

Grundlagen der Programmierung

VL07: Methoden

Prof. Dr. Samuel Kounev
Jóakim v. Kistowski
Norbert Schmitt

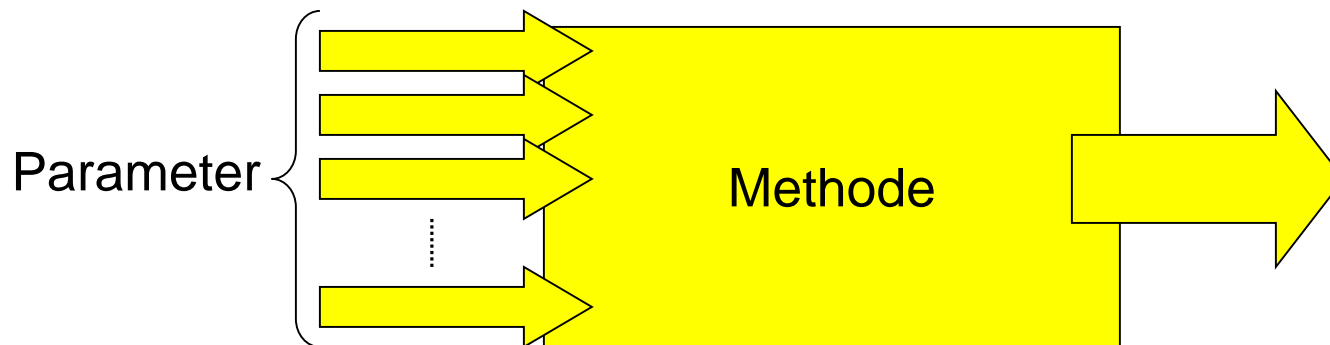


Inhalt

- Methoden, Bedeutung
- Deklaration und Aufruf von Methoden
- Parameterübergabe und -rückgabe
- Sichtbarkeit und Verdecken
- Überladen von Methoden, Signatur
- Aufrufbarkeit
- Variable Argument-Anzahl
- Methode `main`
- Methodenaufruf aus anderen Klassen
- Klassenmethoden
- Instanzmethoden (werden nur erwähnt – Einzelheiten später)

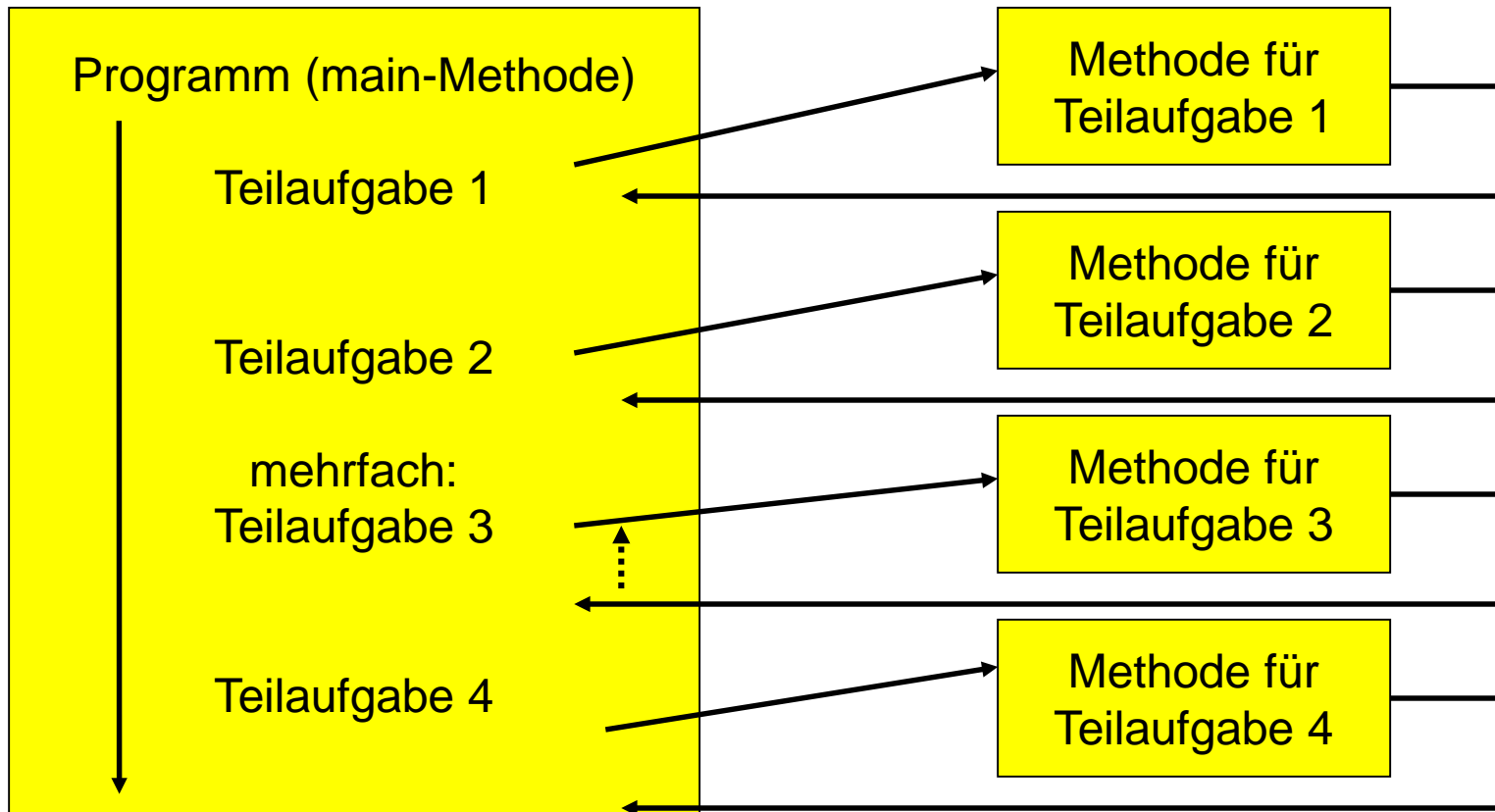
Methoden

- fassen Teile des ausführbaren Programmcodes zu **Unterprogrammen** zusammen
- können durch einen **Methodenaufruf** ausgeführt werden
- können **Parameter (Argumente)** übergeben bekommen
- können ein Ergebnis (**Rückgabewert**) zurückliefern

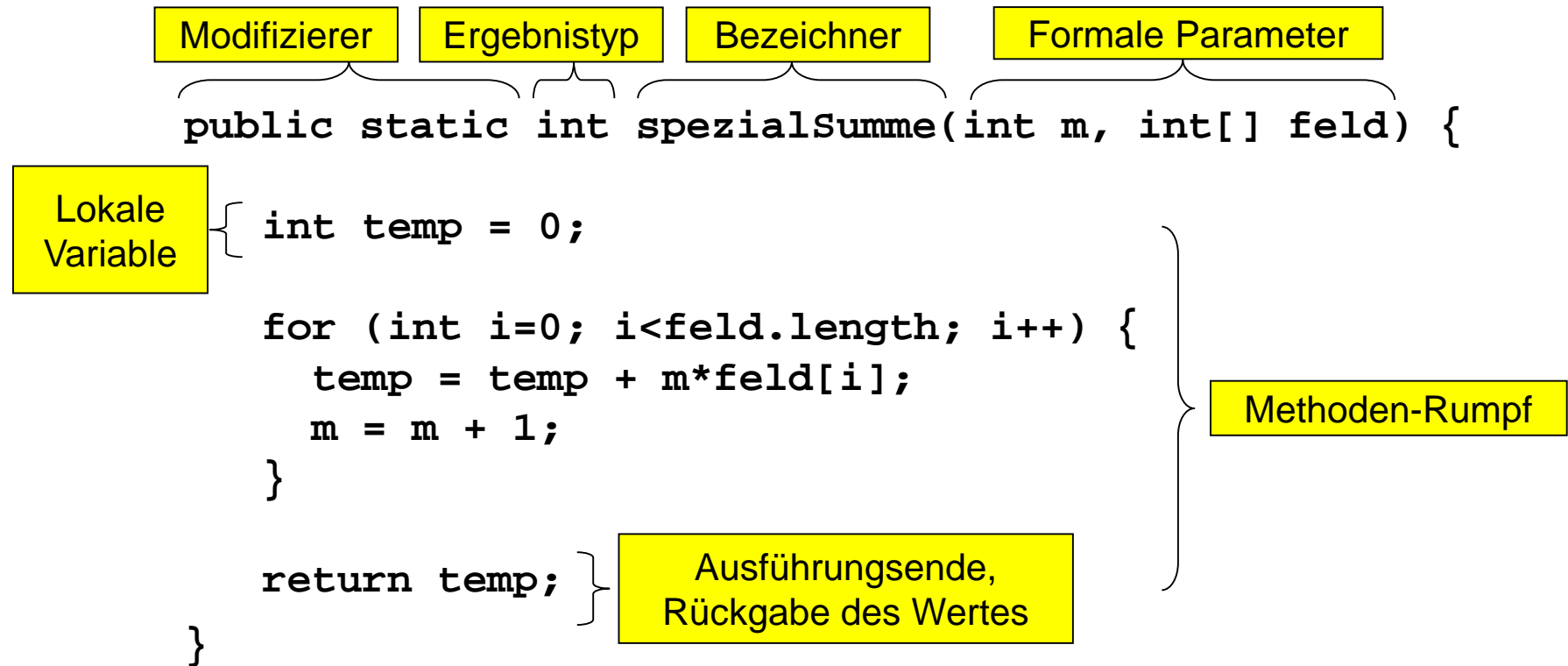


Bedeutung von Methoden

- Strukturierung bzw. Modularisierung des Programms
- Reduzierung des Programmieraufwands
- Leichtere Anpassbarkeit von Programmen



Am Beispiel



- Steht in einem Ausdruck ein Methodenaufruf, so wird bei der Auswertung des Ausdrucks an der entsprechenden Stelle diese Methode ausgeführt und mit dem Rückgabewert weitergerechnet

- Am Beispiel

...

```
int k, ergebnis;
```

```
int[] x = new int[20];
```

```
Scanner input = new Scanner(System.in);
```

```
System.out.println("k = ");
```

```
k = input.nextInt();
```

```
for (int i=0; i<x.length; i++) {
```

```
    System.out.println("x[" + i + "] = ");
```

```
    x[i] = input.nextInt();
```

```
}
```

```
ergebnis = 5 - 3*spezialSumme(3*k,x);
```

...

Methoden-Name

aktuelle Parameter

```
public class MyMethods { // Eine Methoden-Sammlung

    // (1) Quadrat-Funktion
    public static double sqr(double x) {
        return x*x;
    }

    // (2) Quersummen-Funktion
    public static int quersumme (int k) {
        int qs = 0;
        while (k != 0) {
            qs = qs + k % 10;
            k  = k / 10;
        }
        return qs;
    }
}
```

```
// (3) Fakultäts-Funktion
public static long fakultaet(long k) {
    if (k <= 1)
        return 1;
    else {
        long f = 1;
        while (k > 1) {
            f = f * k;
            k--;
        }
        return f;
    }
}
```



```
// (4) Mittelwert-Funktion für Felder
public static double mittelwert(double[] feld) {
    double mw = 0;
    for (int i=0; i < feld.length; i++)
        mw = mw + feld[i];
    return mw / feld.length;
}
```

```
// (5) Kopier-Funktion für Felder (flache Kopie)
public static double[] kopie(double[] x) {
    double[] y = new double[x.length];
    for (int i=0; i < x.length; i++)
        y[i] = x[i];
    return y;
}
```

```
// (6) Vereinfachte println-Methode
public static void println(String s) {
    System.out.println(s);
}

// (7) Wurzel-2-Funktion
public static double wurzel2() {
    return Math.sqrt(2);
}

// (8) Nichtstun-Methode :-)
public static void tuNix() {
    return; // kann auch entfallen!
}

// (9) Testfunktion für eine gerade Zahl
public static boolean istGerade(long k) {
    return (k % 2 == 0);
}
```

```
// (10) main-Methode testet alle obigen Methoden
public static void main(String[] args) {
    double x = 1.7, y;
    long k;
    double[] feldchen = {1.0, 2.0, 3.0, 4.0};
    double[] nochEins;
    y = sqr(x);
    System.out.println(quersumme(12345));
    k = fakultaet(12/3);
    x = mittelwert(feldchen);
    nochEins = kopie(feldchen);
    println("I say hello " + 2 + " u");
    println("Wurzel aus 2 ist " + wurzel2());
    tuNix();
    if (istGerade(k))
        println(k + " ist eine gerade Zahl");
    // k = fakultaet(x);      //unzulässig
    // println(y);           //unzulässig
}
} // Ende der Klasse MyMethods
```

- Erzeuge die formalen Parameter-Variablen der Methode
- Initialisiere diese mit den *Werten* der aktuellen Parameter (Argumente) wie sie beim Aufruf angegeben sind
- Führe den Rumpf der Methode aus
- Gib den Rückgabewert (Ergebniswert) zurück

Call-by-Value / Pass-by-Value Semantik

Die Methode arbeitet mit *Kopien* der aktuellen Parameter, so dass die aktuellen Parameter nicht verändert werden!

Aber Vorsicht

Ist der aktuelle Parameter eine **Referenz**, so können auch von der Methode aus über diese Referenz auf Objekte der aufrufenden Umgebung zugegriffen und dort Veränderungen verursacht werden!

Definition siehe Folie 5

Wird also die Methode

```
public static int spezialSumme(int m, int[] feld)
```

aufgerufen in der Form

```
ergebnis = 5 - 3*spezialSumme(3*k,x);
```

so wird folgendes passieren:

In der Methode `spezialSumme` erhalten

`m` den Wert von `3*k`

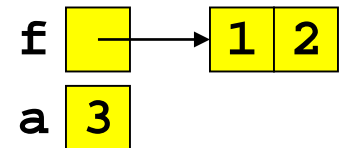
`feld` den Wert von `x` (Referenzkopie!)

Die Anweisungen der Methode werden ausgeführt.

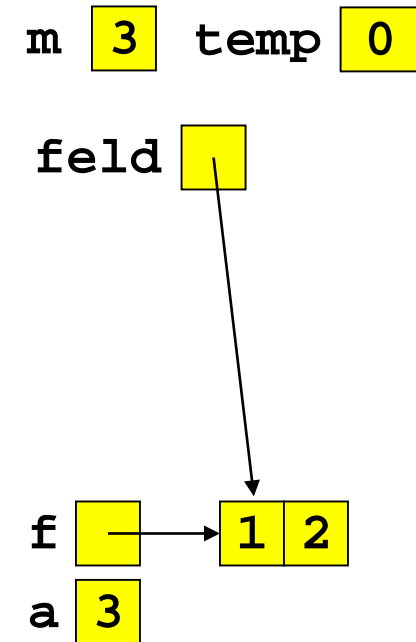
Der Rückgabewert (`rgw`) wird erzeugt. Der Ausdruck wird unter Verwendung von `rgw` weiter ausgewertet gemäß

```
ergebnis = 5 - 3*rgw;
```

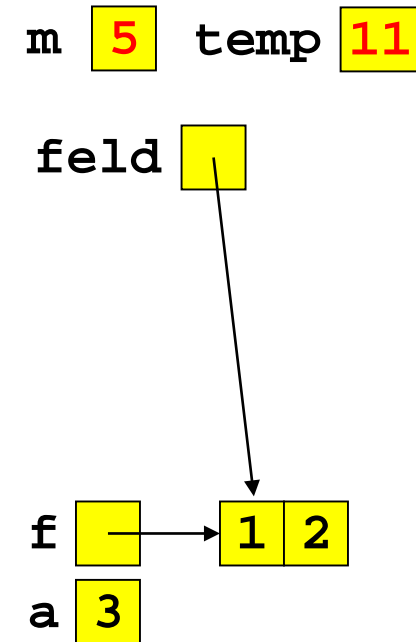
```
public class MethodenAblaufTest {
    public static int
        spezialSumme(int m, int[] feld) {
        int temp = 0;
        for (int i=0; i<feld.length; i++) {
            temp = temp + m*feld[i];
            m = m + 1;
        }
        feld[1] = 5;
        return temp;
    }
    public static void main (String[] args) {
        int[] f = {1, 2};
        int a = 3;
        int s = spezialSumme(a, f);
        System.out.println("f[1] = " + f[1]);
        System.out.println("a = " + a);
        System.out.println("s = " + s);
    }
}
```



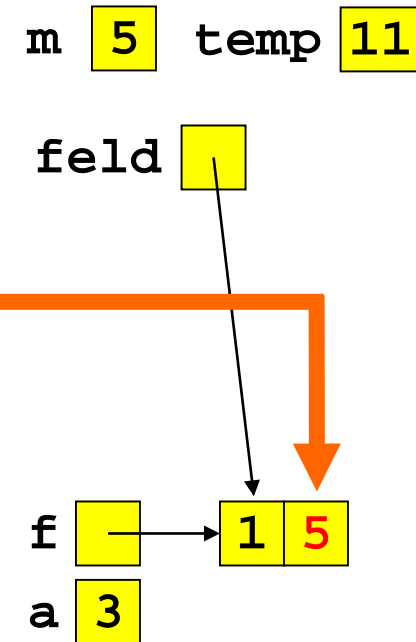
```
public class MethodenAblaufTest {
    public static int
        spezialSumme(int m, int[] feld) {
        int temp = 0;
        for (int i=0; i<feld.length; i++) {
            temp = temp + m*feld[i];
            m = m + 1;
        }
        feld[1] = 5;
        return temp;
    }
    public static void main (String[] args) {
        int[] f = {1, 2};
        int a = 3;
        int s = spezialSumme(a, f);
        System.out.println("f[1] = " + f[1]);
        System.out.println("a = " + a);
        System.out.println("s = " + s);
    }
}
```



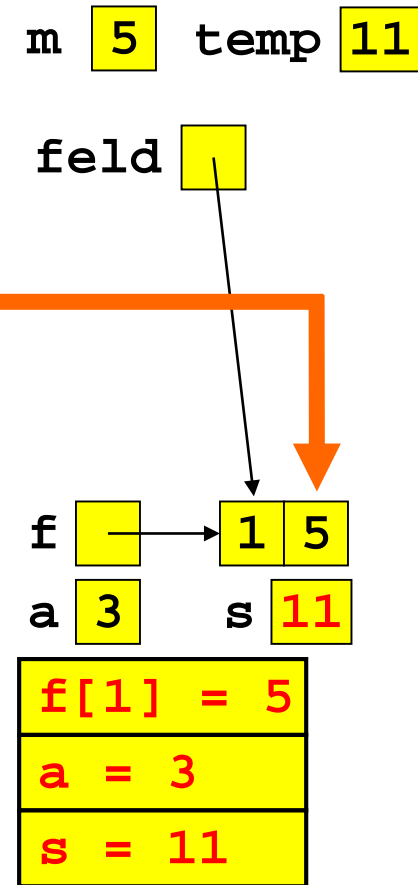
```
public class MethodenAblaufTest {  
    public static int  
        spezialSumme(int m, int[] feld) {  
        int temp = 0;  
        for (int i=0; i<feld.length; i++) {  
            temp = temp + m*feld[i];  
            m = m + 1;  
        }  
        feld[1] = 5;  
        return temp;  
    }  
    public static void main (String[] args) {  
        int[] f = {1, 2};  
        int a = 3;  
        int s = spezialSumme(a, f);  
        System.out.println("f[1] = " + f[1]);  
        System.out.println("a = " + a);  
        System.out.println("s = " + s);  
    }  
}
```



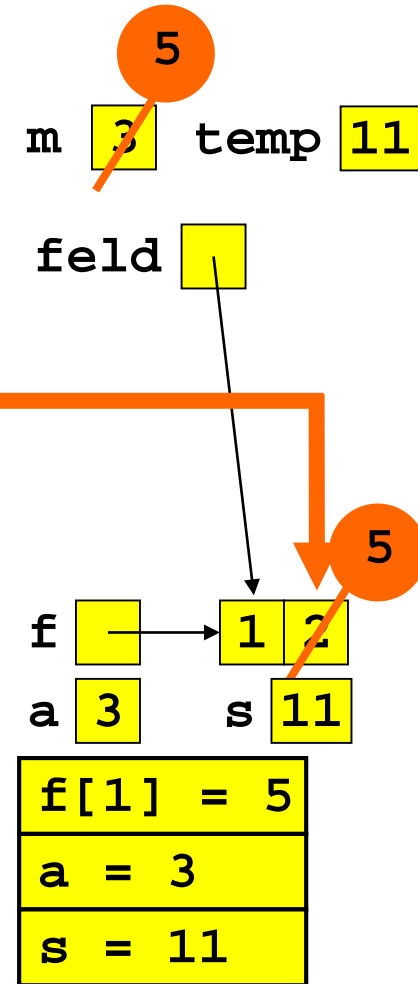

```
public class MethodenAblaufTest {
    public static int
        spezialSumme(int m, int[] feld) {
        int temp = 0;
        for (int i=0; i<feld.length; i++) {
            temp = temp + m*feld[i];
            m = m + 1;
        }
        feld[1] = 5;
        return temp;
    }
    public static void main (String[] args) {
        int[] f = {1, 2};
        int a = 3;
        int s = spezialSumme(a, f);
        System.out.println("f[1] = " + f[1]);
        System.out.println("a = " + a);
        System.out.println("s = " + s);
    }
}
```



```
public class MethodenAblaufTest {  
    public static int  
        spezialSumme(int m, int[] feld) {  
        int temp = 0;  
        for (int i=0; i<feld.length; i++) {  
            temp = temp + m*feld[i];  
            m = m + 1;  
        }  
        feld[1] = 5;  
        return temp;  
    }  
    public static void main (String[] args) {  
        int[] f = {1, 2};  
        int a = 3;  
        int s = spezialSumme(a, f);  
        System.out.println("f[1] = " + f[1]);  
        System.out.println("a = " + a);  
        System.out.println("s = " + s);  
    }  
}
```



```
public class MethodenAblaufTest {
    public static int
        spezialSumme(int m, int[] feld) {
        int temp = 0;
        for (int i=0; i<feld.length; i++) {
            temp = temp + m*feld[i];
            m = m + 1;
        }
        feld[1] = 5;
        return temp;
    }
    public static void main (String[] args) {
        int[] f = {1, 2};
        int a = 3;
        int s = spezialSumme(a, f);
        System.out.println("f[1] = " + f[1]);
        System.out.println("a = " + a);
        System.out.println("s = " + s);
    }
}
```



```
public class TauschenTest {
    public static void tausche1(int a, int b) {
        int help = a;
        a = b; b = help;
    }
    public static void tausche2(int[] p, int[] q) {
        int[] help = p;
        p = q; q = help;
    }
    public static void tausche3(int[] p, int[] q) {
        int help = p[0];
        p[0] = q[0]; q[0] = help;
    }
    public static void main(String[] args) {
        int x = 5, y = 100;
        int[] f = {7}, g = {5};
        tausche1(x, y);
        System.out.println("x=" + x + " y=" + y);
        tausche2(f, g);
        System.out.println("f[0]=" + f[0] + " g[0]=" + g[0]);
        tausche3(f, g);
        System.out.println("f[0]=" + f[0] + " g[0]=" + g[0]);
    }
}
```

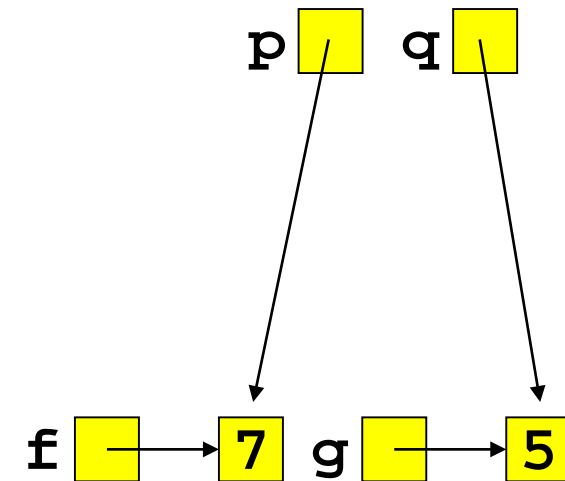
```
public class TauschenTest {  
    public static void tausche1(int a, int b) {  
        int help = a;  
        a = b; b = help;  
    }  
    public static void tausche2(int[] p, int[] q) {  
        int[] help = p;  
        p = q; q = help;  
    }  
    public static void tausche3(int[] p, int[] q) {  
        int help = p[0];  
        p[0] = q[0]; q[0] = help;  
    }  
    public static void main(String[] args) {  
        int x = 5, y = 100;  
        int[] f = {7}, g = {5};  
        tausche1(x, y);  
        System.out.println("x=" + x + " y=" + y);  
        tausche2(f, g);  
        System.out.println("f[0]=" + f[0] + " g[0]=" + g[0]);  
        tausche3(f, g);  
        System.out.println("f[0]=" + f[0] + " g[0]=" + g[0]);  
    }  
}
```

```
a = 5, b = 100  
help = 5  
a = 100  
b = 5
```

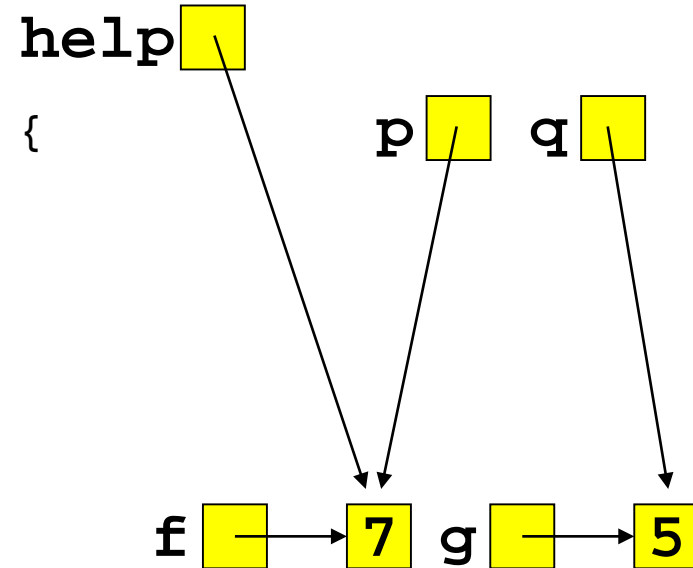
```
public class TauschenTest {  
    public static void tausche1(int a, int b) {  
        int help = a;  
        a = b; b = help;  
    }  
    public static void tausche2(int[] p, int[] q) {  
        int[] help = p;  
        p = q; q = help;  
    }  
    public static void tausche3(int[] p, int[] q) {  
        int help = p[0];  
        p[0] = q[0]; q[0] = help;  
    }  
    public static void main(String[] args) {  
        int x = 5, y = 100;  
        int[] f = {7}, g = {5};  
        tausche1(x, y);  
        System.out.println("x=" + x + " y=" + y);  
        tausche2(f, g);  
        System.out.println("f[0]=" + f[0] + " g[0]=" + g[0]);  
        tausche3(f, g);  
        System.out.println("f[0]=" + f[0] + " g[0]=" + g[0]);  
    }  
}
```

x = 5, y = 100

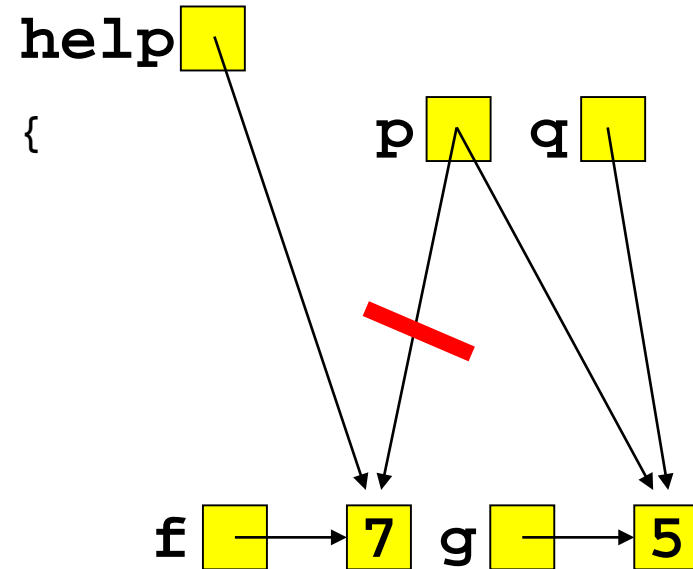
```
public class TauschenTest {  
    public static void tausche1(int a, int b) {  
        int help = a;  
        a = b; b = help;  
    }  
    public static void tausche2(int[] p, int[] q) {  
        int[] help = p;  
        p = q; q = help;  
    }  
    public static void tausche3(int[] p, int[] q) {  
        int help = p[0];  
        p[0] = q[0]; q[0] = help;  
    }  
    public static void main(String[] args) {  
        int x = 5, y = 100;  
        int[] f = {7}, g = {5};  
        tausche1(x, y);  
        System.out.println("x=" + x + " y=" + y);  
        tausche2(f, g);  
        System.out.println("f[0]=" + f[0] + " g[0]=" + g[0]);  
        tausche3(f, g);  
        System.out.println("f[0]=" + f[0] + " g[0]=" + g[0]);  
    }  
}
```



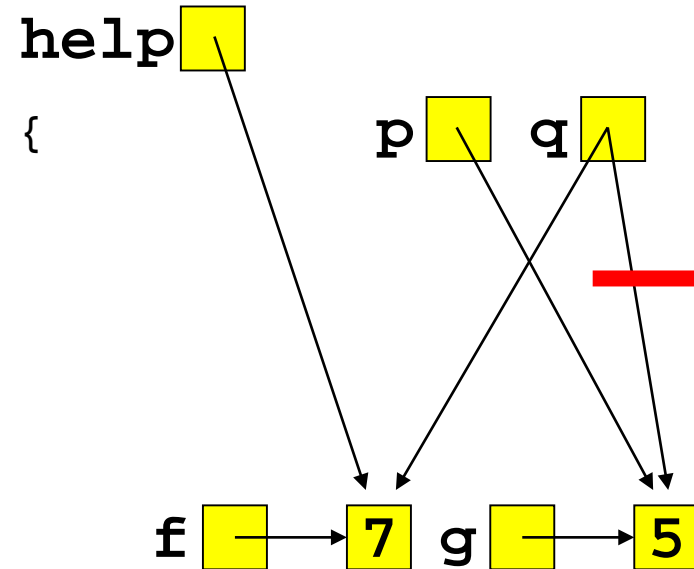
```
public class TauschenTest {  
    public static void tausche1(int a, int b) {  
        int help = a;  
        a = b; b = help;  
    }  
    public static void tausche2(int[] p, int[] q) {  
        int[] help = p;  
        p = q; q = help;  
    }  
    public static void tausche3(int[] p, int[] q) {  
        int help = p[0];  
        p[0] = q[0]; q[0] = help;  
    }  
    public static void main(String[] args) {  
        int x = 5, y = 100;  
        int[] f = {7}, g = {5};  
        tausche1(x, y);  
        System.out.println("x=" + x + " y=" + y);  
        tausche2(f, g);  
        System.out.println("f[0]=" + f[0] + " g[0]=" + g[0]);  
        tausche3(f, g);  
        System.out.println("f[0]=" + f[0] + " g[0]=" + g[0]);  
    }  
}
```




```
public class TauschenTest {  
    public static void tausche1(int a, int b) {  
        int help = a;  
        a = b; b = help;  
    }  
    public static void tausche2(int[] p, int[] q) {  
        int[] help = p;  
        p = q; q = help;  
    }  
    public static void tausche3(int[] p, int[] q) {  
        int help = p[0];  
        p[0] = q[0]; q[0] = help;  
    }  
    public static void main(String[] args) {  
        int x = 5, y = 100;  
        int[] f = {7}, g = {5};  
        tausche1(x, y);  
        System.out.println("x=" + x + " y=" + y);  
        tausche2(f, g);  
        System.out.println("f[0]=" + f[0] + " g[0]=" + g[0]);  
        tausche3(f, g);  
        System.out.println("f[0]=" + f[0] + " g[0]=" + g[0]);  
    }  
}
```



```
public class TauschenTest {  
    public static void tausche1(int a, int b) {  
        int help = a;  
        a = b; b = help;  
    }  
    public static void tausche2(int[] p, int[] q) {  
        int[] help = p;  
        p = q; q = help;  
    }  
    public static void tausche3(int[] p, int[] q) {  
        int help = p[0];  
        p[0] = q[0]; q[0] = help;  
    }  
    public static void main(String[] args) {  
        int x = 5, y = 100;  
        int[] f = {7}, g = {5};  
        tausche1(x, y);  
        System.out.println("x=" + x + " y=" + y);  
        tausche2(f, g);  
        System.out.println("f[0]=" + f[0] + " g[0]=" + g[0]);  
        tausche3(f, g);  
        System.out.println("f[0]=" + f[0] + " g[0]=" + g[0]);  
    }  
}
```

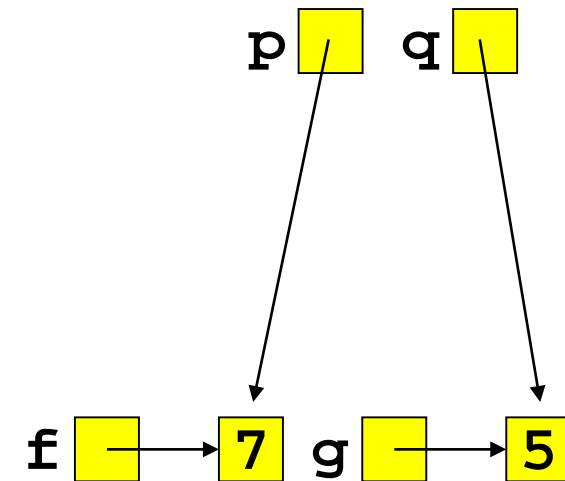


```
public class TauschenTest {  
    public static void tausche1(int a, int b) {  
        int help = a;  
        a = b; b = help;  
    }  
    public static void tausche2(int[] p, int[] q) {  
        int[] help = p;  
        p = q; q = help;  
    }  
    public static void tausche3(int[] p, int[] q) {  
        int help = p[0];  
        p[0] = q[0]; q[0] = help;  
    }  
    public static void main(String[] args) {  
        int x = 5, y = 100;  
        int[] f = {7}, g = {5};  
        tausche1(x, y);  
        System.out.println("x=" + x + " y=" + y);  
        tausche2(f, g);  
        System.out.println("f[0]=" + f[0] + " g[0]=" + g[0]);  
        tausche3(f, g);  
        System.out.println("f[0]=" + f[0] + " g[0]=" + g[0]);  
    }  
}
```

f  → 7 g  → 5

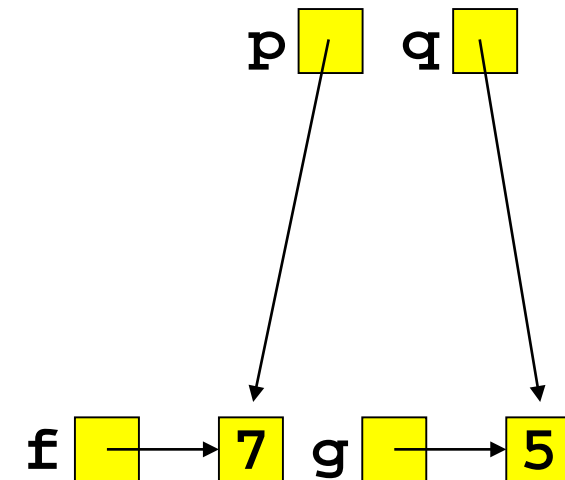
f[0]=7 g[0]=5

```
public class TauschenTest {  
    public static void tausche1(int a, int b) {  
        int help = a;  
        a = b; b = help;  
    }  
    public static void tausche2(int[] p, int[] q) {  
        int[] help = p;  
        p = q; q = help;  
    }  
    public static void tausche3(int[] p, int[] q) {  
        int help = p[0];  
        p[0] = q[0]; q[0] = help;  
    }  
    public static void main(String[] args) {  
        int x = 5, y = 100;  
        int[] f = {7}, g = {5};  
        tausche1(x, y);  
        System.out.println("x=" + x + " y=" + y);  
        tausche2(f, g);  
        System.out.println("f[0]=" + f[0] + " g[0]=" + g[0]);  
        tausche3(f, g);  
        System.out.println("f[0]=" + f[0] + " g[0]=" + g[0]);  
    }  
}
```



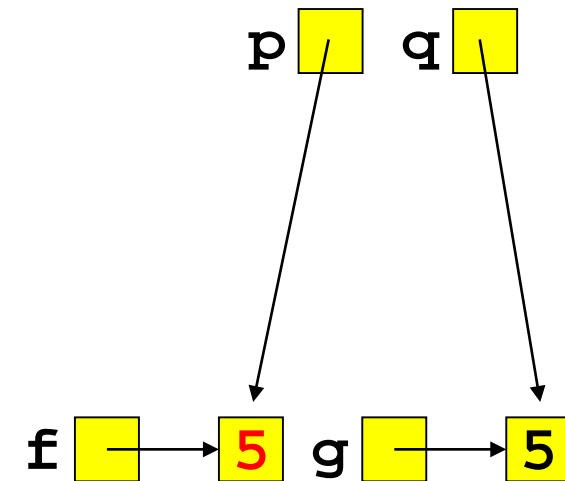
```
public class TauschenTest {  
    public static void tausche1(int a, int b) {  
        int help = a;  
        a = b; b = help;  
    }  
    public static void tausche2(int[] p, int[] q) {  
        int[] help = p;  
        p = q; q = help;  
    }  
    public static void tausche3(int[] p, int[] q) {  
        int help = p[0];  
        p[0] = q[0]; q[0] = help;  
    }  
    public static void main(String[] args) {  
        int x = 5, y = 100;  
        int[] f = {7}, g = {5};  
        tausche1(x, y);  
        System.out.println("x=" + x + " y=" + y);  
        tausche2(f, g);  
        System.out.println("f[0]=" + f[0] + " g[0]=" + g[0]);  
        tausche3(f, g);  
        System.out.println("f[0]=" + f[0] + " g[0]=" + g[0]);  
    }  
}
```

help 7



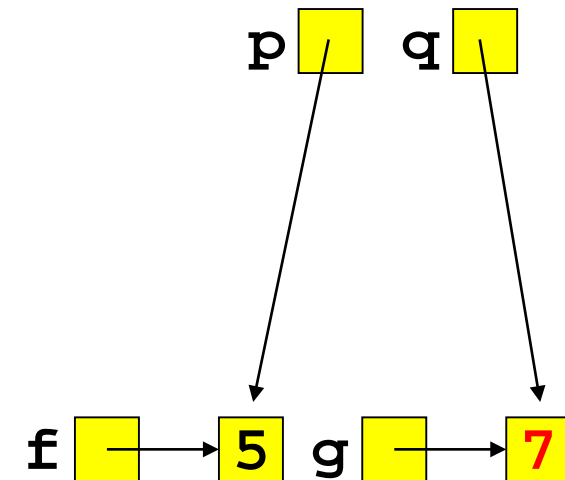
```
public class TauschenTest {  
    public static void tausche1(int a, int b) {  
        int help = a;  
        a = b; b = help;  
    }  
    public static void tausche2(int[] p, int[] q) {  
        int[] help = p;  
        p = q; q = help;  
    }  
    public static void tausche3(int[] p, int[] q) {  
        int help = p[0];  
        p[0] = q[0]; q[0] = help;  
    }  
    public static void main(String[] args) {  
        int x = 5, y = 100;  
        int[] f = {7}, g = {5};  
        tausche1(x, y);  
        System.out.println("x=" + x + " y=" + y);  
        tausche2(f, g);  
        System.out.println("f[0]=" + f[0] + " g[0]=" + g[0]);  
        tausche3(f, g);  
        System.out.println("f[0]=" + f[0] + " g[0]=" + g[0]);  
    }  
}
```

help 7



```
public class TauschenTest {  
    public static void tausche1(int a, int b) {  
        int help = a;  
        a = b; b = help;  
    }  
    public static void tausche2(int[] p, int[] q) {  
        int[] help = p;  
        p = q; q = help;  
    }  
    public static void tausche3(int[] p, int[] q) {  
        int help = p[0];  
        p[0] = q[0]; q[0] = help;  
    }  
    public static void main(String[] args) {  
        int x = 5, y = 100;  
        int[] f = {7}, g = {5};  
        tausche1(x, y);  
        System.out.println("x=" + x + " y=" + y);  
        tausche2(f, g);  
        System.out.println("f[0]=" + f[0] + " g[0]=" + g[0]);  
        tausche3(f, g);  
        System.out.println("f[0]=" + f[0] + " g[0]=" + g[0]);  
    }  
}
```

help 7



```
public class TauschenTest {
    public static void tausche1(int a, int b) {
        int help = a;
        a = b; b = help;
    }
    public static void tausche2(int[] p, int[] q) {
        int[] help = p;
        p = q; q = help;
    }
    public static void tausche3(int[] p, int[] q) {
        int help = p[0];
        p[0] = q[0]; q[0] = help;
    }
    public static void main(String[] args) {
        int x = 5, y = 100;
        int[] f = {7}, g = {5};
        tausche1(x, y);
        System.out.println("x=" + x + " y=" + y);
        tausche2(f, g);
        System.out.println("f[0]=" + f[0] + " g[0]=" + g[0]);
        tausche3(f, g);
        System.out.println("f[0]=" + f[0] + " g[0]=" + g[0]);
    }
}
```

f  → 5 g  → 7

f[0]=5 g[0]=7

Ausgaben

`x=5 y=100`

`f[0]=7 g[0]=5`

`f[0]=5 g[0]=7`

Begründung

Alle drei Methoden vertauschen etwas:

`tausche1` vertauscht nur ihre lokalen Kopien `a` und `b` der Werte `x` und `y` aus der Methode `main`. `x` und `y` bleiben unverändert!




`tausche2` vertauscht nur ihre lokalen Kopien `p` und `q` der Referenzen `f` und `g` aus der Methode `main`. `f` und `g` bleiben unverändert!

`tausche3` vertauscht durch Referenzzugriff über die lokalen Kopien `p` und `q` der Referenzen `f` und `g` aus der Methode `main` die Feldinhalte der durch `f` und `g` referenzierten Felder!

```
public class VerdeckenTest {  
    static int a = 1, b = 2, c = 3;  
    static int m(int a) {  
        int b = 20;  
        System.out.println("a = " + a);  
        System.out.println("b = " + b);  
        System.out.println("c = " + c);  
        return 100;  
    }  
    public static void main (String[] args) {  
        int a = 1000;  
        System.out.println("a = " + a);  
        System.out.println("b = " + b);  
        System.out.println("m(c)= " + m(c));  
    }  
}
```

Sichtbarkeit und Verdecken

In der Klasse `VerdeckenTest` werden verwendet

-  Klassenvariablen
-  formale Argument-Variablen (formale Parameter)
-  lokale Variablen




Innerhalb von Methoden verdecken lokale Variablen und formale Variablen die Klassenvariablen

a = 3 formaler Parameter a (Kopie von Klassen-c)

```
public class VerdeckenTest {  
    static int a = 1, b = 2, c = 3;  
    static int m(int a) {  
        int b = 20;  
        System.out.println("a = " + a);  
        System.out.println("b = " + b);  
        System.out.println("c = " + c);  
        return 100;  
    }  
    public static void main (String[] args) {  
        int a = 1000;  
        System.out.println("a = " + a);  
        System.out.println("b = " + b);  
        System.out.println("m(c)= " + m(c));  
    }  
}
```

Sichtbarkeit und Verdecken

In der Klasse `VerdeckenTest` werden verwendet

-  Klassenvariablen
-  formale Argument-Variablen (formale Parameter)
-  lokale Variablen

`a = 3` Parameter a (Kopie von Klassen-c)

`b = 20` lokales b (verdeckt Klassen-b)




`c = 3` Klassen-c

`a = 3` formaler Parameter a (Kopie von Klassen-c)

`a = 1000` lokales a (verdeckt Klassen-a)

`b = 2` Klassen-b

`m(c) = 100`
Ergebniswert des m-Aufrufs

- In der Klasse **VerdeckenTest** werden verwendet
 -  Klassenvariablen
 -  formale Argument-Variablen (formale Parameter)
 -  lokale Variablen
- Innerhalb von Methoden verdecken lokale Variablen und formale Variablen die Klassenvariablen
- Beim Start von **VerdeckenTest** ergeben sich folgende Ausgaben:

Ausgaben	Begründung
a = 1000	lokales a (verdeckt Klassen- a)
b = 2	Klassen- b
a = 3	formaler Parameter a (Kopie von Klassen- c)
b = 20	lokales b (verdeckt Klassen- b)
c = 3	Klassen- c
m(c) = 100	Ergebniswert des m -Aufrufs

- Als Elemente einer Klasse deklarierte **Variablen** müssen **verschiedene** Namen erhalten!

Beispiel:

```
class A {  
    double y = 18.48;  
    int     y = 12;    // Fehler : Doppeldeklaration  
    ...  
}
```

- Methoden können dagegen mit gleichem Namen vorkommen, wenn sie sich in Ihrer *Signatur* (siehe nachfolgende Definition) unterscheiden. Man spricht dann vom **Überladen** von Methoden (*method overloading*).

- Deklaration von mehreren Methoden mit dem gleichen Namen aber unterschiedlicher Parameter-Liste
- Unterscheidung der überladenen Methoden anhand
 - der Anzahl,
 - der Datentypen und
 - der Reihenfolge

der Parameter (bilden zusammen mit dem Methoden-Namen die so genannte **Signatur** der Methode)

- **Achtung:** Die Namen der Parameter oder die Ergebnistypen der Methoden werden **nicht** zur Unterscheidung herangezogen!

- Bekanntes Beispiel für Überladung: Math.abs

```
public static double abs (double x)
```

```
public static float abs (float x)
```

```
public static long abs (long x)
```

```
public static int abs (int x)
```

```
public static void drucke(float z) {  
    System.out.println("Eine float-Zahl: " + z);  
}
```

```
public static void drucke(int a) {  
    System.out.println("Eine int-Zahl: " + a);  
}
```

```
public static void drucke(int a, int b) {  
    int c = a + b;  
    System.out.println("Summe: " + c);  
}
```

```
public static void main(String[] args) {  
    float f = 1.2f;  
    drucke(1);  
    drucke(f);  
    drucke(4.2f);  
    drucke((int)f, 4);  
}
```

Die **Signatur** einer Methode besteht aus Ihrem **Namen** gefolgt von **der Folge der Typen der formalen Parameter** aus Ihrer Deklaration (in der originalen Reihenfolge unter Beachtung der originalen Anzahlen)

Beispiele:

```
int funktion1() { ... }
```

Signatur: `funktion1()`

```
double funktion1() { ... }
```

Signatur: `funktion1()`

Problem:
Signatur für beide gleich!


```
char buchstaben(char a, int i, String s) { ... }
```

Signatur: **buchstaben(char, int, String)**

```
char buchstaben(char a, int i, int j, String s) {  
    ...  
}
```

Signatur: **buchstaben (char, int, int, String)**

```
int berechne(double x, double y, double z) {  
    ...  
}
```

```
double berechne(double u, double v, double w) {  
    ...  
}
```

Problem:
Signatur für beide gleich!

berechne (double, double, double)

Überladen von Methoden (innerhalb einer Klasse)

- Innerhalb einer Klasse dürfen Methoden gleichen Namens vorkommen, sofern sich deren Signaturen unterscheiden
- Eine Klassendeklaration darf nicht mehr als eine Methode mit einer bestimmten Signatur enthalten
- Ergebnistyp und andere Charakteristika einer Methode sind dabei nicht von Bedeutung

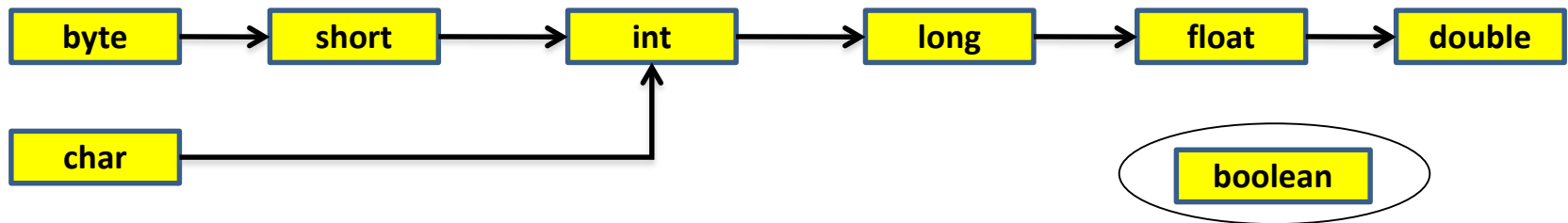
```
class Ueberladen {  
    int g() { ... }  
    int g( int i ) { ... }  
    int g( long i ) { ... }  
    int g( int i, int j ) { ... }  
    int g( double x, int j ) { ... }  
    int g( double y, int j, char a ) { ... }  
    duoble g( double x, double y ) { ... }  
    double g( int j, double x ) { ... }  
    double g( long x ) { ... }  
        // Fehler int g( long i )  
        // existiert bereits  
    ...  
}
```

Überladen von Methoden (innerhalb einer Klasse)

- Kommen mehrere Methoden mit **demselben Namen**, aber **verschiedenen Signaturen** innerhalb einer Klasse vor (es können sowohl deklarierte, als auch geerbte^[1] Methoden sein), so ist der betreffende Name **überladen**.

[1] Vererbung von Methoden wird später behandelt - siehe Abschnitt Objektorientierung

- Beim Aufruf einer überladenen Methode untersucht Java alle Methoden (auch geerbte – wird später behandelt)
 - welche den entsprechenden überladenen Methodennamen tragen,
 - deren Parameteranzahl mit der Argumentanzahl übereinstimmt,
 - für welche die Argumenttypen des Aufrufs durch elementare Typvergrößerungen oder Vergrößerungen von Referenztypen in die Parametertypen umwandelbar sind oder mit diesen übereinstimmen,
 - ob sie zugreifbar sind (abhängig von Modifikatoren `public`).



- Obige Methoden gelten als zum gegebenen aufrufenden Ausdruck **aufrufbare** Methoden. Kann keine aufrufbare Methode gefunden werden, so erfolgt eine **Fehlermeldung** seitens des Compilers.

- Ist für einen Methodenaufruf mehr als eine Methode aufrufbar, so wird zur Laufzeit die **spezifischste** ausgewählt
- Sei $m(S_1, \dots, S_n)$ und $m(T_1, \dots, T_n)$ ein Paar aufrufbarer Methoden. Dabei sei m der Name der Methode und S_i bzw. T_i (mit $1 \leq i \leq n$) seien die Typen der formalen Parameter

$m(S_1, \dots, S_n)$ heißt **spezifischer** als $m(T_1, \dots, T_n)$,

falls für alle i (mit $1 \leq i \leq n$):

T_i durch **elementare Typvergrößerung** oder

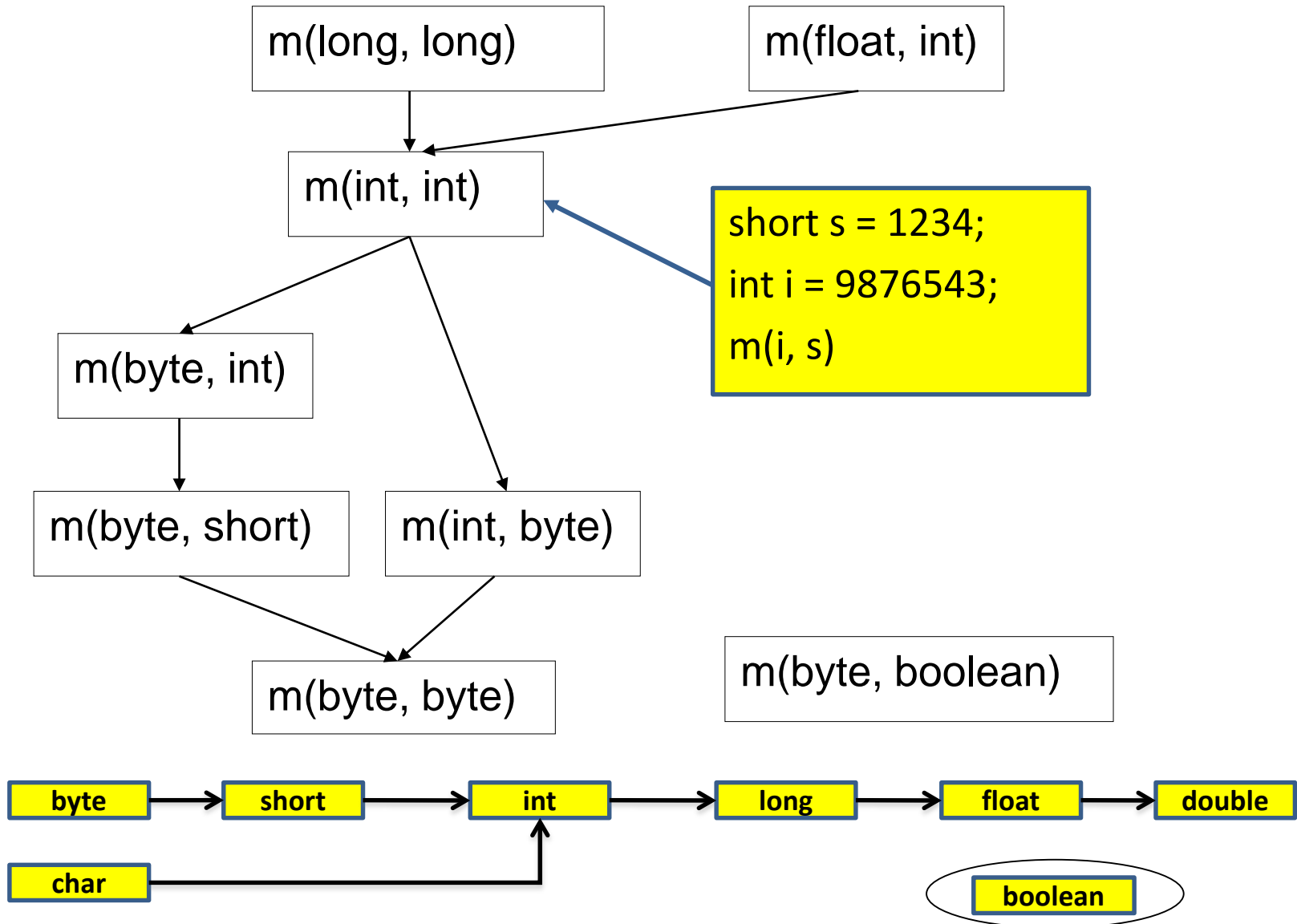
durch **Vergrößerung von Referenztypen** (später) aus S_i entstehen kann,

oder

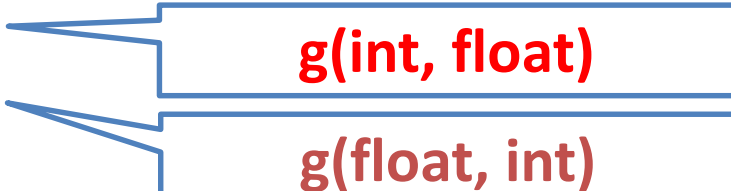
$T_i = S_i$ gilt und

für mindestens ein i_0 (mit $1 \leq i_0 \leq n$) $T_{i_0} \neq S_{i_0}$ gilt.

- Eine Methodendeklaration ist für einen Methodenaufruf **maximal spezifisch**, falls sie aufrufbar ist und keine andere aufrufbare Methode spezifischer als sie ist
- Existiert unter den aufrufbaren Methoden genau eine maximal spezifische Methode, so ist sie die **spezifischste**
- Existieren **zwei** oder **mehr** maximal spezifische Methodendeklarationen, so heißt der Methodenaufruf **mehrdeutig** und verursacht eine entsprechende **Fehlermeldung**
- Existiert die spezifischste Methode, so wird diese aufgerufen und für die aktuellen Parameter ausgeführt




```
class X {  
    int g( int i, float x ) { ... }  
    int g( float x, int i ) { ... }  
  
    int h() {  
        byte b;  
        short s;  
        int i;  
        long l;  
        float f; //Welche Methode wird unten aufgerufen  
        ...      // Welche Probleme treten auf?  
        g( i, f );  
        g( f, i );  
    }  
}
```



`g(int, float)`
`g(float, int)`

`g(b, f);`

`g(int, float)`

`g(i, l);`

`g(int, float) // Datenverlust möglich`

`g(b, b);`

mehrdeutig

`g(i, i);`

mehrdeutig

`g(i, (float) s);`

`g(int, float)`

`g(l, l);`

Nicht aufrufbar

`g(l, (int) l);`

`g(float, int)`
`// aufrufbar, aber`
`// eventuell Datenverlust`

}

}

Einschub:

Vereinfachte Schleifen-Notation (ab Java 5.0)

```
public class Schleifen {  
    public static int summeAlt(int[] w) {  
        int summe = 0;  
        for (int i=0; i < w.length; i++)  
            summe = summe + w[i];  
        return summe;  
    }  
    public static int summeNeu(int[] w) {  
        int summe = 0;  
        for (int x : w)  
            summe = summe + x;  
        return summe;  
    }  
    public static void main(String[] args) {  
        int[] werte = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};  
        System.out.println("Summe: " + summeAlt(werte));  
        System.out.println("Summe: " + summeNeu(werte));  
    }  
}
```

sprich: "für jedes **x** in **w**"

- Häufiges Überladen von Methoden für eine unterschiedliche Anzahl von Parametern des gleichen Typs aufwändig
- Ab Java 5.0 kann der jeweils letzte Parameter in der Signatur von Methoden **variabel** sein!

```
import static java.lang.System.*;

public class Argumente {
    public static int summiere(int... werte) {
        int summe = 0;
        for (int x : werte)
            summe = summe + x;
        return summe;
    }
    public static void main(String[] args) {
        out.println("summiere(1,2): " + summiere(1,2));
        out.println("summiere(1,2,3,4,5): " + summiere(1,2,3,4,5));
    }
}
```

- Die Methode `main` ist nach dem gleichen Schema wie jede andere Methode aufgebaut
- Ihr Rückgabetyp ist `void`, sie liefert also kein Ergebnis zurück
- Der einzige Parameter ist ein eindimensionales Feld vom Typ `String`, das bisher fast immer den Namen `args` trug

```
public class Anrede {  
    public static void main(String[] args) {  
        System.out.println("Hallo, " + args[0] + "!");  
        System.out.println("Der Name " + args[1] + " gefaellt mir gut!");  
        int i=0;  
        for (String p : args) {  
            System.out.println("Die " + i + "-te Eingabe lautet: " + p);  
            i++;  
        }  
    }  
}
```

Die Methoden aus anderen Klassen aufrufen: **Klassenmethoden**

- In Programmen werden oft Methoden aus anderen Klassen benutzt
- Beispiel: die Methoden der Klasse `Math` aus V03
- Alle diese Methoden waren durch das Schlüsselwort `static` gekennzeichnet
- Dieses Schlüsselwort macht die jeweilige Methode zur **Klassenmethode**

Damit wird die Methode für die Klasse selbst und mit der Klasse auch für andere Klassen verfügbar

```
public class MeineMethoden {

    public static void mal5nehmen(int n) {
        n = n*5;
        System.out.println("n = " + n);
    }

    public static int fakultaet(int n) {
        if (n == 0)
            return 1;
        for (int i = n-1; i>0; i--)
            n = n*i;
        return n;
    }

    public static void main(String[] args){
        int n = 7;
        mal5nehmen(n);
        System.out.println("n! = " + fakultaet(n));
    }
}
```

- Die Klasse `MeineMethoden` enthält die Klassenmethoden:

`mal5nehmen`, `fakultaet` und `main`

- Sollen in einem weiteren Programm die Klassenmethoden `mal5nehmen` oder `fakultaet` aufgerufen werden, so genügt es diese mit dem vorangestellten Klassennamen `MeineMethoden` aufzurufen
- Der Compiler findet diese dann im bereits compilierten Code der Klasse `meineMethoden`, und zur Laufzeit können sie dann von dort eingebunden und ausgeführt werden

```
public class TesteMethoden {  
    public static void main(String[] args) {  
        int x = 5;  
        MeineMethoden.mal5nehmen(x);  
        System.out.println(x + "! = " + MeineMethoden.fakultaet(x));  
    }  
}
```


- Wird das Schlüsselwort **static** in der Methodendeklaration weggelassen, so wird die betreffende Methode zur **Instanzmethode**
- Sie kann dann nur mit einem Objekt der betreffenden Klasse aufgerufen werden
- Einzelheiten und Erklärungen folgen später im Teil Objektorientierung

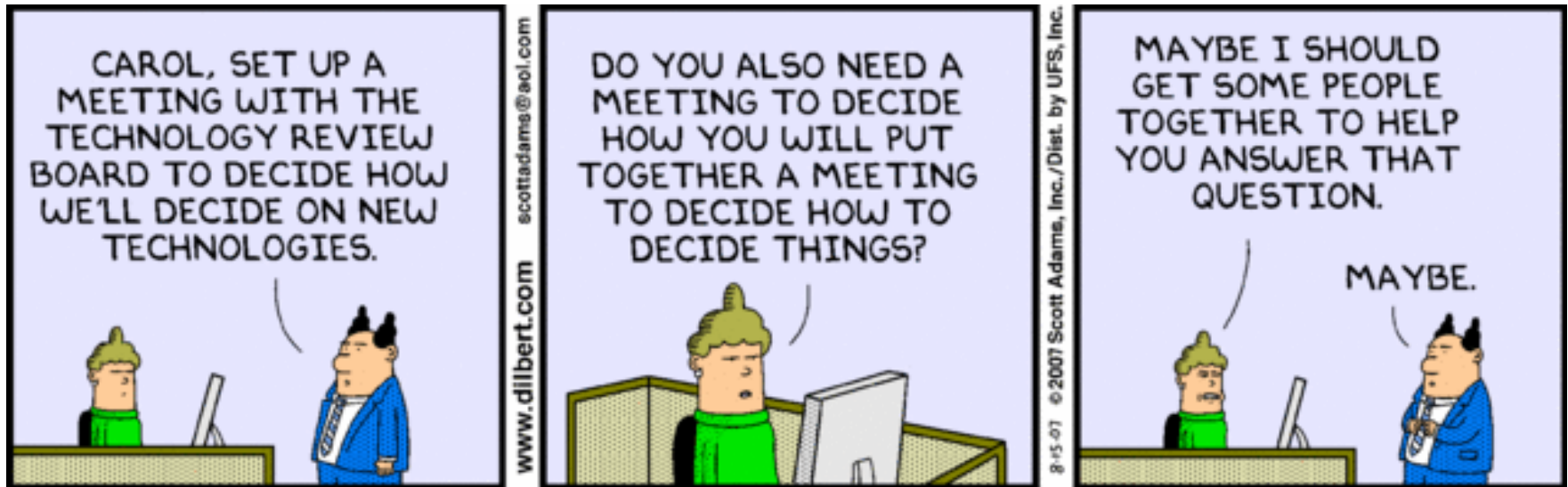
Beispiel:

```
public class Multiplizierer {  
    public int faktor = 0;  
    public int mul(int n) {  
        return faktor * n;  
    }  
}
```

```
public class TesteMultiplizierer {  
    public static void main(String[] args) {  
        Multiplizierer m7 = new Multiplizierer();  
        Multiplizierer m8 = new Multiplizierer();  
        m7.faktor = 7;  
        m8.faktor = 8;  
        System.out.println("7*5 = " + m7.mul(5));  
        System.out.println("8*5 = " + m8.mul(5));  
    }  
}
```

- Weitere **Einzelheiten und Erklärungen** folgen **später** in den Teilen zur Objektorientierung
- Ein Beispiel sind die Methoden der Klasse `java.lang.String`, die ebenfalls **später** erklärt werden

Meetings about meetings...



→ Next topic: Recursion!

Danksagung

- Vorlesungsmaterialien von Prof. Dr. Detlef Seese wurden als Basis verwendet
- Unterstützung bei der technischen und inhaltlichen Gestaltung des Vorlesungsmaterials leisteten:

Jóakim v. Kistowski

Dietmar Ratz, Joachim Melcher, Roland Küstermann, Jana Weiner, Hagen Buchwald, Matthes Elstermann, Oliver Schöll, Niklas Kühl, Tobias Diederich