

## **3. Grundkonzepte der Objektorientierung (2):**

# **Klassenvariablen, Klassenmethoden**

Java-Beispiele:

TimeC.java

ScheduleC.java

Time.java (aus: III.2)

# Schwerpunkte

- Klassenvariablen – Instanzvariablen
- Klassenmethoden – Instanzmethoden
- 'static' - eine glückliche Begriffsbildung?

# Zur Motivation: Klasse 'Time' mit erweiterter Aufgabenstellung

bisher: Time.java (III.2)

Time.java  
Schedule.java

```
class Time {  
    private int hour, minute;  
    public Time() ...  
    public addMinutes (int m) ...  
    public printTime () ...  
    public boolean before (Time t) ...  
    ...  
}
```

ADT 'Time'

```
class Schedule {  
    public static main ... {  
        Time t1, t2, t3, t4;  
        ...  
    }  
}
```

Anwendung  
von 'Time':  
4 Terminkalender  
zusammenstellen  
→ 4 Zeitobjekte

# Aufruf der Terminplanung

```
% java Schedule
erster Plan:
8:30AM      PI1
10:00AM     Pause
10:15AM     TI1
11:45AM     Pause
noon        Ma1
1:40PM      Pause
1:55PM      Ma2
letzte (aktuelle) Tageszeit in Minuten:
2:15PM      = 855. Minute des Tages

zweiter Plan:
midnight    Nebenfach
1:40AM      Proseminar
letzte (aktuelle) Tageszeit in Minuten:
3:10AM      = 190. Minute des Tages

dritter Plan:
2:15PM      Sport
3:55PM      Freizeit
letzte (aktuelle) Tageszeit in Mi
7:15PM      = 1155. Minute des Tages

vierter Plan:
noon        zweites Nebenfach
1:40PM      frei
letzte (aktuelle) Tageszeit in Mi
5:00PM      = 1020. Minute des Tages
```

4 Pläne –  
4 Zeitobjekte

Bisher:  
Zeitangaben  
nach  
englischen  
Konventionen

- Erweiterte Aufgabe:  
Zeitangaben nach englischen oder deutschen Konventionen:
- Aber einheitlich für alle Zeitobjekte zu einem bestimmten Zeitpunkt
  - Man kann „umschalten“ zwischen D und E bzw. E und D

# Aufruf der Terminplanung

```
% java Schedule
erster Plan:
8:30AM      PI1
10:00AM     Pause
10:15AM     TI1
11:45AM     Pause
noon        Ma1
1:40PM      Pause
1:55PM      Ma2
letzte (aktuelle) Tageszeit in Minuten:
2:15PM      = 855. Minute des Tages

zweiter Plan:
midnight    Nebenfach
1:40AM      Proseminar
letzte (aktuelle) Tageszeit in Minuten:
3:10AM      = 190. Minute des Tages

dritter Plan:
14:15       Sport
15:55       Freizeit
letzte (aktuelle) Tageszeit in Mi
19:15       = 1155. Minute des Tages

vierter Plan:
noon        zweites Nebenfach
13:40       frei
letzte (aktuelle) Tageszeit in Mi
17:00       = 1020. Minute des Tages
```

4 Pläne –  
4 Zeitobjekte

Bisher:  
Zeitangaben  
nach  
englischen  
Konventionen

umschalten: 

- Erweiterte Aufgabe:  
Zeitangaben nach englischen oder deutschen Konventionen:
- Aber einheitlich für alle Zeitobjekte zu einem bestimmten Zeitpunkt
  - Man kann „umschalten“ zwischen D und E bzw. E und D

# Zur Motivation: Klasse 'Time' mit erweiterter Aufgabenstellung

bisher: Time.java (III.2)

```
class Time {
    private int hour, minute;
    public Time() ...
    public addMinutes (int m) ...
    public printTime () ...
    public boolean before (Time t) ...
}
```

```
class Time {
    ...
    printTime() {
        //druckt Zeit nach englischen Konventionen
    }
}
```

**zu ändern**



# Zur Motivation: Klasse 'Time' mit erweiterter Aufgabenstellung

bisher: Time.java (III.2)

```
class Time {  
    ...  
    printTime() {  
        //druckt Zeit nach englischen Konventionen  
    }  
}
```

neu:

- Ausgabe der Zeitangaben nach englischen (**1:40 PM**) oder deutschen (**13:40**) Konventionen
- Anwenderprogramm kann wählen: einheitliches Format für alle Zeitobjekte (vgl. Schedule: 4 Zeitpläne → 4 Zeitobjekte) zu einem bestimmten Zeitpunkt
- Anwenderprogramm kann das Format umschalten (engl. -> dt., dt. -> engl.)

Lösungsidee ?

# Erster Lösungsansatz

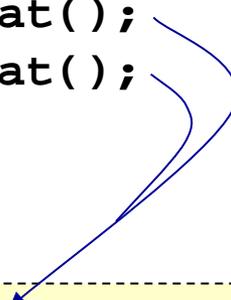
```
class Time {  
    ...  
    private boolean englishTime = true;  
  
    public void printTime () {  
        if (englishTime)  
            printEnglishTime();  
        else  
            printGermanTime();  
    }  
  
    private void printEnglishTime () ...  
  
    private void printGermanTime () ...  
  
    public void switchTimeFormat () {  
        englishTime = !englishTime;  
    }  
}
```



Aufgabe vollständig  
erfüllt?

# Probleme

```
public static void main (...) {  
    TimeC t1 = new TimeC(8,30);  
    TimeC t2 = new TimeC();  
  
    t1.printTime();  
    t2.printTime();  
    t1.switchTimeFormat();  
    t2.switchTimeFormat();  
    t1.printTime();  
    t2.printTime();  
}
```



- Einheitliches Format für alle Zeitobjekte nicht gewährleistet (hier: alle Zeitobjekte müssen einzeln umschalten)
- Grund: Instanzvariable 'englishTime' existiert mehrfach (je Objekt einmal)

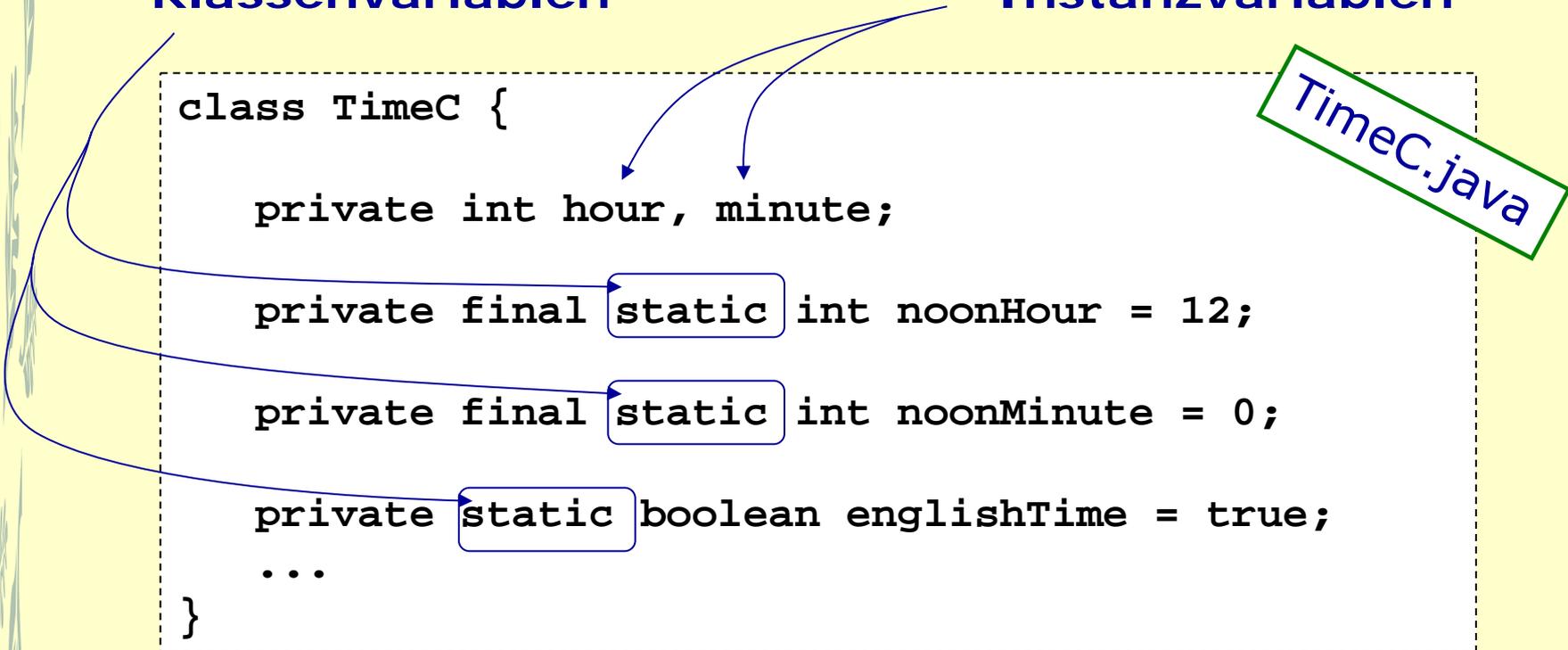
→ Variable 'englishTime' sollte nur einmal existieren: Klassenvariable

# Variablen: zwei Arten

## Klassenvariablen

## Instanzvariablen

```
class TimeC {  
    private int hour, minute;  
    private final static int noonHour = 12;  
    private final static int noonMinute = 0;  
    private static boolean englishTime = true;  
    ...  
}
```



Klassenvariablen (static):  
existieren einmal je Klasse  
→ zum Austausch von  
Informationen zwischen mehreren  
Objekten derselben Klasse

Instanzvariablen (non-static):  
für jedes Objekt angelegt  
→ beschreiben den Zustand  
der Instanz  
(des jeweiligen Objekts)

# Lebensdauer: Existenz der Variablen

```
class TimeC {  
    private int hour, minute;  
  
    private final static int noonHour = 12;  
    private final static int noonMinute = 0;  
    private static boolean englishTime = true;  
    ...  
}
```

Klassen-Variablen

Instanz-Variablen

Existenz zum Programmstart

noonHour	12
noonMinute	0
englishTime	true

der Klasse TimeC zugeordnet

Existenz mit der Erzeugung eines Objekts:

```
t1 = new TimeC(8,30);  
t2 = new TimeC(0,0);
```



den Objekten zugeordnet

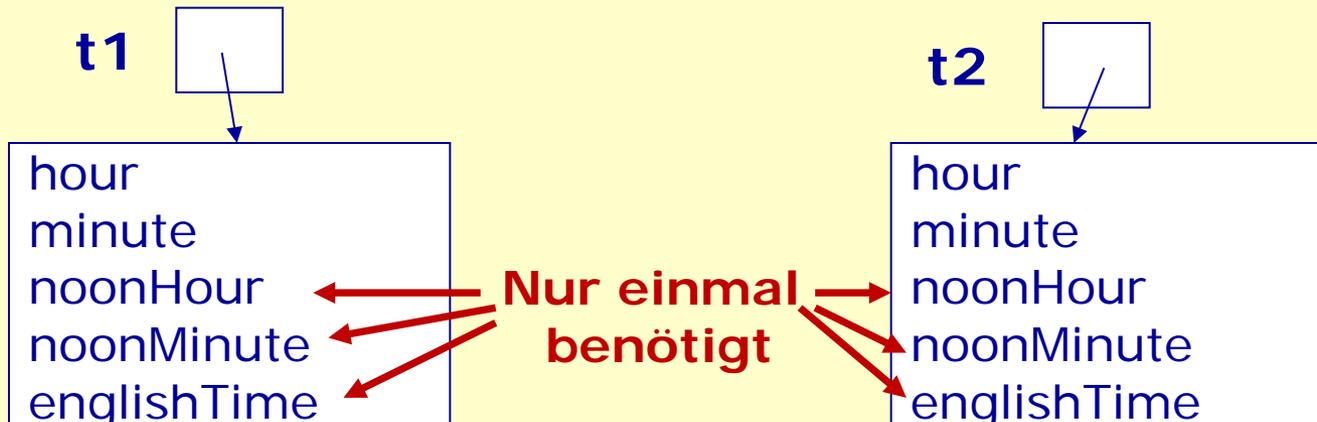
# Sinn der Klassenvariablen (1)

Gemeinsame Information aller Objekte  
nur einmal abspeichern  
(Effizienz und Programmlogik)

Zum Vergleich: alternative Lösung nur mit Instanzvariablen

```
class Time {  
    private int hour, minute;  
  
    private final int noonHour = 12;  
    private final int noonMinute = 0;  
    private boolean englishTime = true;  
}
```

ohne static



# Sinn der Klassenvariablen (2)

Imperative Programmierung realisieren:  
keine Objektbildung beabsichtigt

Daten (Variablen) werden nur einmal benötigt

```
class Zeitplan {  
  
    static int hour, minute;  
  
    static addMinutes (int m)    {  
        ...  
        hour = totalMinutes / 60;  
    }  
}
```

- Variablen hour, minute existieren vom Programmstart an
- Keine Instanzbildung von Zeitplan vorgenommen
- Programmierstil wie Pascal, C – Algorithmen im Mittelpunkt (z. B. Quicksort, Hanoi ... dort keine OO sinnvoll)

# Methoden: zwei Arten

TimeC.java

```
class TimeC {  
    ... // Variablen  
  
    public addMinutes (int m) {  
        int totalMinutes = ...  
        hour = ...;  
        minute =...;  
    }  
    public static void switchTimeFormat() {  
        englishTime = !englishTime;  
    }  
}
```

Instanzmethode

Klassenmethode

- **Instanzmethode:** an Objekt (Instanz) gebunden  
→ existiert so oft, wie Instanzen erzeugt wurden
- **Klassenmethode:** an Klasse gebunden  
→ existiert nur einmal

# Sinn der Klassenmethoden

Algorithmus (Methode) muss nur einmal existieren

```
class TimeC {  
    private int hour, minute; // Instanzvariablen  
    static boolean englishTime; // Klassenvariablen  
  
    public addMinutes (int m) {  
        int totalMinutes = (60*hour + minute + m);  
    }  
    public static void switchTimeFormat() {  
        englishTime = !englishTime;  
    }  
}
```

Darf switchTimeFormat auch  
Instanz-Methode sein?  
Falls ja: Konsequenz?

## Klassenmethoden:

können nur Klassenvariablen bearbeiten

## Instanzmethoden:

alle Variablen (Klassen- u. Instanzvar.) können bearbeitet werden

# Aufruf von Klassenmethoden: Bezug zur Klasse

```
class TimeC {  
    private int hour, minute;  
  
    public printTimeInMinutes()    {...}  
    public static void switchTimeFormat()    {...}  
}
```

```
class ScheduleC {  
    main (...)    {  
        TimeC t1 new TimeC(8,30);  
        TimeC t2 new TimeC(0,0);
```

ScheduleC.java

Instanzmethoden: an die Instanz gebunden

```
t2.printTimeInMinutes();  
TimeC.switchTimeFormat ();  
...
```

Klassenmethoden: an Klasse gebunden

# Variablen und Methoden: Erzeugung und Zuordnung

(Zusammenfassung)

```
class C {  
    int v1, v2;  
    static int vs1, vs2;  
    void m1() {...}  
    static void m2() {...}  
}
```

Existenz zum  
Programmstart  
(nur einmal)

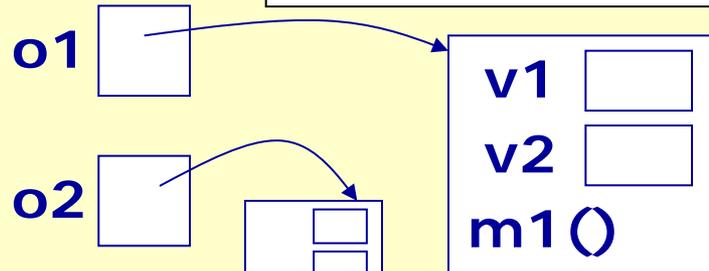
Existenz mit Erzeugung des  
Objekts  
(u.U. mehrfach)

Kann als  
Objekt  
betrachtet  
werden

vs1   
vs2   
m2()

C.vs1; C.m2();

```
o1 = new C();  
o2 = new C();
```



# 'static' - eine glückliche Begriffsbildung?

Klassenvariable: static → an Klasse gebunden

Instanzvariable: ohne static → an Instanz gebunden

Instanzvariable ohne eigenes Schlüsselwort !

Begriff 'static': stammt von C – dort in ganz anderem Zusammenhang (ohne Bezug zur OO)  
→ in Java: methodisch schlechte Wahl  
(fehleranfällig – 'static' ohne Semantik)

Smalltalk:

**Instance Variables:**

day Integer

year Integer

**Class Variables:**

MonthNames     Array of Symbol