

*Namen und Bezeichner, Variablen, Zuweisungen, Konstanten, Datentypen,  
Operationen, Auswertung von Ausdrücken, Konversion, Kontrollfluss: if..then, while, do .. while, for  
Monte-Carlo Simulation, Zufallszahlen, Linearer Kongruenzgenerator  
Methoden*

## **2. JAVA I**

# Namen und Bezeichner

Ein Programm braucht einen Namen

```
public class RandomWalk{
```

- Konvention: jedes Wort im Namen mit Grossbuchstaben

**Bezeichner:** Ein Name für "Objekte" im Programm.

- Namen müssen mit eine Buchstaben beginnen (oder \_ oder \$)
- Danach Folge von Buchstabe oder Zahl
  - **legal:**        `_myName`        `TheCure`        `ANSWER_IS_42`        `$bling$`
  - **illegal:**      `me+u`            `49ers`            `side-swipe`        `Ph.D's`

# Schlüsselwörter

sind von der Sprache Java belegt und dürfen nicht verwendet werden:

abstract	default	if	private	this
boolean	do	implements	protected	throw
break	double	import	<b>public</b>	throws
byte	else	instanceof	return	transient
case	extends	int	short	try
catch	final	interface	<b>static</b>	<b>void</b>
char	finally	long	strictfp	volatile
<b>class</b>	float	native	super	while
const	for	new	switch	
continue	goto	package	synchronized	

# Variablen

- repräsentieren (wechselnde) Werte
- haben Name, Typ, Wert und Adresse
- sind im Programmtext sichtbar

## Deklaration:

Typname Variablenname [Initialisierung];

## Beispiel

```
int a;
```

definiert Variable mit

- Name: a
- Typ: int
- Wert: default Wert (hier: 0)
- Adresse: Durch Compiler / Laufzeit bestimmt

# Zuweisungsoperator =

Der "="-Operator ist in Java eine Zuweisung, kein Vergleich!

Einen Moment bitte, eine Zuweisung ist ein Operator?

Tatsache: Zuweisung gibt den Wert der zugewiesen wurde zurück.

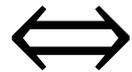
In Pascal ist eine Zuweisung eine Anweisung (Statement) und kann nicht als Teil eines Ausdrucks verwendet werden. In Java ist das anders.

## Beispiele

```
int a = 3;  
double b;  
b = 3.141;  
int c = a = 0;  
String name = "Inf";
```

# Arithmetische Zuweisungen

$a += b$



$a = a + b$

analog für  $-$ ,  $*$ ,  $/$

## Beispiele

```
x += 3;      // x = x + 3;
```

```
name += "x"  // name = name + "x"
```

```
num *= 2;    // num = num * 2;
```

```
for (int i = 0; i < 128; i *= 2) {  
    ...  
}
```

# Konstanten

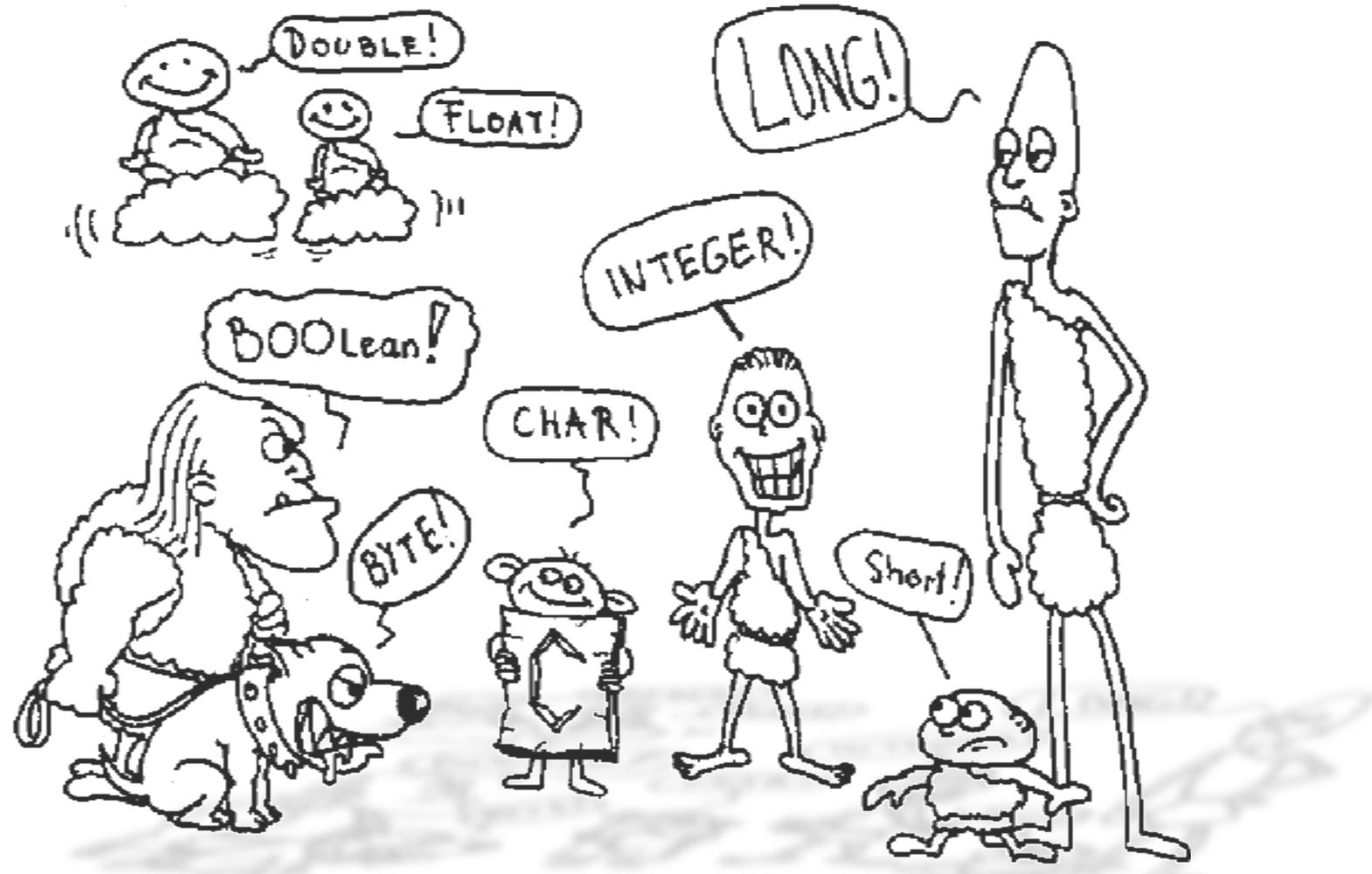
Schlüsselwort **final**

```
final double E = 2.7182818284;
```

```
final int Length = 100;
```

```
final String name = "Informatik";
```

# Datentypen



# Primitive Typen

## Ganze Zahlen:

- **byte** (8 Bits)
- **short** (16 Bits)
- **int** (32 Bits)
- **long** (64 Bits)



Bereich: -2147483648 ... 2147483647  
 $-2^{31} \dots 2^{31}-1$

## Fliesskommazahlen

- **float** (32 Bits)
- **double** (64 Bits)



Beispiele: 18.0 , -0.18e2 , .341E-2

## Zeichen (Unicode):

- **char** (16 Bits)

## Booleans:

- **boolean**



Werte: **true** , **false**  
Operatoren: **&&**, **|**, **!**

# Numerische Operatoren

**+** **-** **\*** **/** **%**

modulo

# Ganzzahldivision

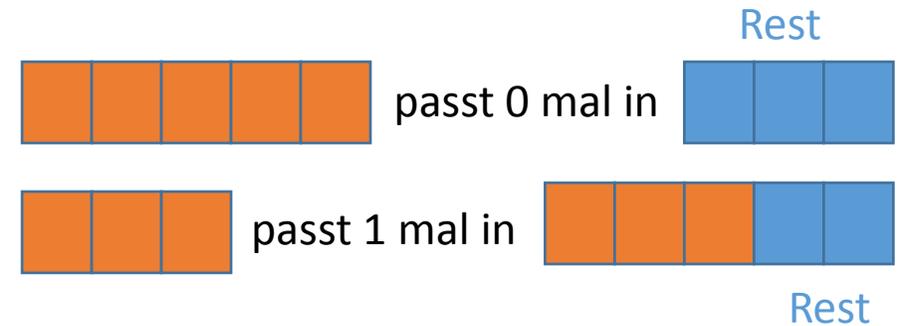
int / int  $\rightarrow$  int !

3 / 5 ergibt 0 (int)

3 % 5 ergibt 3

5 / 3 ergibt 1 (int)

5 % 3 ergibt 2



## Allgemeine Rechenregeln für Modulo-Operationen

$$(a+b) \% c = ((a \% c) + (b \% c)) \% c$$

$$(a*b) \% c = ((a \% c) * (b \% c)) \% c$$

Bsp.:

$$\begin{aligned} 17 \% 5 &= ((3*5) \% 5 + 2 \% 5) \% 5 \\ &= ((3 \% 5) * (5 \% 5) + 2 \% 5) \% 5 \\ &= (3 * 0 + 2) \% 5 \end{aligned}$$

# Auswertung von arithmetischen Ausdrücken

Was benötigen wir?

- Präzedenz:

wer bindet stärker?

$$3 + 5 * 2 = 3 + (5 * 2)$$

Punkt vor Strich  
Arithmetisch vor Logisch  
! vor && vor ||

- Assoziativität

zuerst links oder zuerst rechts?

$$3 - 5 - 2 = (3 - 5) - 2$$

von links nach rechts  
ausser für Zuweisung

- (Stelligkeit)

unär oder binär

-3 ist unär, 1 - 3 ist binär

unär:  
vorgangestelltes -, + oder !

der Operatoren (plus explizite Klammerung)

# Auswertung: wie in der Mathematik

$$3 * 5 + 2 - (5 + 7) / 10 == 5$$

$$3 * 5 + 2 - 12 / 10 == 5$$

$$15 + 2 - 1 == 5$$

$$17 - 1 == 5$$

$$16 == 5$$

false

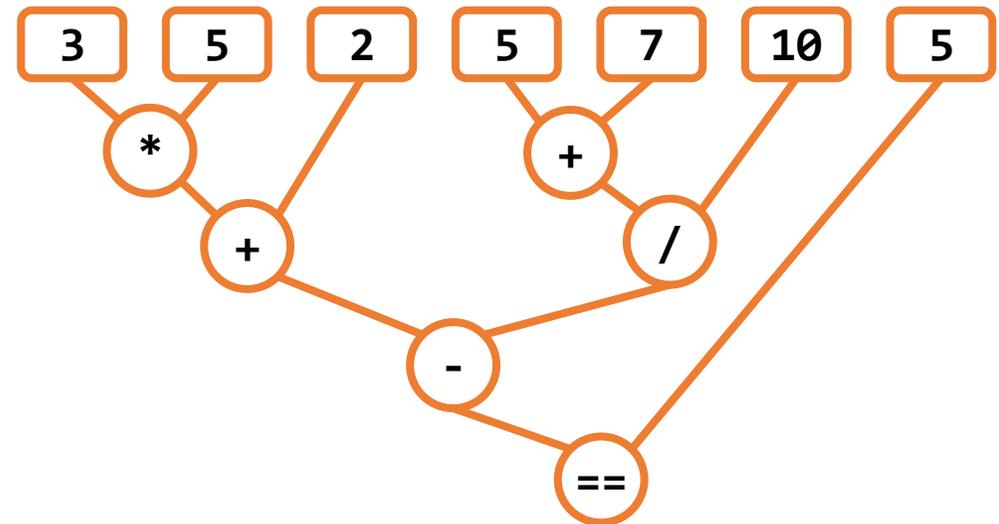
Arithmetik vor Logik, Klammern

Punkt vor Strich

von links nach rechts

von links nach rechts

Logischer Ausdruck



# Beispiel für mod / div: Zeitberechnung

Schreiben Sie ein Programm, welches die momentane Zeit im Format stunde:minute:sekunde ausgibt (z.B. 13:22:15) .

`System.currentTimeMillis()`

gibt die Zeit seit 1. Januar 1970 Mitternacht in Millisekunden zurück

# Zeitberechnung

```
public static void main(String[] args) {  
    long totalMilliseconds = System.currentTimeMillis();  
  
    long sekunde = totalMilliseconds / 1000 % 60;  
    long minute = totalMilliseconds / 1000 / 60 % 60;  
    long stunde = totalMilliseconds / 1000 / 60 / 60 % 24;  
  
    System.out.println(stunde + ":" + minute + ":" + sekunde);  
}
```

# Inkrement / Dekrement Operatoren

```
int x = 10;
```

```
++X ;                    $\Leftrightarrow$  X = X + 1;
```

```
X++ ;                    $\Leftrightarrow$  X = X + 1;
```

Prä-Inkrement

```
y = ++X ;                $\Leftrightarrow$  X = X + 1; y = X;
```

```
y = X++ ;                $\Leftrightarrow$  y = X; X = X + 1;
```

Post-Inkrement

Analog für Dekrement --X

# Typkonversion

Java is a **stark typisiert**

- Der Compiler kann schon während der Übersetzung Typinkompatibilitäten entdecken

```
int myInt;  
float myFloat = -3.14159;  
myInt = myFloat;
```

Fehler zur Kompilationszeit



The type incompatibility problem

# Implizite Konversion

Implizite Konversion bei Zuweisung vom "kleineren" zum "grösseren" Typ (double > float > long > int)

```
int myInteger;
```

```
...
```

```
float myFloat = myInteger;
```

# Konversion bei binären Operationen

Bei einer binären Operation mit numerischen Operanden von verschiedenem Typ werden die Operanden nach folgenden Regeln konvertiert

Haben beide Operanden denselben Typ, findet keine Konversion statt,

**andernfalls:**

Ist einer der Operanden double, so wird der andere nach double konvertiert,

**andernfalls:**

Ist einer der Operanden float, so wird der andere nach float konvertiert,

**andernfalls:**

Ist einer der Operanden long, so wird der andere nach long konvertiert,

**andernfalls:**

Beide Operanden werden nach int konvertiert.

# Explizite Konversion (type cast)

```
int myInt;  
float myFloat = -3.14159f;  
  
myInt = (int)myFloat;
```

**Explizite Konversion (cast) int → float**  
Nachkommastellen werden abgeschnitten  
→ Rundung zur 0 hin

# Kontrollfluss

Bisher (in dieser Vorlesung)

- linearer Kontrollfluss "von oben nach unten"
- unbedingte Ausführung jeder Anweisung

# If-Anweisung

```
if (Bedingung)  
    Anweisung  
else if (Bedingung)  
    Anweisung  
else  
    Anweisung
```

Bedingung:

Ausdruck vom Typ **boolean**

**Beispiel:**

```
if (anzahl > 0)  
    mean = sum / anzahl;
```

# Relationale Operatoren

<, >, <=, >=,  ==, !=

(typ, typ) → boolean

# Anweisungsblöcke

Wo Anweisungen gefordert sind, können Anweisungsblöcke stehen

```
if (Bedingung) {  
    // if Zweig  
}  
else if (Bedingung){  
    // else if Zweig  
}  
else {  
    // else Zweig  
}
```

## Beispiel:

```
if (anzahl > 0){  
    mean = sum / anzahl;  
    ssq = sumsq / anzahl;  
}
```

# Achtung Falle

```
if (anzahl > 0)
    mean = sum / anzahl;
    ssq = sumsq / anzahl;
```

=

```
if (anzahl > 0){
    mean = sum / anzahl;
}
ssq = sumsq / anzahl;
```

!!

# Alle Pfade müssen zurückkehren

Das geht nicht

```
public static int max(int a, int b) {  
    if (a > b) {  
        return a;  
    }  
}
```

**Error: not all paths return a value**

Das geht auch nicht:

```
public static int max(int a, int b) {  
    if (a > b) {  
        return a;  
    } else if (b >= a) {  
        return b;  
    }  
}
```

Der Compiler schliesst, dass beide Bedingungen zu false auswerten könnten. Mathematisch ist das hier unmöglich.

# Schleifen: while

**while** (*Bedingung*)  
*Anweisung*

## Beispiel

```
int num = 143;
```

```
int factor = 2;
```

```
while (num % factor != 0)
```

```
    factor++;
```

```
System.out.println(factor + "/" + num);
```

# Schleifen: do ... while

**do**

*Anweisung*

**while** (*Bedingung*);

## Beispiel

```
Scanner scanner = new Scanner(System.in);  
String password;
```

```
do{
```

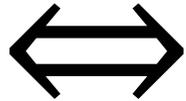
```
    System.out.print("enter magic word:");  
    password = scanner.next();
```

```
} while (!password.equals("InfoII"));
```

```
System.out.println("Door open.");  
scanner.close();
```

# Schleifen: for

```
for (Initialisierung; Bedingung; Fortschritt)  
    Anweisung
```



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

# Schleifen: for

①

②

④

**for** (*Initialisierung; Bedingung; Fortschritt*)  
*Anweisung*

③

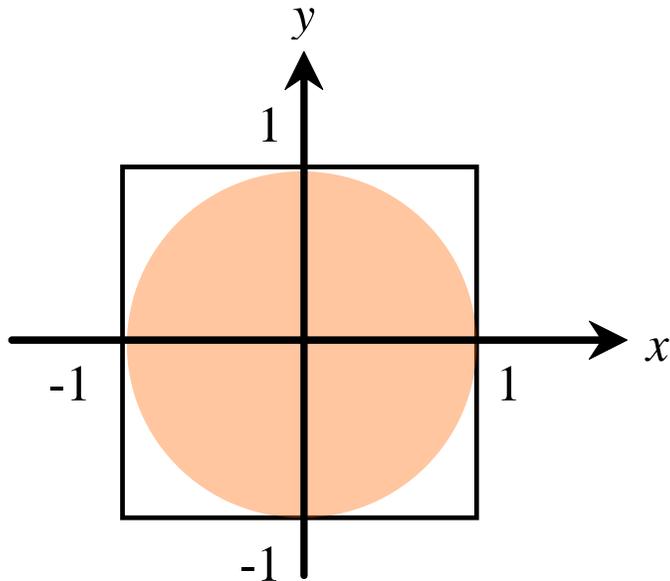
# Beispiele

```
for (int i = 0; i<10; ++i)
{
    System.out.println(i + "*" + i + "=" + i*i);
}
```

```
for (int x = 1; x<=128; x*= 2)
{
    System.out.println("x= " + x);
}
```

# Monte Carlo Simulation

Monte Carlo Simulation: Verwendung des Zufalls zum Lösen von Problemen. Breites Anwendungsfeld in der angewandten Mathematik und sämtlichen Natur- und Ingenieurwissenschaften.  
Einfaches Beispiel: Approximation von  $\pi$ .



$$\frac{\text{Kreisfläche}}{\text{Fläche des Quadrates}} = \frac{\pi}{4}$$

Simuliere Zufallsvariable mit Gleichverteilung auf dem Quadrat.

$$\frac{\#Treffer}{\#Versuche} \cdot 4 \approx \pi$$

# Pseudo-Zufallszahlen

Computersimulation stützt sich ab auf **Pseudozufallszahlen** mit folgenden Anforderungen:

- (1) Gute Approximation einer bekannten Verteilung,
  - z.B. Gleichverteilung auf  $[0,1)$
- (2) Erzeugung möglichst "unabhängiger" Zufallswerte
  - Echte Unabhängigkeit ist bei Pseudozufallszahlen nie gegeben
- (3) schnelle Erzeugung, hohe Präzision
- (4) manchmal: Reproduzierbarkeit

# Einfachste Pseudozufallszahlen: Linearer Kongruenzgenerator

Wähle drei ganze positive Zahlen: **Faktor**  $a$ , **Inkrement**  $b$  und **Modul**  $m$ .

Generiere mit einem **Startwert** ("seed")  $s$  eine Folge von Zufallszahlen wie folgt

$$u_0 = s$$

$$u_{k+1} = (a \cdot u_k + b) \bmod m$$

Diese Pseudozufallszahlen sind unter gewissen Bedingungen an  $a$ ,  $b$  und  $m$  annähernd gleichverteilt auf  $[0, \dots, m-1]$ .

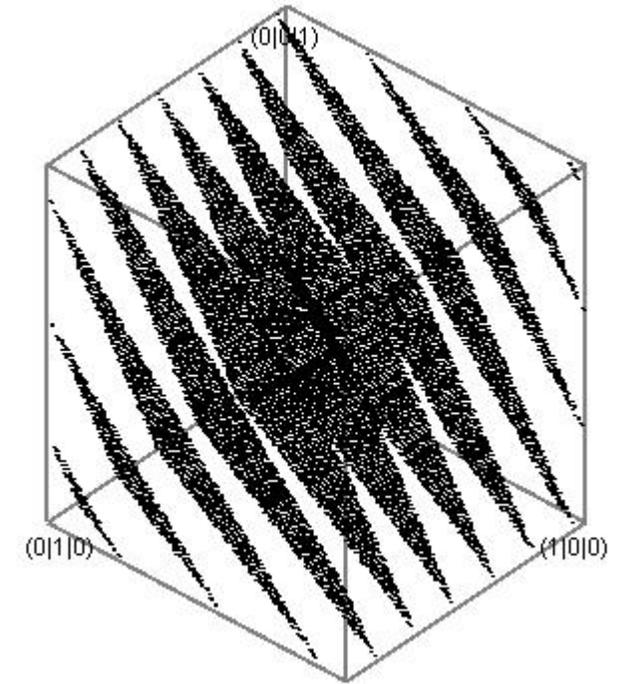
Reellwertige Pseudozufallszahlen  $X_k := \frac{u_k}{m}$  sind approximiert uniformverteilt auf  $[0,1)$ .

# Lineare Kongruenzgeneratoren

Lineare Kongruenzgeneratoren verletzen die "Unabhängigkeitsbedingung" (2) mit Ihren Hyperebenenverhalten besonders stark.

Das gilt insbesondere bei schlechter Wahl der Parameter.

Es gibt bessere Generatoren. Für unsere Zwecke hier genügt der LCG vorerst.



Hyperebenenverhalten in drei Dimensionen  
(Quelle: Wikipedia)

# Beispiel: Ausgabe von Zahlen des LCG

**fehlerhaft !**

```
public static void main(String[] args){  
    // numbers from some historic BSD random generator  
    final long m = 2147483647;  
    final long b = 12345;  
    final long a = 1103515245;  
    long u = 0;  
    for (int i = 0; i<10; ++i)  
    {  
        u = (u * a + b) % m;  
        double d = u / m;  
        System.out.println(u + " : " + d);  
    }  
}
```

## Ausgabe:

```
12345 : 0.0  
1406938949 : 0.0  
178066070 : 0.0  
1543701248 : 0.0  
427461576 : 0.0  
562845833 : 0.0  
1609490218 : 0.0  
377220791 : 0.0  
2040027864 : 0.0  
186150528 : 0.0
```

**Wo ist der Fehler?**

# Beispiel: Ausgabe von Zahlen des LCG

```
public static void main(String[] args){  
    // numbers from some historic BSD random generator  
    final long m = 2147483647;  
    final long b = 12345;  
    final long a = 1103515245;  
    long u = 0;  
    for (int i = 0; i<10; ++i)  
    {  
        u = (u * a + b) % m;  
        double d = (double)u / m;  
        System.out.println(u + " : " + d);  
    }  
}
```

**Interessante Eigenschaft von  
Zufallszahlengeneratoren:  
Bei bekanntem Startwert ist  
der Versuch reproduzierbar!**

# Aufbau einer Klasse

## Eine Klasse besteht aus

- Instanzen- und Klassenvariablen
- **Methoden**
  - Methoden übernehmen die Rolle von Funktionen / Prozeduren in Pascal
  - Konstruktoren sind spezielle Methoden, sie werden beim Erzeugen einer Klasse automatisch aufgerufen
- Code im Klassenkörper
  - wird beim Instanzieren der Klasse ausgeführt
  - falls `static` so wird er bei erstmaliger Verwendung der Klasse ausgeführt

```
public class A{  
  
    public static int F(int x)  
    {  
        ...  
        return j  
    }  
  
    A()  
    {  
        ...  
    }  
  
    { // Code im Klassenkörper  
        ...  
    }  
}
```

# Aufbau einer Klasse

Bei eigenständigen Programmen muss es eine main-Methode geben:

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

Jede Klasse kann eine solche main-Methode enthalten. Sie wird beim *Aufruf der Klasse* ausgeführt.

Klassen können getrennt übersetzt werden

Variablen im Klassenkörper (ausserhalb von Methoden) sind global zu allen Methoden der Klasse

```
public class A{  
  
    ... main (....)  
    {  
  
        ...  
        return j  
    }  
  
    ...  
    int k;  
  
}
```

# Methoden

## Methoden haben immer

- Name
- Rückgabotyp
- (potentiell leere)  
Parameterliste

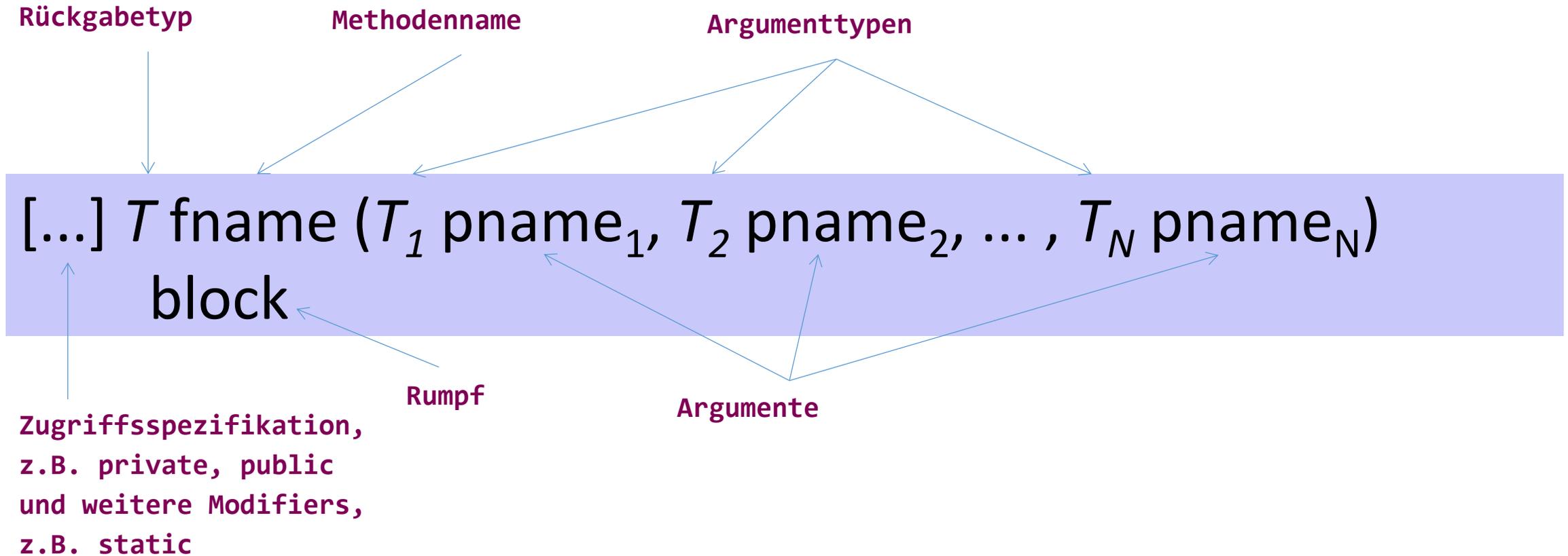
## und optional

- Zugriffsspezifizierer
- Weitere Modifizierer

## Beispiel:

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

# Methodendeklaration



# Statische Methoden

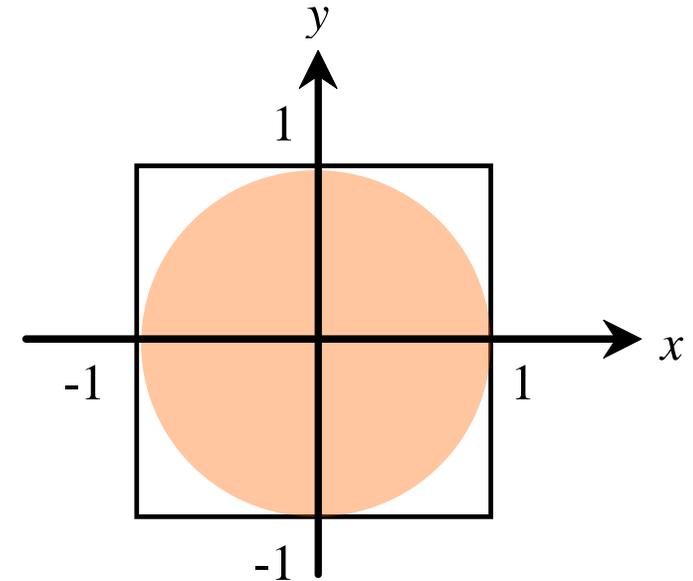
```
class RandomNumbers {  
    // Konstanten  
    static final long m = 2147483647;  
    static final long b = 12345;  
    static final long a = 1103515245;  
    // (globale) Variable  
    static long u = System.currentTimeMillis();  
  
    // returns a pseudo random number in interval [0,1)  
    public static double Uniform(){  
        u = (u * a + b) % m;  
        return(double)u/m;  
    }  
}
```

# Aufruf in der gleichen Klasse

```
class RandomNumbers {  
    ... // wie oben  
  
    public static double Uniform(){  
        ... // wie oben  
    }  
  
    public static double Exponential(double mu){  
        double u = Uniform();  
        return -Math.log(1-u)/mu;  
    }  
}
```

# Wir schätzen $\pi$ mit Monte Carlo Simulation

```
public class MonteCarloPi {  
    public static void main(String[] args) {  
        final int Trials = 10000000;  
        int hits = 0;  
  
        for (int i = 0; i < Trials; i++) {  
            double x = RandomNumbers.Uniform() ;  
            double y = RandomNumbers.Uniform() ;  
            if (x * x + y * y <= 1)  
                hits++;  
        }  
        double pi = 4.0 * hits / Trials;  
        System.out.println("estimate of PI is " + pi + " =? " + Math.PI);  
    }  
}
```



Klassenname.Methodenname

# Credits

Teile dieser Folien inspiriert durch Vorlesungsfolien zu Parallel Programming 2015 von Otmar Hilliges.