

---

# Eiffel: Beispiel 1a

```
class
    TIME_OF_DAY
        -- Absolute time within a day
feature
    hour: INTEGER is
        -- Hour of day, 00 to 23
        do
            Result := minutes // 60
        end -- hour
    minute: INTEGER is
        -- Minute in hour, 00 to 59
        do
            Result := minutes \% 60
        end -- minute
```

---

## Eiffel: Beispiel 1b

```
set (hh: INTEGER, mm: INTEGER) is
  require
    0 <= hh and hh < 24
    0 <= mm and mm < 60
  do
    minutes := hh * 60 + mm
  ensure
    hour = hh
    minute = mm
  end -- set
adjust (hh_by: INTEGER, mm_by: INTEGER) is
  -- Advance (+) or retard (-) either or both
  do
    minutes := minutes + hh_by * 60 + mm_by
    normalise
  end -- adjust
```

---

## Eiffel: Beispiel 1c

```
feature {NONE}
    minutes: INTEGER
        -- Minutes since midnight

    normalise is
        -- Restore invariant after adjustment
    do
        minutes := minutes \ 1440
        if minutes < 0 then
            minutes := minutes + 1440
        end
    end -- normalise
```

---

# Eiffel: Beispiel 1d

```
invariant
```

```
    0 <= hour and hour < 24
```

```
    0 <= minute and minute < 60
```

```
    0 <= minutes and minutes < 1440
```

```
end -- class TIME_OF_DAY
```

```
...
```

```
    finish_time: TIME_OF_DAY;
```

```
    ...
```

```
    create finish_time -- !!finish_time
```

---

# Eiffel: Creation

```
class TIME_OF_DAY

creation
  set

feature
  ...

  lunchtime: TIME_OF_DAY
  ...
  create lunchtime.set (12, 30)
  -- !!lunchtime.set (12, 30)
```

---

## Eiffel: Clone

```
ff_start, ent_start: TIME_OF_DAY
...
create ff_start  -- !!ff_start
...
if starting_together then
    ent_start := clone (ff_start)
end
```

---

# Eiffel: Rename

```
class
  HOUR12
      -- Hour component of time of day
inherit
  DIGIT12
      rename
          increment as advance,
          decrement as retard
      end
end -- class HOUR12
```

---

# Eiffel: Export

```
class
  DIGITF

inherit
  DIGIT
    export
      {NUMBER} more_sig
    end

end -- class DIGITF
```



---

# Eiffel: Überschreiben

```
class HOUR12 -- Hour component of time of day
inherit
    DIGIT12
        redefine
            symbol
        end
feature
    symbol: STRING is -- 12, 1, 2... 10, 11
        do
            ...
        end -- symbol
end -- class HOUR12
```

---

# Eiffel: Generizität

```
digits: ARRAY [DIGIT]
create digits.make (1, 3)  -- !!digits.make(1,3)
```

auch gebundene Generizität:

```
class
  SORTED_LIST [ET -> COMPARABLE]
  ...
```

---

# Eiffel: Input and Output

```
class FILTER
creation run
feature
  run is -- Echo until end of file
  do
    from
      io.read_line
    until
      equal (io.last_string, "")
    loop
      io.put_string (io.last_string)
      io.put_new_line
      io.read_line
    end -- loop
  end -- run
end -- class FILTER
```

---

## Eiffel: Beispiel 2a

```
class
  MON12
      -- Month of year
inherit
  DIGIT12
      redefine
          symbol, increment, decrement
      end
creation
  make,
  connect_day
```

---

## Eiffel: Beispiel 2b

```
feature
  increment is
    -- As parent, but note change
    do
      Precursor
      changed
    end -- increment
  decrement is
    -- As parent, but note change
    do
      Precursor
      changed
    end -- decrement
```

---

## Eiffel: Beispiel 2c

```
feature {NONE} -- new features, protected
  day: DIGITV -- Day of month, we control range
  changed is -- Month has changed: update day range
  do
    if day /= Void then
      inspect value + 1
      when 1, 3, 5, 7, 8, 10, 12 then
        day.set_range (1, 31)
      when 2 then
        day.set_range (1, 28) -- leap?
      when 4, 6, 9, 11 then
        day.set_range (1, 30)
      end
    end
  end
end -- changed
```

---

## Eiffel: Beispiel 2d

```
full_names: ARRAY [STRING] is
    -- Full month names
    once
        Result := <<"January", "February", "March",
            "April", "May", "June",
            "July", "August", "September",
            "October", "November", "December">>
    end
end -- class MON12
```

---

## Quelle (kurze Sprachbeschreibung)

Simon Parker: Eiffel for beginners

[www.maths.tcd.ie/~odunlain/eiffel/eiffel\\_course/eforb.htm](http://www.maths.tcd.ie/~odunlain/eiffel/eiffel_course/eforb.htm)



---

# Nebenläufigkeit allgemein

- im Prinzip sind Nebenläufigkeit und Synchronisation unabhängig von Objektorientiertheit
- Monitore setzen auf Modulen bzw. Objekten auf
- Grundidee von Smalltalk kommt von aktiven Objekten (= Prozessen)
- in der Praxis werden durch Synchronisationsbedingungen Unterschiede zwischen Vererbung und Subtyping offensichtlich (Vererbungsanomalie)

---

# Monitore in Java 1

- jedes Objekt ist Monitor:  
zu jedem Zeitpunkt höchstens eine „synchronized“ Methode (oder Anweisung) ausführbar, alle anderen müssen warten bis vorherige beendet
- eine explizite Warteschlange pro Objekt:  
explizites einhängen (wait) und aufwecken (notify)
- Mechanismus auf niedriger Ebene und sehr oft falsch verwendet (z.B. unsichere „double checks“)

---

## Monitore in Java 2

- folgende Gleichung stimmt fast nie:  
Objekt = Kapselungseinheit = Monitor = Warteschlange
- Ratschläge:
  - alle Methoden „synchronized“, oder keine
  - eigene Warteschlange ( $\neq$  Monitor) für jeden Zweck
  - Effizienzüberlegungen (double check) bei Synchronisation ignorieren oder sehr genau analysieren

---

# Vererbungsanomalie

- ererbter Synchronisationscode muss häufig überschrieben werden, da
  - neuer Zustand berücksichtigt werden muss, der in Oberklasse nicht existiert hat
  - in Unterklasse zwischen Zuständen unterschieden werden muss, die in Oberklasse gleich sind
  - alternative Zweige in Unterklasse zu allgemein sind
- manchmal Hinweis auf Verletzung der Ersetzbarkeit, aber nicht immer
- Synchronisationscode klar von inhaltlichem Code trennen

---

# SCOOP Modell (Eiffel)

```
class interface BOUNDED_QUEUE [T] feature
  empty, full: BOOLEAN
  put(x:T) require not full
             ensure not empty
  remove    require not empty
             ensure not full
  item: T   require not empty
end -- class interface BOUNDED_QUEUE
```

```
separate class BOUNDED_BUFFER [T] inherit
  BOUNDED_QUEUE [T]
end
```