LE - 3
Methoden und Testing

Ilya Shabanov
Nadim El Sayed

Freitagsrunde 4!

Was wir bisher gelernt haben :

- **Variablen und Arrays: Deklaration / Definition.**
- **Schleifen (for, while ...).**
- **Fallunterscheidungen (if-else).**
- **Zuweisungen...**

➢ **Jetzt wollen wir sie (Anweisungen) strukturiert benutzen !**

Methoden

Einführungsbeispiel :

- **Binomialkoeffizienten berechnen**

$$\binom{n}{k} = \frac{n!}{(n-k)! \cdot k!}$$

Einführungsbeispiel - Fakultät :

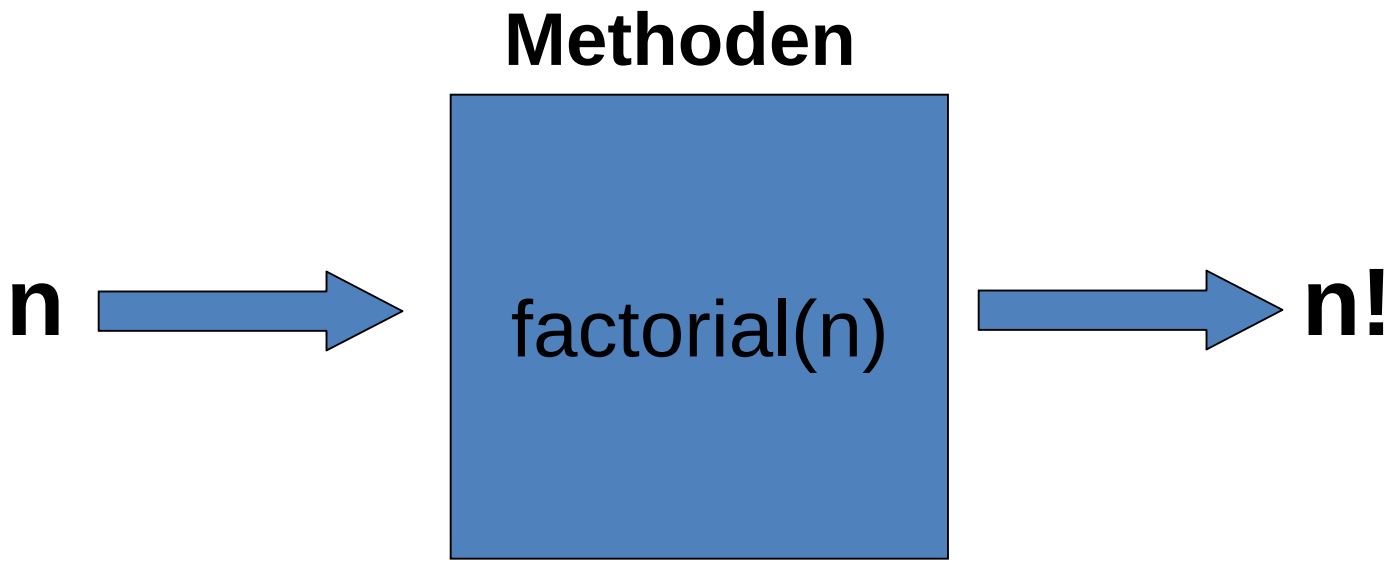
```
public static void main( String[] args ) {
    int n = 5, k = 2;
    int nFac = 1 , kFac = 1 , nMinusKFac = 1; // init variables

    for( int i = n ; i > 0 ; i++){    // calculate n factorial
       nFac = nFac * i;
    }

    for( int i = k ; i > 0 ; i++){    // calculate k factorial
       kFac = kFac * i;
    }

    for( int i = n-k ; i > 0 ; i++){  // calculate (n-k) factorial
       nMinusKFac = nMinusKFac * i;
    }

    int result = nFac/(nMinusKFac * kFac); // calculate n choose k
    System.out.println(n + " über " + k + " ist " + result );
}
```

Einführungsbeispiel - Fakultät :

```
public static void main( String[] args ) {
    int n = 5, k = 2;
    int nFac = 1 , kFac = 1 , nMinusKFac = 1; // init variables

    for( int i = n ; i > 0 ; i++){    // calculate n factorial
       nFac = nFac * i;
    }

    for( int i = k ; i > 0 ; i++){    // calculate k factorial
       kFac = kFac * i;
    }

    for( int i = n-k ; i > 0 ; i++){  // calculate (n-k) factorial
       nMinusKFac = nMinusKFac * i;
    }

    int result = nFac/(nMinusKFac * kFac); // calculate n choose k
    System.out.println(n + " über " + k + " ist " + result );
}
```

Einführungsbeispiel - Fakultät :

- **Der Code für die Fakultätsberechnung wiederholt sich!**
- **Wie kann man diesen Code mehrmals benutzen?**



Was ist eine Methode? :

- **Eine Methode fasst mehrere Anweisungen zusammen.**
- **Besitzt einen Namen, mit dessen Aufruf diese Anweisungen ausgeführt werden.**
- **Fässt häufig benutzen Code in einer Einheit zusammen.**
- **Methoden bestehen aus :**
  - **Kopf (Deklaration)**
  - **Rumpf (Definition)**

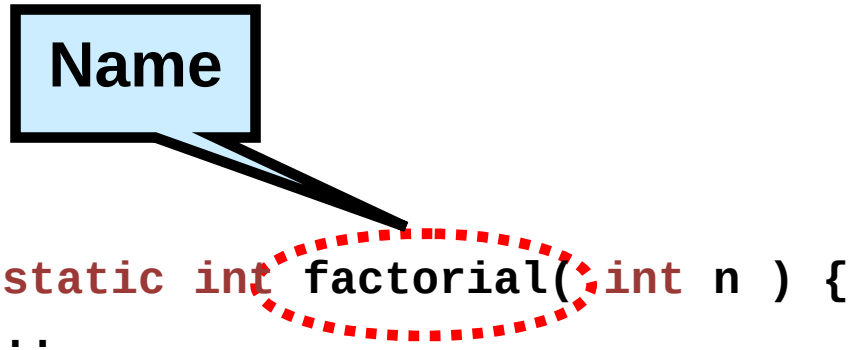
Fakultät reloaded:

```
public static int factorial( int n ) {
    int nFactorial = 1;
    for( int i = n ; i > 0 ; i++){    // calculate n factorial
       nFactorial = nFactorial * i;
    }
    return nFactorial;
}
```

- **Unser Code sieht nun viel einfacher aus:**

```
public static void main( String[] args ) {
    int n = 5, k = 2;
    int nFac = factorial( n ); //calculate n factorial
    int kFac = factorial( k ); //calculate k factorial
    int nMinusKFac = factorial( n-k ); //calculate (n-k) factorial

    int result = nFac/(nMinusKFac * kFac); // calculate n choose k
    System.out.println(n + " über " + k + " ist " + result );
}
```

Methodenkopf - Name:




```
public static int factorial( int n ) {
    .....
}
```

- **Der Name wird zum Aufrufen der Methode verwendet.**
- **Er sollte daher aussagekräftig sein (am besten ein Verb).**
- **Es dürfen nur Buchstaben, Zahlen und ‚_‘ verwendet werden.**

Methodenkopf - Parameter:

Parameterliste

```
public static int factorial( int n ) {
    .....
}
```

- **Die Eingabewerte werden in der Methode als Variablen verwendet.**
- **Mehrere Parameter werden durch Kommas getrennt.**
- **Es ist auch möglich auf Parameter zu verzichten.**

Methodenkopf - Rückgabe:

Rückgabetyp

```
public static int factorial( int n ) {
    .....
}
```

- **Definiert den Typ der Rückgabe: boolean, int, double etc.**
- **Nur eine Variable kann zurückgegeben werden.**
- **Falls nichts zurückgegeben werden soll, schreibt man „void“.**

Methodenkopf - Modifikatoren:

Modifikatoren

```
public static int factorial( int n ) {
    ....
}
```

- **Legen die Art (und Sichtbarkeit) der Methode fest.**
- **Wir schreiben zunächst nur „public static“.**
- **Mehr dazu im OOP Kapitel (LE 5 & 6).**

Methodenrumpf (body):

- **Wird zwischen {} eingeschlossen und steht nach dem Kopf.**

- **Enthält die Anweisungen, die (von den Parametern) zur Rückgabe führen.**

- **Die übergebenen Parameter werden innerhalb des Rumpfs als Variablen behandelt.**

Wie geben wir die Ergebnisse zurück? :

```
public static int factorial( int n ) {
    int nFactorial = 1;
    for( int i = n ; i > 0 ; i++){    // calculate n factorial
       nFactorial = nFactorial * i;
    }
    return nFactorial;
}
```

- **Die return Anweisung beendet die Ausführung.**
- **Nach return steht ein Ausdruck des Rückgabetyps.**
- **Falls keine Rückgabe (void), lässt man return weg.**

Was passiert mit den Variablen in der Methode? :

- **In der Methode deklarierte Variablen sind nach Ausführung nicht mehr bekannt.**

```
public static void main( String[] args ) {

    declareVariables();
    System.out.println( “String: ” + str + “ Number: ” + num );
}

// Method changes <number> to zero
public static void declareVariables() {
    String str = “Was auch sei 4711 ist immer dabei.”;
    int num = 4711;
}
```

- **Führt zu einem Compilerfehler!**
- **Denn str und num sind bei der Ausgabe nicht mehr bekannt.**

Vorteile von Methoden :

- **Code wird lesbarer und robuster.**
- **Aufruf unabhängig von der Implementierung.**
- **Implementierung kann nachträglich geändert oder verbessert werden bei gleichem Aufruf.**
- **Für größere Projekte ist das eine Möglichkeit der Arbeitsteilung.**
- **Testen einzelner Methoden des Programms möglich.**

Resumé :

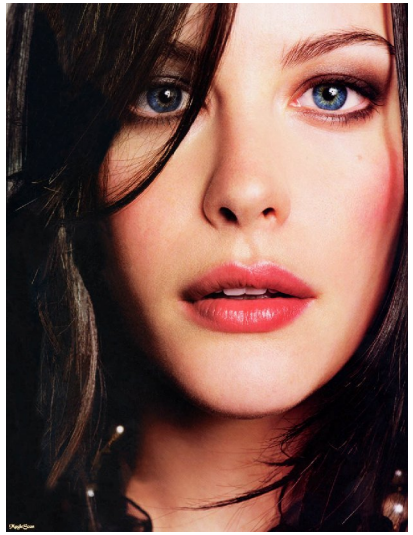
- **Kopf**
  - **Besteht aus Name, Parametern, Rückgabetyp.**
  - **WAS für eine Methode ist es? WAS macht sie ?**
  - **Dem Benutzer muss das auf einen Blick klar sein!**
- **Rumpf**
  - **Beinhaltet die Implementierung (Menge v. Anweisungen).**
  - **WIE macht die Methode das, was sie soll?**
  - **Für den Benutzer nicht zwingend relevant.**

- **Einen korrekten Kopf zu schreiben bringt euch bereits Punkte in der Klausur!**

Methoden Beispiele :

- **Methode ohne Parameter und ohne Rückgabe.**

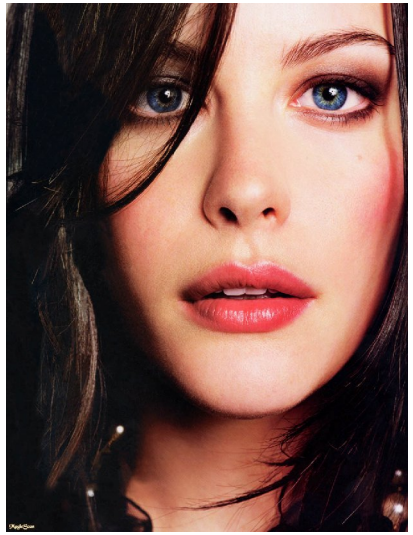
```
public static void main( String[] args ) {
    flirtWithLivTyler();
}

public static void flirtWithLivTyler() {
    System.out.println( “Hey Liv, you‘re hot!“ );
}
```



~ omg

- **Methode hat kein return, da keine Rückgabe.**
- **Aufruf über Namen gefolgt von () .**

Methoden Beispiele :

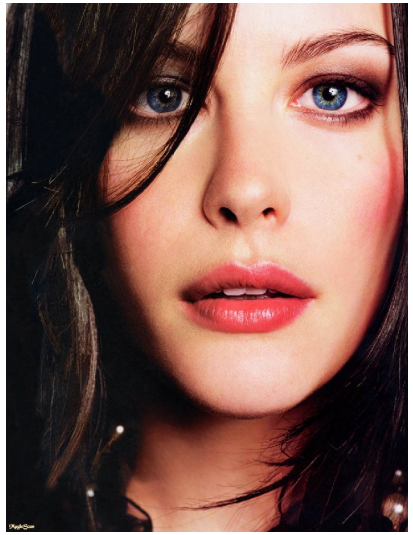


- **Methode ohne Parameter, mit Rückgabe.**

```
public static boolean askForCoffee() {
    System.out.println( "Noooo!" );
    return false;
}
public static void main( String[] args ) {
    System.out.println( "Do you want to drink a coffee with me?" );
    boolean interested = askForCoffee();
    System.out.println( "Liv Tyler is interested: " + interested );
}
```

- **Methode hat ein return gefolgt von einem boolean Ausdruck.**
- **Der Methodenaufruf wird wie ein boolescher Wert behandelt.**
- **Speicherung der Rückgabe in einer Variablen.**

Methoden Beispiele :

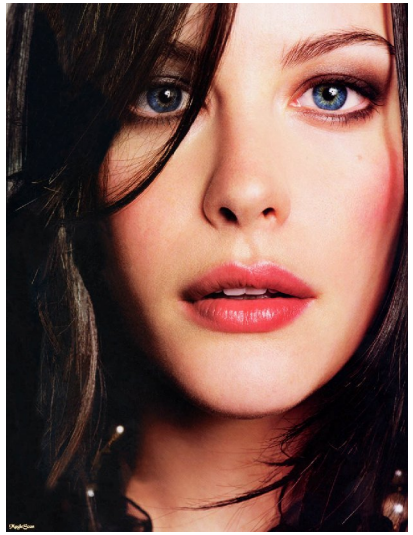


- **Methode mit Parameter, ohne Rückgabe.**

```
public static void main( String[] args ) {
    String recepient = “Lovely Liv Tyler”;
    String message = “C’mon, just one coffee!”;
    writeFanMail( recepient , message );
}

public static void writeFanMail( String addr , String mail ) {
    // ... Sending mail code
    System.out.println( “Message:” + mail + “ sent to:” + addr );
}
```

- **Methode hat kein return, da keine Rückgabe.**
- **Aufruf über Namen mit Klammern, wo die Parameter stehen.**

Methoden Beispiele :



- **Methode mit Parameter und mit Rückgabe.**

```
public static boolean isOlderThanLiv( int age ) {
    if( age > 30 ){
       return true;
    }
    return false;
}
public static void main( String[] args ) {
    int myAge = 22;
    boolean olderThanLiv = isOlderThanLiv( myAge );
    System.out.println( "Am I older than Liv? " + olderThanLiv );
}
```

- **Methode hat ein return.**
- **Aufruf über Namen mit Klammern, wo die Parameter stehen.**
- **Speichern der Rückgabe in einer Variablen.**

Call-by-Value :

```
public static void main( String[] args ) {

    int n = 5;
    System.out.println( "n ist: " + n );
    setToZero( n );
    System.out.println( "n ist: " + n );
}

// Method changes <number> to zero
public static void setToZero( int number ) {
    number = 0;
}
```

```
Ausgabe:
    n ist: 5
    n ist: 5
```

- **Die Variable wird nicht verändert, da die Methode mit einer Kopie arbeitet.**
- **Das gilt aber nur für primitive Datentypen wie int, double, boolean usw. (in Java immer kleingeschrieben)**

Call-by-Value :

```
public static void main( String[] args ) {

    int[] numArr = new int[1];
    numArr[0] = 7;
    System.out.println( "[0] ist: " + numArr[0] );
    setToZero( numArr );
    System.out.println( "[0] ist: " + numArr[0] );
}

// Method changes <number> to zero
public static void setToZero( int[] numArray ) {
    numArray[0] = 0;
}
```

```
Ausgabe:
    [0] ist: 7
    [0] ist: 0
```

- **Das Array wird verändert, da es kein primitiver Datentyp ist.**
- **Bei nicht-primitiven Datentypen wird nicht kopiert.**

Fakultät revolutions:

- **Rekursion funktioniert auch in Java!**

```
public static int factorial( int n ) {
    if( n <= 1 ){
        return 1;
    }
    return n * factorial( n – 1 );
}
```

Namen

Was macht eigentlich diese Methode
(und warum liegt hier eigentlich Stroh?) :

```
public static boolean myMethod(int a) {

        if (a == 0){
                return true;
        }else if (a > 0) {
                boolean boolValue = myMethod(a - 1);

                // boolValue is the result of myMethod(a - 1)
                return !boolValue;
        }else {
                boolean boolValue = myMethod(a + 1);

                // we return the complement of boolValue
                return !boolValue;
        }
}
```

Nun wird's klarer... :

```
public static boolean isEven( int num ) {

        if (num == 0){
                return true;
        }else if (num > 0) {
                boolean predecessorEven = isEven(num - 1);
                // predecessor is even <=> number is not even
                return !boolValue;
        }else {
                boolean successorEven = isEven(num + 1);
                // successor is even <=> number is not even
                return !successorEven;
        }
}
```

- **Diese Methode berechnet ob eine Zahl gerade ist, indem sie (rekursiv) schaut ob der Vorgänger ungerade ist.**

Kommentare & Namen : Do‘s und Dont‘s

	Do‘s	Dont‘s
Kommentare	▪ Schwer ersichtliche Gedankengänge erläutern. ▪ Erklärung der Bedeutung und Anwendung von komplexen Methoden/Klassen. ▪ Erläuterung des generellen Ablaufs des Programms.	▪ Kommentare, die beschreiben, **was** man tut und nicht **warum** man es tut. ▪ Kommentare, die belanglos sind und sich wiederholen.
Namen	▪ Namen, die selbsterklärend sind und ihre Funktion repräsentieren. ▪ Namen können Kürzel enthalten, die den Typ, die Sichtbarkeit oder die Aufgabe der Variablen angeben.	▪ Namen auf Deutsch. ▪ Namen, welche mit Großbuchstaben anfangen, denn das geht nur bei Klassen. ▪ Zu lange Namen.

Testen

Testen :

Ach wozu testen, passt schon! – ***Worte eines Langzeitstudenten***

Testen ist Pflicht um gute Software herzustellen, denn Fehler treten immer auf, egal wie gut (oder eingebildet) man ist.

Was sind die Hauptziele beim Testen von Software?

1. Funktionalität testen:

- **Tut das Programm das, was es soll ?**
- **Funktioniert es auch für sinnlose Eingaben ?**

2. Stabilität testen:

- **Terminiert das Programm für alle Eingaben ?**

Funktionalität testen:

```
// Method calculates <base> to the power of <exp>

public static double pow(double base, int exp) {

    if ( exp > 0 ) {
       return base * pow( base , exp – 1 );
    }

    return 1;
}
```

Erwartetes Ergebnis	Tatsächliches Ergebnis
pow(3 , 2) = 9.0	pow(3 , 2) = 9.0
pow(0 , 0) = 1.0	pow(0 , 0) = 1.0
pow(2 , -1) = 0.5	pow(2 , -1) = 1.0
pow(-1 , -1) = -1.0	pow(-1 , -1) = 1.0

Funktionalität testen:

Die Methode fühlt sich scheinbar nicht zuständig für negative Exponenten.

```
// Method calculates <base> to the power of <exp>

public static double pow( double base, int exp) {

    if ( exp > 0 ) {
       return base * pow( base , exp – 1 );
    }
    else if ( exp < 0 ) {
       return 1 / pow( base , -1 * exp );
    }

    return 1;
}
```

Resumé:

- **Randwerte einsetzen.**

  >> Kehrwert von 0

- **Werte einsetzen, für die das Programm einfach nicht gedacht ist .**

  >> Fakultät einer negativen Zahl

- **Viel hilft viel.**

- **Nach der Korrektur immer nochmals testen.**

Für Bug-begeisterte :

Was passiert eigentlich wenn wir `pow(0,-1)`, also „0 hoch –1“ berechnen wollen?

Stabilität testen:

```
// Method calculates <num> modulo <div>

public static int mod( int num , int div  ) {

    while( num > div ){
       num = num – div;
    }
    return num;
}
```

- **Bei einem Test versucht man `mod(4,-1)` zu berechnen.**

  >> Stunden später: das Programm läuft noch...

Was tun? :

- **Einfügen vieler println Anweisungen („Printline Debugging“).**

```
// Method calculates <num> modulo <div>

public static int mod( int num , int div  ) {
    System.out.println("Es geht los mit num:" + num +
                       " div:" + div );
    while( num > div ){
        System.out.println("num: " + num);
        num = num – div;
    }
    System.out.println("Geschafft, Ergebnis:" + num);
    return num;
}
```

Was macht das Programm also im Verlauf der Berechnung ?:

```
// Method calculates <num> modulo <div>

public static int mod( int num , int div  ) {
    System.out.println("Es geht los mit num:" + num +
                       " div:" + div );
    while( num > div ){
       System.out.println("num: " + num);
       num = num – div;
    }
    System.out.println("Geschafft, Ergebnis:" + num);
    return num;
}
```

Ausgabe:

```
Es geht los mit num:4 div:-1

num: 4
num: 5
num: 6
num: 7
...
```

- **Num wird also immer größer ! Aha....**

Resumé:

- **Printline Debugging.**
- **Ausgabe aller relevanten Variablen.**
- **Println Anweisungen nur auskommentieren, nicht löschen!**
- **Später: Debugger benutzen.**

Viel Spaß bei der Übung!