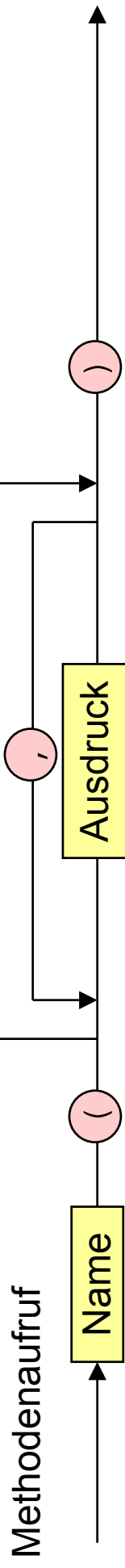
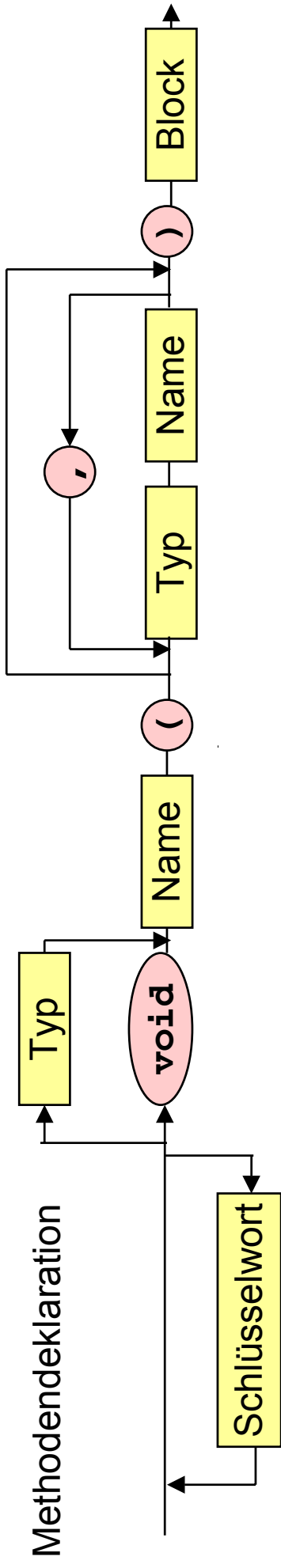

II.2. Objekte, Klassen und Methoden

- 1. Grundzüge der Objektorientierung
- 2. Methoden, Unterprogramme und Parameter
- 3. Datenabstraktion
- 4. Konstruktoren
- 5. Vordefinierte Klassen

2. Methoden

- **Generelles zum Aufruf von Methoden**
- **Parameterübergabemechanismen**
(call by value, call by reference)
- **Speicherorganisation bei Methodenaufruf und Parameterübergabe (Laufzeitkeller)**
- **Statische und nicht-statische Methoden und Attribute**
- **Gültigkeit von Bezeichnern**

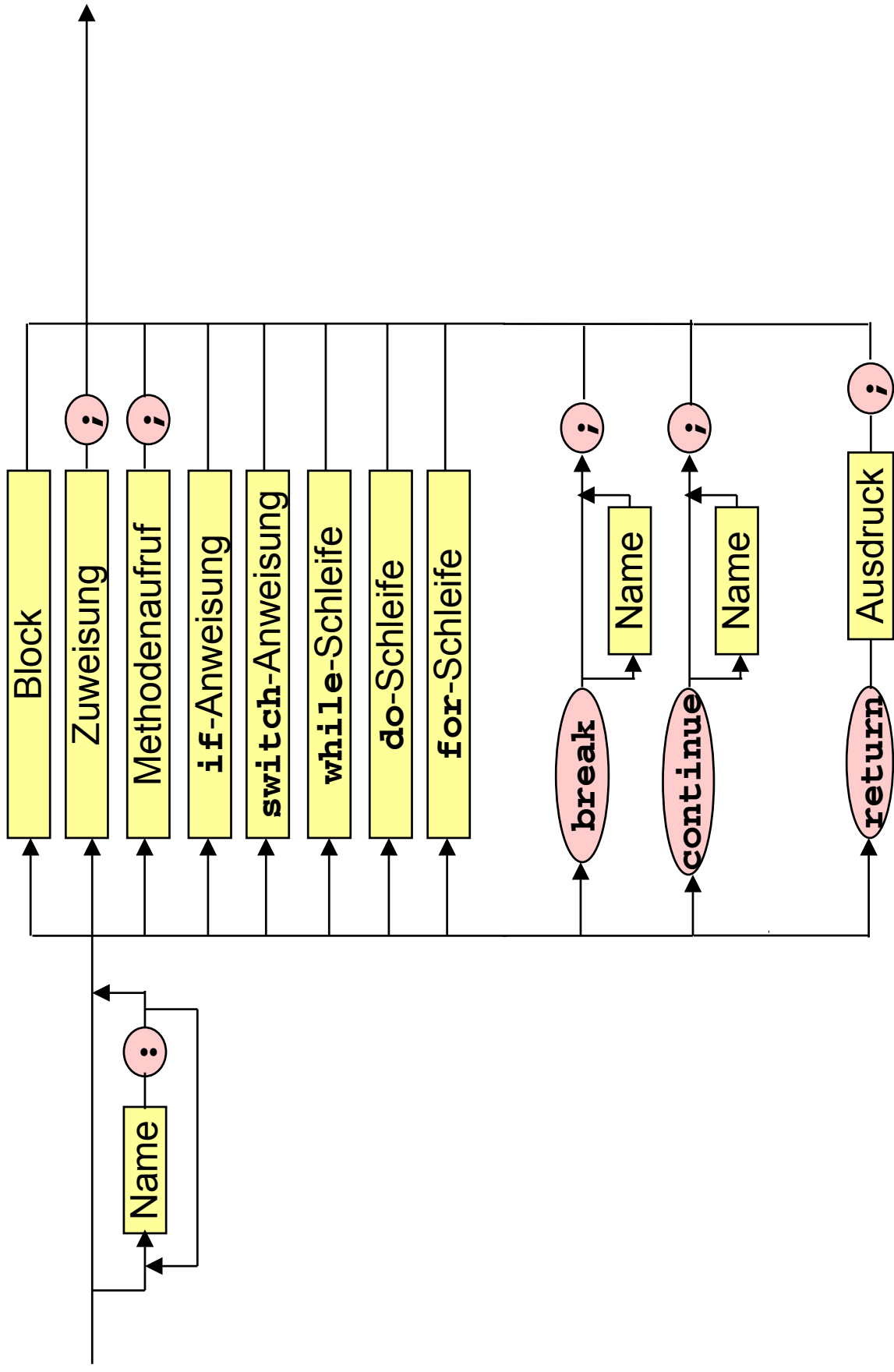
Methodendeklaration und -aufruf



Aufruf von Methoden

```
public class Zins_Programm {  
  
    public static double zins (double kapital) {  
        // berechnet 3 Prozent Zinsen  
        return 1.03 * kapital;  
    }  
  
    public static void main (String [] args) {  
        double betrag1 = 1000,  
               betrag2 = 570.22,  
  
               gewinn = zins (betrag1 + betrag2) ;  
  
        System.out.println (gewinn) ;  
    }  
  
}
```

Anweisung



Prozedur-Methoden

```
public class Druck_Programm {  
  
    /* Druckprozedur.  
     * Gibt alle Werte in a aus.  
     */  
    public static void drucke (int[] a) {  
        for (int i = 0; i < a.length; i++)  
            System.out.print(a[i] + " ");  
  
        System.out.print("\n");  
    }  
  
    public static void main (String[] args) {  
        int[] x = new int [4];  
        x[0] = 5; x[1] = 2; x[2] = 7; x[3] = 4;  
        drucke (x);  
    }  
}
```

Call by value und call by reference

Call by value:

- Aktueller Parameter (im Methodenaufruf) wird ausgewertet.
- Wert des aktuellen Parameters wird in den formalen Parameter der Methode kopiert.
- Änderungen des formalen Parameters der Methode bewirken **keine Änderung** des aktuellen Parameters.

Call by reference:

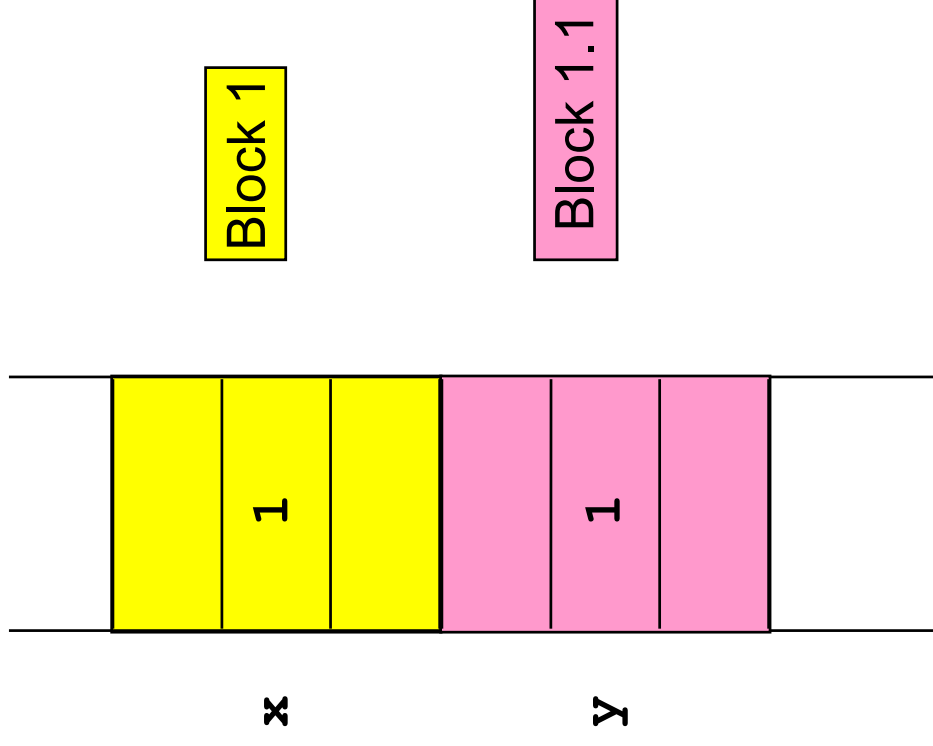
- Aktueller Parameter ist Variable.
- Formaler Parameter der Methode wird **Verweis** auf den aktuellen Parameter.
- Jede Änderung des formalen Parameters der Methode bewirkt auch eine Änderung des aktuellen Parameters.

Call by value - Parameterübergabe

```
public class Call_by_value {  
  
    public static void f (double r) {  
  
        r = 4.6;  
  
    }  
  
    public static void main (String [] args) {  
  
        double s = 2.1;  
        System.out.println("s: " + s);  
  
        f(s);  
        System.out.println("s: " + s);  
  
    }  
  
}
```

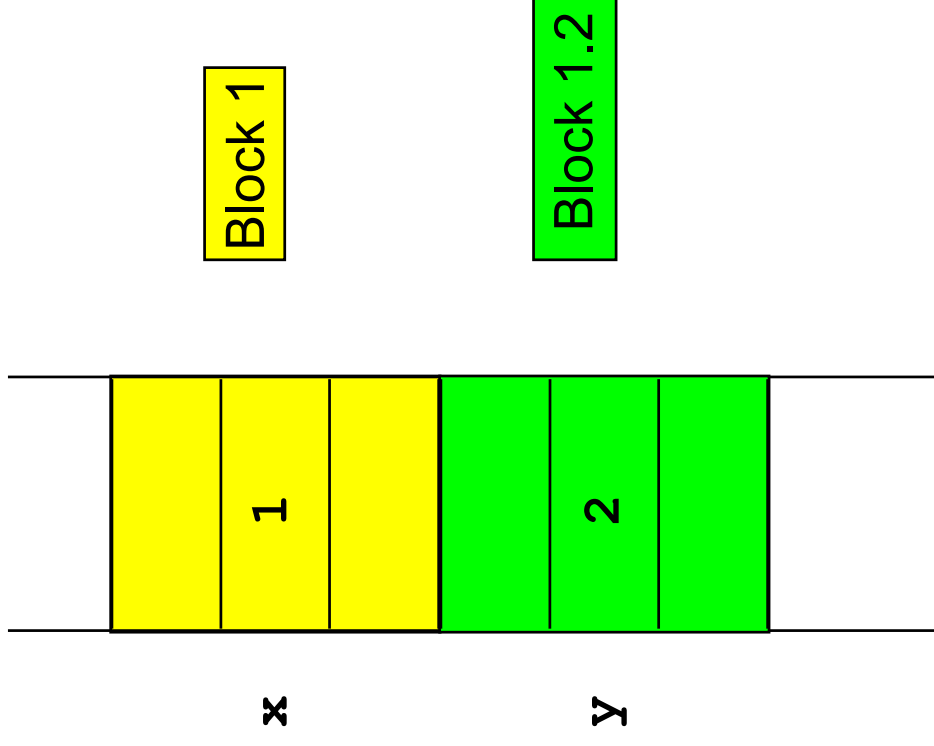

Laufzeitkeller

```
int x;  
  
  {int y = 1;  
    x = y;  
  }  
  
  {int y = 2;  
    ...  
  }
```



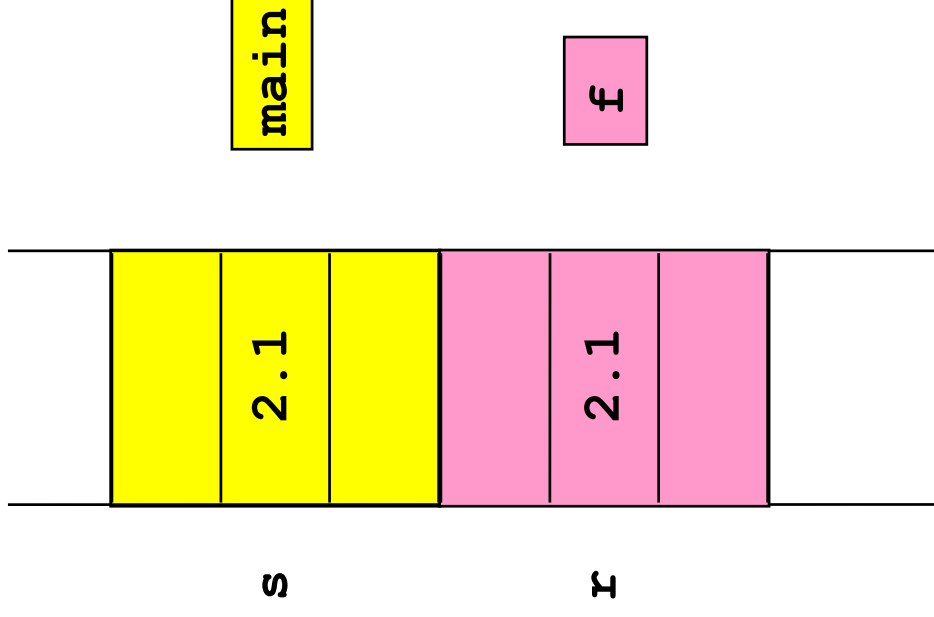
Laufzeitkeller

```
int x;  
  
  {int y = 1;  
    x = y;  
  }  
  
  {int y = 2;  
    ...  
  }
```



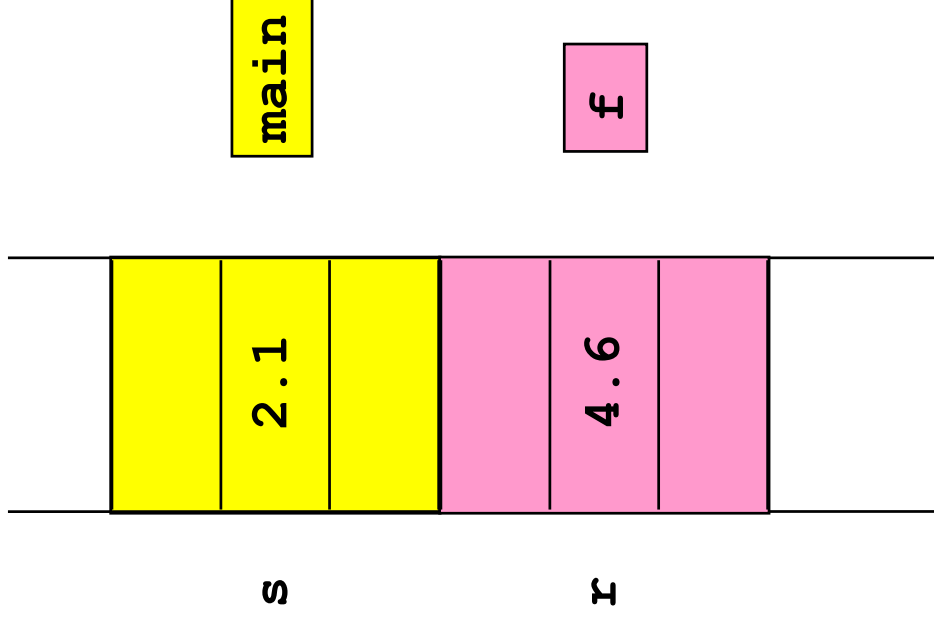
Laufzeitkeller bei Methoden

```
public static void main (...) {  
  
    double s = 2.1;  
  
    f(s);  
  
}  
  
public static void f (double r) {  
  
    r = 4.6;  
  
}
```



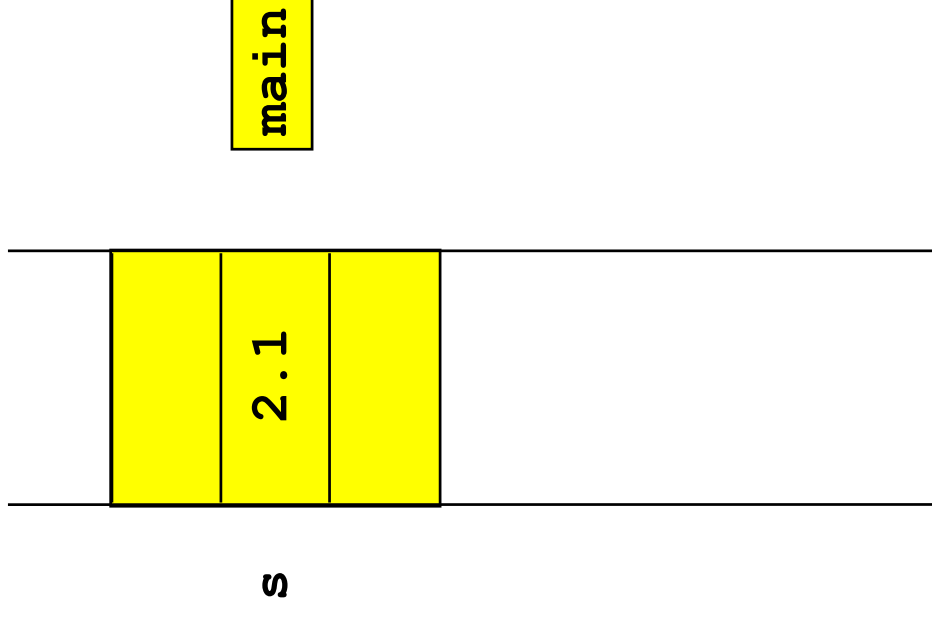
Laufzeitkeller bei Methoden

```
public static void main (...) {  
  
    double s = 2.1;  
  
    f(s);  
  
}  
  
public static void f (double r) {  
  
    r = 4.6;  
  
}
```



Laufzeitkeller bei Methoden

```
public static void main (...) {  
  
    double s = 2.1;  
  
    f(s);  
  
}  
  
public static void f (double r) {  
  
    r = 4.6;  
  
}
```

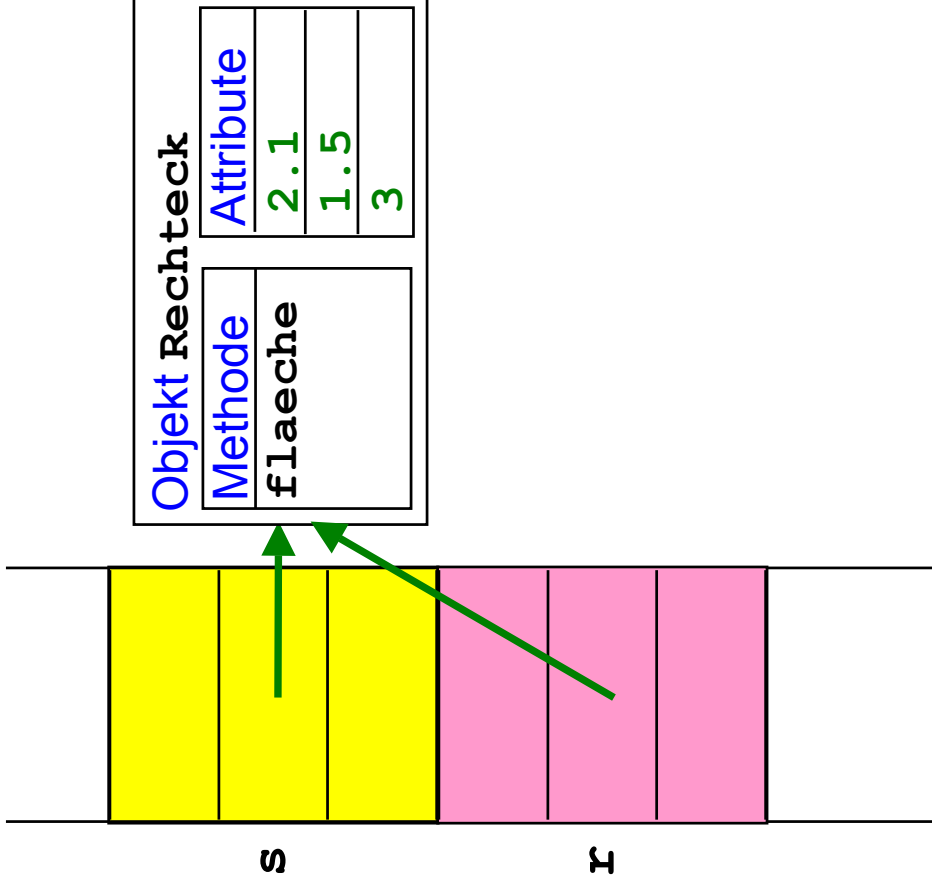


Call by reference - Parameterübergabe

```
public class Call_by_reference {  
  
    public static void f (Rechteck r) {  
        r.laenge = 4.6;  
    }  
  
    public static void main (String [] args) {  
        Rechteck s = new Rechteck ();  
        s.laenge = 2.1; s.breite = 1.5; s.strichstaerke = 3;  
        System.out.println (s.laenge + ", " + s.breite +  
            ", " + s.strichstaerke);  
  
        f(s);  
        System.out.println (s.laenge + ", " + s.breite +  
            ", " + s.strichstaerke);  
    }  
}
```

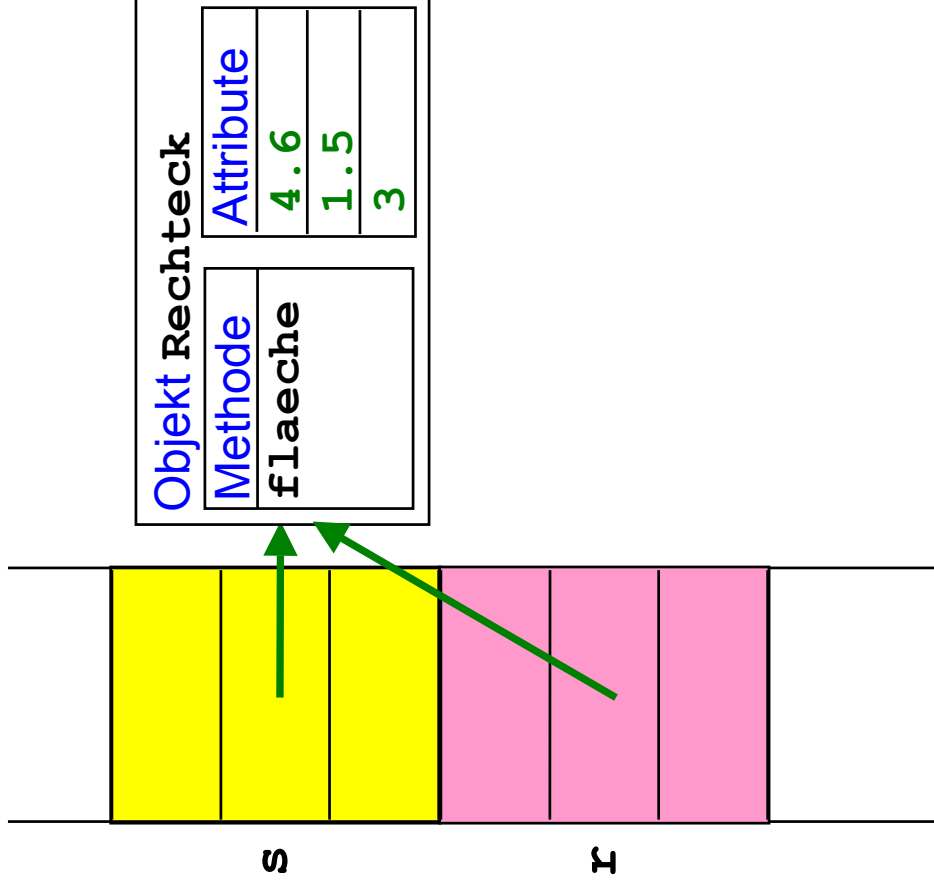
Laufzeitkeller bei Methoden

```
public static void main (...) {  
  
    Rechteck s = new Rechteck ();  
  
    s.laenge = 2.1;  
    s.breite = 1.5;  
    s.strichstaerke = 3;  
  
    f(s);  
  
}  
  
public static void f (Rechteck r) {  
    r.laenge = 4.6;  
}
```



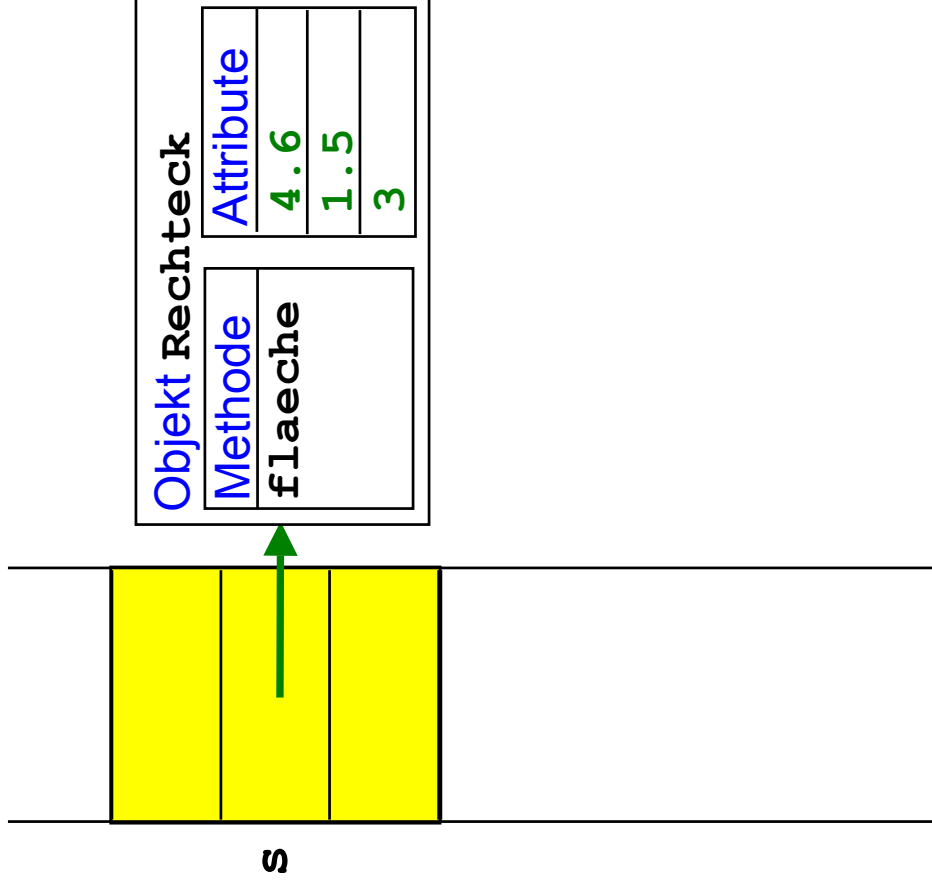
Laufzeitkeller bei Methoden

```
public static void main (...) {  
  
    Rechteck s = new Rechteck ();  
  
    s.laenge = 2.1;  
    s.breite = 1.5;  
    s.strichstaerke = 3;  
  
    f(s);  
  
}  
  
public static void f (Rechteck r) {  
    r.laenge = 4.6;  
}
```



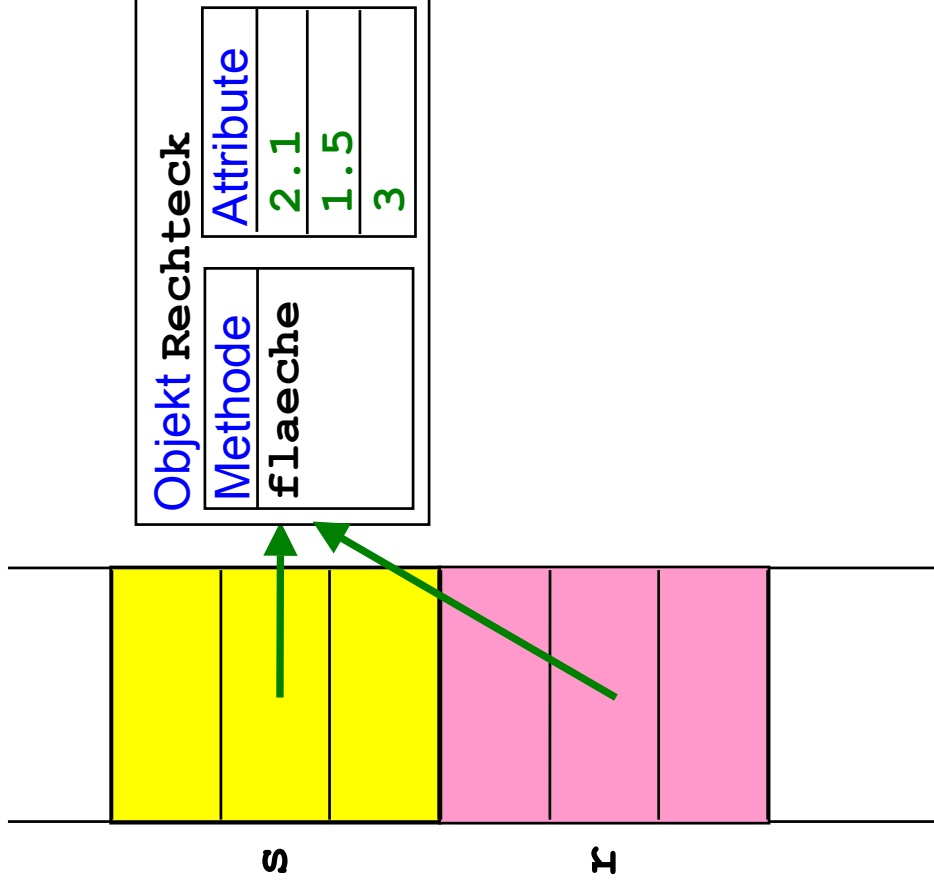
Laufzeitkeller bei Methoden

```
public static void main (...) {  
  
    Rechteck s = new Rechteck ();  
  
    s.laenge = 2.1;  
    s.breite = 1.5;  
    s.strichstaerke = 3;  
  
    f(s);  
  
}  
  
public static void f (Rechteck r) {  
    r.laenge = 4.6;  
}
```



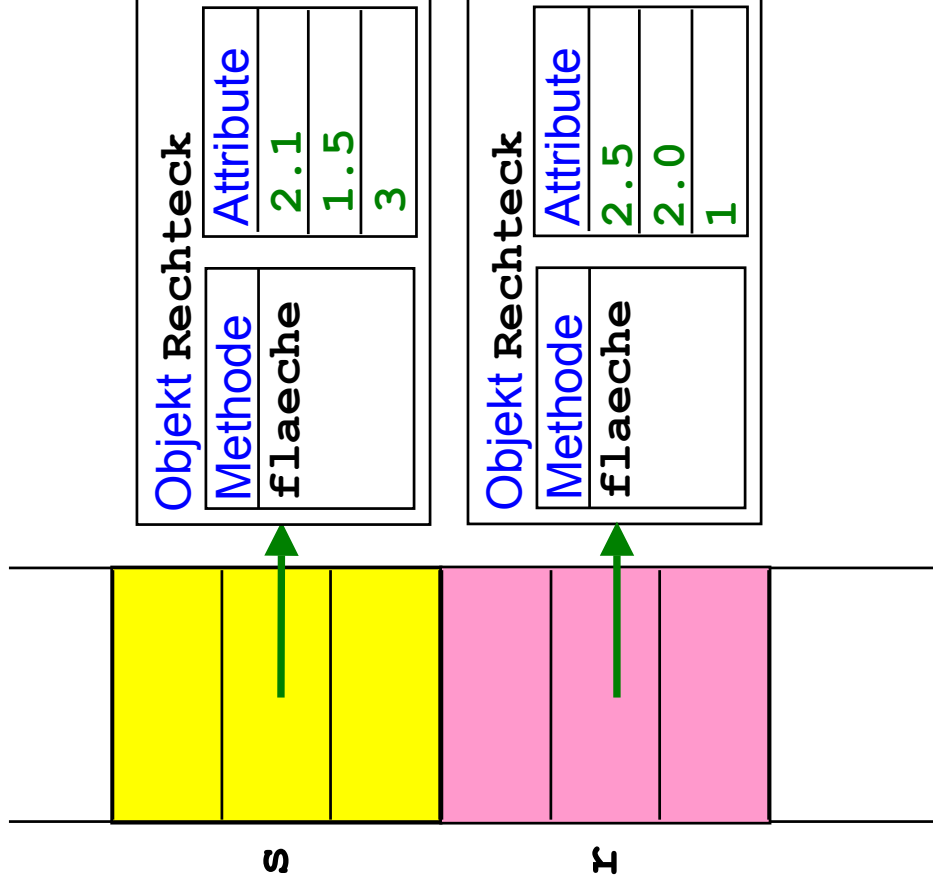
Parameterübergabe in Java

```
public static void main (...) {  
    Rechteck s = new Rechteck ();  
  
    s.laenge = 2.1;  
    s.breite = 1.5;  
    s.strichstaerke = 3;  
  
    f(s);  
}  
  
public static void f (Rechteck r) {  
    r = new Rechteck ();  
  
    r.laenge = 2.5;  
    r.breite = 2.0;  
    r.strichstaerke = 1;  
}
```



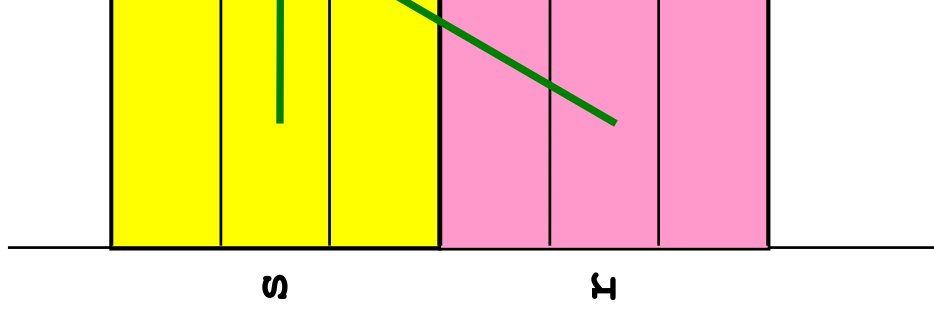
Parameterübergabe in Java

```
public static void main (...) {  
    Rechteck s = new Rechteck ();  
  
    s.laenge = 2.1;  
    s.breite = 1.5;  
    s.strichstaerke = 3;  
  
    f(s);  
}  
  
public static void f (Rechteck r) {  
    r = new Rechteck ();  
  
    r.laenge = 2.5;  
    r.breite = 2.0;  
    r.strichstaerke = 1;  
}
```



Parameterübergabe in Java

```
public static void main (...) {  
    Rechteck s = new Rechteck ();  
  
    s.laenge = 2.1;  
    s.breite = 1.5;  
    s.strichstaerke = 3;  
  
    f(s);  
}  
  
public static void f (Rechteck r) {  
    r = new Rechteck ();  
  
    r.laenge = 2.5;  
    r.breite = 2.0;  
    r.strichstaerke = 1;  
}
```



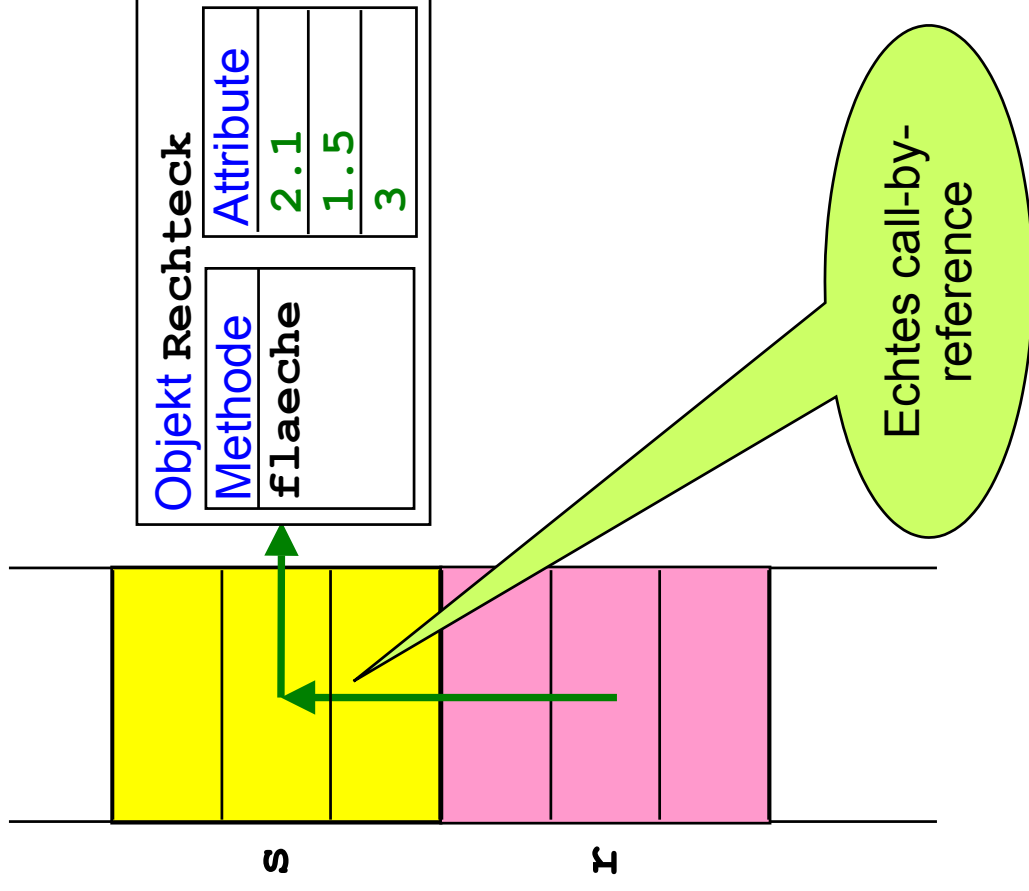
Objekt Rechteck	
Methode	
flaeche	

Attribute	
2.1	
1.5	
3	

Java verwendet
immer call-
by-value

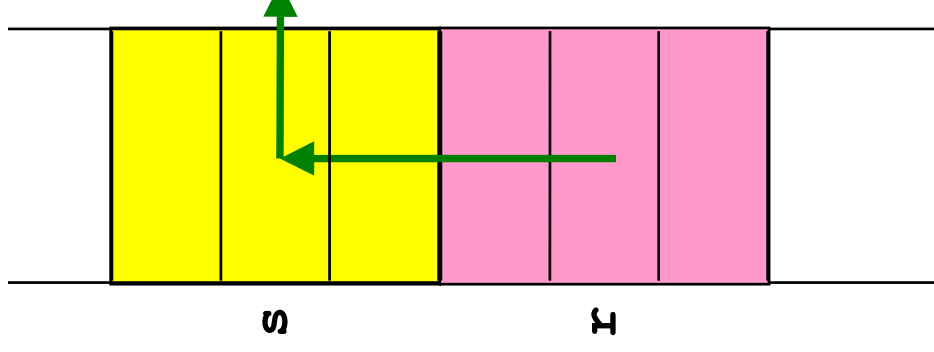
Echtes call by reference

```
public static void main (...) {  
    Rechteck s = new Rechteck ();  
  
    s.laenge = 2.1;  
    s.breite = 1.5;  
    s.strichstaerke = 3;  
  
    f(s);  
}  
  
public static void f (Rechteck r) {  
    r = new Rechteck ();  
  
    r.laenge = 2.5;  
    r.breite = 2.0;  
    r.strichstaerke = 1;  
}
```



Echtes call by reference

```
public static void main (...) {  
    Rechteck s = new Rechteck ();  
  
    s.laenge = 2.1;  
    s.breite = 1.5;  
    s.strichstaerke = 3;  
  
    f(s);  
}  
  
public static void f (Rechteck r) {  
    r = new Rechteck ();  
  
    r.laenge = 2.5;  
    r.breite = 2.0;  
    r.strichstaerke = 1;  
}
```



Objekt Rechteck	
Methode	Attribute
flaeche	2.5
	2.0
	1

Parameterübergabe in Java

- **Wert- / Referenzvariablen abhängig vom Datentyp:**
 - Primitive Datentypen: Wertvariablen
 - Nicht-primitive Datentypen (Arrays, Klassentypen): Referenzvariablen
- **In anderen Programmiersprachen:**
 - Wert- / Referenzvariablen unabhängig vom Datentyp
 - Beliebige Manipulation von Referenzen (Zeigern)
- **Parameterübergabe ist immer Werteübergabe (call by value)**
- **Bei nicht-primitiven Datentypen:**
 - Wegen der Referenzvariablen entspricht dies eingeschränkter Form der Referenzübergabe (call by reference)
- **In anderen Programmiersprachen:**
 - Wert- / volle Referenzübergabe unabhängig vom Datentyp

Sortiermethode

```
public class Sort {  
    public static void sortiere (int [] a) {  
        for (int i = 0; i < a.length - 1; i ++)  
            //Vertausche a[i] mit kleinstem Nachfolger  
            for (int j = i+1; j < a.length; j++)  
                if (a[i] > a[j]) { //Nachfolger kleiner als a[i]?  
                    //Vertausche a[i] und a[j]  
                    int z = a[i]; a[i] = a[j]; a[j] = z;  
                }  
    }  
  
    public static void drucke (int [] a) { ... }  
  
    public static void main (String [] args) {  
        int[] x = new int [4]; x[0] = 5; x[1] = 2; x[2] = 7; x[3] = 4;  
        drucke (x);  
        sortiere (x);  
        drucke (x);  
    }  
}
```


Statische Attribute und Methoden

```
public class Rechteck {
```

```
    static int flaechenberechnung = 0;
```

```
    double laenge, breite;    int strichstaerke;
```

```
    double flaeche () {
```

```
        flaechenberechnung ++;
```

```
        return laenge * breite;}  
}
```

```
double flaeche;
```

```
Rechteck r = new Rechteck (), s = new Rechteck ();
```

```
System.out.println (Rechteck.flaechenberechnung);
```

```
r.laenge = 2.5;    r.breite = 2.0;    r.strichstaerke = 1;
```

```
flaeche = r.flaeche ();
```

```
System.out.println (Rechteck.flaechenberechnung);
```

```
s.laenge = 2.1;    s.breite = 1.5;    s.strichstaerke = 3;
```

```
flaeche = s.flaeche ();
```

```
System.out.println (Rechteck.flaechenberechnung);
```

Klassen in Java

Klassen werden für verschiedene Zwecke verwendet:

- **1. Datentypen**
 - (Bsp: Rechteck)
- **2. Modularisierung von logisch zusammenhängenden Programmteilen**
 - (Bsp: Sort, enthält Methoden, die für das Sortieren benötigt werden.)
- **Aufruf von Objekt-Attributen und -Methoden (nicht static):**
 - `r.laenge, r.flaeche()`, ...
- **Aufruf von Klassen-Attributen und -Methoden (static):**
 - `Rechteck.flaechenberechnung, Sort.sortiere(x)`, ...

toString - Methode

```
public class Rechteck {  
    ...  
    public String toString () {  
        return "Länge: " + laenge +  
            " , Breite: " + breite +  
            " , Strichstärke: " + strichstaerke;  
    }  
}
```

```
Rechteck r = new Rechteck (), s = new Rechteck ();  
  
r.laenge = 2.5;    r.breite = 2.0;    r.strichstaerke = 1;  
System.out.println (r);  
  
s.laenge = 2.1;    s.breite = 1.5;    s.strichstaerke = 3;  
System.out.println (s);
```

Gültigkeit von Bezeichnern

```
public class Gueltigkeit {
    static Dreieck x;
    public static void main (String [] x) {
        System.out.println (x[0]);
        Dreieck d = new Dreieck ();
        Dreieck.setze (d, 2, Math.sqrt(5), 1);
        System.out.println (d);
    }
}
```

```
class Dreieck {
    double x, y, z;
    static void setze (Dreieck d, double x, double y, double z) {
        d.x = x;    d.y = y;    d.z = z;    }
    double flaeche () {
        double p = (x*x - y*y + z*z) / (2*x);
        double y = Math.sqrt (z*z - p*p);
        return x*y/2;    }
}
```