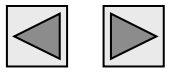
Softwareentwicklung

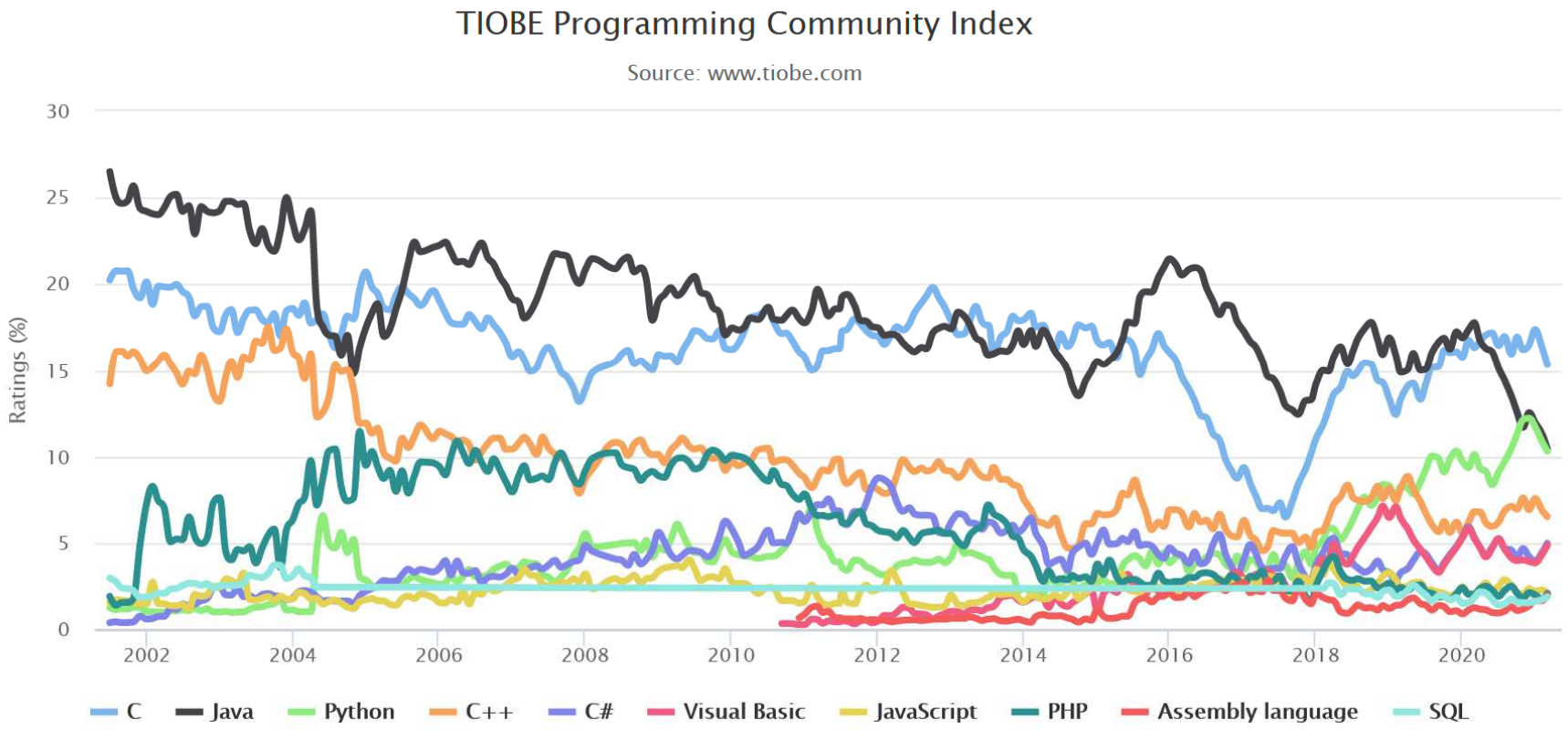
Programmierung mit Java

Teil 1



Übersicht

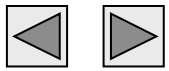
Warum gerade Java? → http://www.tiobe.com/tiobe_index



Übersicht

- Allgemeine Einführung in die Java-Programmierung
  - JRE (JVM+API), JDK
  - Variablen, Konstanten, Datentypen, Arithmet. Operationen
  - Kontrollstrukturen (for, while, do-while, if-else, switch)

- Einführung in Java-spezifische Konzepte
  - Objektorientierte Programmierung
    - Objekte, Klassen, Methoden
    - Vererbung und Polymorphismus
  - Ausnahmebehandlung
  - Ein- / Ausgabeoperationen
  - Grafische Benutzerschnittstelle

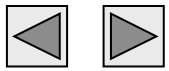


Schritte bis zum Ablauf eines Programms

1. Erstellung des Quelltextes als Textdatei mit der Dateiendung .java
   ➔ Das Programm wird in den Rechner eingegeben.

2. Kompilierung des Quelltextes in Java-Bytecode
   ➔ Das „menschenlesbare“ Programm wird in ein „maschinenlesbares“ Programm übersetzt.

3. Interpretieren des Java-Bytecodes durch die sogenannte *Java Virtual Machine (JVM)*
   ➔ Das Programm wird vom Computer ausgeführt.



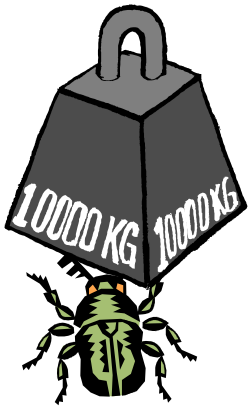
Werkzeuge

- Java Developer Kit JDK
  - Java-Compiler **javac**
    zum Übersetzen von Quelltexten in Java-Bytecode
  - Java-Interpreter **java**
    zur Ausführung von Java-Bytecode-Programmen

- Einfacher Editor (z.B. Notepad)

  Besser: Integrierte Entwicklungsumgebung
  z.B. **Netbeans, Eclipse, Intellij, …**

  Editor zur Erstellung von Quelltexten
  - Automatisches Aufrufen des Compilers und Interpreters
  - Debugger zur Fehlersuche



Das erste Java-Programm: „Hello World“

Programm „*HelloWorld.java*“

```
class HelloWorld {
   public static void main (String args[]) {
     System.out.println("Hallo Welt!");
   }
}
```

Ausgabe:

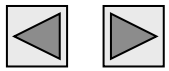
```
Hallo Welt!
```

Javaprogramm compilieren:

```
javac HelloWorld.java   (erzeugt HelloWorld.class)
```

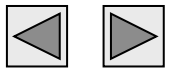
Javaprogramm starten:

```
java HelloWorld
```



Kommentare im Programmquelltext

- Einzeilige Kommentare
  - Beispiel: **// Dies ist ein Kommentar**
- Längere Kommentare
  - Beispiel:
```
/* Dies ist
   ein Kommentar */
```
- Dokumentationskommentare (für JavaDoc)
  - Beispiel:
```
/**
 *  Ein Kommentar für javadoc
 *  Eine <em>HTML-Liste</em>
 *  <ol><li>Eintrag1</li><li>Eintrag2</li></ol>
 */
```



Struktur von Java-Programmen

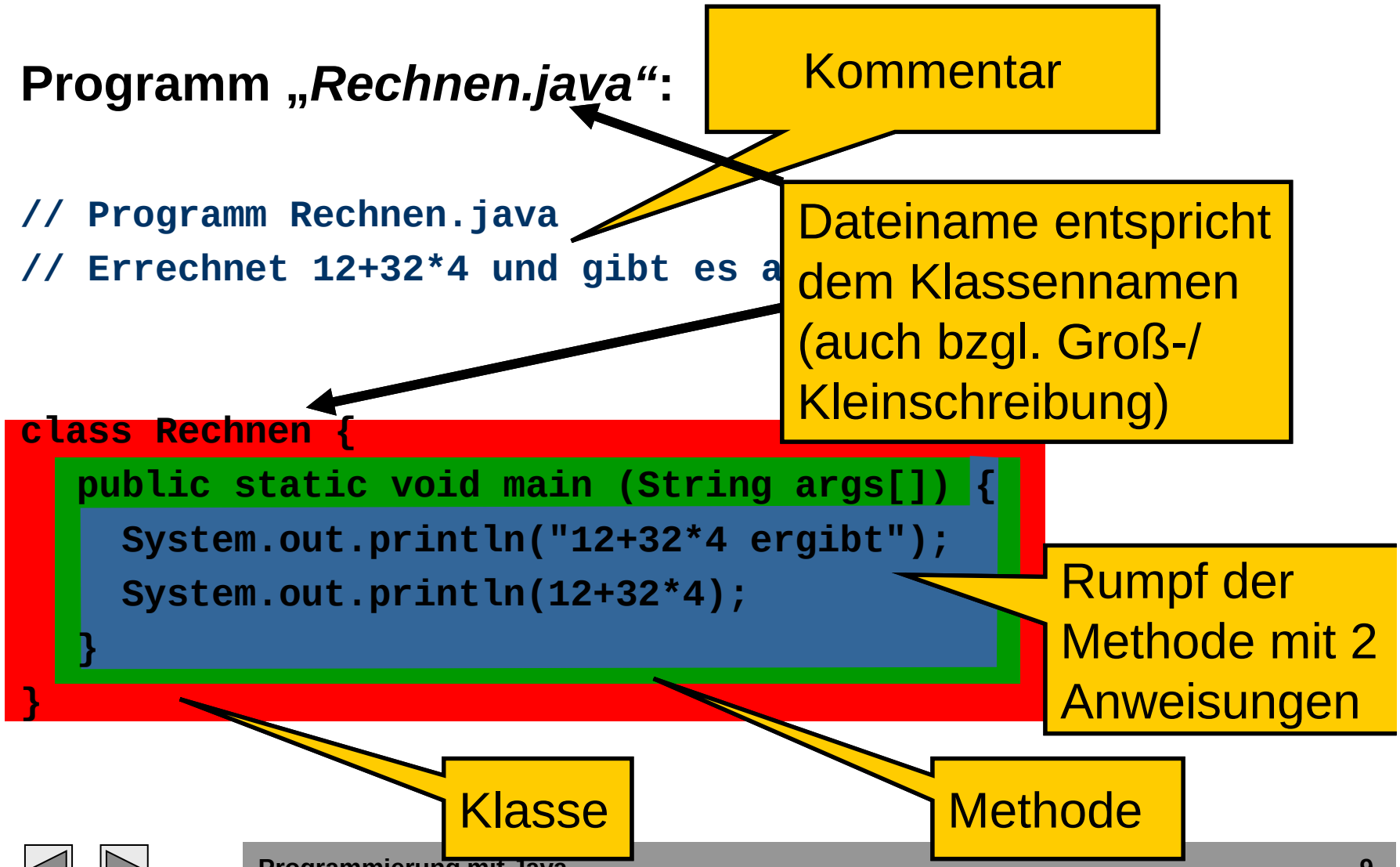
- Programme bestehen aus **Klassen** (class)
  - im Beispiel: nur eine Klasse „HelloWorld"
- Klassen bestehen aus **Methoden**
  - im Beispiel: nur eine Methode „main“
- Methoden bestehen aus einer **Folge von Anweisungen,** getrennt durch Semikolon
  - im Beispiel: **System.out.println(...);**
- Genau eine Klasse hat Methode "**main**" (Hauptprogramm)
- Jede Klasse muss in einer eigenen, gleichnamigen Quelltextdatei definiert werden (.java)

  ➔ergo: pro Klasse eine Datei und umgekehrt

Groß-/Kleinschreibung relevant!
z.B. **H**elloWorld ≠ **h**elloWorld

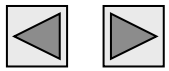
Erläuterung am Programm *Rechnen.java*

Programm „*Rechnen.java*“:



Bezeichner

- **Bezeichner** sind vom Programmierer zu vergebene Namen für
  - Klassen, Methoden, Pakete, Variablen
- **Syntax:** Beliebig lange Zeichenfolge aus
  - Buchstaben, Ziffern,
  - Dem Dollarzeichen **$** oder dem Unterstrich _
- Bezeichner müssen mit einem Buchstaben oder dem Dollarzeichen **$** oder dem Unterstrich _ anfangen
- Keine Leerzeichen
- Keine Schlüsselwörter
  - Schlüsselwörter sind reservierte Wörter wie z.B. *class*, *static*, *void*, *true, false, null*, ...



Variablen: Deklaration

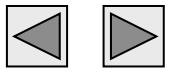
- für Variablen muss angegeben werden, von welchem Typ (z. B. ganze Zahl, Kommazahl, Buchstaben) sie sind
- dies wird als **Deklaration** bezeichnet
- **Variablendeklaration:** ***Typ Variablenbezeichner;***

Beispiele:

```
int anzahl;              // Variable anzahl  ist eine ganze Zahl
char endbuchstabe;       // Variable endbuchstabe ist ein Buchstabe
double z1, z2;           // Variablen z1 und z2 sind Kommazahlen
```

Primitive Datentypen

Typ	Länge	Bereich	
byte	8 Bits	-128 bis 127	Ganze Zahlen
short	16 Bits	-32768 bis 32767	
int	32 Bits	-2147483648 bis 2147483647	
long	64 Bits	noch viel größer als int	
float	32 Bits	+/- 3.4E+38 (8 Stellen genau)	Komma-zahlen
double	64 Bits	+/- 1.8E+308 (17 Stellen genau)	
char	16 Bits	65536 verschiedene Zeichen	
boolean	1 Bit	true oder false	Logik



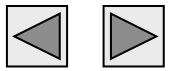
Arithmetische Operatoren

- **+, -, *, / :** Addition, Subtraktion, Multiplikation, Division
- **Division:** ganzzahlig**,** wenn beide Operanden ganzzahlig sind, sonst *float/double*

```
int i = 9 / 4;                  //i ist 2
double d = 3.6 / 6;             //d ist 0.6
```

- **Modulo-Operator %:** Rest der Ganzzahldivision

```
int i = 5;
int j = i % 3;                  // j ist 2
```



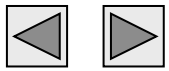
Übungsaufgabe:

Temperaturumrechnung

Schreiben Sie ein Programm, das eine Temperatur in Grad Celsius in Grad Fahrenheit umrechnet, das Ergebnis in einer Variablen speichert und ausgibt. Analog soll von Fahrenheit nach Celsius umgerechnet werden.

Tipp: Formel zur Umrechnung von Fahrenheit nach Celsius:

$$Celsius = 5/9 * (Fahrenheit - 32)$$

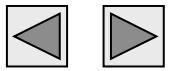


Lösung der Übungsaufgabe:

Temperaturumrechnung Celsius - Fahrenheit und umgekehrt: Celsius = 5/9 * (Fahrenheit - 32)

Programm „*Temperatur.java*“

```
class Temperatur {
   public static void main (String args[]) {
     double celsiusIn = 39.6;
     double fahrenheitIn = 451.0;
     double fahrenheitOut, celsiusOut;
     celsiusOut    =  5.0 / 9.0 * (fahrenheitIn - 32);
     fahrenheitOut =  celsiusIn * 9 / 5 + 32;
     System.out.print("Die Temperatur ");
     System.out.print(fahrenheitIn);
     System.out.print(" entspricht in Celsius ");
     System.out.print(celsiusOut);
     …
   }
}
```



Typkonversion

- **Vom kleineren zum größeren: Problemlos**

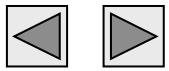
```
double d;
int i = 3;
d = i + 7;
```

- **Vom größeren zum kleineren: Explizit**

```
double d = 3.0;
int i;
i = (int) (d + 7);
```

Wert_mit_neuem_Typ* = (Typname) *Wert_mit_altem_Typ

explizite Typkonvertierung wird als ***Casting*** bezeichnet

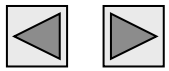


Zahlenwerte in arithmetischen Ausdrücken

- **ganzzahlige Konstanten** (z.B. 113, -73147, 0) werden immer als **int** interpretiert
- **Gleitkommakonstanten** (z.B. 3.1415, -4.56e-10, 0.0) werden immer als **double** interpretiert
- Bei Zuweisung eines konstanten Zahlenwertes zu einer Variablen eines kleineren Typs muss eine explizite Typumwandlung angegeben werden.

  Beispiele:

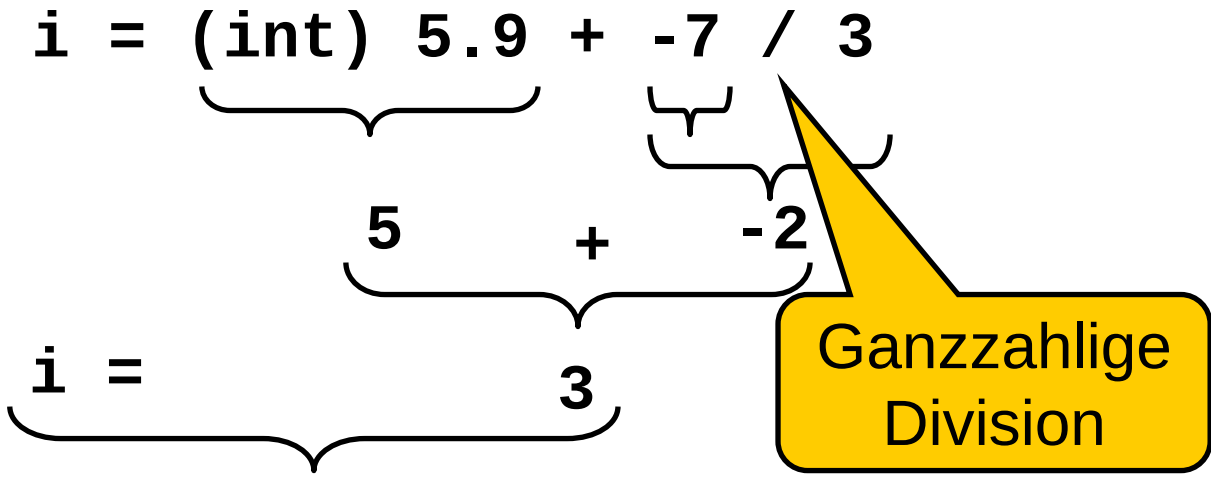
```
float f = (float) 3.0;

short kleineZahl = (short) 10;
```



Operatorenreihenfolge (Priorität)

Priorität	Operatoren		Assoziativität
1	+ -	Vorzeichen (einstellig)	rechts
1	(Typ)	Typkonversion	rechts
2	* / %	Multiplikation, Division, Rest	links
3	+ -	Addition, Subtraktion	links
4	=	Wertzuweisung	links

Beispiel:



Operatorenreihenfolge (Assoziativität)

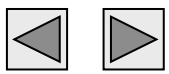
Priorität	Operatoren		Assoziativität
1	+ -	Vorzeichen (einstellig)	rechts
1	(Typ)	Typkonversion	rechts
2	* / %	Multiplikation, Division, Rest	links
3	+ -	Addition, Subtraktion	links
4	=	Wertzuweisung	links

Beispiel: double i = 5 / 10 / 1.0

Ganzzahlige Division

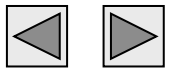
0 / 1.0

i = 0



Mathematische Funktionen: die Klasse "Math"

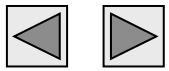
- Die wichtigsten mathematischen Funktionen sind in der Klasse **Math** definiert.
- Funktionen (präziser: Methoden) von Math können in Ausdrücken verwendet werden
  - Aufruf der Funktion *f(x)* durch Voranstellen von des Klassennamens „*Math.*“
  - **Beispiel:** `double a = Math.sin(34.21);`
- Die Klasse Math ist Bestandteil der Standard-Funktionsbibliothek *java.lang*
- Die Funktionen sind in der Online-Dokumentation von Java (des JDK) erläutert.



Funktionen der Klasse Math (1)

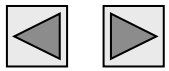
- **Trigonometrische Funktionen**
  - *double sin(double x)* Sinus
  - *double cos(double x)* Cosinus
  - *double tan(double x)* Tangens
  - *double asin(double x)* Arcussinus ($\sin^{-1}$)
  - *double acos(double x)* Arcuscosinus ($\cos^{-1}$)
  - *double atan(double x)* Arcustangens ($\tan^{-1}$)

  Winkel werden im Bogenmaß angegeben!
- **Potenzierung, Wurzeln, Logarithmen**
  - *double exp(double x)* Eulerfunktion e^x
  - *double log(double x)* natürlicher Logarithmus
  - *double pow(double x, double y)* x^y
  - *double sqrt(double x)* Quadratwurzel



Funktionen der Klasse Math (2)

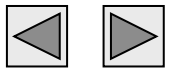
- **Minimum und Maximum**
  - *int min(int x, int y)* minimum(x,y)
  - *long min(long x, long y)* minimum(x,y)
  - *float min(float x, float y)* minimum(x,y)
  - *double min(double x, double y)* minimum(x,y)
  - *int max(int x, int y)* maximum(x,y)
  - *long max(long x, long y)* maximum(x,y)
  - *float max(float x, float y)* maximum(x,y)
  - *double max(double x, double y)* maximum(x,y)
- **Mathematische Konstanten (Pi, E)**
  - *double PI* Kreiszahl π
  - *double E* Eulerzahl e



Funktionen der Klasse Math (3)

- **Runden und Abschneiden**
  - *int abs(int x)* | x |
  - *long abs(long x)* | x |
  - *float abs(float x)* | x |
  - *double abs(double x)* | x |
  - *double ceil(double x)* ⌈ x ⌉
  - *double floor(double x)* ⌊ x ⌋
  - *int round(float x)* ⌊ x+0.5 ⌋

- **Zufallszahlen**
  - *double random()* 0 ≤ Zufallszahl < 1

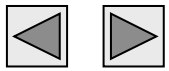


Übungsaufgabe

Simulation eines Würfels:

Schreiben Sie ein Programm, das bei jedem Aufruf eine zufällige ganze Zahl zwischen 1 und 6 ausgibt.

Tipp: Verwende die Methode *random()* der Klasse *Math*!



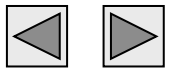
Lösung der Übungsaufgabe Würfel

Schreiben Sie ein Programm, das bei jedem Aufruf eine zufällige ganze Zahl zwischen 1 und 6 ausgibt.

Lösung: Verwendung von Math.random()

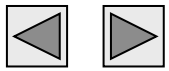
Programm „*Zufall.java*“

```
class Zufall {
   public static void main (String args[]) {
     double z = Math.random();              // 0 <= z < 1
     int wuerfel = (int) (z*6.0+1.0);       // 1 <= wuerfel <= 6
     System.out.print("Zufallszahl: ");
     System.out.println(wuerfel);
   }
}
```

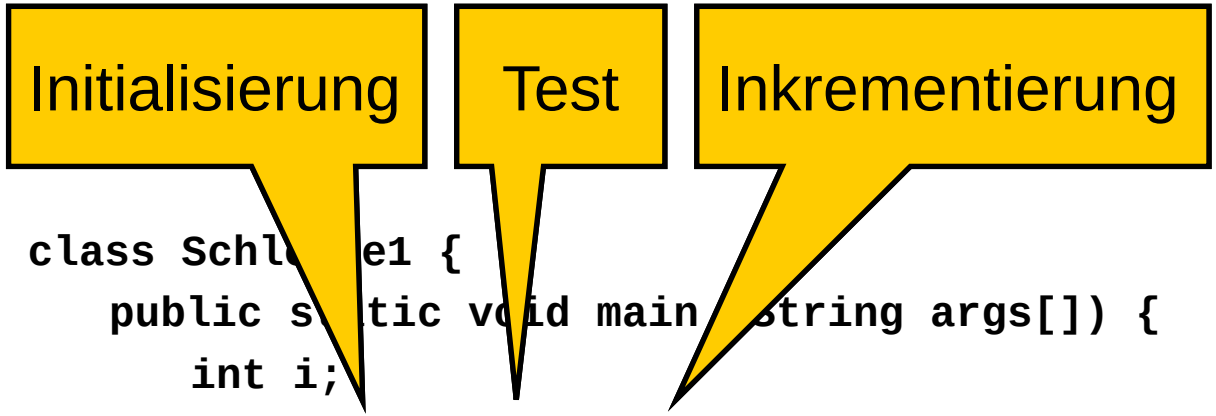


Kontrollstrukturen: for-Schleife

- dient zur Iteration (wiederholten Ausführung) von Anweisungen
- Syntax:
  for (*Initialisierung* ; *Test* ; *Inkrementierung*)
  ***Anweisung*;** // Schleifenrumpf
- ***Initialisierung*** ist eine Anweisung, in der typischerweise eine Zählvariable (i.d.R. ganzzahlig) mit einem Startwert initialisiert wird.
- ***Test*** ist ein boolean-Ausdruck (d.h. er ist entweder *wahr* oder *falsch*). Der Schleifenrumpf wird solange wiederholt, wie der Ausdruck wahr (true) ist.
- ***Inkrementierung*** ist eine Anweisung, in der typischerweise die Zählvariable inkrementiert (erhöht) oder dekrementiert (erniedrigt) wird.



Beispiel zur for-Schleife




```
class Schl   e1 {
   public s  tic vo id main  String args[]) {
      int i;
      for (i=1; i<=5; i=i+1) {
         System.out.println("Hallo Welt!");
      }
   }
}
```

Ausgabe:

```
Hallo Welt!
Hallo Welt!
Hallo Welt!
Hallo Welt!
Hallo Welt!
```

Kontrollstrukturen: if-else

- dient zur bedingten Ausführung von Programmteilen
- Syntax:

```
if ( boolean-Ausdruck )
{
      Anweisung1;              // true-Teil
}
else
{
      Anweisung2;              // false-Teil
 }
```

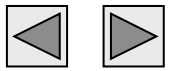
Beispiel zur for-Schleife und if-then-else

Programm:

```
class Schleife2 {
    public static void main (String args[]) {
        int i;
        for (i=1; i<=8; i=i+1) {
            System.out.print(i);
            if (i % 2 == 0)    // ist i modulo 2 = 0?
                System.out.println(" ist gerade");
            else
                System.out.println(" ist ungerade");
        }
    }
}
```

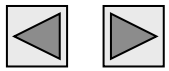
Ausgabe:

```
1 ist ungerade
2 ist gerade
3 ist ungerade
4 ist gerade
5 ist ungerade
6 ist gerade
7 ist ungerade
8 ist gerade
```



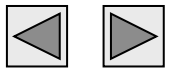
Übungsaufgabe

Erzeugen Sie 1000 zufällige double-Werte und geben Sie das Minimum, das Maximum sowie den Durchschnitt aller Werte aus.



Lösung zur Übungsaufgabe

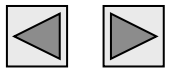
```
public class Statistik {
  public static void main (String args[]) {
    double zufallswert, mittelwert;
    double min = 1;  //Zufallswerte zwischen 0 und 1
    double max = 0;
    double summe = 0;
    for (int i=0; i<1000; i=i+1){
      zufallswert = Math.random();
      summe = summe + zufallswert;
     if (zufallswert < min) min = zufallswert;
     if (zufallswert > max) max = zufallswert;
   }
    mittelwert = summe / 1000;
    System.out.print("Minimum: " + min + " Maximum: " +
        max + " Mittelwert: " + mittelwert);
  }
}
```



Alternative Lösung:

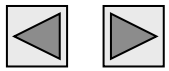
- statt
```
if (zufallswert < min) min = zufallswert;
if (zufallswert > max) max = zufallswert;
```

-
```
min = Math.min(min,zufallswert);
max = Math.max(max,zufallswert);
```



Boolesche Ausdrücke

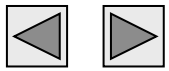
- Bedingungen in **if** oder **for**: **Boolesche Ausdrücke**
- **Boolesche Ausdrücke liefern als Rückgabewert** nur einen der zwei (Wahrheits-)Werte *true* oder *false*
- Beispiele:
  - if(i == 0) ...
  - for(i = 0 ; i < 1000 ; i = i + 1)...



Relationale Operatoren

- Relationale Operatoren bilden spezielle Boolesche Ausdrücke
- Relationale Operatoren setzen ihre Operanden zueinander in Beziehung und liefern den Wahrheitswert der Beziehung zurück.
  - Beispiele: 5 < 3 ergibt *false*
    (4-2)==2 ergibt *true*

Mathem. Notation	Java	Erläuterung
=	==	Identität (Gleichheit)
≠	!=	Ungleichheit
<	<	kleiner als
≤	<=	kleiner oder gleich als
>	>	größer als
≥	>=	größer oder gleich als



Komplexe Boolesche Ausdrücke

- Bildung komplexer Boolescher Ausdrücke mit aussagenlogischen Operatoren (nicht, und, oder):

- **Nicht / Negation:**
  in Java: ! *Wert*

  ¬ true = false
  ¬ false = true

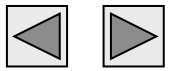
- **Und-Verknüpfung:**
  in Java: *Wert*$_1$ && *Wert*$_2$

  false ∧ false = false
  false ∧ true = false
  true ∧ false = false
  true ∧ true = true

- **Oder-Verknüpfung:**
  in Java: *Wert*$_1$ || *Wert*$_2$

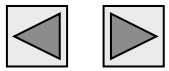
  false ∨ false = false
  false ∨ true = true
  true ∨ false = true
  true ∨ true = true

- Beispiel: ((x < 7) || (y == 6)) && !(z < 5)
- Wert bei x = 2 ; y = 6 ; z = 1?



Operatorenreihenfolge (Priorität)

Priorität	Operatoren		Assoziativität
1	+ -	Vorzeichen (unär)	rechts
1	!	logische Negation	rechts
1	(Typ)	Typkonversion	rechts
2	* / %	Multiplikation, Division, Rest	links
3	+ -	Addition, Subtraktion	links
4	< <=	kleiner, kleiner gleich	links
4	> >=	größer, größer als	links
5	==	Gleichheit	links
5	!=	Ungleichheit	links
6	&&	logisches Und	links
7	||	logisches Oder	links
8	=	Wertzuweisung	rechts



Kontrollstrukturen: while- und do-while-Schleife

while-Schleife:

Syntax:

```
while ( boolean-Ausdruck )
{
    Anweisung;
}
```

- Schleife mit Eintrittsbedingung
  - *Anweisung* wird 0 bis x-mal ausgeführt

do-while-Schleife:

Syntax:

```
do {
    Anweisung;
} while ( boolean-Ausdruck );
```

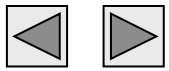
- Schleife mit Wiederholungsbedingung
  - *Anweisung* wird mindestens einmal ausgeführt

Kontrollstrukturen: switch-Anweisung

- dient zur Realisierung von Fallunterscheidungen
- Syntax:

```
switch (Ausdruck )  {
        case Wert1 : Anweisung1; // 1. Fall Ausdruck=Wert1
        case Wert2 : Anweisung2; // 2. Fall Ausdruck=Wert2
        . . .
        default: Ansonsten-Anweisung; // ansonsten
        }
```

- es werden alle Anweisungen ab der ersten erfüllten case-Bedingung ausgeführt (inkl. der *Ansonsten-Anweisung*)
- *Ausdruck* muss vom Typ *byte*, *short*, *int* oder *char* sein.
  Anmerkung: Ab Java 7 (Version 1.7) auch *String* erlaubt
- *WertX* muss eine Konstante vom gleichen Typ wie *Ausdruck* sein

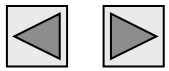


Die break-Anweisung (1)

- dient zum Abbruch von switch-Fallunterscheidungen
- Syntax:

```
switch ( Ausdruck )  {
      case Wert1 :  Anweisung1; break;
      case Wert2 :  Anweisung2; break;
      . . .
      default: Ansonsten-Anweisung;
      }
```

- Der Programmablauf wird bei Ausführung eines break-Befehls hinter der switch-Anweisung fortgesetzt!



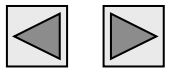
Beispiel zur switch-Anweisung

Programm:

```
class Switch2 {
   public static void main (String args[])  {
    int i;
    for (i=0; i < 10; i=i+1)
        switch(i) {
            case 1: System.out.println("eins"); break;
            case 2: System.out.println("zwei"); break;
            case 5: System.out.println("fünf"); break;
            default: System.out.println(i);
            }
   }
}
```

Ausgabe:

```
0
eins
zwei
3
4
fünf
6
7
8
9
```



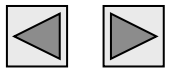
Die break-Anweisung (2)

- kann auch zum Abbruch von Schleifen (for, while, do-while) eingesetzt werden

  Beispiel while-Schleife:

```
while ( boolean-Ausdruck1 )  {
        . . .
        if ( boolean-Ausdruck2 ) break;  // Abbruch
        . . .
}
```

- Der Programmablauf wird hinter der Schleifenanweisung fortgesetzt

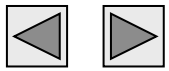


Übungsaufgabe zu switch

Schreibe ein Programm, das den Wert einer ganzzahligen Variablen w im Bereich von 20 bis 99 so als Text ausgibt, wie er ausgesprochen wird. Gebt dabei alle Werte von 20 bis 99 entsprechend aus.

Ausgabe:

w = 20	Ausgabe: "zwanzig"
.....	
w = 98	Ausgabe: „achtundneunzig"
w = 99	Ausgabe: "neunundneunzig"



Lösung der Übungsaufgabe zu switch

```
public static void main (String args[]) {
       for (int i = 20; i <100; i++){
           int a = i % 10;   //Einer-Stelle
           int b = i / 10;   //Zehner-Stelle
           switch (a){       //Anfang mit Einer-Stelle
                case 0: System.out.print("");break;
                case 1: System.out.print("einund");break;
                case 2: System.out.print("zweiund");break;
                ..............................
                case 9: System.out.print("neunund");break;
           }
           switch (b){
                case 2: System.out.println("zwanzig");break;
                case 3: System.out.println("dreißig");break;
                ......................
                case 9: System.out.println("neunzig");break;
           }
       }
}
```