

Grundlagen der Programmierung



TECHNISCHE
UNIVERSITÄT
DARMSTADT

Vorlesungsskript zum Sommersemester 2020

5. Vorlesung (18. Mai 2020)



Dr. Jin Gerlach

Christian Olt

Fachgebiet Wirtschaftsinformatik | Software & Digital Business

Fachbereich Rechts- und Wirtschaftswissenschaften

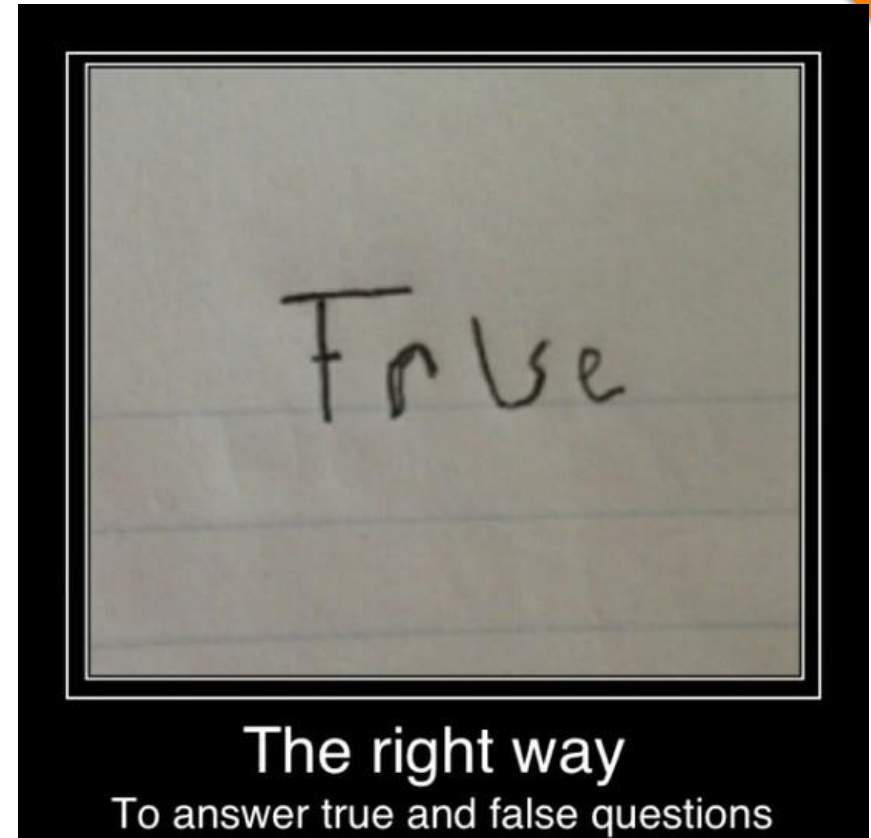
Technische Universität Darmstadt

True oder false?



ISCHE
TAT
T
D.
<http://pingo.upb.de>
#9456

1. Die do-while-Schleife ist eine kopfgesteuerte Schleife.
2. Jede for-Schleife kann in eine while-Schleife umgewandelt werden.
3. Die Eigenschaft „length“ eines gegebenen Arrays liefert den Index des letzten Elements des Arrays.
4. Ein Array muss erst erzeugt werden, bevor es verwendet werden kann.





do-while-Schleife

- Eine „do-while“-Schleife ist eine sogenannte „fußgesteuerte“ Schleife (Versus der kopfgesteuerten „while“-Schleife)
- **Syntax:** `do {Anweisung} while (Bedingung) ;`
- Wiederhole Anweisung, solange Bedingung (d.h. ein Boolean-Ausdruck) gleich `true` ist

- Beispiel:

```
int n = 1;
```

```
do {
```

```
    n = n + 1;
```

```
    } while (n <= 3);
```

```
// nach Durchlauf der Schleife ist n = 4
```



„for“-Schleifen vs. „while“-Schleifen

- Jede for-Schleife kann in eine while-Schleife transformiert werden!
- Die **for-Schleife**

```
for (Initialisierung; Bedingung; Aktualisierung)  
    {Anweisung}
```

ist **äquivalent** zu der **while-Schleife**

```
Initialisierung ;  
while (Bedingung) {  
    Anweisung ;  
    Aktualisierung ;  
}
```



Strukturierte Datentypen: Arrays

- Ein Array (Feld) ist ein strukturierter Datentyp
- Ein Array fasst mehrere Werte (gleichen Datentyps) zu einer Einheit zusammen. Er ist mit einem „Regal“ vergleichbar, in dem die Plätze durchnummeriert (d.h. mit einem Index versehen) sind.
- Zu jedem beliebigen Datentyp T existiert ein zugehöriger Array-Datentyp $T[]$, z. B. `int[]`, `char[]`, ...

Index	0	1	2	3
Datentyp	<code>int</code>	<code>int</code>	<code>int</code>	<code>int</code>
Inhalt	40	50	20	5





Arrays erzeugen

- Ein Array muss erst erzeugt werden, bevor er verwendet werden kann

- **Statische Erzeugung:**

```
int[] zahlen =           // Deklaration
    {40, 50, 20, 5};     // Statische Erzeugung
```

- **Dynamische Erzeugung:**

```
int[] zahlen =           // Deklaration
    new int[4];           // Dynamische Erzeugung mit Defaultwerten
zahlen[0] = 40;
zahlen[1] = 50;
...
```

- Das erzeugte Array wird zur Initialisierung der Array-Variablen verwendet

Mehrdimensionale Felder



TECHNISCHE
UNIVERSITÄT
DARMSTADT

Wiederholung

Erweitert

- Mehrdimensionale Felder sind Felder von Feldern
- Felder können beliebig geschachtelt werden
- Bei Deklaration, Erzeugung und Benutzung müssen entsprechend viele Paare von eckigen Klammern angegeben werden
- Beispiel:

```
int[][] nDim = { {10, 20, 5}, {1, 2} };
```

Beispiel: mehrdimensionales Array



```
int[][] nDim = { {10, 20, 5}, {1, 2} };  
System.out.println(nDim[0][0]); // 10  
System.out.println(nDim[0][1]); // 20  
System.out.println(nDim[0][2]); // 5  
System.out.println(nDim[1][0]); // 1  
System.out.println(nDim[1][1]); // 2
```

	Spalte 0	Spalte 1	Spalte 2
Zeile 0	10	20	5
Zeile 1	1	2	



Mehrdimensionale Arrays und Schleifen

- Mehrdimensionale Arrays werden typischerweise in mehreren verschachtelten for-Schleifen durchlaufen
- Bildschirmausgabe eines zweidimensionalen Arrays (einer Matrix):

```
int [][] matrix = { {1,0,0}, {0,1,0}, {0,0,1} };  
for (int i=0; i < matrix.length; i++) {  
    for (int j=0; j < matrix[i].length; j++) {  
        System.out.print(matrix[i][j]);  
    }  
    System.out.println();  
}
```

Kapitel 2: Grundlagen der Programmierung



TECHNISCHE
UNIVERSITÄT
DARMSTADT

- „Modularität“ als Programmierkonzept
- Methoden in Java
- Überladene Methoden
- Casting: Typenumwandlung in Java

Lernziele:

- Das Konzept „Modularität“ verstehen, erklären und anwenden können
- Gegebene Java-Methoden verwenden und eigene Java-Methoden erstellen können
- Explizites und implizites Casting erklären und anwenden können



Beispiel: Der euklidische Algorithmus

Algorithmus zur Bestimmung des größten gemeinsamen Teilers (ggT) von Euklid (ca. 300 v. Chr):

Solange x ungleich y ist, wiederhole:

Wenn x größer als y ist, dann:

ziehe y von x ab und weise das Ergebnis x zu.

Andernfalls:

ziehe x von y ab und weise das Ergebnis y zu.

Wenn x gleich y ist, dann:

x (bzw. y) ist der gesuchte größte gemeinsame Teiler.



Beispiel: Programm zur Berechnung der ggT von 90 und 81 sowie von 5930 und 43!



TECHNISCHE
UNIVERSITÄT
DARMSTADT

```
public static void main(String[] args){
    int s = 5930;
    int t = 43;
    int u = 90;
    int v = 81;
    while (s != t){
        if(s > t){
            s = s - t;
        } else {
            t = t - s;
        }
    }
    System.out.println(s);
    while (u != v){
        if(u > v){
            u = u - v;
        } else {
            v = v - u;
        }
    }
    System.out.println(u);
}
```



Konzept „Modularität“

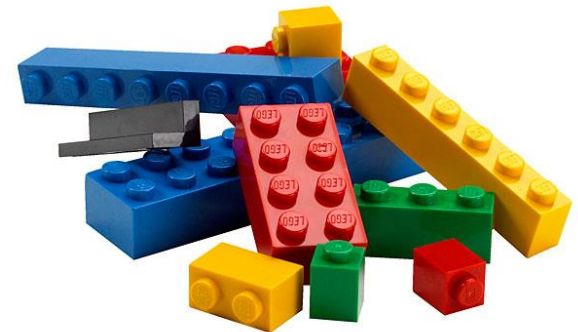
■ Gütekriterien:

1. Zerlegbarkeit
2. Zusammensetzbarkeit
3. Verständlichkeit
4. Kontinuität
5. Protektion



■ Modularität in **Java**:

- Methoden
- Klassen
- Pakete
- Bibliotheken

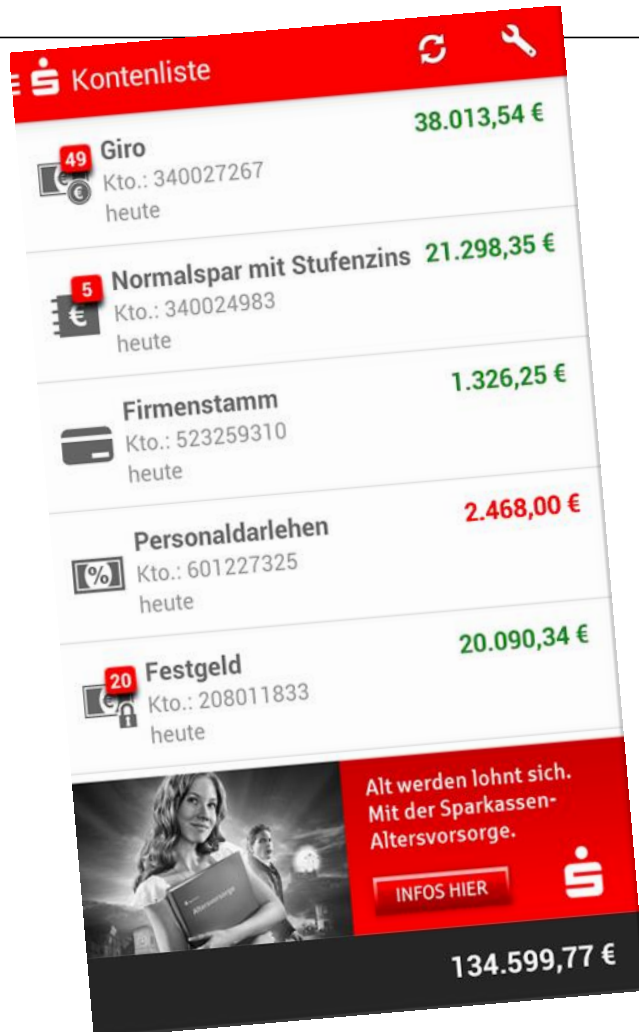


Meyer, B. 1997. *Object-Oriented Software Construction*. Prentice Hall New York.

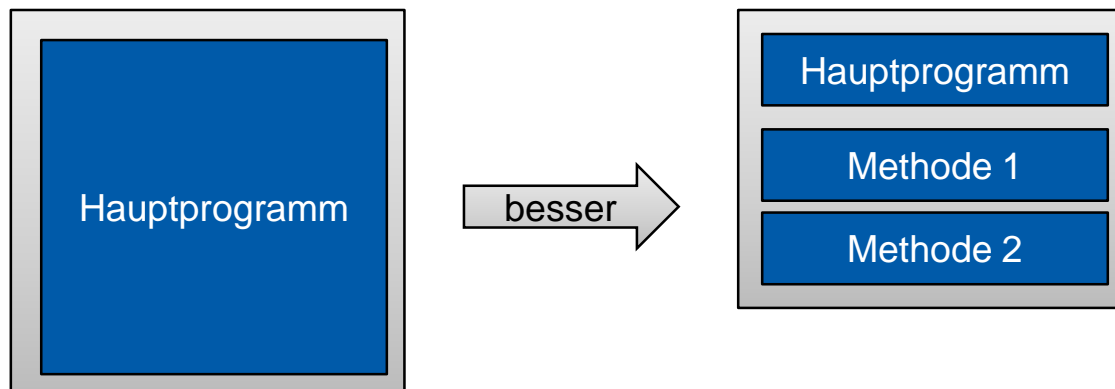
Methoden (Beispiele)



TECHNISCHE
UNIVERSITÄT
DARMSTADT



- Methoden sind **Unterprogramme**
- Sie stellen (oftmals häufig wiederkehrende) **Hilfsfunktionen** bereit, die somit nur einmal implementiert, aber beliebig oft durch das Programm aufgerufen werden können



- Wir haben bereits Methoden benutzt, z. B. `println(...)`

Syntax einer Methode

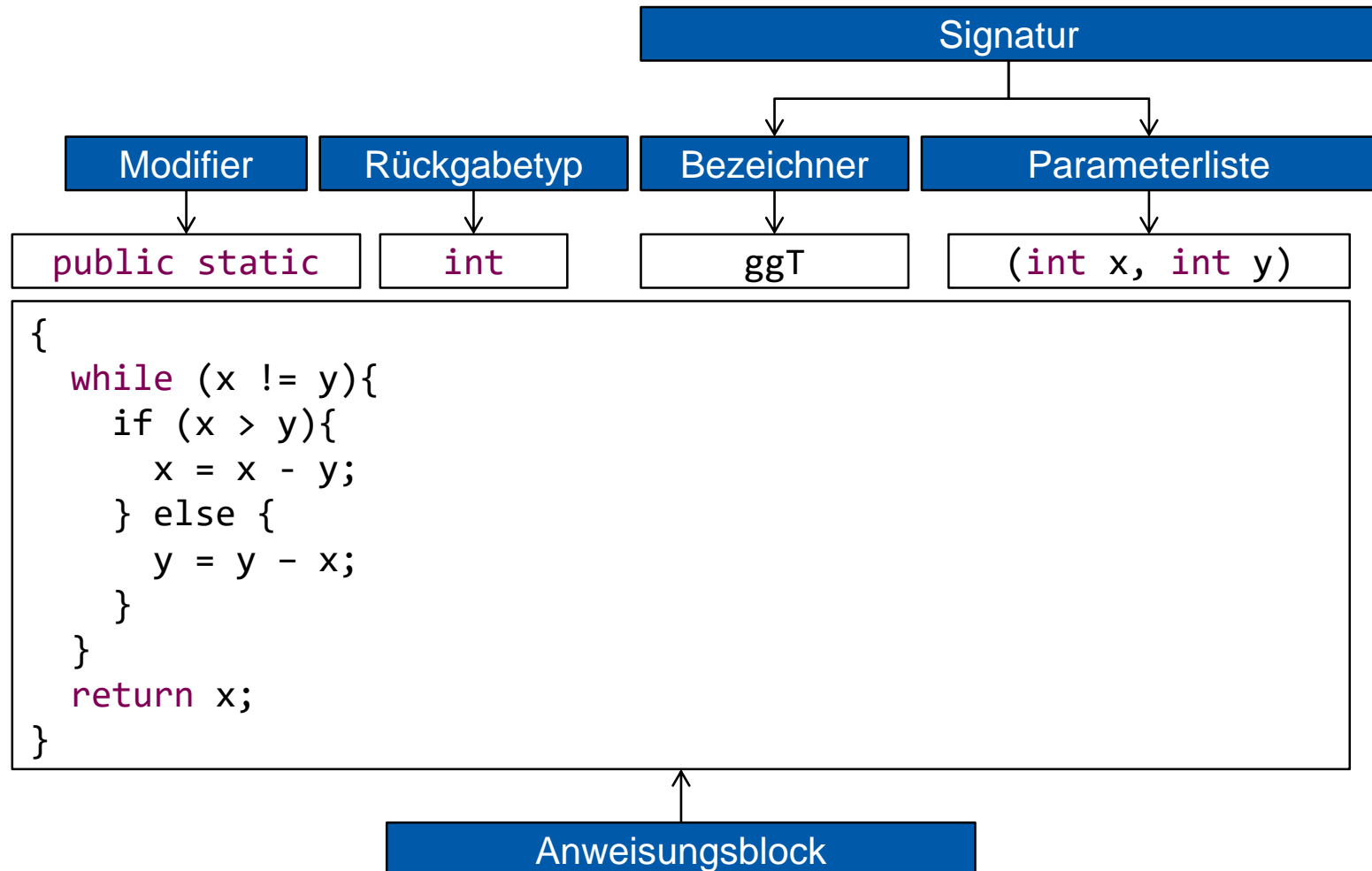
- Syntax:

Modifizier Rückgabetyp **Bezeichner**(Parameterliste) **Anweisungsblock**

- Beispiel:

```
public static int ggT(int x, int y){  
    while (x != y){  
        if (x > y){  
            x = x - y;  
        } else {  
            y = y - x;  
        }  
    }  
    return x;  
}
```


Elemente einer Methode



Parameter einer Methode

- Enthält eine **Signatur** eine Liste mit einem oder mehreren **Parametern**, so müssen der Methode beim Aufruf Werte (in der entsprechenden Reihenfolge) von denselben Datentypen übergeben werden
- Die Parameter können im **Anweisungsblock wie Variablen** verwendet werden. Hinweis: Diese Variablen sind nur innerhalb der entsprechenden Methode „sichtbar“
- Benötigt eine Methode **keine Parameter**, so sind die beiden runden Klammern der Signatur **leer**

Rückgabebetyp einer Methode

- Der Rückgabebetyp legt fest, welchen **Datentyp** der Rückgabewert hat, den die Methode bei der Beendigung an den Aufrufer (z. B. das Hauptprogramm) zurückgibt
- Soll eine Methode keinen Wert zurückgeben, so hat sie den Rückgabebetyp `void`
- Der Befehl **`return`** bestimmt, welcher Wert zurückgegeben wird:
`return myValue;`
- Der Aufrufer kann diesen Wert als Ergebnis des Methodenaufrufs weiter benutzen

Beispiel: Programm zur Berechnung der ggT von 90 und 81 sowie von 5930 und 43!



TECHNISCHE
UNIVERSITÄT
DARMSTADT

```
public static void main(String[] args){
    int s = 5930;
    int t = 43;
    int u = 90;
    int v = 81;
    while (s != t){
        if(s > t){
            s = s - t;
        } else {
            t = t - s;
        }
    }
    System.out.println(s);

    while (u != v){
        if(u > v){
            u = u - v;
        } else {
            v = v - u;
        }
    }
    System.out.println(u);
}
```

```
public static int ggT(int x, int y){
    while (x != y){
        if(x > y){
            x = x - y;
        } else {
            y = y - x;
        }
    }
    return y;
}

public static void main(String[]
args){
    System.out.println(ggT(90, 81));
    System.out.println(ggT(5930, 43));
}
```

Aufgabe: Methodenaufrufe



ISCHE
TÄT
T
D
<http://pingo.upb.de>
#9456

Gegeben ist die folgende Methode:

```
public static int doIt(int s) {  
    System.out.println(s);  
    return (s + 42);  
}
```


Welche der Aufrufe sind ausführbar?

1. `int x = doIt(42);`
2. `doIt(4);`
3. `int y = doIt(42.1);`
4. `int z = doIt();`
5. `int a = doIt(1, 2);`

Ablauf eines Methodenaufrufs (I)

Wird eine Methode aufgerufen, so unterbricht der Anweisungsblock, in dem der Aufruf stattfindet, seine Abarbeitung bis alle Anweisungen der aufgerufenen Methode abgearbeitet sind

```
public static int kgV(int a, int b){  
    int c = ggT(a, b);  
    int erg = (a * b) / c;  
    return erg;  
}
```



```
public static void main(String[] args){  
    int a = kgV(90, 81);  
}
```


```
public static int ggT(int x, int y){  
    while (x != y){  
        if(x > y){  
            x = x - y;  
        } else {  
            y = y - x;  
        }  
    }  
    return y;  
}
```

Ablauf eines Methodenaufrufs (I)

Animation

Wird eine Methode aufgerufen, so unterbricht der Anweisungsblock, in dem der Aufruf stattfindet, seine Abarbeitung bis alle Anweisungen der aufgerufenen Methode abgearbeitet sind

```
public static int kgV(int a = 90, int b = 81){  
    int c = 9;  
    int erg = 810;  
    return 810;  
}
```



```
public static void main(String[] args){  
    int a = 810;  
}
```

```
public static int ggT(int x = 90, int y = 81){  
    while (x != y){  
        if(x > y){  
            x = x - y;  
        } else {  
            y = y - x;  
        }  
    }  
    return y = 9;  
}
```

- Eine Methode heißt **überladen**, wenn in derselben Klasse gleichnamige Methoden mit unterschiedlicher Signatur existieren (*Overloading*)
- Die überladenen Methoden können unterschiedliche Implementierungen enthalten
- Auswahl der auszuführenden Methode erfolgt anhand der **Signatur**
- Beispiel:

```
public static double average(double a, double b) {  
    return (a + b) / 2;  
}
```

```
public static double average(double a, double b, double c) {  
    return (a + b + c) / 3;  
}
```


Kompiliert der folgende Code?



```
public class MethodReturn {  
    public static int get42() {  
        return 42;  
    }  
  
    public static long get42() {  
        return 42;  
    }  
}
```

Bekannte überladene Methode: `println`

```
public class Overloading {  
    public static void main(String[] args) {  
        System.out.println();  
    }  
}
```

- `println()` : void - `PrintStream`
- `println(boolean arg0)` : void - `PrintStream`
- `println(char arg0)` : void - `PrintStream`
- `println(char[] arg0)` : void - `PrintStream`
- `println(double arg0)` : void - `PrintStream`
- `println(float arg0)` : void - `PrintStream`
- `println(int arg0)` : void - `PrintStream`
- `println(long arg0)` : void - `PrintStream`
- `println(Object arg0)` : void - `PrintStream`
- `println(String arg0)` : void - `PrintStream`

Press 'Ctrl+Space' to show Template Proposals

- ***println***

```
public void println()
```

Terminates the current line by writing the line separator string. The line separator string is defined by the system property `line.separator`, and is not necessarily a single newline character (`'\n'`).

Press 'Tab' from proposal table or click for focus

Ausgaben von Zeichenketten mit `System.out.print()` und `System.out.println()`

- Die Methoden `System.out.print()` und `System.out.println()` unterscheiden sich lediglich dadurch, dass bei der Variante `println()` eine neue Zeile angefangen wird.
- Beide Ausgabe-Methoden können neben numerischen Ausdrücken z. B. auch Text ausgeben – dieser Text steht in Anführungszeichen.
- Beispiel: `System.out.print("Text der ausgegeben wird");`
- Sollen Textbausteine und Variablen kombiniert werden, kann dies mit dem Operator `+` erreicht werden.
- Beispiel:
`int j = 2015;`
`System.out.println("Wir haben das Jahr " + j + "!");`



Details zu Datentypen

- Der **Datentyp** einer Variablen definiert:
 - den Wertebereich
 - einen Standardwert (auch: Default-Wert)
 - eine Menge zugehöriger Operationen
- Der Compiler kann prüfen, dass Variablen nur erlaubte Werte enthalten und nur erlaubte Operationen angewendet werden, dadurch können Fehler vermieden werden
- Beispiel:
`int x;`
 - Variable `x` ist vom Datentyp `int`, der Standardwert ist `0`, der Wertebereich ist `[-2147483648; 2147483647]` und ganzzahlig, erlaubt sind alle arithmetischen Operationen

Typenumwandlung („Casting“)

- Zwar ist Java eine „getypte“ Sprache (d.h. es werden Datentypen verwendet) aber Datentypen können bei Bedarf konvertiert werden (dies wird als „**Casting**“ bezeichnet)
- Beispiel:

```
int i = 5;  
double d = i;
```
- Java unterscheidet zwei Arten des Castings:
 - **implizite** (automatische) Typumwandlung
 - **explizite** (manuelle) Typumwandlung



- Daten eines „untergeordneten“ Datentyps werden automatisch dem „übergeordneten“ Datentyp angepasst
- Ein Datentyp ist einem anderen „übergeordnet“ wenn sein Wertebereich den des anderen enthält, z.B. $W(\text{short}) \subset W(\text{long})$, $W(\text{int}) \subset W(\text{double})$
- `byte` \rightarrow `short` \rightarrow `int` \rightarrow `long` \rightarrow `float` \rightarrow `double`
- Das implizite Casting findet statt:
 - Bei der Auswertung arithmetischer Operationen mit Operanden unterschiedlichen Typs
 - Bei Zuweisungen, wenn der Typ des zugewiesenen Ausdrucks nicht dem der Variablen entspricht
- Beispiel:

```
int i = 5;  
double d = i;
```

- Ein Wert oder eine Variable eines übergeordneten Datentyps kann **explizit** einer Variable eines untergeordneten Datentyps (mit möglichem Verlust von Informationen) zugewiesen werden
Beispiel: Bei der Umwandlung eines Gleitkommawertes in einen ganzzahligen Wert, werden die Nachkommastellen einfach weggelassen
- Der gewünschte Typ für eine Typanpassung wird vor der umzuwandelnden Variable in Klammern () angegeben (Casting hat **hohe Priorität!**)

Beispiel:

```
double d = 3.1415;  
int n = (int) d;      // n = 3
```