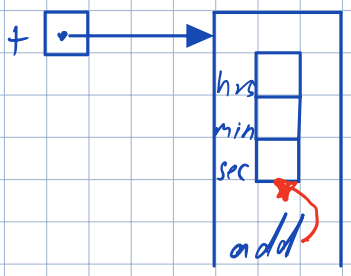
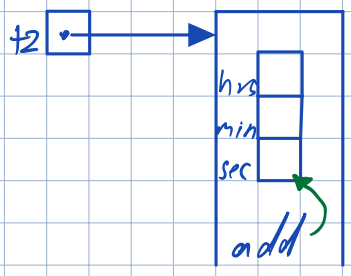
```
class Timer {

    int hrs;
    int min;
    int sec;

    void add(int s){
        this.sec += s;
        if(sec>59){
            this.min += this.sec \ 60;
            this.sec = this.sec % 60;
        }
    }
}
```

```
Timer t = new Timer();
t.sec = 75;
t.add(3);
```




```
Timer t2 = new Timer();
t2.sec = 62;
t2.add(9);
```



Klasse = Daten + Methoden

Parameter

```
t.add(13);
```

Empfänger / Receiver

```
void add(int sec)
// gibt es das this: t
this.sec += sec
```

```
Timer t1 = new Timer();
t1.add(13);

Timer t2 = new Timer();
t2.add(14);
```

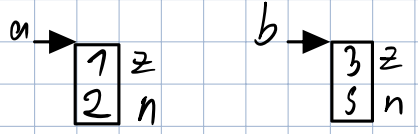
```
class Timer {
    :
    void add(int sec) {
        t2
        this.sec += sec;
        t2
        :
    }
}
```

Bsp:

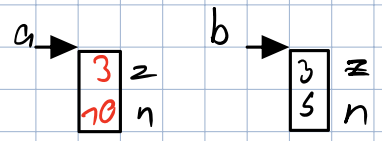
```
Fraction a = new Fraction(); a.z = 1; a.n = 2;
Fraction b = new Fraction(); b.z = 3; b.n = 5
```

a = 1/2 b = 3/5

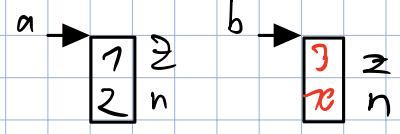
```
class Fraction {
    int z; // Zähler
    int n; // Nenner
```




```
a.mult(b);
```




```
b.mult(a);
```



Es wird immer der Zustand des Empfängers verändert

```
void mult(Fraction f){
    this.z = this.z * f.z;
    this.n = this.n * f.n;
}
```

Konstruktoren

⇒ spezielle Methoden, die beim erzeugen eines Objekts automatisch aufgerufen werden.

```
class Fraction {
    int z;
    int n;
    Fraction(int z; int n;){
        this.z = z;
        this.n = n;
    }
    Fraction(){
        this.z = 0;
        this.n = 1;
    }
    void mult(Fraction f){
        ...
    }
}
```

kein Name!

Alternative: this(0,1); ↳ Konstruktor Aufruf

```
Fraction f = new Fraction(1,2);
```

reserviert Speicher

z = 1, n = 2

Aufruf Konstruktor

```
Fraction fz = new Fraction();
```

↳ z = 0, n = 1

Beispiel Timer:

```
class Timer{
  int hrs, min, sec;

  Timer(int h, int m, int s){
    hrs = h; min = m; sec = s;
  }

  Timer(){
    this(0,0,0);
  }

  void clear() {...}

  void add() {...}
}
```

this.hrs = h;
this.min = m;
this.sec = s;!

ruft den Konstruktor mit 3 Parametern auf

Konstruktoren werden automatisch aufgerufen, wenn ein Objekt ihrer Klasse erzeugt wird

```
Timer t = new Timer(3,25,10);
```

1. Erzeugt ein Timer-Objekt
2. Ruft den Konstruktor mit 3 int-Parametern auf, der die Felder wie folgt initialisiert;

   hrs = 3; min = 25; sec = 10;

```
t = new Timer();
```

1. Erzeugt neues Timer-Objekt
2. Ruft den parameterlosen Konstruktor auf, der den Konstruktor mit 3 Parametern aufruft;

   hrs = 0; min = 0; sec = 0;

Vor Aufruf des Konstruktors werden noch alle Initialisierungen ausgeführt, die bei der Deklaration der Felder angegeben wurden.

Java hat "Standard-Konstruktor"! Man kann Obj. erstellen auch wenn kein Konst. impl. wurde

Statik

```
class Timer {
    static int maxHrs = 24;
    int hrs;
    int min;
    int sec;
    .....
}
```

neu 1x vorhanden (in d. Klasse)

⇐ Statisches Feld / Klassenfeld

← Objektfelder ⇒ in jedem Objekt vorhanden

```
Timer t = new Timer();
t.hrs;
```

Timer.months = 24;

Klasse Timer

maxHrs = 24

Timer. (eigentlich)

Timer-objekt

hrs = 3
min = 27
sec = 50

Timer-objekt

hrs = 17
min = 0
sec = 33

Alle Objekte (d.h. ihre Methoden) können auf die statischen Felder ihrer Klasse zugreifen

z.B maxHrs = 12;

Statische Methoden

```
class Timer {
    ....
    static void switchMaxHrs() {
        maxHrs = 36 - maxHrs;
    }
    void clear() {....}
    void add(int s) {...}
```

Statische Methode (Klassenmethode) wird auf die selbst angewendet

Aufrufe über Klassennamen:
Timer.switchMaxHrs();

Objektmethoden werden auf Objekte der Klasse angewendet

Aufruf über Variablennamen: t.add(10);

Verwendung von statischen Methoden

- zum Abfragen / Setzen / Ändern von statischen Feldern
- für Operationen, die sich nicht auf ein bestimmtes Objekt beziehen (z.B. main-Methode)

Besonderheiten

- Statische Methoden dürfen nur auf statische Felder zugreifen (nicht auf Objektfelder)