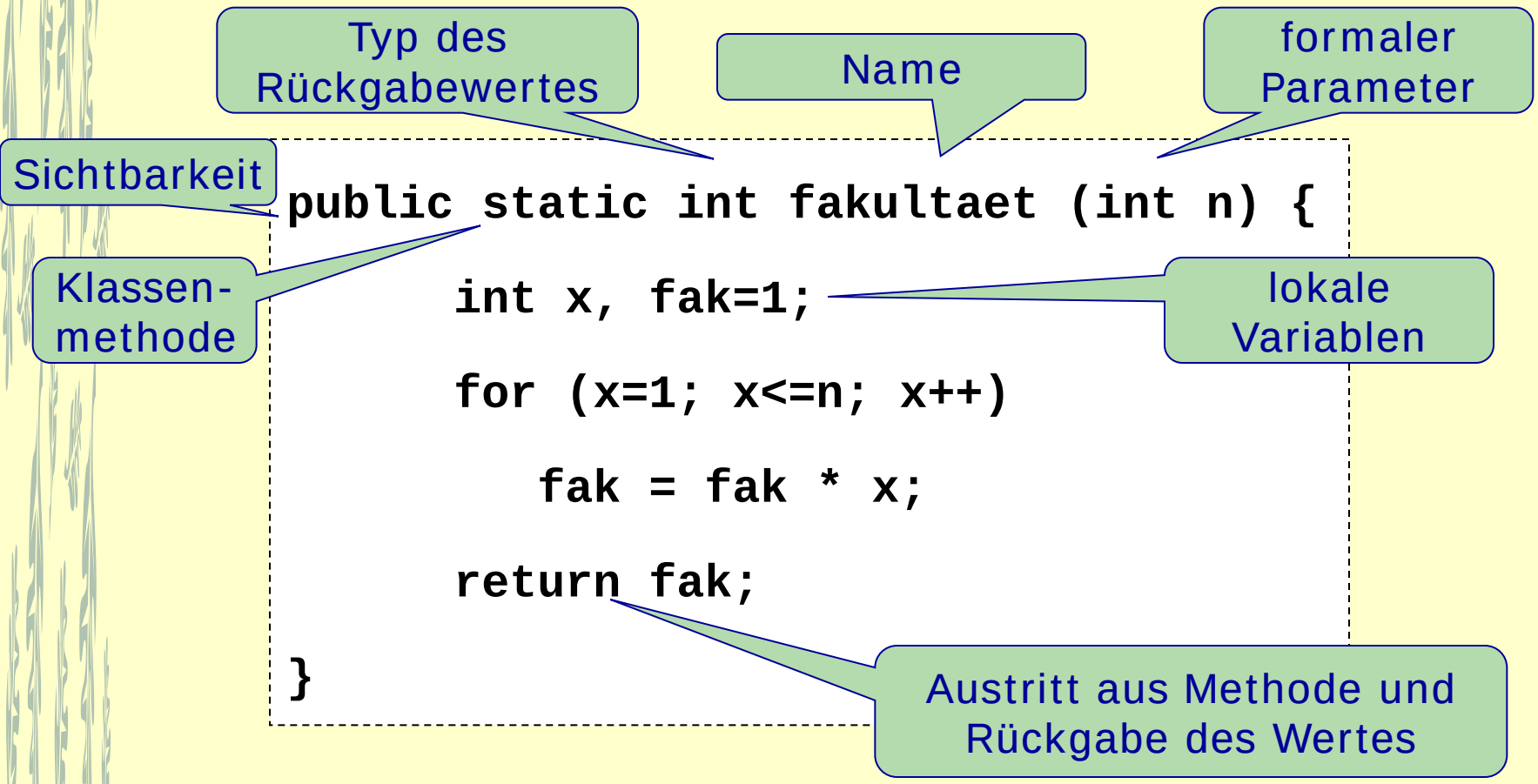
Fakultaet.java

```
class Fakultaet {

   public static int fakultaet (int n) {
       int x, fak = 1;

       for (x=1; x<=n; x++)
          fak = fak * x;
       return fak;
   }

   public static void main (String[] args) {
       int x, y;

       x = fakultaet(3);
       System.out.println("3! = " + x);
       y = fakultaet(5);
       System.out.println("5! = " + y);
   }
}
```

Methodendeklaration:
Beschreibung des Algorithmus

Methodenaufruf:
Aktivierung des Algorithmus

Methodendeklaration: Beschreibung des Algorithmus




```
public static int fakultaet (int n) {
        int x, fak=1;
        for (x=1; x<=n; x++)
           fak = fak * x;
        return fak;
}
```

Methodenaufruf: Aktivierung des Algorithmus

1. Fall: Wert wird zurückgegeben;
mit Zieltyp (z. B. int)
→ als Ausdruck
→ rechts von Zuweisung

```
x = fakultaet(3);

...

...

System.out.println("Hello!");
```

aktueller Parameter

2. Fall:
ohne Zieltyp (= void)
→ als Anweisung

Parameterübergabe: aktueller -> formaler Parameter

Aktueller Parameter → formaler Parameter

```
class Fakultaet {

   public static int fakultaet (int n) {
       int x, fak = 1;

       for (x=1; x<=n; x++)
          fak = fak * x;
       return fak;
   }

   public static void main (String[] args) {
       int x, y;

       x = fakultaet(3);
       System.out.println("3! = " + x);
       y = fakultaet(5);
       System.out.println("5! = " + y);
   }
}
```

Fakultaet.java

Methodendeklaration:
Beschreibung des Algorithmus

Methodenaufruf:
Aktivierung des Algorithmus

Parameterübergabe: aktueller -> formaler Parameter

Pascal:

```
PROCEDURE ex (x: INTEGER; VAR y: INTEGER);
   BEGIN x := 1; y := 1 END;

a := 10; b := 10;
ex (a, b);
```

Werte von a und b?

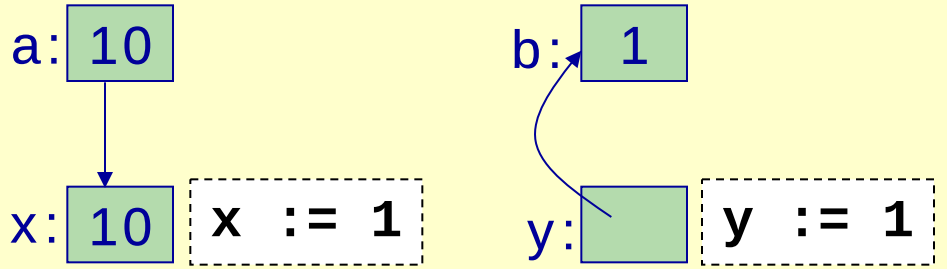
→ a = 10 b = 1

Werteparameter (x): nur Wert des aktuellen Parameters an formalen übergeben

→ Wertänderung des formalen Parameters lässt aktuellen Parameter unverändert

Referenzparameter (y): Adresse übergeben

→ Wertänderung wirkt auf aktuellen Parameter



auch: Adressparameter, Var-Parameter

Parameterübergabe: Java

```
fakultaet(x);
System.out.println("Grad C");
```

- Grundsätzlich: nur Werteparameter
  (insb. alle elementaren Typen, String)

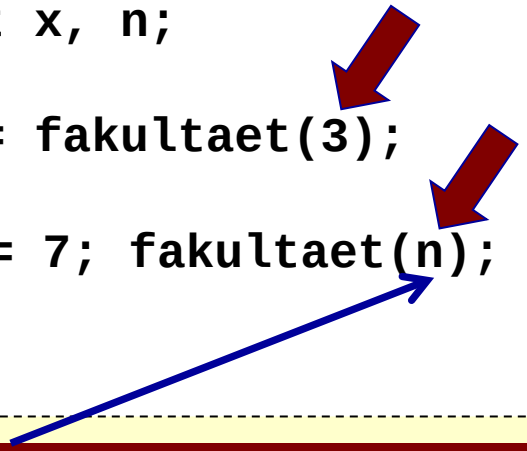
- Objekte (insb. Felder / Arrays):
  Referenzparameter

Also: Art der Parameterübergabe in Java:

- abhängig vom Typ des Parameters
- keine Schlüsselwörter zur Unterscheidung (z. B. VAR)

Werteparameter: bei elementaren Typen

```
   public static int fakultaet (int n) {
       int x, fak = 1;

       for (x=1; x<=n; x++)
          fak = fak * x;
       return fak;
   }

   public static void main (String[] args) {
       int x, n;

       x = fakultaet(3);

       n = 7; fakultaet(n);
  }
}
```



Aktueller Werteparameter n kann sich nicht ändern

Referenzparameter: z.B. Arrays/Felder

(Vorgriff im Stoff)

Methode: Array auf 0 gesetzt

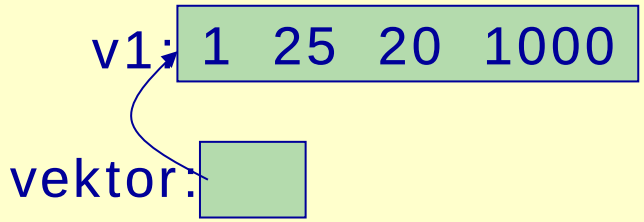
```
static void delete (int[] vektor) {

   for (int i = 0; i < vektor.length; i++) {
      vektor [i] = 0; // vektor auf 0 setzen
   }
}
```

Aufruf:

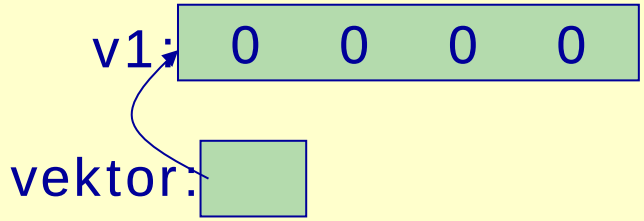
```
int [] v1 = new int[4];
v1 = {1, 25, 20, 1000}
delete(v1);
```

Aktueller Referenzparameter ändert sich, falls formaler Parameter geändert wird.

Aufruf von delete(v1):



Nach delete(v1):



Ein komplexes Java-Programm: Terminkalender

Wie findet man geeignete Teilalgorithmen?

Zeitplan.java

Nutzen des Beispiels:
- **Methodik:** Teilalgorithmen bilden bei komplexen Programmen
- Teil III: **OO-Konzepte** → dasselbe Programm noch einmal objektorientiert strukturiert (**Vergleich!**)

Programm: Terminkalender zusammenstellen

Zeitplan.java

Aufgabe:

Man verwalte eine fortlaufende Folge von Terminen durch Zusammenstellung der jeweiligen Uhrzeit, gefolgt von der Bezeichnung der dort beginnenden Veranstaltung. Mit der Eintragung der Veranstaltung wird die gesamte Zeit der Dauer der Veranstaltung reserviert, d. h. die nächste freie Zeit liegt erst danach.

Die aktuelle Zeit soll nach den englischen Konventionen ausgegeben werden (AM: Vormittagszeit bis einschließlich 11:59, PM: Nachmittagszeit, *noon* für 12 Uhr, *midnight* für 0.00 Uhr).

Die Abschlusszeit des Plans soll ausgegeben werden: Tageszeit und die dazugehörige Minute des Tages.

Beispielanwendung

Ausgabe bei Programmaufruf:

```
% java ZeitPlan
Terminkalender: Zeitangabe + Text
--------------------------------
8:30AM            V PI1
10:00AM           Pause
10:15AM           V ThI1
11:45AM           Pause
12:15PM           U PI1
letzte (aktuelle) Tageszeit in Minuten:
1:45PM = 825. Minute des Tages
```

Welche Teilalgorithmen bieten sich an?

Ausgabe bei Programmaufruf:

```
% java ZeitPlan
Terminkalender: Zeitangabe + Text
---------------------------------
8:30AM              V PI1
10:00AM             Pause
10:15AM             V ThI1
11:45AM             Pause
12:15PM             U PI1
letzte (aktuelle) Tageszeit in Minuten:
1:45PM = 825. Minute des Tages
```

Gesamtalgorithmus

Geeignete Teilalgorithmen?

Kunst der imperativen Programmierung

Kunst der imperativen Programmierung:

Geeignete Teilaufgaben (Algorithmen) finden und in Prozeduren (Methoden, Funktionen) umsetzen.

Also: Orientierung an sinnvollen Teilaufgaben, die einen Beitrag zur Lösung der Gesamtaufgabe leisten.

Achtung: Optimale Zerlegung gelingt meist nicht beim ersten Versuch.

Daten und Teilalgorithmen

Idee:

→ Verwaltung der aktuellen Zeit:
2 globale Variablen (hour, minute)

→ Methoden (Algorithmen): für Teilaufgaben

1. Die aktuelle Zeit erhöhen (addMinutes)
2. Druck der Uhrzeit nach den englischen Konventionen
3. Druck der Uhrzeit und der entsprechenden Minutenzahl
4. Die Uhrzeit in Minuten umrechnen
5. Neuen Eintrag vornehmen (Dauer, Veranstaltung)

Zuordnung der Teilaufgaben zur Ausgabe des Programms

Ausgabe bei Programmaufruf:

```
% java ZeitPlan1
Terminkalender: Zeitangabe + Text
--------------------------------
8:30AM              V PI1
10:00AM             Pause
10:15AM             V ThI1
11:45AM             Pause
12:15PM             U PI1
letzte (aktuelle) Tageszeit in Minuten:
1:45PM = 825. Minute des Tages
```

1. Die aktuelle Zeit erhöhen (addMinutes)
2. Druck der Uhrzeit nach den englischen Konventionen
3. Druck der Uhrzeit und der entsprechenden Minutenzahl
4. Die Uhrzeit in Minuten umrechnen
5. Neuen Eintrag vornehmen (Dauer, Veranstaltung)

Zeitplan-Programm: Überblick über die Methoden

ZeitPlan.java

```
class ZeitPlan {
   private static int hour, minute;

   private static void addMinutes (int m)
   private static void printTime ()
   private static void printTimeInMinutes ()
   private static int timeInMinutes ()
   private static void includeNewEntry
              (int intervalInMinutes, String event)

   public static void main (String[] args)
       ...
       includeNewEntry(90, "V PI1");
       includeNewEntry(15, "Pause");
       includeNewEntry(90, "V ThI1");
       ...
   }
}
```

1. Die aktuelle Zeit erhöhen (addMinutes)
2. Druck der Uhrzeit nach den englischen Konventionen
3. Druck der Uhrzeit und der entsprechenden Minutenzahl
4. Die Uhrzeit in Minuten umrechnen
5. Neuen Eintrag vornehmen (Dauer, Veranstaltung)

Die aktuelle Zeit erhöhen

```
private static int hour, minute;


private static void addMinutes (int m) {

    // erhoeht die aktuelle Zeit um m Minuten

    int totalMinutes = (60*hour + minute + m) % (24*60);?

    if (totalMinutes < 0)
       totalMinutes = totalMinutes + 24*60; ?

    hour = totalMinutes/60;
    minute = totalMinutes%60;
}
```

-1 % (24 *60) = -1
Uhrzeit vom Vortag: aber negativ;
also: m kann auch negativ sein,
d.h. addMinutes auch nutzbar für Subtraktion

Druck der Uhrzeit nach den englischen Konventionen

```
private static void printTime () {
    // druckt die aktuelle Zeit nach
    // englischen Konventionen: AM, PM, noon, midnight

    if ((hour == 0) && (minute == 0))
      System.out.print("midnight");
    else if ((hour == 12) && (minute == 0))
      System.out.print("noon  ");
    else {
      if      (hour == 0) System.out.print(12);
      else if (hour > 12) System.out.print(hour-12);
      else                System.out.print(hour);

      if (minute < 10) System.out.print(":0" + minute);
      else             System.out.print(":" + minute);

      if (hour < 12) System.out.print("AM");
      else           System.out.print("PM");
    }
}
```

Druck der Uhrzeit und der entsprechenden Minutenzahl

```
private static void printTimeInMinutes () {

     // druckt aktuelle Zeit mit Entsprechung in Minuten

    printTime ();
    System.out.println("   = " + timeInMinutes()
                       + ". Minute des Tages ");
  }

private static int timeInMinutes () {

    // ermittelt die Anzahl von Minuten seit 0:00 Uhr,
    // die der aktuellen Zeit entspricht

    int totalMinutes = (60*hour + minute) % (24*60);  ?

    if (totalMinutes < 0)
      totalMinutes = totalMinutes + 24*60;  ?
    return totalMinutes;
}
```

Neuen Eintrag vornehmen (Zeit, Veranstaltung)

```
private static void includeNewEntry
      (int intervalInMinutes, String event) {

   printTime();
   System.out.println("\t\t" + event);
   addMinutes(intervalInMinutes);
}
```

```
public static void main (String[] a
   hour = 8; minute = 30; //Anfangs

   System.out.println("Terminkalend
   System.out.println("-----------
   includeNewEntry(90, "V PI1");
   includeNewEntry(15, "Pause");
   includeNewEntry(90, "V ThI1");
   includeNewEntry(30, "Pause");
   includeNewEntry(90, "U PI1 ");
   System.out.println("letzte (aktuelle) Tageszeit in Minuten: ");
   printTimeInMinutes ();
}
```

```
% java ZeitPlan1
Terminkalender: Zeitangabe + Text
--------------------------------
8:30AM          V PI1
10:00AM         Pause
10:15AM         V ThI1
11:45AM         Pause
12:15PM         U PI1
letzte (aktuelle) Tageszeit in Minuten:
1:45PM = 825. Minute des Tages
```

Lebensdauer
Sichtbarkeit

Lebensdauer von Variablen

Lebensdauer:
Zeitraum der Existenz der Variablen zur Laufzeit
(d. h.: des Speicherplatzes)

Globale (static-) Variablen:
gesamte Laufzeit des Programms

Lokale Variablen:
vom Beginn bis zum Ende des Methoden-aufrufs (Wert geht dann verloren)

```
class ZeitPlan {
   private static int hour, minute;
      . . .
   private static void addMinutes (int m)   {
      int totalMinutes = . . . ;

      hour = ...;
      minute = ... ;
   }
   public static void main (String[] args) {
      addMinutes(30);
   }
}
```

Lebensdauer: Schlussfolgerungen

Globale Variablen (in Klassen):

- gemeinsamer 'Datenspeicher' mehrerer Methoden

Lokale Variablen:

- Hilfsvariablen eines Algorithmus
- Speicherplatz nur während der Abarbeitung der Methode reserviert

Sichtbarkeit von Bezeichnern

Sichtbarkeit:
Wo kann ein Bezeichner (EBNF: Identifikator) für Variablen und Methoden im Programm ***verwendet*** werden?

- **Innerhalb der Klasse:**
  - lokale Variablen nur in der Methode sichtbar
  - globale Variablen: in der ganzen Klasse sichtbar (möglich: verdeckt durch lokale Variablen gleichen Namens)

- **Programm besteht aus mehreren Klassen:**
  - private-Variablen/Methoden: n u r in ihrer Klasse
  - public-Variablen/Methoden: auch in anderen Klassen

Sichtbarkeit von Variablen

- **Innerhalb der Klasse:**
  - lokale Variablen nur in der Methode
  - globale: in der ganzen Klasse
    (möglich: verdeckt durch lokale Variablen gleichen Namens)

```
class Zeitplan {
    private static int hour, minute;
        . . .
    private static void addMinutes (int m)   {
        int totalMinutes = . . . ;
        totalMinutes = ... ;
        minute = ... ;
    }
    public static void main (String[] args) {
        minute = 30;  // nicht totalMinutes = ...
    }
}
```

Sichtbarkeit von Variablen und Methoden

- **Programm besteht aus mehreren Klassen:**
  - private-Variablen/Methoden: n u r in ihrer Klasse
  - public-Variablen/Methoden:auch in anderen Klassen

```
class Zeitplan {
   private static int hour, minute;
       . . .
   private static void addMinutes (int m)   {
       ...
   }
   public static void main (String[] args) {
       ...
   }
}
```

ZeitPlan.java – Was soll's und wie weiter?

"Programmieren lernen durch Studieren beispielhafter Programme"

- Zerlegung einer Gesamtaufgabe:
  5 algorithmische Teilaufgaben
  → 5 Methoden

- Prinzipien:
  - globale Variablen = gemeinsame Daten mehrerer Methoden
  - lokale Variablen = Hilfsvariablen eines Algorithmus
  - Konzepte:
    Lebensdauer, Sichtbarkeit, formale und aktuelle Parameter
  - Wiederverwendung in Teil III: dasselbe Beispiel nicht imperativ, sondern objektorientiert (Vergleich!)

Wie weiter mit dem Zeitplan-Programm?

- Programm aufrufen
- Programm verstehen
- Programm leicht ändern:
  - neue Funktion:
    `private static void printTimeGerman ()`
  - neue Funktion
    `private boolean morning()`
    testet, ob die aktuelle Zeit <= 12.00 Uhr
  - Bessere Formatierung: nicht

```
8:30AM          V PI1
10:00AM          Pause
```

- Interaktive Variante: Veranstaltung und Zeitbedarf werden im Dialog abgefragt und nicht in main() fest vorgegeben.