# Funktionen: Methoden mit Rückgabewert

- Motivation
- Begriffe am Beispiel
- Best Practices
- Ergebnistyp und Ergebniswert
- Parameter

# Motivation

- Große Programme werden leicht unleserlich
- Änderungen in großen Programmen sind schwierig
  - Lokalisierung: Wo muss geändert werden?
  - Wiederholungen: Gibt es ähnliche Abschnitte, in denen ebenfalls geändert werden muss?
- Konsequenz: Fehler...!

- Nebenbei: Was heißt eigentlich „groß“?

# Ein Ansatz zur Lösung

Programme werden in kleinere „Einheiten“ aufgeteilt; Namen dafür sind (je nach Programmiersprache):

- *Methode*, Method
- Funktion, Function
- Prozedur, Procedure
- Unterprogramm, Subroutine

Konsequenzen

- Bessere Überschaubarkeit -> Verständlichkeit -> Änderbarkeit
- Paralleles Arbeiten mehrerer Entwickler

# Negativ-Beispiel: Blutalkoholkonzentration

```
public static void main(String[] args) {
    print("Masseanteil des Alkohols in g: ");
    double a;
    BufferedReader reader = new BufferedReader(new InputStreamReader(System.in));
    try {
        String line = reader.readLine();
        if (line == null)
            line = "";
        a = Double.parseDouble(line);
    } catch (NumberFormatException e) {
        throw new MISException("illegal double value");
    } catch (IOException e) {
        throw new MISException(e.toString());
    }

    print("Masse der Person in kg: ");
    double m;
    try {
        String line = reader.readLine();
        if (line == null)
            line = "";
        m = Double.parseDouble(line);
    } catch (NumberFormatException e) {
        throw new MISException("illegal double value");
    } catch (IOException e) {
        throw new MISException(e.toString());
    }

    print("Geschlecht (m für männlich oder w für weiblich): ");
    String gender;
    gender = readString();

    print("Alter in Jahren: ");
    double t;
    try {
        String line = reader.readLine();
        if (line == null)
            line = "";
        t = Double.parseDouble(line);
    } catch (NumberFormatException e) {
        throw new MISException("illegal double value");
    } catch (IOException e) {
        throw new MISException(e.toString());
    }

    print("Größe der Person in cm: ");
    double h;
    try {
        String line = reader.readLine();
        if (line == null)
            line = "";
        h = Double.parseDouble(line);
    } catch (NumberFormatException e) {
        throw new MISException("illegal double value");
    } catch (IOException e) {
        throw new MISException(e.toString());
    }

    // Watson-Formel
    double gwk;
    if (gender.equals("m"))
        gwk = 2.447 - 0.09516 * t + 0.1074 * h + 0.3362 * m;
    else if (gender.equals("w"))
        gwk = -2.097 + 0.1069 * h + 0.2466 * m;
    else
        throw new PRException("Ungültige Eingabe des Geschlechts");

    double r = 1.055 * gwk / (0.8 * m);

    // Widmark-Formel
    double w = a / (m * r);

    System.out.println("Blutalkoholkonzentration: " + w + " Promille");
}
```

```
String readLine(boolean ignoreLF) throws IOException {
    StringBuffer s = null;
    int startChar;

    synchronized (lock) {
        ensureOpen();
        boolean omitLF = ignoreLF || skipLF;

    bufferLoop:
        for (;;) {

            if (nextChar >= nChars)
                fill();
            if (nextChar >= nChars) { /* EOF */
                if (s != null && s.length() > 0)
                    return s.toString();
                else
                    return null;
            }
            boolean eol = false;
            char c = 0;
            int i;

            /* Skip a leftover '\n', if necessary */
            if (omitLF && (cb[nextChar] == '\n'))
                nextChar++;
            skipLF = false;
            omitLF = false;

        charLoop:
            for (i = nextChar; i < nChars; i++) {
                c = cb[i];
                if ((c == '\n') || (c == '\r')) {
                    eol = true;
                    break charLoop;
                }
            }

            startChar = nextChar;
            nextChar = i;

            if (eol) {
                String str;
                if (s == null) {
                    str = new String(cb, startChar, i - startChar);
                } else {
                    s.append(cb, startChar, i - startChar);
                    str = s.toString();
                }
                nextChar++;
                if (c == '\r') {
                    skipLF = true;
                }
                return str;
            }

            if (s == null)
                s = new StringBuffer(defaultExpectedLineLength);
            s.append(cb, startChar, i - startChar);
        }
    }
}
```

**schwierige Lokalisierung**

# Negativ-Beispiel: Blutalkoholkonzentration

```
public static void main(String[] args) {
    print("Masseanteil des Alkohols in g: ");
    double a;
    BufferedReader reader = new BufferedReader(new InputStreamReader(System.in));
    try {
        String line = reader.readLine();
        if (line == null)
            line = "";
        a = Double.parseDouble(line);
    } catch (NumberFormatException e) {
        throw new MISException("illegal double value");
    } catch (IOException e) {
        throw new MISException(e.toString());
    }

    print("Masse der Person in kg: ");
    double m;
    try {
        String line = reader.readLine();
        if (line == null)
            line = "";
        m = Double.parseDouble(line);
    } catch (NumberFormatException e) {
        throw new MISException("illegal double value");
    } catch (IOException e) {
        throw new MISException(e.toString());
    }

    print("Geschlecht (m für männlich oder w für weiblich): ");
    String gender;
    gender = readString();

    print("Alter in Jahren: ");
    double t;
    try {
        String line = reader.readLine();
        if (line == null)
            line = "";
        t = Double.parseDouble(line);
    } catch (NumberFormatException e) {
        throw new MISException("illegal double value");
    } catch (IOException e) {
        throw new MISException(e.toString());
    }

    print("Größe der Person in cm: ");
    double h;
    try {
        String line = reader.readLine();
        if (line == null)
            line = "";
        h = Double.parseDouble(line);
    } catch (NumberFormatException e) {
        throw new MISException("illegal double value");
    } catch (IOException e) {
        throw new MISException(e.toString());
    }

    // Watson-Formel
    double gwk;
    if (gender.equals("m"))
        gwk = 2.447 - 0.09516 * t + 0.1074 * h + 0.3362 * m;
    else if (gender.equals("w"))
        gwk = -2.097 + 0.1069 * h + 0.2466 * m;
    else
        throw new PRException("Ungültige Eingabe des Geschlechts");

    double r = 1.055 * gwk / (0.8 * m);

    // Widmark-Formel
    double w = a / (m * r);

    System.out.println("Blutalkoholkonzentration: " + w + " Promille");
}
```

viele ähnliche Programmteile

# Überschaubare Variante des gleichen Programms

```
public static void main(String[] args) {
    print("Masseanteil des Alkohols in g: ");
    double a = readDouble();

    print("Masse der Person in kg: ");
    double m = readDouble();

    print("Geschlecht (m für männlich oder w für weiblich): ");
    String gender;
    gender = readString();

    print("Alter in Jahren: ");
    double t = readDouble();

    print("Größe der Person in cm: ");
    double h = readDouble();

    double gwk = watsonFormula(gender, t, h, m);

    double r = bloodDensity(gwk, m);

    double w = widmarkFormula(a, m, r);

    println("Blutalkoholkonzentration: " + w + " Promille");
}
```

```
public static double readDouble() {
    BufferedReader reader = new BufferedReader(new InputStreaml
    try {
        String line = reader.readLine();
        if (line == null)
            line = "";
        return Double.parseDouble(line);
    } catch (NumberFormatException e) {
        throw new MISException("illegal double value");
    } catch (IOException e) {
        throw new MISException(e.toString());
    }
}
```

```
static double bloodDensity(double gwk, double m) {
    return 1.055 * gwk / (0.8 * m);
}
```

```
static double widmarkFormula(double a, double m, double r) {
    return a / (m * r);
}
```

# Erklärungen am Beispiel 1

Die *Methode watsonFormula* wird hier *aufgerufen*

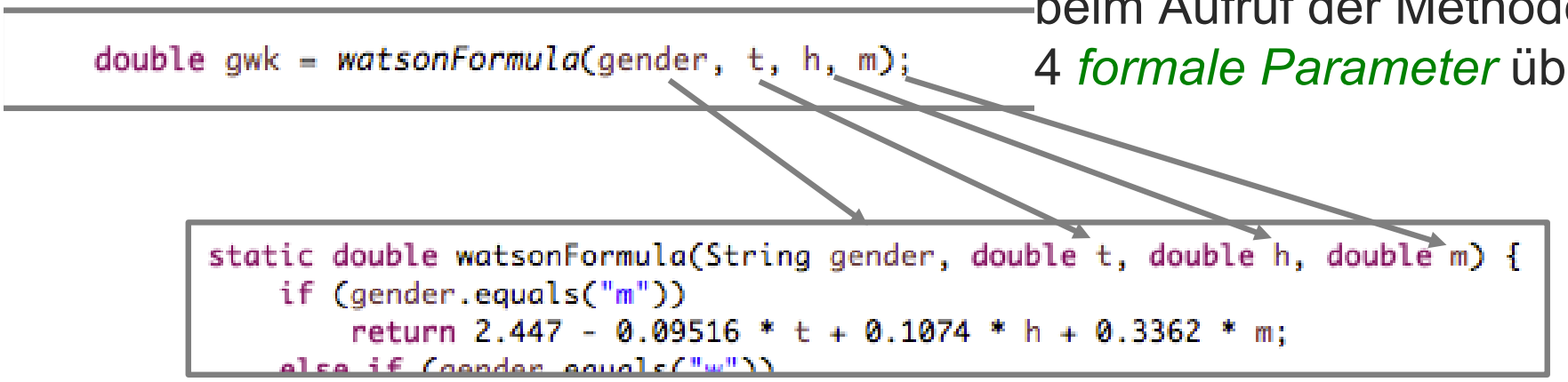
```
    print("Masse der Person in kg: ");
    double m = readDouble();

    print("Geschlecht (m für männlich oder w für weiblich): ");
    String gender;
    gender = readString();

    print("Alter in Jahren: ");
    double t = readDouble();

    print("Größe der Person in cm: ");
    double h = readDouble();

    double gwk = watsonFormula(gender, t, h, m);

    double r = bloodDensity(gwk, m);

    double w = widmarkFormula(a, m, r);

    println("Blutalkoholkonzentration: " + w + " Promille");
}
```

Sie erhält beim Aufruf 4 *aktuelle Parameter*: gender, t, h und m

# Erklärungen am Beispiel 2

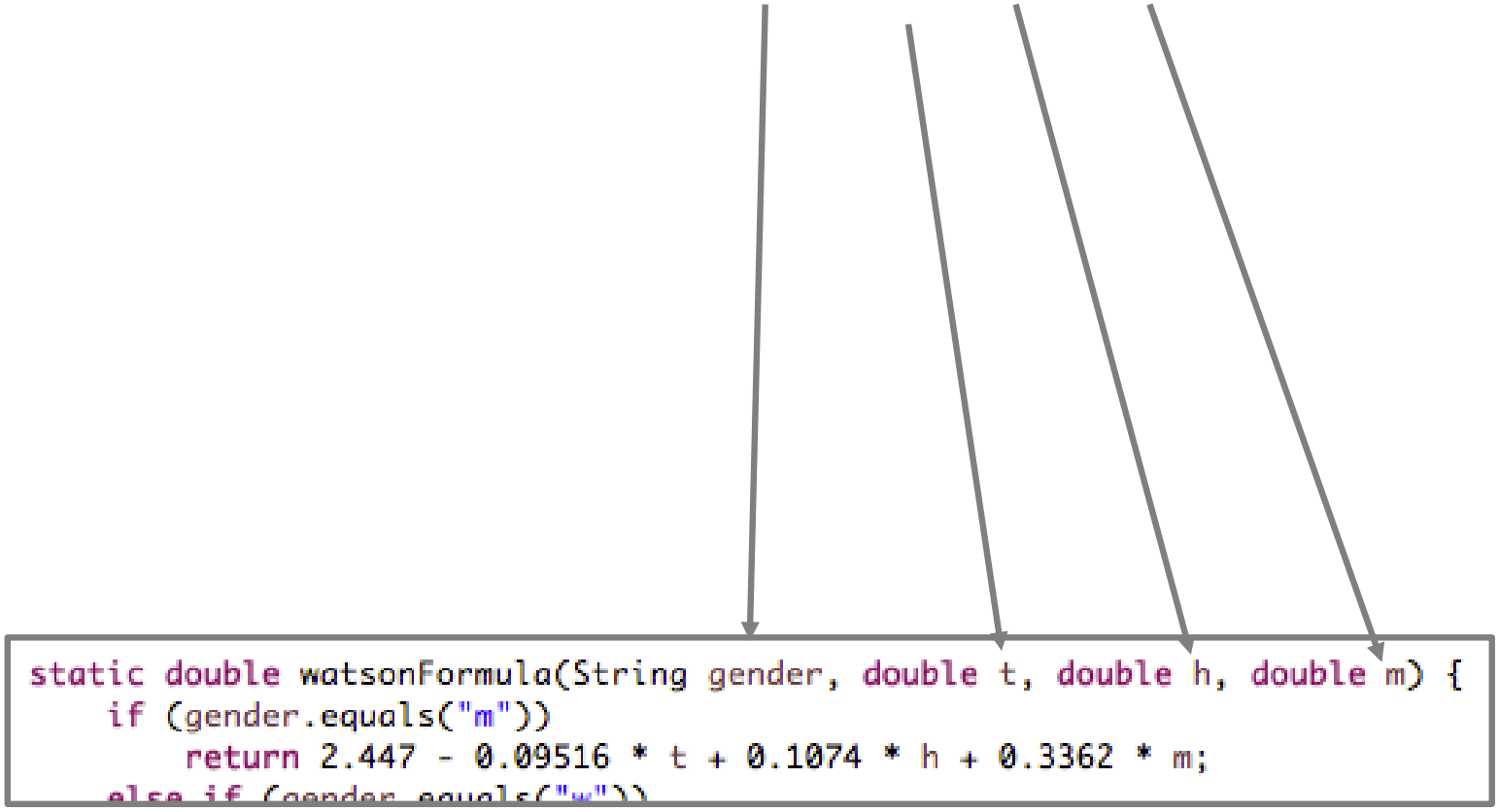
- Die *Zuordnung* aktueller -> formaler Parameter bei der Übergabe richtet sich nach der *Reihenfolge*;
  die *Namen* haben *nichts* damit zu tun
- In diesem Beispiel bietet es sich an, aktuelle und formale Parameter gleich zu nennen

Die 4 *aktuellen Parameter* werden beim Aufruf der Methode an 4 *formale Parameter* übergeben



# Erklärungen am Beispiel 3

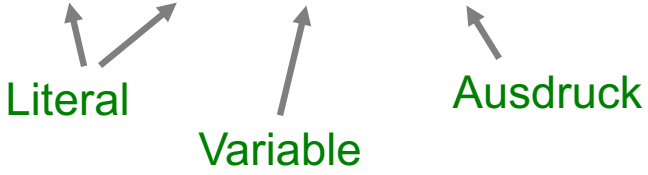
- Die *Übergabe entspricht* einer *Zuweisung* des Wertes des aktuellen Parameters an den formalen Parameter
- Der Aufruf könnte auch lauten:
  gwk = watsonFormula(“m”, 12.0, m, h * 2);



```
static double watsonFormula(String gender, double t, double h, double m) {
    if (gender.equals("m"))
        return 2.447 - 0.09516 * t + 0.1074 * h + 0.3362 * m;
    else if (gender.equals("w"))
```

# Erklärungen am Beispiel 4

- Die *Übergabe entspricht* einer *Zuweisung* des Wertes des aktuellen Parameters an den formalen Parameter
- Der Aufruf könnte auch lauten:
  gwk = watsonFormula(“m”, 12.0, m, h * 2);




```
static double watsonFormula(String gender, double t, double h, double m) {
    if (gender.equals("m"))
        return 2.447 - 0.09516 * t + 0.1074 * h + 0.3362 * m;
    else if (gender.equals("w"))
```

# Erklärungen am Beispiel 5

- Die Übergabe entspricht einer Zuweisung des Wertes des aktuellen Parameters an den formalen Parameter
- Der Aufruf könnte auch lauten:
  gwk = watsonFormula(“m”, 12.0, m, h * 2);

Der Typ eines aktuellen Parameters muss *zuweisungskompatibel* zum *Typ* des zugehörigen *formalen Parameters* sein

```
double gwk = watsonFormula(gender, t, h, m);
```

```
static double watsonFormula(String gender, double t, double h, double m) {
    if (gender.equals("m"))
        return 2.447 - 0.09516 * t + 0.1074 * h + 0.3362 * m;
    else if (gender.equals("w"))
```

# Erklärungen am Beispiel 6

Unsere bisherigen Typen erlauben folgende Zuweisungen / Parameter-Übergaben:

- String -> String
- boolean -> boolean
- double -> double
- int -> int
- int -> double

Der Typ eines aktuellen Parameters muss *zuweisungskompatibel* zum *Typ* des zugehörigen *formalen Parameters* sein

```
double gwk = watsonFormula(gender, t, h, m);
```

```
static double watsonFormula(String gender, double t, double h, double m) {
    if (gender.equals("m"))
        return 2.447 - 0.09516 * t + 0.1074 * h + 0.3362 * m;
    else if (gender.equals("w"))
```

# Leider...

- Es gibt nicht ***die*** richtige Aufteilung eines Programms in verschiedene Methoden
- Ob eine Lösung gut ist...
  - hängt von der konkreten Aufgabenstellung ab
  - kann man oft erst beurteilen, wenn nachträglich an einem Programm etwas geändert werden muss
- Was es gibt, sind sogenannte *best practices*, z.B. Design Pattern, Guidelines, Prinzipien
- Es gibt *keine* immer anwendbaren „Kochrezepte“

# Beispiel für unterschiedliche Aufteilungen

V1

```
static double watsonFormula(String gender, double t, double h, double m) {
    if (gender.equals("m"))
        return 2.447 - 0.09516 * t + 0.1074 * h + 0.3362 * m;
    else
        return -2.097 + 0.1069 * h + 0.2466 * m;
}
```

```
double gwk = watsonFormula(gender, t, h, m);
```

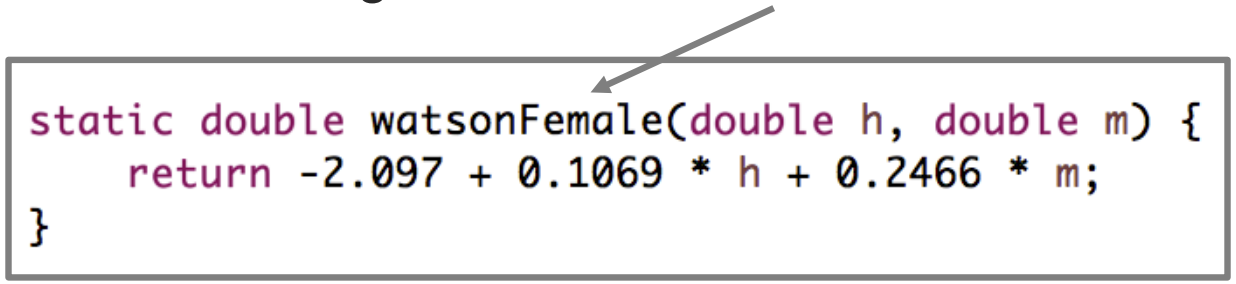
```
static double watsonMale(double t, double h, double m) {
    return 2.447 - 0.09516 * t + 0.1074 * h + 0.3362 * m;
}

static double watsonFemale(double h, double m) {
    return -2.097 + 0.1069 * h + 0.2466 * m;
}
```

V2

```
double gwk;
if (gender.equals("m"))
    gwk = watsonMale(t, h, m);
else
    gwk = watsonFemale(h, m);
```

# Formalia

- Prinzipiell wird bei einer Methoden-Vereinbarung eine Folge von Anweisungen mit einem Namen versehen:



```
static double watsonFemale(double h, double m) {
    return -2.097 + 0.1069 * h + 0.2466 * m;
}
```

- Unter diesem Namen ist die Methode dann überall im Programm sichtbar und kann aufgerufen werden:

```
gwk = watsonFemale(h, m);
```

- Der Effekt eines Methodenaufrufs ist derselbe, als würden die Anweisungen, die bei der Vereinbarung der Methode festgelegt wurden, am Ort des Aufrufs ausgeführt

# Ergebnis*typ* und –*wert* 1

- Als Ergebnis einer Methode wird ein Java-Ausdruck mit einem beliebigen Typ zurückgegeben
- Der *Typ* wird im Kopf der Methode vor dem Namen angegeben

```
static double watsonFemale(double h, double m) {
    return -2.097 + 0.1069 * h + 0.2466 * m;
}
```

- Um einen Ergebnis*wert* aus einer Methode zurückzugeben, benutzen wir die *return-Anweisung mit einem Ausdruck*
- Bei der Ausführung des Programms wird dieser Ausdruck berechnet und sein Wert als Methodenergebnis geliefert

```
static double watsonFemale(double h, double m) {
    return -2.097 + 0.1069 * h + 0.2466 * m;
}
```

# Ergebnis*typ* und –*wert* 2

- Der Ergebniswert kann von *mehreren Stellen* innerhalb einer Methode zurückgegeben werden
- Dafür dürfen *mehrere return*-Anweisungen verwendet werden

```
static double watsonFormulaV2(String gender, double t, double h, double m) {
    if (gender.equals("m"))
        return 2.447 - 0.09516 * t + 0.1074 * h + 0.3362 * m;
    return -2.097 + 0.1069 * h + 0.2466 * m;
}
```

- Der Ausdruck der (ersten) *ausgeführten return*-Anweisung bestimmt den Ergebniswert
- Die Methode wird unmittelbar nach der *return*-Anweidung beendet, d.h. eventuelle weitere Anweisungen in der Methode werden nicht mehr ausgeführt

# Ergebnis*typ* und –*wert* 3

Der Typ eines Ausdrucks, der nach einer *return*-Anweisung steht, muss an den Typ der Methode zuweisbar sein

Beispiel:
Lautet die Vereinbarung einer Methode

static *Typ name* ( ... ) {
    ...
    return *Ausdruck*;
}

so muss die folgende Vereinbarung mit Zuweisung erlaubt sein:

*Typ* var = *Ausdruck*;

```
static double watsonFemale(double h, double m) {
    return -2.097 + 0.1069 * h + 0.2466 * m;
}
```

Ansonsten meldet der Java-Compiler einen Fehler...

# Ergebnis*typ* und –*wert* 4

```
    static int max ( int x, int y ) {
        if (x > y)
            return "Maximum = " + x;
        else
            return "Maximum = " + y;
    }

}
```

Problems | Javadoc | Tasks | Properties | Console | SVN Repos

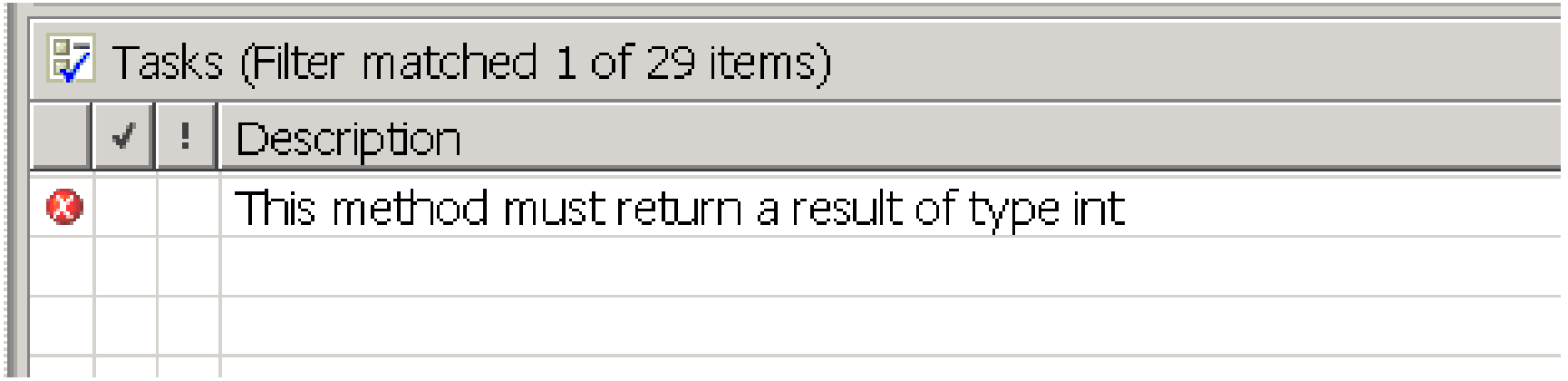
2 errors, 0 warnings, 0 infos (Filter matched 2 of 6.091 items)

| | Description |
|---|---|
| ⊗ | Type mismatch: cannot convert from String to int |
| ⊗ | Type mismatch: cannot convert from String to int |

# Ergebnis*typ* und –*wert* 5

Wenn eine Methode ein Ergebnis verspricht, muss sie das auch liefern, sonst meldet der Java-Compiler einen Fehler; Beispiel:

```
static int max ( int value1, int value2 ) {
    int result;
    if ( value1 >= value2 )
        result = value1;
    else
        return value2;
}
```



# Parameter für Methoden

- Viele Methoden akzeptieren einen oder mehrere *Parameter*, mit denen sie arbeiten

  Beispiele

  – Die Methode

  *stringAusdruck*.equals( *variableOderAusdruck* )

  akzeptiert einen Parameter vom Typ *String (*und liefert *true* genau dann, wenn der Parameter gleich dem *stringAusdruck* ist, sonst *false*)

  – Die Methode

  println( value1 )

  akzeptiert einen Parameter vom Typ *int*, *double*, *boolean* oder *String* (und gibt seinen Wert auf der Konsole aus)

# Beispiel für Parameter in Methoden

- Eine Methode *max* soll das Maximum zweier Werte vom Typ *int* bestimmen und als Ergebnis liefern
- Die Methode kann wie folgt aussehen

```
static int max( int a, int b ) {

    if ( a > b )
            return a;
    else
            return b;
}
```

Man nennt "a" und "b"
*formale Parameter*
der Methode „max"

- Die Methode „max" kann in "main" wie folgt aufgerufen werden

```
public static void main ( String [ ] args ) {
    int x = 200, y = 4712;
    print( max( 3, 5/2 ) );
    int max = max( 2+3, 4 );
    print( max( x, y ) );
}
```

Man nennt die übergebenen Werte
*aktuelle Parameter*
für die Methode „max"

# Zuordnungen von aktuellen zu formalen Parametern

- Bei *formalen Parametern* handelt es sich immer um „Platzhalter“ für die Verwendung in der Methode, denen bei jedem Aufruf ein (anderer) Wert übergeben wird
  -> Die formalen Parameter sieht man nur *innerhalb* der Methode
- Als *aktuelle Parameter* können beliebige Werte übergeben werden, also Variable, Ergebnisse von Ausdrücken (z.B. anderer Methodenaufrufe) etc.
  -> Die aktuellen Parameter sieht man nur *außerhalb* der Methode
- Aktuelle und formale Parameter werden einander *beim Aufruf zugeordnet*; das funktioniert *wie eine Zuweisung* beim Methodenaufruf

# Zuordnungen von aktuellen zu formalen Parametern

- Die Methode max

```
static int max( int a, int b )
{    ...    }
```

zwei formale Parameter

Typen: int, int

- Mögliche Aufrufe von max

```
int x = 200, y = 4712, z = 42;

print( max( 3, 5/2 ) );

print( max( x, y ) );

print( max( max( x, y ), z ) );
```

jeweils zwei aktuelle Parameter vom Typ int

- Es gilt
  - Die Anzahl aktueller muss *gleich* der *Anzahl* formaler Parameter sein
  - Einander zugeordnete aktuelle und formale Parameter müssen *zuweisungsverträglich* sein, d.h.

    es muss nach den Regeln von Java möglich sein, einen Wert vom Typ des aktuellen Parameters an eine Variable vom Typ des formalen Parameters zuzuweisen

# Namen formaler Parameter

- Formale Parameter sind nur in der Methode sichtbar, in der sie vereinbart werden
  - nicht davor
  - nicht danach
  - nicht in der aufrufenden Methode
  - nicht sonstwo

- Die Namen formaler Parameter müssen *in ihrer Methode eindeutig* sein, das heißt sie müssen sich untereinander und von den Namen aller Variablen in der Methode unterscheiden