Aufgabe 2.20:
Schreibe ein Programm , das nach Eingabe von Double-Zahlen in ein Feld das arithmetische Mittel der Feldkomponenten ausgibt .

Aufgabe 2.21:
Schreibe ein Programm, das nach Eingabe von Double-Zahlen in ein Feld das arithmetische Mittel und die Summe aller Feldkomponenten ausgibt .

Aufgabe 2.22:
Schreibe ein Programm, das nach Eingabe von Double-Zahlen in ein Feld die kleinste und die größte Feldkomponente ausgibt .

Aufgabe 2.23:
Schreiben Sie ein Programm, das nach Eingabe von Double-Zahlen in ein Feld das geometrische Mittel der Feldkomponenten ausgibt .

Aufgabe 2.24:
Schreibe ein Programm, das ein Feld mit 100 Integer-zahlen 0<=z<=10 füllt. Das Programm soll ausgeben, wie oft eine eingegebene Zahl in diesem Feld vorkommt.

Methoden

Methoden dienen als Bausteine für komplexe Programme.
Methoden können als Funktionen aufgefasst werden: Einer Funktion wird i.a. etwas übergeben und sie gibt i.a. etwas aus.
Eine Methode dient dazu, eine genau definierte Aufgabe zu erfüllen. Dazu können der Methode Daten übergeben werden, die sie verarbeitet

Beispiel 1 (Der Methode wird eine Zahl übergeben. Sie gibt aber nichts (void) an das Programm zurück. Sie bewirkt, dass die Quadratzahl von x ausgegeben wird.)

```
static void hochzwei_1(int x)
{
  System.out.println(x*x);
}
```

Beispiel 2 (Der Methode wird eine Zahl übergeben. Die Quadratzahl von x wird an das Programm zurückgegeben.)

```
static int hochzwei_2(int x)
{
  return x*x
}
```

Mit der Anweisung `System.out.print(hochzwei_1(6));` **wird die Methode hochzwei_1 mit dem Parameter 6 aufgerufen. Der Aufruf bewirkt die Ausgabe von 36.**

Syntax für eine Methode in Java:

```
Attribute Rückgabetyp MethodenName(Liste der formalen Parameter)
{
  Anweisungen;
  return Ausdruck;                    // optional
}
```

Der Methodenkopf in der ersten Zeile enthält

- immer einen **Rückgabetyp**: Er legt fest, welchen Datentyp die Methode zurückgibt: Dies kann ein primitiver Typ oder ein Objekttyp sein. Falls die Methode keinen Rückgabewert hat, wird als Typ **void** angegeben. Der Ausdruck in der return-Anweisung muss dem Rückgabetyp entsprechen.
- den **Namen der Methode**
- danach in runden Klammern folgend eine **Parameterliste** für die Übergabe von Daten an die Methode. Sollen keine Daten übergeben werden, bleibt das Klammernpaar leer.
- Optional können im Methodenkopf zu Beginn auch **Attribute** angegeben werden: die Sichtbarkeits-attribute **public** und **private** legen fest, ob eine Methode in anderen Klassen verwendet werden kann oder nicht. Das Attribut **static** legt eine Methode als sogenannte ***Klassenmethode*** fest. Entfällt dieses Attribut, so ist die Methode eine ***Instanzmethode***. Zunächst arbeiten wir nur mit Klassenmethoden.

Parameter und Wertrückgabe einer Methode

Hat eine Methode einen Rückgabetyp (d.h. der Rückgabetyp wurde nicht mit void deklariert), muss irgendwo im Methodenkörper ein Wert zurückgegeben werden. Herfür wird das Schlüsselwort return verwendet.

Allgemeine Regeln zur Methodendeklaration:

- **Methoden können nur in Klassen deklariert werden.**
- **Methoden können nicht in Methoden deklariert werden, also insbesondere nicht innerhalb von main().**
- **Trenne zwei Methoden immer durch eine Leerzeile.**
- **Beschränke jede Methode auf die Ausführung genau einer klar definierten Aufgabe.**
- **Verwenden für Methoden und Parameter aussagekräftige Namen!**
- **Ein Rückgabetyp muss immer angegeben werden. Ist dieser nicht void, so muss eine return-Anweisung vorhanden sein!**

Aufgabe 2.30

Scheibe Metoden (Klassenfunktionen) für folgende mathematische Funktionen:

$$f(x) = \log_c(x) = \frac{\ln(x)}{\ln(c)}$$

$$g(x) = \sqrt[n]{x} = x^{\frac{1}{n}}$$

$$h(n) = n! = 1 \cdot 2 \cdot 3 \cdots n; \quad 0! = 1; \; 1! = 1$$

$$S(x) = \sinh(x) = e^x - e^{-x}$$

$$C(x) = \cosh(x) = e^x + e^{-x}$$

$$t(x) = \tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

Aufgabe 2.31

Schreibe ein Programm, das für jede die Funktion S eine Tabelle für $x \in [1;5]$ mit der Schrittweite dx=0.5 ausgibt.

Beispiel:

Sinh(1)	**=**	**2.35**
Sinh(1.5)	**=**	**4.25**
Sinh(2)	**=**	**7.25**
.....		
Sinh(5)	**=**	**148.41**

Call-by-value oder Call-by-reference?

Hier geht es um folgende Frage: Was geschieht, wenn innerhalb einer Methode der Wert des übergebenen Parameters verändert wird? Wirkt sich dies auch auf die Variable in der aufrufenden Methode aus?
Testprogramm:

```
class CallByValue
{
  public static void main(String[] args)
  {
    int a = 1, hilf;
    System.out.println("Wert von a vorher: " + a);
    hilf = erhoehen(a);
    System.out.println("Wert von a nachher: " + a);
    System.out.println("Wert von hilf: " + hilf);
  }
  static int erhoehen(int i)
  {
    i=i+10;
    return i;
  }
}
```

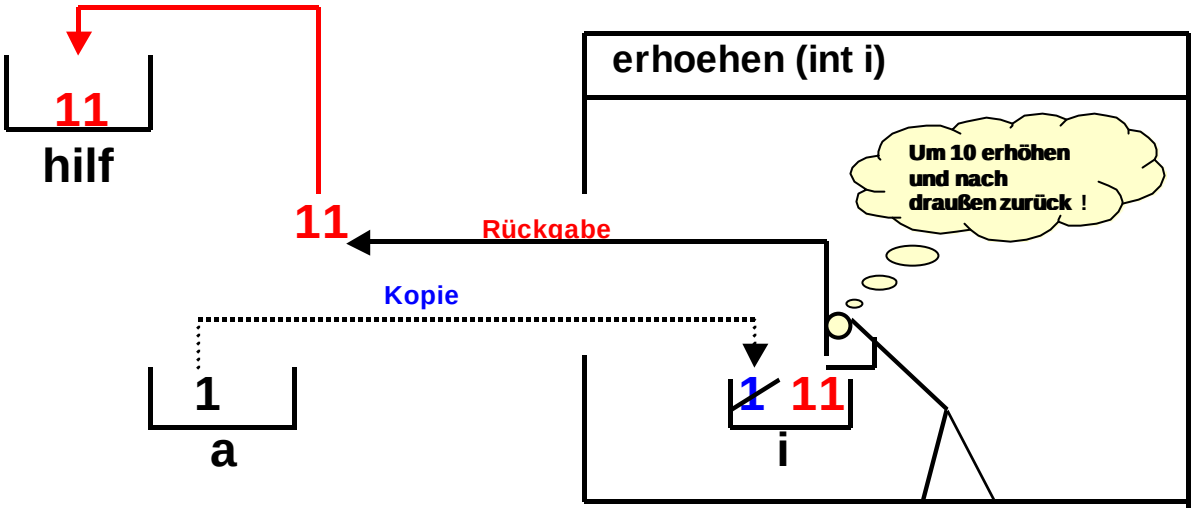
Der Wert von der Variablen a ist unverändert 1 !
Die Methode hat also nur eine Kopie der Variable a

bearbeitet, das Original aber unverändert gelassen!

Ein solcher Methodenaufruf heißt Call-by-value.

Dieser tritt in Java immer dann auf, wenn die Parameter von einem primitiven Datentyp sind.
Die Variable hilf hingegen erhält den Wert 11.

Called by value:

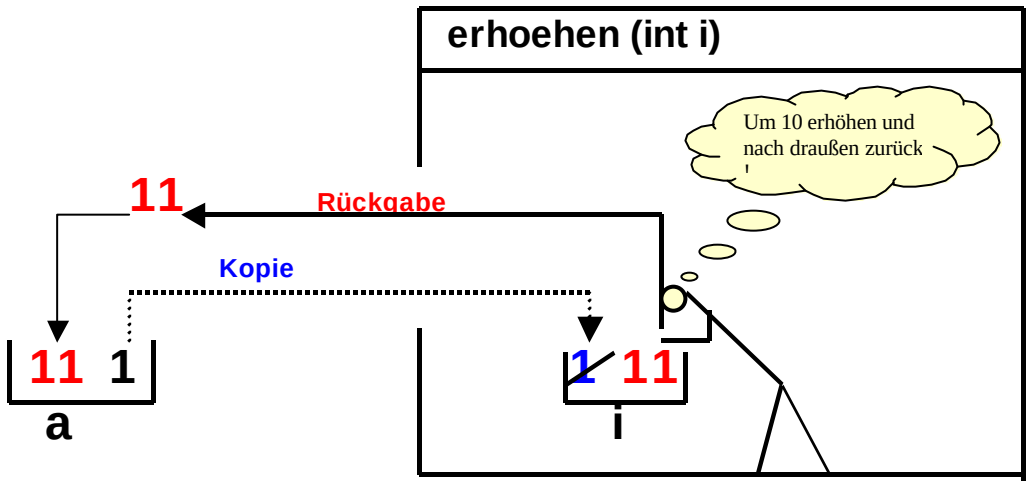


Beim Aufruf der Methode mit erhoehen(a) wird eine Kopie des Inhalts der Variablen a der formalen Variablen i als Wert übergeben. Der „*Methodenmanager*“ weiß was er zu tun hat (den Wert von i um 10 erhöhen und den neuen Wert wieder nach draußen zurückgeben).

Der „Methodenmanager“ hat keinen Zugriff auf Variablen außerhalb der Methode. Diese sind für ihn unsichtbar!

Ist eine Veränderung der Variablen a gewünscht, so ist die Anweisung hilf=erhoehen(a); durch a = erhoehen(a); zu ersetzen!

Die Ausgabe des Programms wäre dann:
Wert von a vorher: 1
Wert von a nachher: 11



Call by reference:

Erfolgt der Aufruf mit einem strukturierten Datentyp, (etwa mit dem Feldtyp) dann handelt es sich immer um einen Call by reference –Aufruf !

Beispiel:

```
class CallByReference {
  public static void main(String[] args){
    int [] feld = {1,2,3,4,5,6};

    System.out.println("Demo Call by reference : " );
    System.out.println("Feldinhalt vorher  : " );
    for (int i=0; i<=5; i++) {
      System.out.print(feld[i]+" ");
    }
    System.out.println(" " );
    erhoehen(feld);
    System.out.println("Feldinhalt nachher: " );
    for (int i=0; i<=5; i++){
      System.out.print(feld[i]+" ");
     }
    System.out.println(" " );
  } // End of main

static void erhoehen(int[ ] a)
  {
    for (int i=0; i<=5; i++) {
     a[i]=a[i]+1;
    }
```

```
  }
} // End of class
```

Möglich wäre auch folgende Methodendefinition:

```
static int[ ] erhoehen2(int[ ] F)
 {
  for (int i=0; i<=5; i++) {
   F[i]=F[i]+1;
  }
  return F;
 }
```

Auch hier ist F nur ein formaler Parameter.

Dem „Methodenmanager“ wird hier die Speicheradresse (Referenz) der Variablen übergeben, die er entsprechend manipulieren soll. (a ist hier nur ein formaler Parameter, die in Wirklichkeit nicht existiert !)