II.2. Objekte, Klassen und Methoden

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

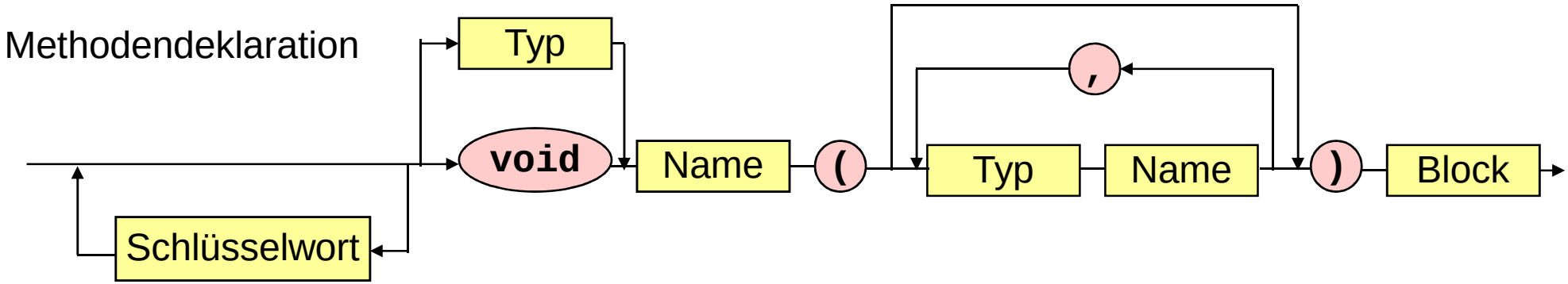
2. Methoden

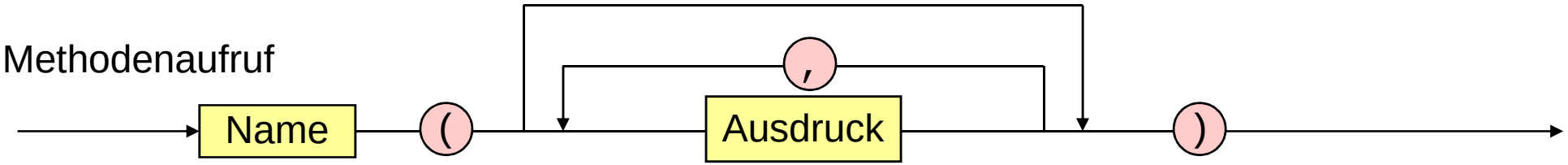
- Generelles zum Aufruf von Methoden
- Parameterübergabemechanismen
  (call by value, call by reference)
- Speicherorganisation bei Methodenaufruf und Parameterübergabe (Laufzeitkeller)
- vararg Parameter
- Statische und nicht-statische Methoden und Attribute
- Aufzählungstypen
- Gültigkeit von Bezeichnern

2. Methoden

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

Methodendeklaration und -aufruf

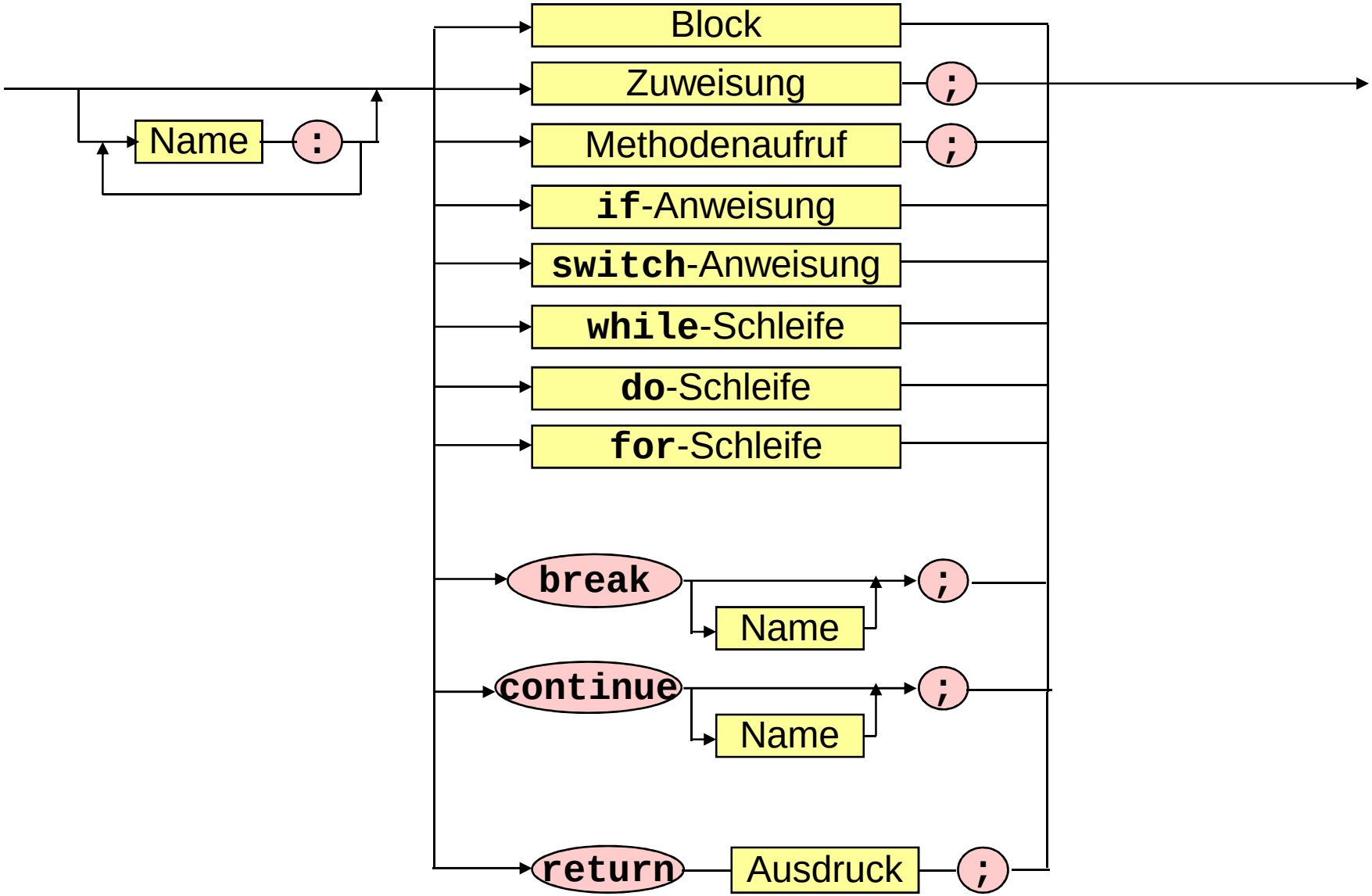




Aufruf von Methoden

```
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


Block
Zuweisung
;
Name
:
Methodenaufruf
;
if-Anweisung
switch-Anweisung
while-Schleife
do-Schleife
for-Schleife
break
;
Name
continue
;
Name
return
Ausdruck
;

Prozedur-Methoden

```
public class Druck_Programm {

     /* Druckprozedur.
      * Gibt alle Werte in a aus.
      */
     public static void drucke (int[] a) {
          for (int x : a)
              System.out.print(x + " ");

          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);
     }
}
```

2. Methoden

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

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:

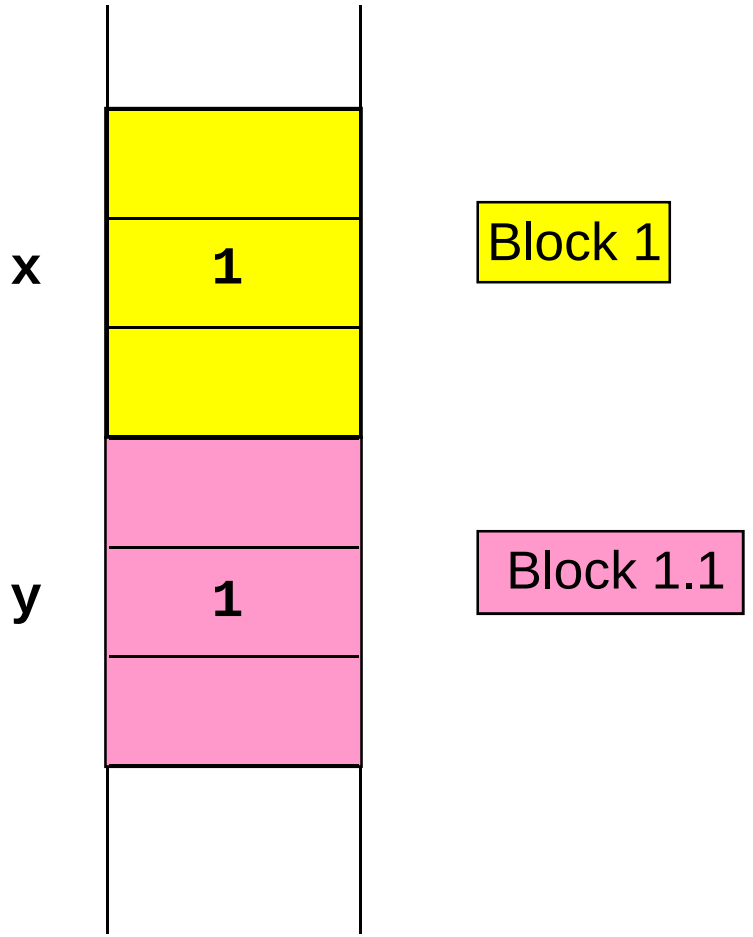
- Aktueller Parameter ist Variable.
- Formaler Parameter der Methode wird *Verweis* auf den aktuellen Parameter.
- Jede Änderung des formalen Parameters der Methode bewirkt auch eine Änderung des aktuellen Parameters.

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);
     }

}
```

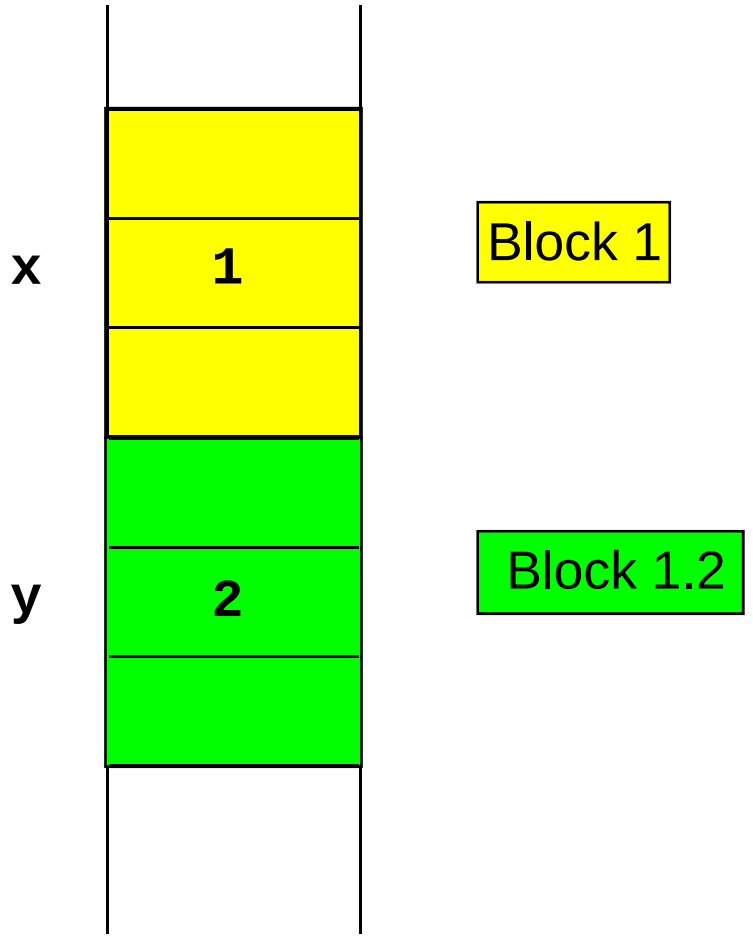
Laufzeitkeller

```
int x;

     {int y = 1;

          x = y;
      }

     {int y = 2;

          ...
     }
```



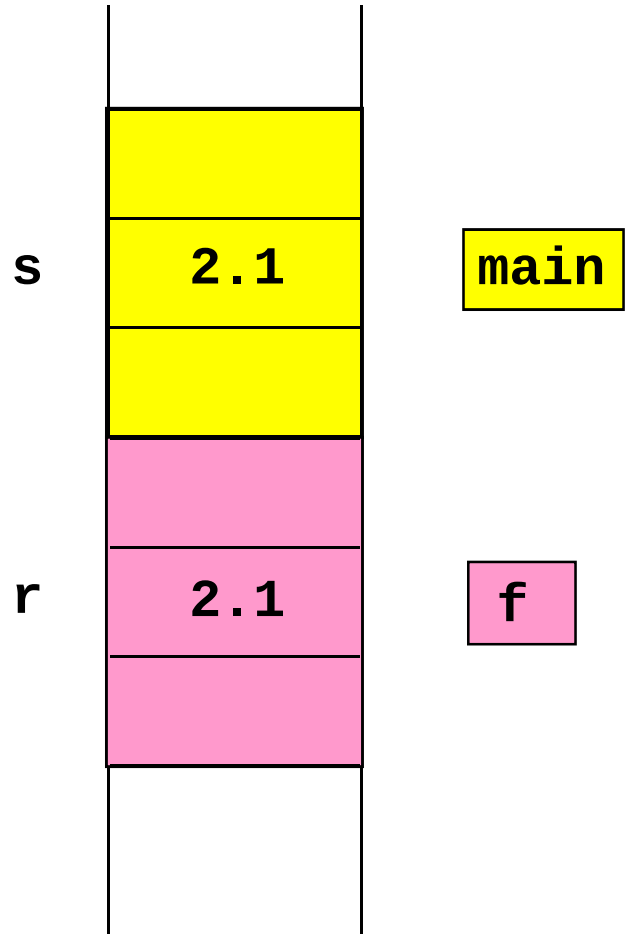
Laufzeitkeller

```
int x;

     {int y = 1;

          x = y;
      }

     {int y = 2;

          ...
     }
```



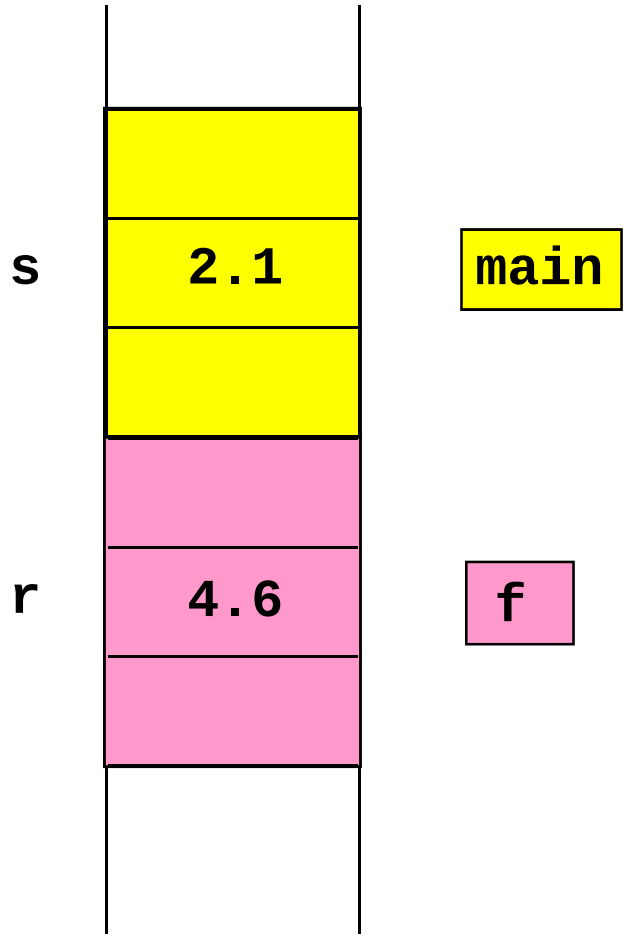
Laufzeitkeller bei Methoden

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

      double s = 2.1;

      f(s);

}



public static void f (double r) {

      r = 4.6;

}
```



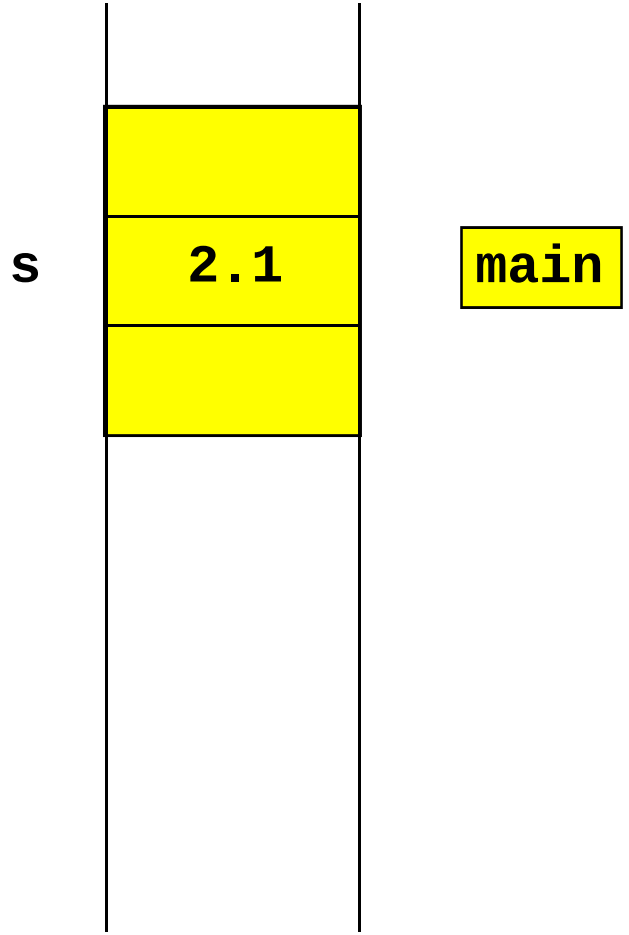
Laufzeitkeller bei Methoden

```
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;

}
```

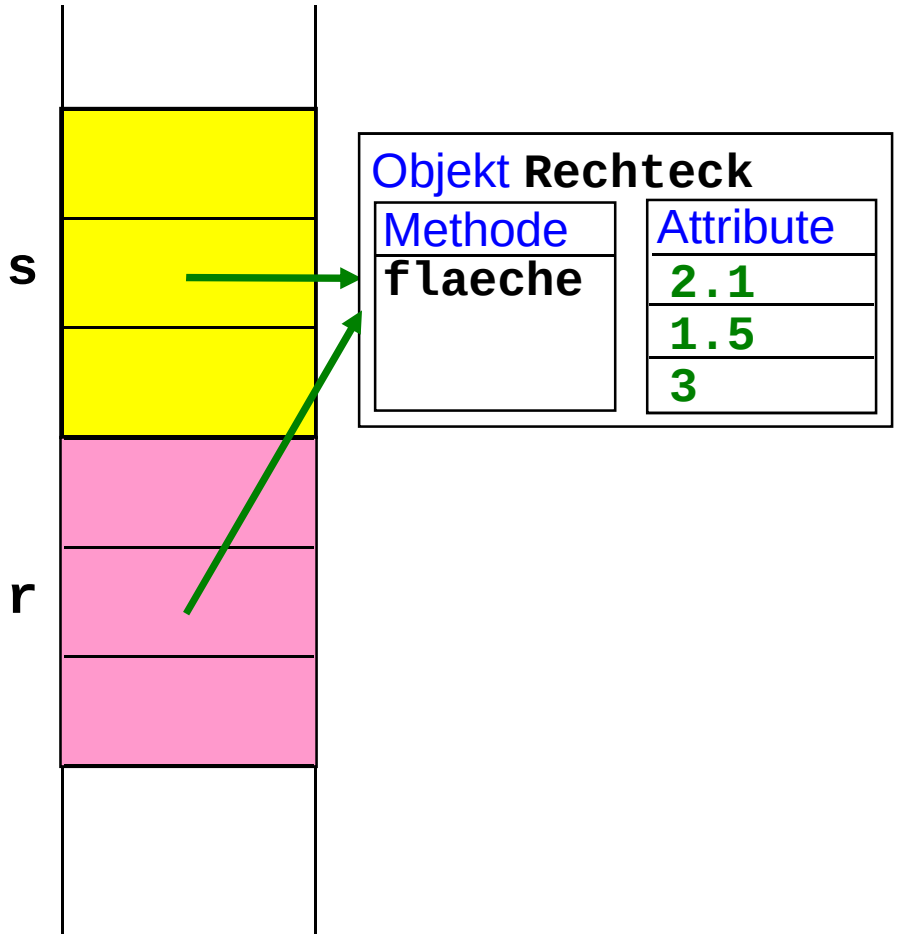


Call by reference - Parameterübergabe

```
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);
     }
}
```

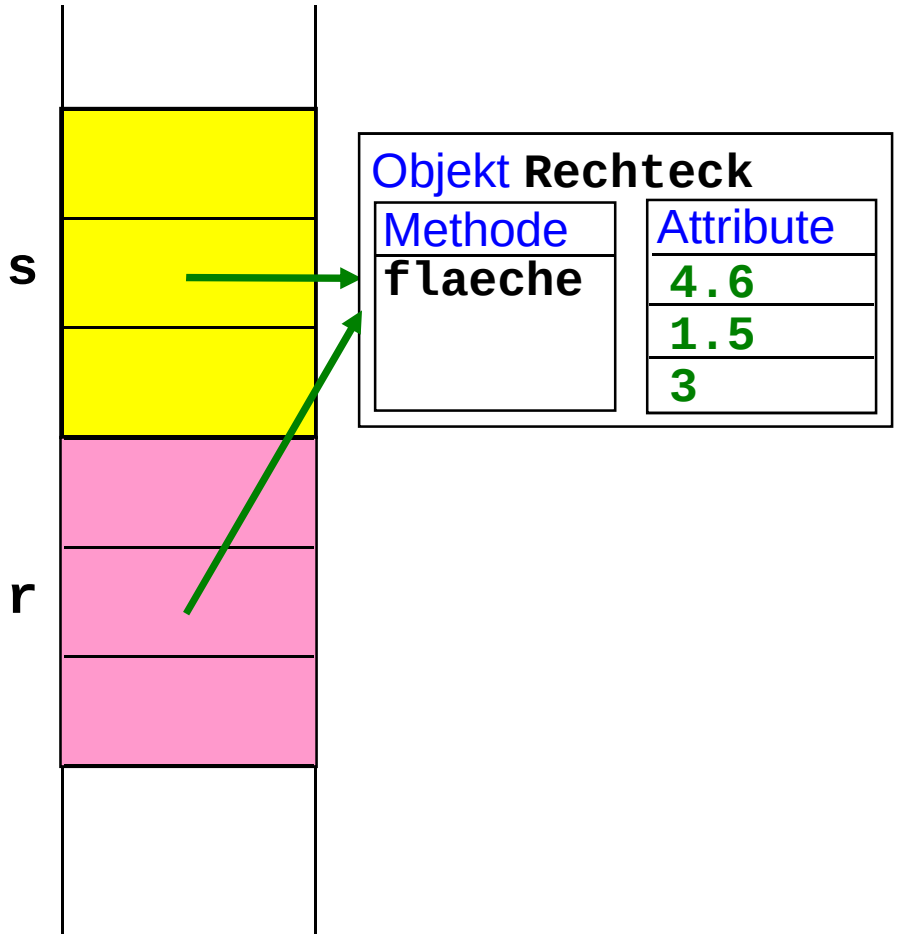
Laufzeitkeller bei Methoden

```
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;
}
```



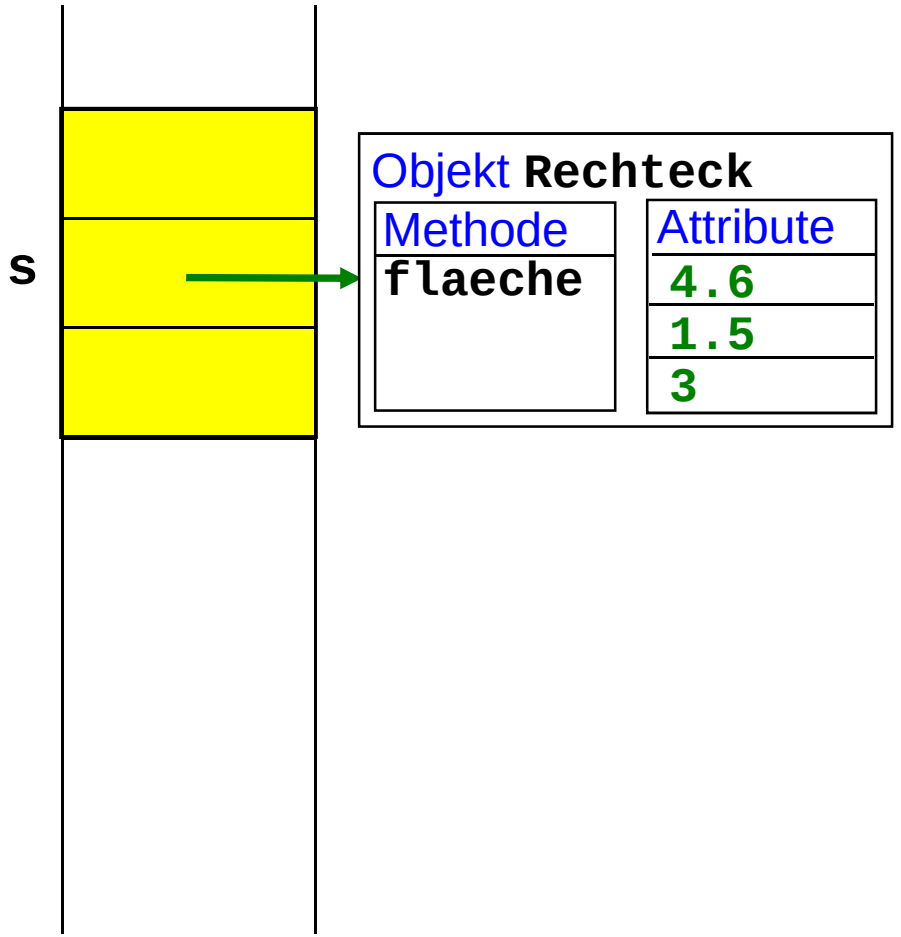
Laufzeitkeller bei Methoden

```
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;
}
```



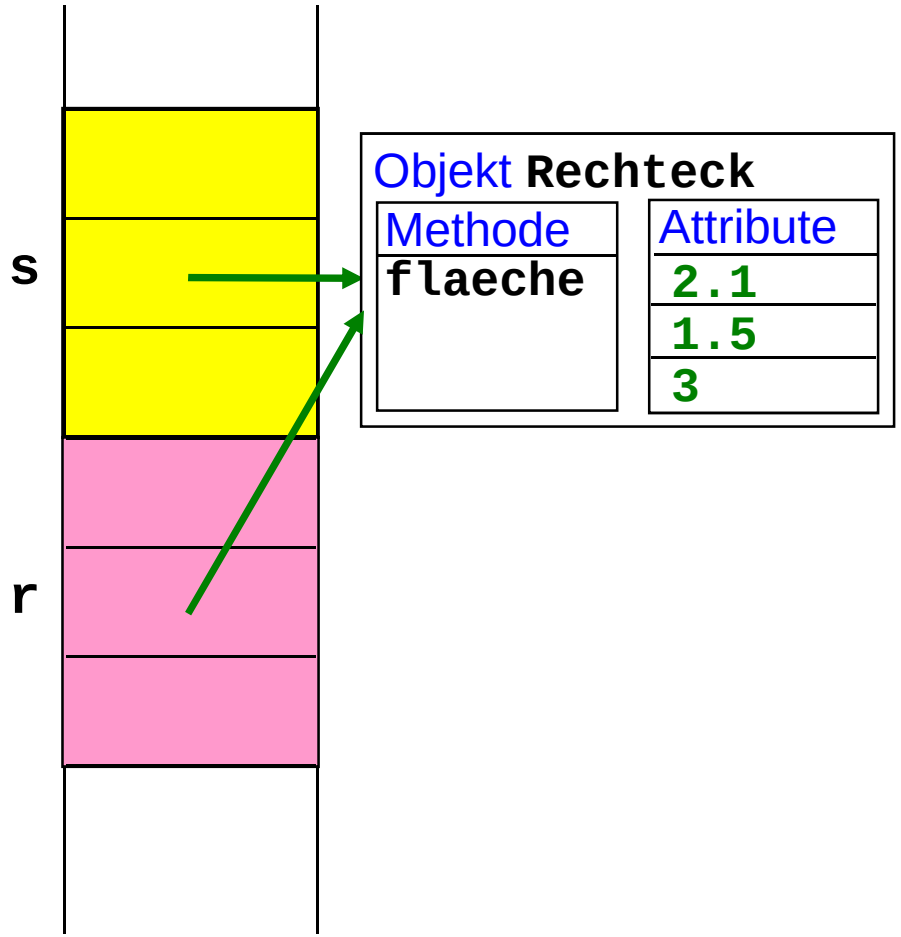
Laufzeitkeller bei Methoden

```
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;
}
```



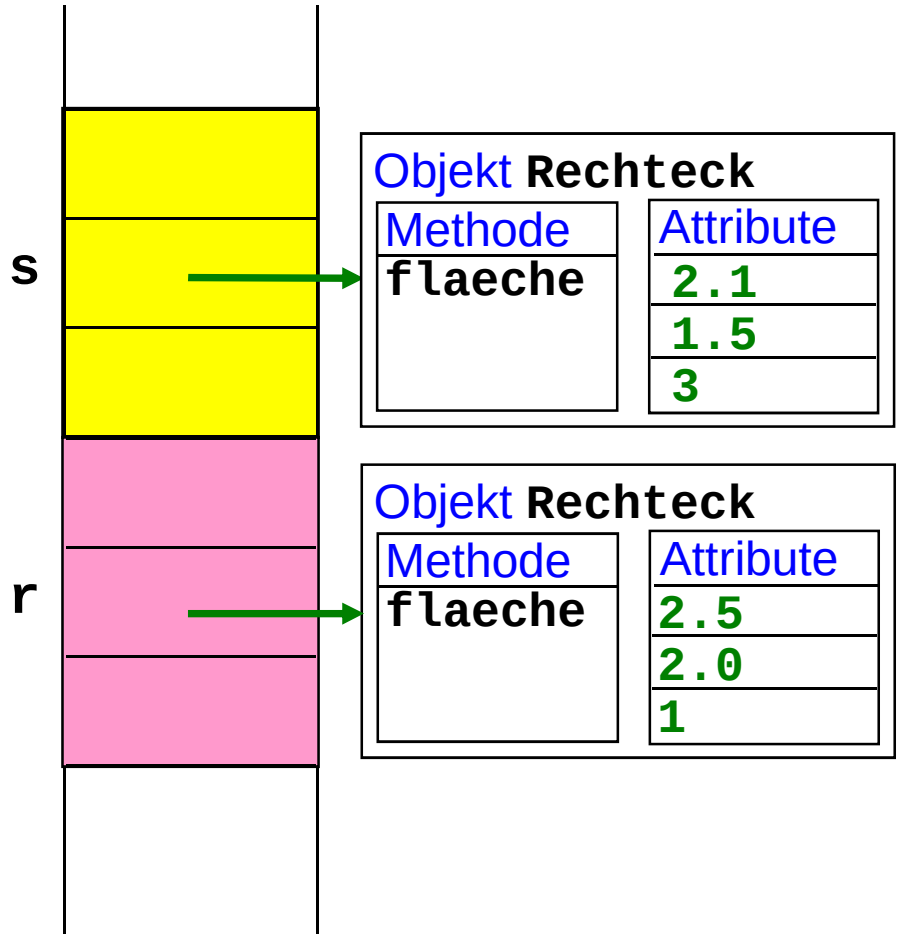
Parameterübergabe in Java

```
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;
}
```



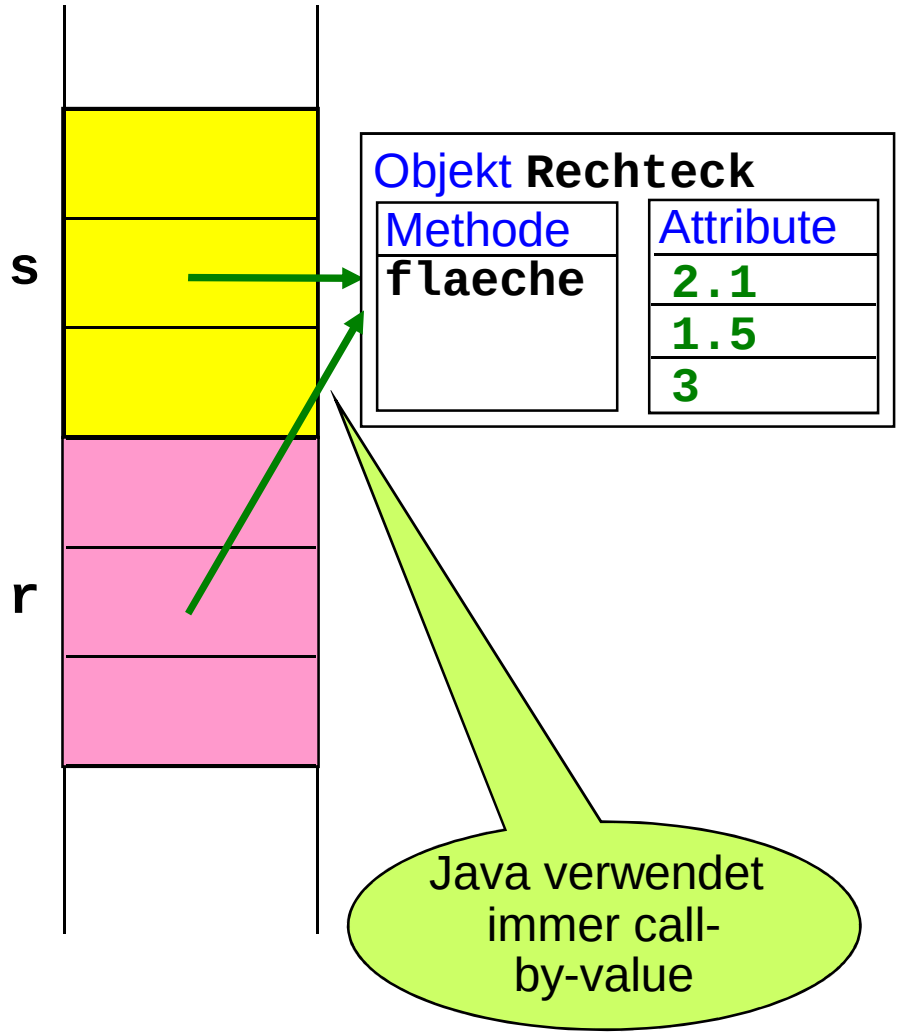
Parameterübergabe in Java

```
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;
}
```



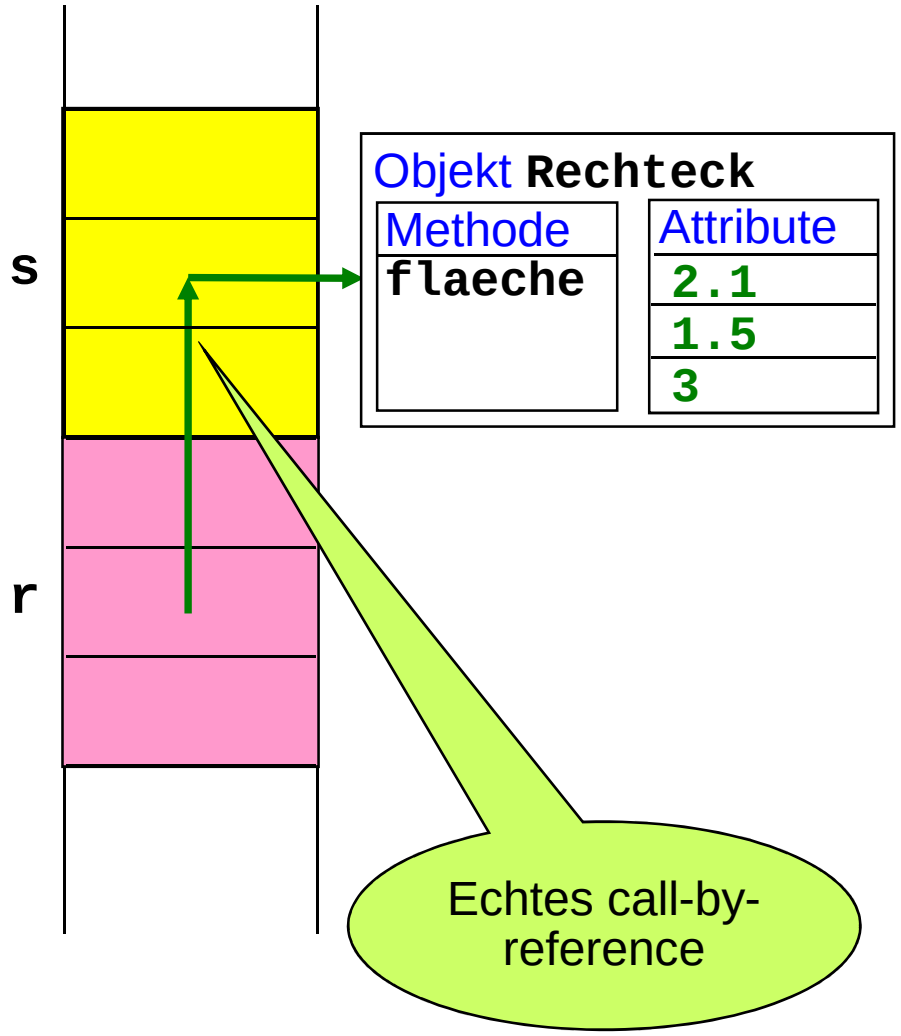
Parameterübergabe in Java

```
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;
}
```



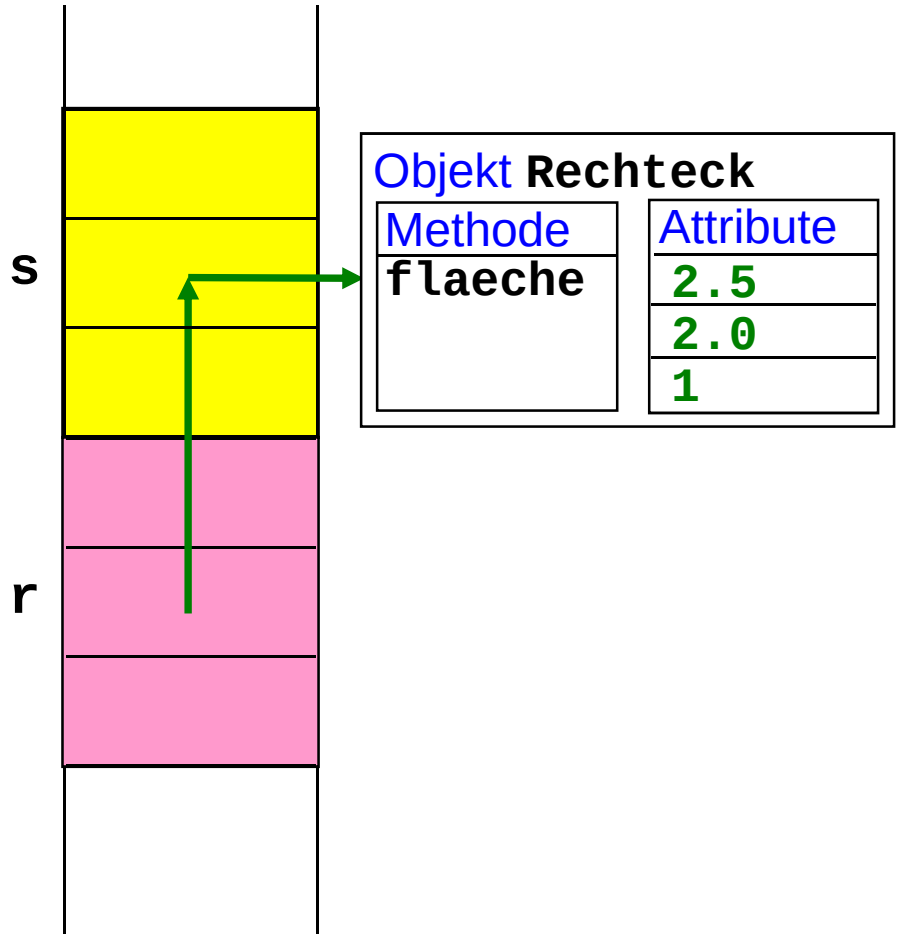
Echtes call by reference

```
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;
}
```



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);
  }
}
```

2. Methoden

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

vararg Parameter

- **vararg: variable length argument list**
  - Methoden mit beliebiger Anzahl von Argumenten

```
public static int addiere (int... args) {
      int x = 0;
      for (int i : args)
            x += i;
      return x;
}
```

args ist vom Typ **int []**

- **addiere (2, 3, 4)** ergibt **9**
- **addiere ()** ergibt **0**
- **addiere (new int [] {2,3,4})** ergibt **9**

vararg Parameter

- **vararg: variable length argument list**
  - Methoden mit beliebiger Anzahl von Argumenten

```
public static int add_mult (int y, int... args) {
      int x = 0;
      for (int i : args)
            x += i;
      return x * y;
}
```

- **`add_mult (2, 3, 4)`** ergibt **`14`**
- **`add_mult ()`** ergibt **`Typfehler`**
- **`add_mult (2)`** ergibt **`0`**

2. Methoden

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

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);
```

Aufzählungstypen

- Typ mit endlich vielen (wenigen) Objekten

```
enum Tag {MO, DI, MI, DO, FR, SA, SO}
```

```
class Tag {
      final static Tag MO = new Tag ();
      final static Tag DI = new Tag ();
                        ...              }
```

- Verwendung ähnlich wie andere Klassen

```
Tag t = Tag.MO;

Tag s;
s = Tag.DI;
```

Aufzählungstypen

- Typ mit endlich vielen (wenigen) Objekten

```
enum Tag {MO, DI, MI, DO, FR, SA, SO}
```

```
class Tag {
      final static Tag MO = new Tag ();
      final static Tag DI = new Tag ();
                        ...              }
```

- Vordefinierte Methoden
  - `static Tag valueOf (String s)`
    - `valueOf("MO")` ergibt `Tag.MO`
  - `static Tag [] values ()`
    - `Tag.values()` ergibt `Array {MO, DI, ..., SO}`
  - `int ordinal()`
    - `Tag.DO.ordinal()` ergibt `3`

Aufzählungstypen

```
public enum Tag {

      MO, DI, MI, DO, FR, SA, SO;

      public static boolean istWochenende (Tag t) {
            return t == SA || t == SO;
      }

      public static void aktivitaet (Tag t) {
            switch (t) {
                  case SO:
                        System.out.println("ausruhen"); break;
                  case SA:
                        System.out.println("Hausputz"); break;
                  case DO: case FR:
                        System.out.println("aufraeumen"); break;
                  default:
                        System.out.println("arbeiten"); break;
            }
}
```

Klassen in Java

Klassen werden für verschiedene Zwecke verwendet:

- **1. Datentypen**
  - (Bsp: **`Rechteck`**, **`Tag`**)
- **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)`**, **`Tag.MO`**...

toString - Methode

```
public class Rechteck {

  ...
 public String toString () {
    return  "Laenge: " + laenge +
            ", Breite: " + breite +
            ", Strichstaerke: " + 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);
```

2. Methoden

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

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;    }
```