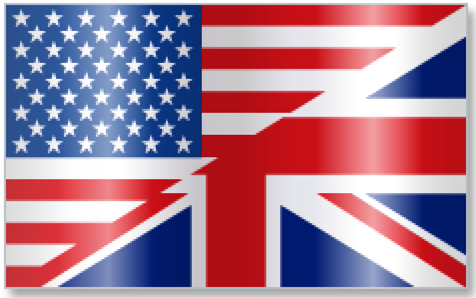
Einführung in die strukturierte und objektbasierte Programmierung (620.200, »ESOP«)

Assoc. Prof. Dr. Mathias Lux

ITEC / AAU



Modalities

- This is the theoretical lecture.
- There is an exam at the end of the semester
  - Most likely on 10.02. 2017, 10-12, HS A see online system.
  - It'll last 100 minutes
  - Don't forget to enroll to the exam!

Schedule

- Thursdays, 14-16, HS C (s.t.)
  - If it's not taking place, there'll be an email and the campus system will be updated.

Practical course & turorial

- Starts next week.
  - Bring your computer if you have one.
- The MORE course
  - Takes place in a computer lab
  - It‘s in English and it will revisit the theoretical part too.

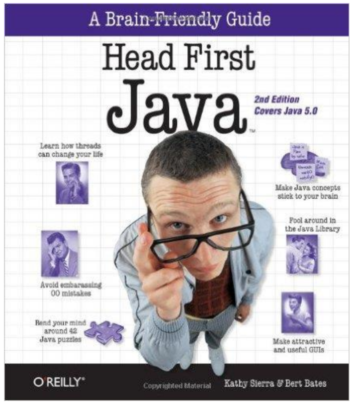
Readings (German)

Hanspeter Mössenböck, *Sprechen Sie Java? Eine Einführung in das systematische Programmieren*
5. Auflage, dpunkt.verlag, 2014
ISBN 978-3-86490-099-0

Readings (English)

Kathy Sierra, Bert Bates (2005) Head First Java (Englisch) Taschenbuch, O'Reilly and Associates;

- This book covers object oriented programming, so there is a gap in the first part. For this I recommend Introduction to Programming Using Java, Seventh Edition by David J. Eck. This book is an extensive introduction to programming based on Java. Read over chapters 1, 2, and 3 to get the necessary background knowledge on variables



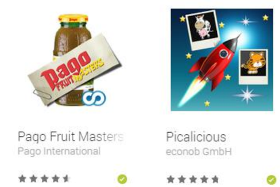
Java Documentation

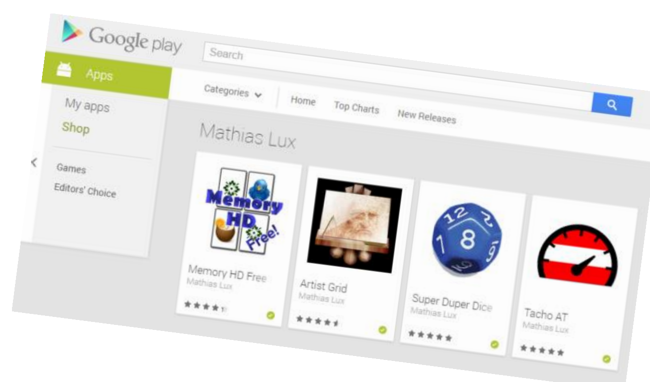
- Java API Doc
  - http://docs.oracle.com/javase/8/docs/api/
- Java Tutorials
  - http://docs.oracle.com/javase/tutorial/

How should I learn Java?

1. Learn to have fun programming. It makes it easier.
2. Invest time in the Java Tutorials and the readings.
3. Go to the course.

Motivation – Why Lux?





Motivation

- It's necessary for research & development
  - Grand Challenge projects, prototypes
- Projects for multimedia production, ie. Processing
- Games, apps, etc.

What is “programming”?

... describing the solution of a problem in such an exact way, that a computer can solve the problem.

Cp. recipes, manuals, etc.

Quelle für die folgenden Folien: ***Grundlagen der Programmierung,*** *Prof. Dr. Hanspeter Mössenböck*

Programming is

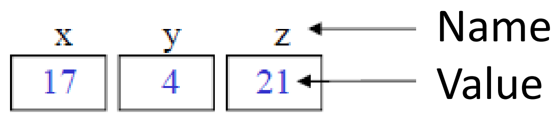
- a creative process
- an engineering skill
- a complex task if you want to doi it right.

What is a program

program = data + commands

Data

- Set of address-able memory cells



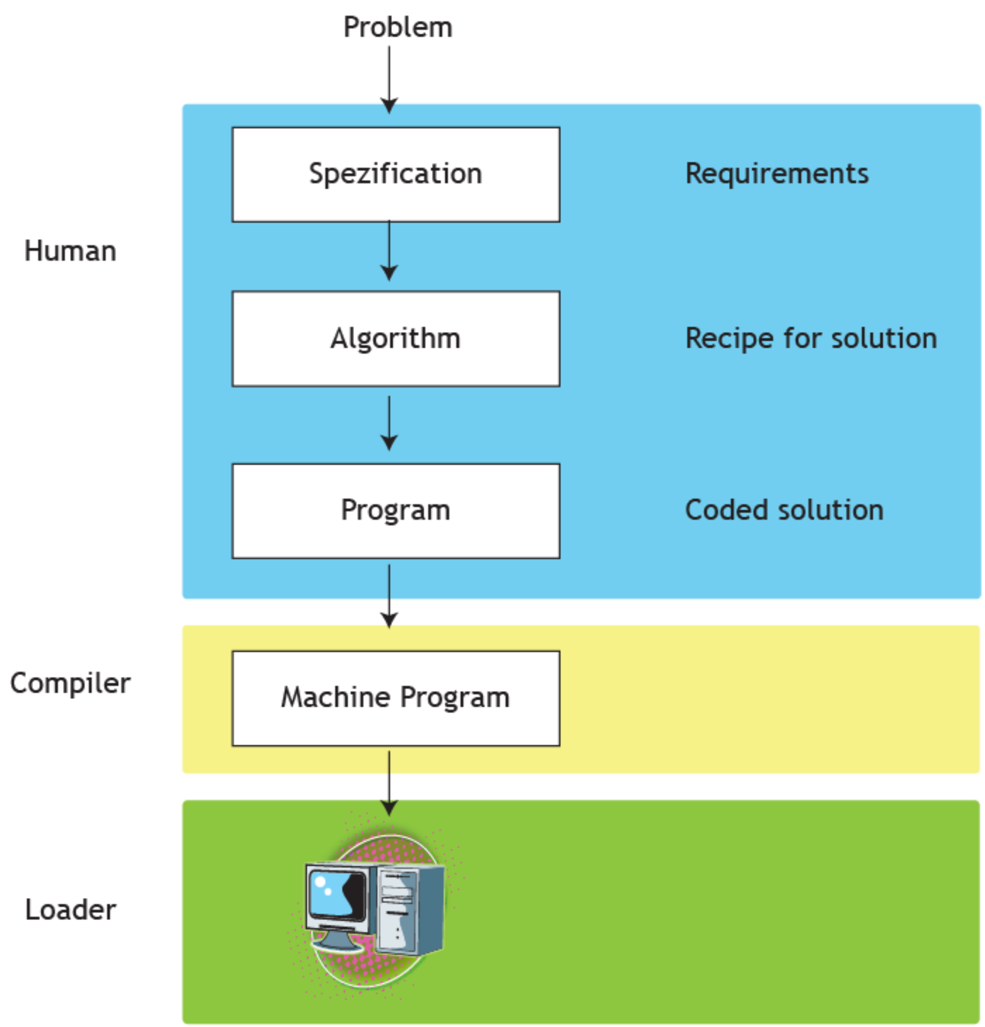
- Data is stored in binary format, eg. 17 = 10001
- Binary format is universal
  - numbers, text, image, audio, ...
- 1 Byte = 8 Bit
- 1 Wort = 4 Byte (typically)

Commands

- Operations on memory cells

Machine language		Programming Language
$ACC \leftarrow x$	// load memory cell x	$z = x + y;$
$ACC \leftarrow ACC + y$	// add memory cell y	
$z \leftarrow ACC$	// store result in memory cell z	

How to create a program?



Algorithm

- Precise, step by step solution to a problem

name

parameters

```
Sum up numbers from 1 to max (in:max, out:sum)
1. sum <- 0
2. number <- 1
3. Iterate as long as number smaller or equal max
    1. sum <- sum + number
    2. number <- number + 1
```

steps

- program = specification of an algorithm in a programming language

Variables

- Variables are named container for values.

x	y
99	3

- Values can change

$x \leftarrow x + 1$

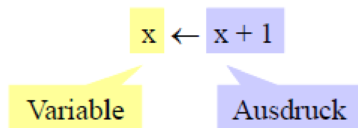
x
100

- Variables have a data type
  - which is the set/range of values allowed for a variable.

Type		Values	
□	Number	17	54
○	Letter	'a'	'x'

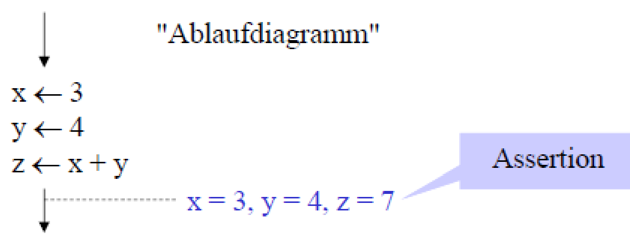
Statements

- Assignement



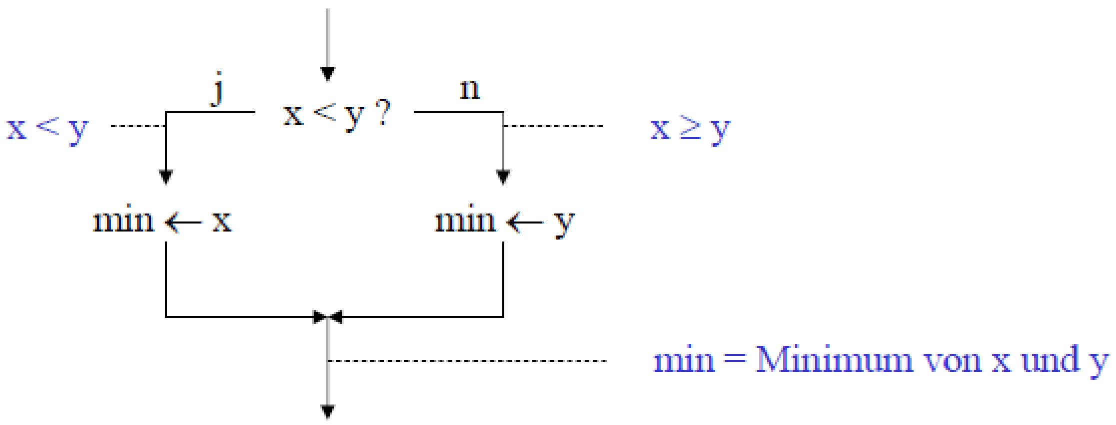
1. compute value
2. assign result to variable

- Sequence of statements



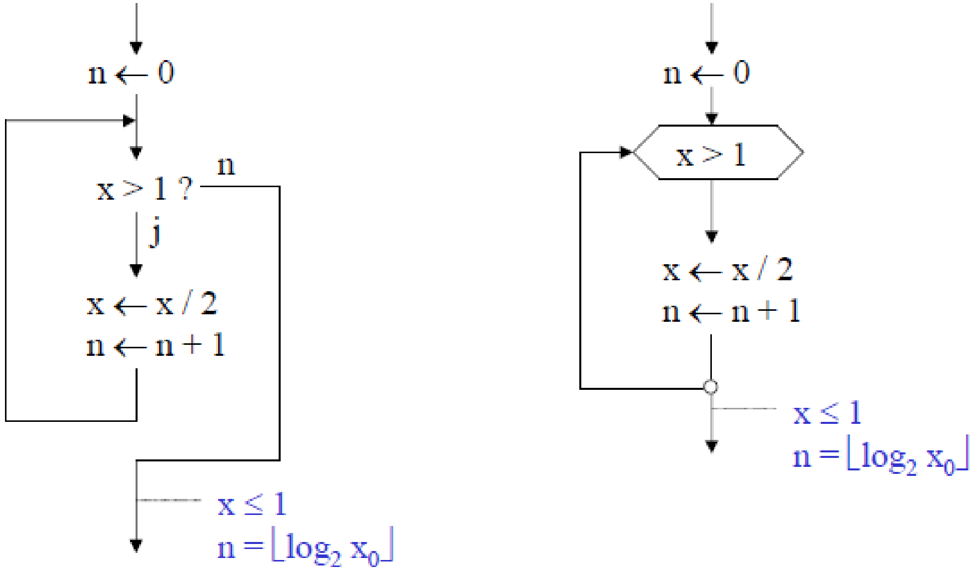
Statements

- Condition / Choice

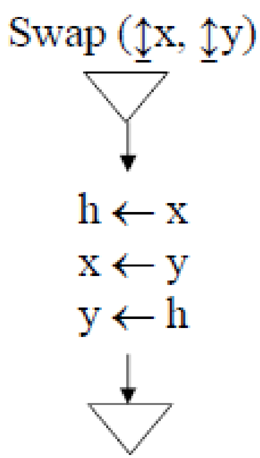


Statements

- Iterations, Loops



Example: swap values



proof of concept

x	y	h
~~3~~	~~2~~	3
2	3	

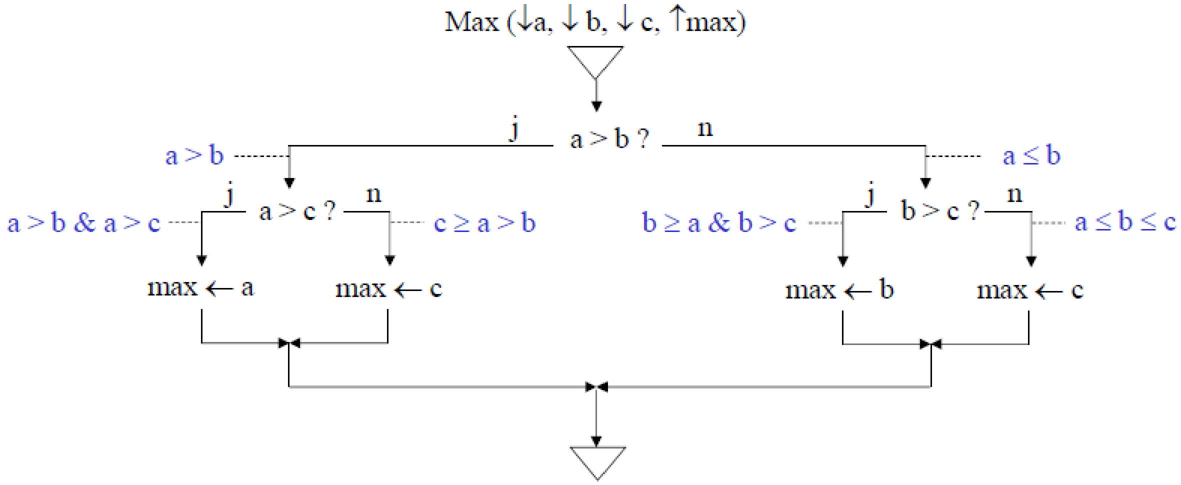
Example: swap values

```
int x = 10;
int y = -5;
int h;

println(x);
println(y);

h = x;
x = y;
y = h;

println(x);
println(y);
```

- Source Code for Processing
- Processing is „like Java“

- int … data type
- ; … ends a statement
- println() … function for printing text on screen.

Example: maximum of three numbers



Example: maximum of three numbers

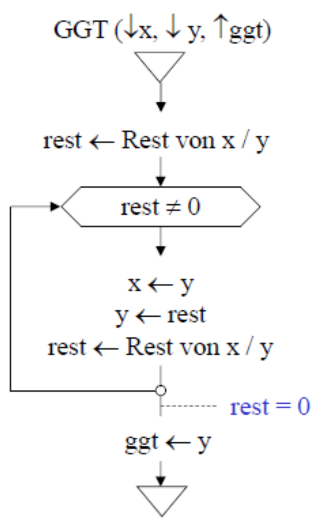
```
int a = 11;
int b = 12;
int c = 13;
int max;

if (a<b) {
  if (b<c) {
    max = c;
  } else {
    max = b;
  }
} else {
  if (a<c) {
    max = c;
  } else {
    max = a;
  }
}

println(max);
```

- Source Code für Processing

- if (test) {..}
- else {..}

Example: Euclidean algorithm

- Greatest common divisor (ggt) of two numbers.



proof of concept

x	y	rest
~~28~~	~~20~~	8
~~20~~	~~8~~	4
8	4	0

Why does this work?
(ggt divides x) & (ggt divides y)
-> x = i*ggt, y = j*ggt, (x-y) = (i-j)*ggt
-> ggt divides (x-y)
-> ggt divides (x-q*y)
-> ggt divides rest of x/y
-> ggt(x,y) = ggt(y, rest)

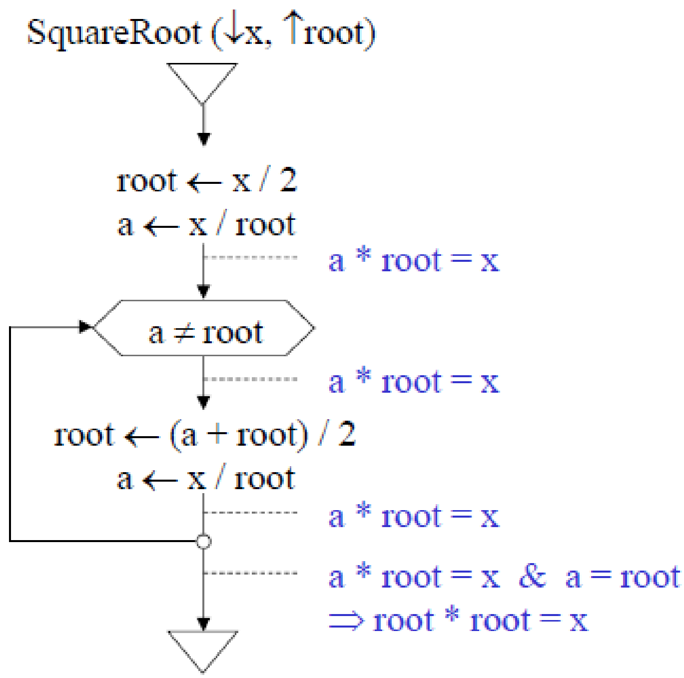
Example: Euclidean algorithm

```
int x = 21;
int y = 14;

int rest = x % y;

while (rest != 0) {
  x = y;
  y = rest;
  rest = x % y;
}

println(y);
```

- Source Code for Processing

- While (test) {..}
- % … modulo

Example: square root



proof of concept

x	root	a
10	~~5~~	~~2~~
	~~3.5~~	~~2.85714~~
	~~3.17857~~	~~3.14607~~
	~~3.16232~~	~~3.16223~~
	3.16228	3.16228

Example: square root

```
float x = 10;

float root = x / 2;
float a = x / root;

while (a != root) {
  root = (a + root) / 2;
  a = x / root;
}

println(root);
```

- Source Code for Processing

- float … data type
- / … Division

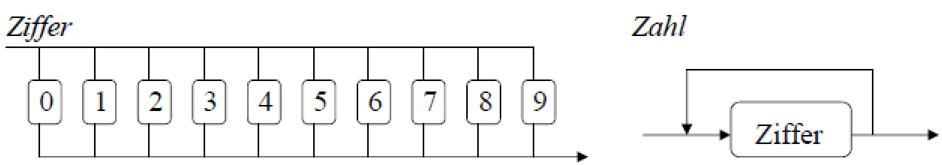
- Hint: Don‘t test float on equality!
  - |a-root| < 0,00001

Specification of programming languages

- Syntax
  - rules to build sentences
  - e.g. assignment = variable <- statement
- Semantics
  - Actual meaning of sentences
  - e.g.: compute statement and assign result to variable.

Specification of programming languages

- Grammar
  - Set of syntax rules
  - eg. grammar for discrete positive numbers.
    - Ziffer = "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9".
    - Zahl = Ziffer {Ziffer}.



EBNF (Extended Backus-Naur-Form)

Examples

- *Grammar for floating point values*
  - number= numeral {numeral}.
  - float = number"." number["E" ["+" | "-"] number].
- *Grammar for If-statements*
  - IfStatement = "if" "(" Statement")" Statement ["else" Statement].

Usage	Notation
definition	=
concatenation	,
termination	;
termination	. [1]
alternation	|
option	[...]
repetition	{ ... }
grouping	(...)
terminal string	" ... "
terminal string	' ... '
comment	(* ... *)
special sequence	? ... ?
exception	-

src. Wikipedia

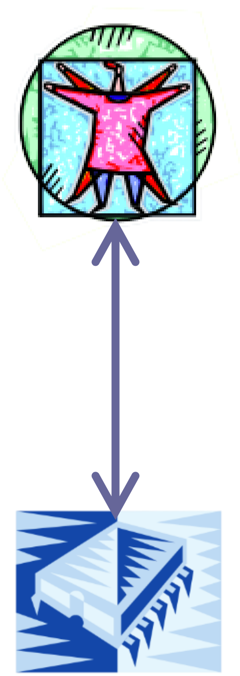
Programming Languages

- Formal languages that can be translated to machine language with a program.
  - A program is a „text“ written in a formal language
- There are a lot of different languages
  - Java, Python, C, C++, Objective C, Pascal, Modula, Perl, Basic, C#, JavaScript, Dart, Erlang, LUA uvm.

Programming Languages

- Compiler: program is translated
  - by a program
  - to machine code
  - Eg. C, C++
- Interpreter:
  - program is executed step by step by another program
  - Eg. Python, Ruby, JavaScript, Perl, LUA

Specification of Algorithms



Graphical or verbal notation
Higher programming languages (like Java)
Assembly languages
Machine code
Hardware, electric signals

Verbale Notation

- Description in natural language

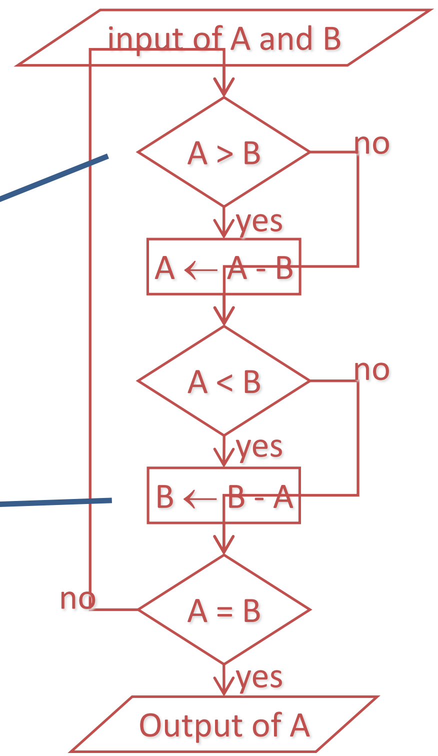
```
Euclidian Algorithm ggT(A, B)
  0. Input of A and B
  1. If A larger than B, then subtract B from A
     and assign the result to A.
  2. If A smaller than B then subtract A from B
     and assign the result to B.
  3. If A is not equal B then go to step 1
  4. The result is A (or B)
```

Flowchart

Input and Output

Branch / Condition

Assignment



Flowchart

Contra

- Often unstructured, no formal framework.
- Not good for working in teams, hard to read for others
- Hard to update and revise.

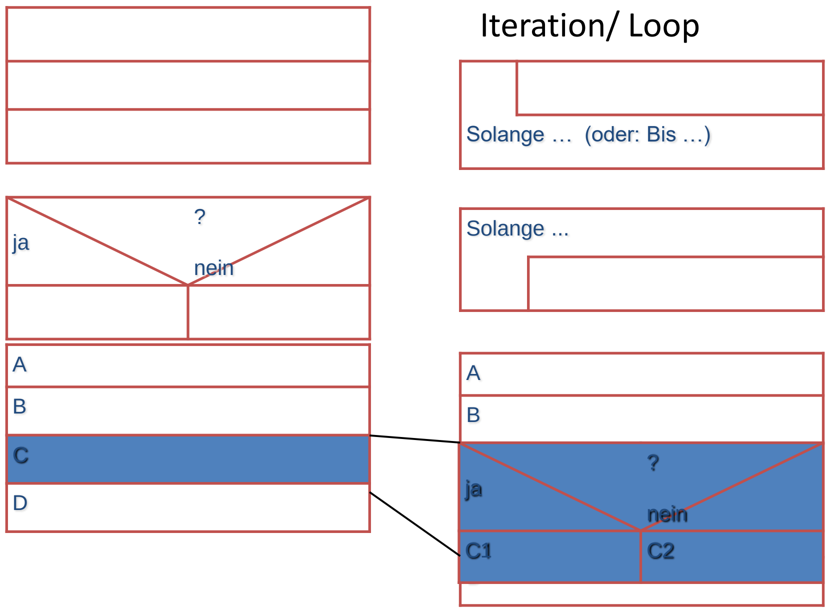
Nassi-Shneiderman-Chart

- More structured due to stronger restrictions.

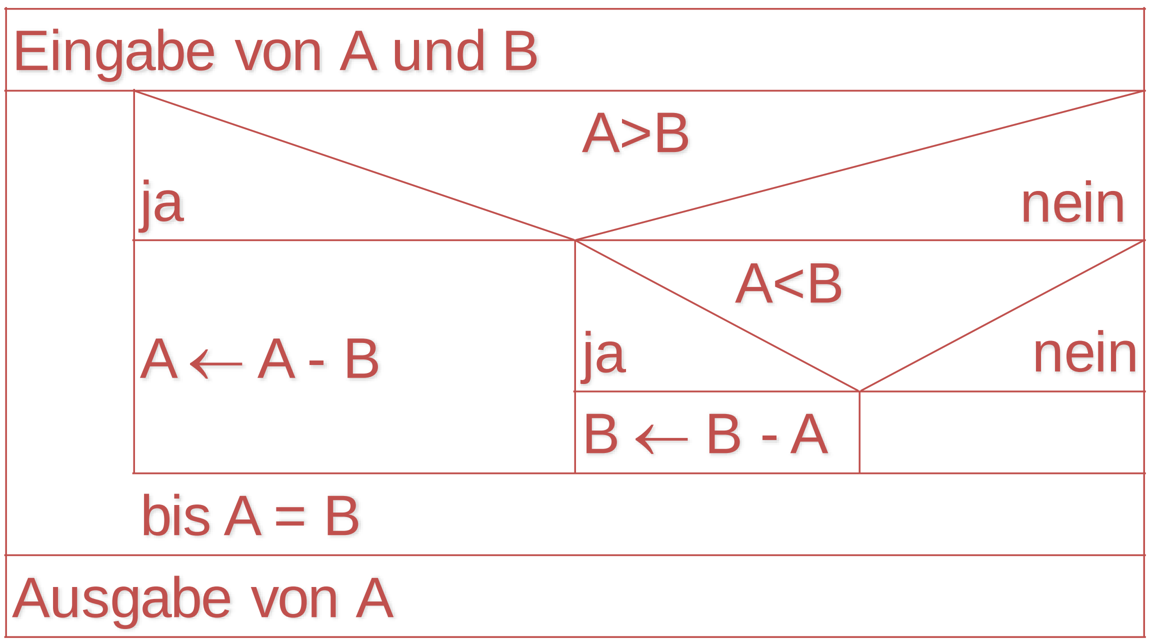
- Sequence

- Branch / Condition

- + nesting!



Nassi-Shneiderman-Chart: Euclidian Algorithm



Pseudocode

- Semi-formal languages
- Examples:

```
WHILE   A not equal B
    IF A > B
    THEN subtract B from A
    ELSE
           subtract A from B
    ENDIF
ENDWHILE
ggT := A
```

ESOP - Simple Programs

Assoc. Prof. Dr. Mathias Lux
ITEC / AAU

Agenda

- Symbols
- Variables, Constants
- Assignments
- Operators

Quelle für die folgenden Folien: ***Grundlagen der Programmierung,*** *Prof. Dr. Hanspeter Mössenböck*

Symbols: names

Naming of variables, types, functions, etc. within a program.

- consist of letters, digits and ‚_‘
- always start with a letter
- arbitrary length
- case sensitive

- Examples
  - x, x17, my_Var, myVar

Symbols: key words

- Name key parts of the program
- cannot be used as names

- Examples:
  - if, while, for, enum, class, static, ...

Symbols: numbers

- Discrete numbers
  - (decimal or hexadecimal)
- Floating point numbers

- Examples
  - 376 ... decimal
  - 0x1A5 ... hexadecimal
  - 3.14 ... floating point

Symbols: strings

- Any strings between quotation marks.
- Must not exceed end of lines
- " needs to be excaped to \"

- Examples
  - "a simple string"
  - "she said \"Hallo\""

Symbols: strings

- String
  - in Java not a base data type but an onject.
- char … single Unicode letter
  - 2 Bytes
  - simple apostrophe, eg. ‘L’, ‘)’, ...

Declaration of variables

- Each variable must be declared before use
  - Name and type are given to the compiler
  - Compiler allocates memory
- Examples:
  - `int x;` ... declares variable x of type int (integer)
  - `short a, b;` ... declares two variables of type short (short integer)

Integer types

byte	8 bit	-2^7 .. 2^7-1	(-128 .. 127)
short	16 bit	-2^15 .. 2^15-1	(-32.768 .. 32.767)
int	32 bit	-2^31 .. 2^31-1	(-2.147.483.648 ..)
long	64 bit	...	...

- Declaration & initialisation
  - `int x = 100;`
    declares integer x and assign value of 100.
  - `short a = 0, b = 1;`
    declares two short variables with initial values.

Constants

- Init variables that cannot be changed later
  - static final int max = 100;
- Why would you do that?
  - readability
    - max easier to read than 100
  - maintainability
    - if the same value is used several times.
- Constants are declared in class scope
  - will be explained later in the course

Comments

- line comments
  - Start with // .. and with end-of-line (EOL)
- block comments
  - use /* ... */, can span over multiple lines.

- Comments & Readability
  - comment for later understanding
  - do not comment what‘s obvious

```
// Hier ist ein Zeilenkommentar

int x = 15; // Initialisierung an dieser Stelle erforderlich!
short y = -12;

/* ************************************************
   Dieses Programm wurde von Mathias Lux geschrieben
   ************************************************ */
```

Language for comments and names

- Think about your audience
  - English is better than German
- Do not mix languages!

- Special care with
  - swear words, email adresses, people names, licenses!

Explore Gist Blog Help dermotte

Search

shit | Search

- Repositories 203
- Code 26,340
- Issues 2,815
- Users 31

Languages

C	224,353
HTML	34,242
JavaScript	
GAS	25,739
Less	23,338
C++	18,093
PHP	15,306
XML	9,977
Ruby	9,436
Markdown	8,795

Advanced search Cheat sheet

We've found 26,340 code results

Sort: Best match

romankalb/PMapp – main.js JavaScript
Last indexed on 3 Aug

```
debug("Shit");
```

matthewcv/nodestuff – mmcolors.js JavaScript
Last indexed on 31 Jul

```
alert('shit');
```

lwl8851206/HelloWorld – test.js JavaScript
Last indexed on 29 Jul

```
function shit(){}
```

ACSvsFMI/theDoctors – shit.js JavaScript
Last indexed on 25 Jul

```
alert('shit')
```

nitirajrathore/testrepo – tits.js JavaScript
Last indexed on 2 Aug

```
alert("fucking dick shit")
```

gpestana/legacy_slick.js – core_tests.js JavaScript
Last indexed on 1 Aug

```
/*Tests and shit.. */
```

bmelon11/myrepo – boo.js JavaScript
Last indexed on 28 Jul

```
console.log("eat shit")
```

apiengine/apiengine-client – page.js JavaScript
Last indexed on 23 Jul

```
some profile shit goes here
```

AchintyaAshok/NYT---Intern-Project-Front-End – storyView.js JavaScript
Last indexed on 28 Jul

```
console.log('django is shit');
```

JamieAppleseed/jamieappleseed.com – application.js JavaScript
Last indexed on 8 Aug

```
(function(win){
  // do shit
})(this);
```

Choice of variables and constants

- Coding conventions exist for
  - readability of code
  - maintainability and preservation
- Naming conventions see:
  http://www.oracle.com/technetwork/java/javase/documentation/codeconventions-135099.html#367
- Tipps:
  - Names that make sense (cp. comments)
  - Better shorter than longer (cp. support by IDE).

No good naming ..

aeonsf/footlocker – dpc-hashcrack.py
Last indexed on 31 Jul

Python

```python
				return licker

def toptobottom(crack):
	i = 0
	while i < (len( asshole )/2):
		if len(crack) == 32:
			if crack == md5(asshole[i]):
...
			if crack == md5( asshole [i]):
				print "\n\t[p1] 3===D  passwd is = %s\n"% asshole [i]
				break
		elif len(crack) == 40:
```

hallfox/teampython – ex10b.py
Last indexed on 3 Aug

Python

```python
escape4 = "%s is a total  asshole ."

asshole  = "Tyler \t\nFUCK\n Sontag \\"

singlequotepreformatting = '''
This looks a lot cleaner and minimalistic.
...
For now on, let's use the single quotes instead.
'''

print escape1
print escape2
print escape3
print escape4 %  asshole
print singlequotepreformatting
```

Assignments

$$\underbrace{x}_{\text{variable}} = \underbrace{y + 1}_{\text{statement}} ;$$

1. compute statement
2. store in variable

- left and right side have to be compatible
  - either the same type (int, byte, ...)
  - or type left $\supseteq$ type right
- hierarchy of integer types
  - long $\supseteq$ int $\supseteq$ short $\supseteq$ byte

Assignments

- Examples

```
int i, j; short s; byte b;
i = j;            // ok: same type
i = 300;          // ok (numeric expressions are int)
b = 300;          // not ok: 300 > byte
i = s;            // ok
s = i;            // not ok
```

Static Type Check

- Compiler checks:
  - variables stay in allowed value range.
  - operators are applied on the right types / values.

Arithmetic Expressions

- Simplified grammar

```
Expr = Operand {BinaryOperator Operand}.
Operand = [UnaryOperator] ( identifier | number | "(" Expr ")" ).
```

- eg. `- x + 3 * (y + 1)`

Arithmetic Expressions

- Binary Operators

+	sum	
-	subtraction	
*	multiplikation	
/	division	`5/3 = 1`
%	modulo	`5%3 = 2`

- Unary operators

+	identity	`(+x) = x`
-	invert sign	

Types in Arithmetic Expressions

- Order of operations
  - multiplication and division (*, /, %) over addition and subtraction (+, -)
    - eg. 2 + 3 * 4 = 14
  - left association
    - eg. 7 - 3 - 2 = 2
  - unary operators over binaray operators
    - eg.: -2 * 4 + 3 ergibt -5
- Resulting types
  - input type can be byte, short, int, long
  - resulting type
    - if one operand is long -> result is type long,
    - otherwise -> type int

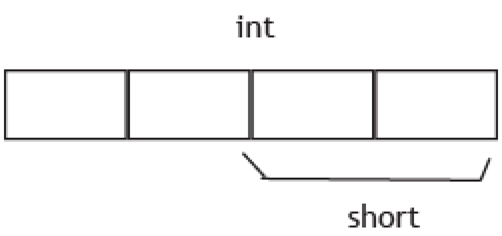
Examples

```
short s; int i; long x;
x = x + i;             // long
i = s + 1;             // int (1 is int)
s = s + 1;             // false!
s = (short)(s + 1);    // type cast necessary
```

Type Cast

```
(type) expression
```

- changes *expression* to *type*
- result can be truncated



Increment / Decrement

- access variable plus operation
  - x++ ... returns x and then adds +1
  - ++x ... adds 1 to x and then returns x
  - x-- , --x ... the same with subtraction.
- can be a statement on ist own right
  - x = 1; x++; // x = 2 the same as: x = x + 1;

- examples
  - x = 1; y = x++ * 3; // x = 2, y = 3 is: y = x * 3; x = x + 1;
  - x = 1; y = ++x * 3; // x = 2, y = 6 is: x = x + 1; y = x * 3;
- only works on variables, not expressions.
  - y = (x + 1)++; // wrong!

The power of two (shifts)

Shift-operators allow for efficient multiplication and division by powers of two.

Multiplikation	
x * 2	x << 1
x * 4	x << 2
x * 8	x << 3
x * 16	x << 4
...	...

Division	
x / 2	x >> 1
x / 4	x >> 2
x / 8	x >> 3
x / 16	x >> 4
...	...

Division only works out for positive numbers.

The power of two (shifts)

Examples

x = 3;	x = -3;	x = 15;
0000 0011	1111 1101	0000 1111
x = x << 2; // 12	x = x << 1; // -6	x = x >> 2; // 3
0000 1100	1111 1010	0000 0011

Assignment operators.

- arithmetic operations can be combined with assignments.

	short	long
+=	x += y;	x = x + y;
-=	x -= y;	x = x - y;
*=	x *= y;	x = x * y;
/=	x /= y;	x = x / y;
%=	x %= y;	x = x % y;

check http://docs.oracle.com/javase/specs/jls/se8/html/jls-15.html#jls-15.26

String Operators

- Strings can be concatenated with ‘+’
  - “Mathias” + “ “ + “Lux”
- Other operators do not apply
  - Especially not comparisons
  - “Mathias” != “Lux” ... checks addresses!

Bit Operators

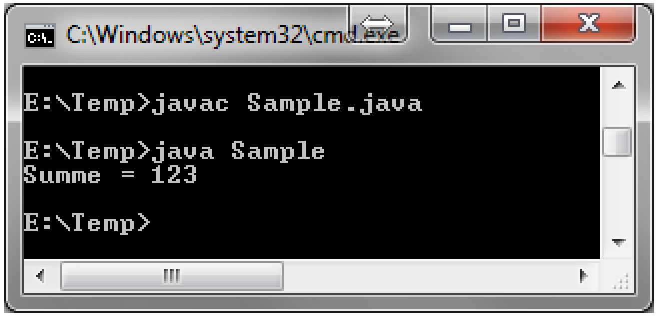
- Bits of operands are modified
  - Example(Java uses two‘s complement)
    - `byte a = 17;                   // 00010001`
    - `byte b =  7;                   // 00000111`
- Supported operations
  - Disjunction:
    - `byte or = a | b;  // 23`
  - Conjunction:
    - `byte and = a & b; //  1`
  - Antivalence:
    - `byte xor = a ^ b; // 22`
  - Complement:
    - `byte notB = ~b;   // -8`

Java-Programs

```
class ProgramName {
    public static void main (String[] arg) {
        ... // Declarations
        ... // Statements
    }
}
```

Text has to be in file named *ProgramName.java*

```
// Example:
class Sample {
    public static void main (String[] arg) {
        int a = 23;
        int b = 100;
        System.out.print("Sum = ");
        System.out.println(a + b);
    }
}
```



Compile and Run with JDK

- Compile
  - `C:\> cd MySamples`
    change to source file
  - `C:\MySamples> javac Sample.java`
    create class file (compile)
- Execute
  - `C:\MySamples> java Sample`
    run class file
  - `Sum = 123`

Example: IDEA IDE

- Strings, comments and variables
  – Spell check, consistency, type check
- Live Templates
  – psvm + <tab>
- Automated naming of Variables
  – <Strg>-<Space>

ESOP - Conditions & Loops

Assoc. Prof. Dr. Mathias Lux

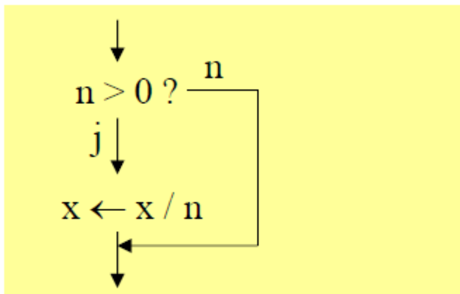
ITEC / AAU

Agenda

- Conditions
  - If – Else, Switch
- Loops
  - While, Do-While, For

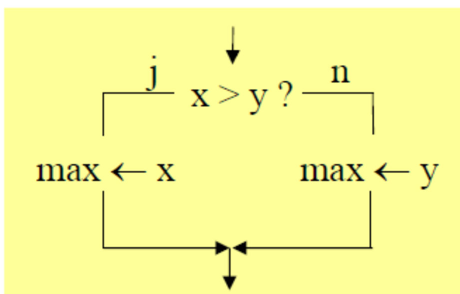
Quelle für die folgenden Folien: ***Grundlagen der Programmierung,*** *Prof. Dr. Hanspeter Mössenböck*

If-Else




```
if (n > 0) x = x / n;
```

without else




```
if (x > y)
   max = x;
else
   max = y;
```

with else

Syntax

IfStatement = "**if**" "(" Expression ")" Statement ["**else**" Statement].

Blocks

If there is more than one statement in the if or the else part of a condition, we need to define blocks with {…}.

```
Statement = Assignment | IfStatement | Block | ... .
Block = "{" {Statement} "}".
```

Blocks

- Example

Indentation

```
if (x < 0) {
  negNumbers++;
  System.out.print(-x);
} else {
  posNumbers++;
  System.out.print(x);
}
```

Best Practice:
{…} for single statements too

Indentations

- For readability
  - visualize strucutre
- how much?
  - 1 tab oder 2 spaces

- Short If-statements in a single line:
  - `if (n != 0) x = x / n;`
  - `if (x > y) max = x; else max = y;`

Dangling Else

```
if (a > b)
   if (a != 0) max = a;
else
   max = b;
```

```
if (a > b)
   if (a != 0) max = a; else max = b;
```

- Two ifs, one else. Where does the else belong to?
- In Java else goes with the if immediately before it.

- Alternative: use blocks.

Short If

- `(Expression)?Statement:Statement`

```
int x = 3;
int y = 4;
int max = (x<y)?y:x;

println(max);
```

Comparison

- Compare two values
- Returns *true* or *false*

		Example
`==`	equal	`x==3`
`!=`	not equal	`x!=y`
`>`	larger than	`4>3`
`<`	smaller than	`x+1<0`
`>=`	larger or equal	`x>=y`
`<=`	smaller or equal	`x<=y`

Combining Comparisons

&& logic AND

x	y	x && y
true	true	true
true	false	false
false	true	false
false	false	false

|| logic OR

x	y	x || y
true	true	true
true	false	true
false	true	true
false	false	false

! logic NOT

x	!x
true	false
false	true

- Example
  - `if (a >= 0 && a <= 10 || a >= 100 && a <= 110) b = a;`

Boolean Operators

- ! Is stronger && and ||
- && is stronger than||

- brackets for association of clauses
  - `if (a > 0 && (b==1 || b==7)) ...`

Data Type `boolean`

- data type (just like `int`)
  - values are *true* and *false*
- Examples

```
boolean p, q;
p = false;
q = x > 0;
p = p || q && x < 10;
```

DeMorgan Rules

- ! (a && b) ⇔ ! a || ! b
- ! (a || b) ⇔ ! a && ! b

```
if (x >= 0 && x < 10) {
    ...
} else { // ! (x >= 0 && x < 10)
    ...
}
```

⇨ ! (x >= 0) || ! (x < 10)

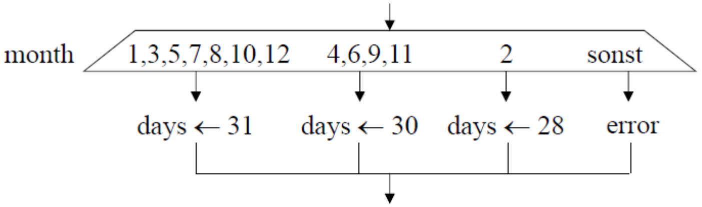
⇨ x < 0 || x >= 10

Examples boolean & `if`

- Expression is evaluated to `true` or `false`
  - `if (true) …`
  - `if (!true) …`
  - `if ((x >=1) == true) …`

Switch Statement

- Multiple branches



- In Java

```java
switch (month) {
   case 1: case 3: case 5: case 7: case 8: case 10: case 12:
    days = 31; break;
   case 4: case 6: case 9: case 11:
    days= 30; break;
   case 2:
    days = 28; break;
   default:
    System.out.println("error");
}
```

Switch Statement

- Conditions
  - expression has to be integer, char or String
  - case labels have to be constants
  - case label data has to fit expression
  - case labels need to pair wise different
- Break statement
  - Jumps to the end of the switch block
  - If break is missing, everything after it is executed
    - typical error

Switch Expression

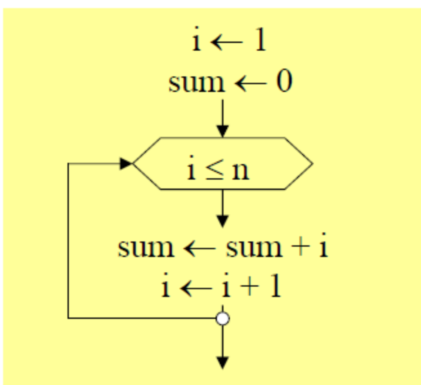
```
switch (month) {
  case 1: case 3: case 5: case 7: case 8: case 10: case 12:
    days = 31; break;
  case 4: case 6: case 9: case 11:
    days= 30; break;
  case 2:
    days = 28; break;
  default:
    System.out.println("error");
}
```

Switch-Syntax

```
Statement = Assignment | IfStatement | SwitchStatement | ... | Block.
SwitchStatement = "switch" "(" Expression ")" "{" {LabelSeq StatementSeq} "}".
LabelSeq = Label {Label}.
StatementSeq = Statement {Statement}.
Label = "case" ConstantExpression ":" | "default" ":".
```

While Loop

- Loops a sequence of statements
- As long as a condition evaluates to `true`.




```
i = 1;
sum = 0;
while ( i <= n ) {
    sum = sum + i;
    i = i + 1;
}
```

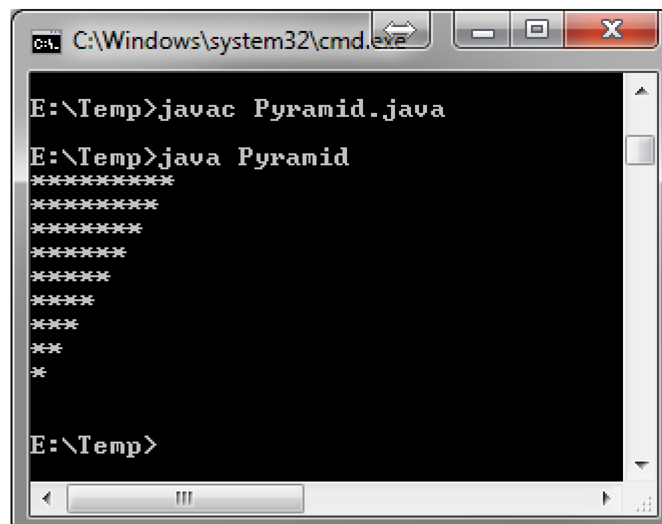
continuation condition

loop body

```
Statement = Assignment | IfStatement | SwitchStatement | WhileStatement | ... | Block.
WhileStatement = "while" "(" Expression ")" Statement .
```

While Loop

```java
class Pyramid {
    public static void main (String[] arg) {
        int i = 10;
        while (i-->0) {
            int j = 0;
            while (j++<i) {
                System.out.print("*");
            }
            System.out.println();
        }
    }
}
```

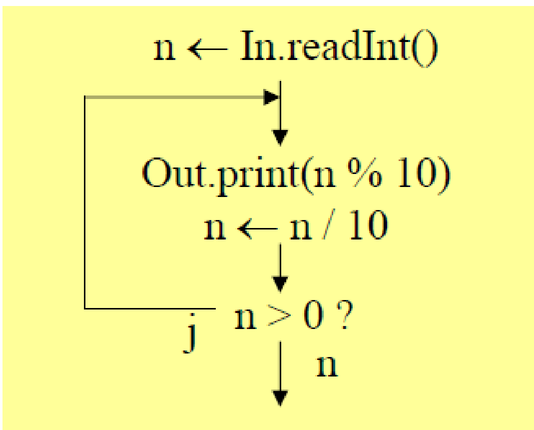


Termination

- Loops should terminate
  - no endless loop `while (true) { ... }`
- Common problems for endless loops
  - variable in continuation condition is not changed
  - continuation condition never evaluates to `false`
    - eg. `while (x!=0) { x -= 5; }`
- Approach: model & test for typical problems

Do-While Loop

- Continuation condition is tested at the end of the loop
- Loop body is run at least once




```
int n = In.readInt();
do {
    Out.print(n % 10);
    n = n / 10;
} while ( n > 0 );
```

proof of concept

n	n % 10
~~123~~	3
~~12~~	2
~~1~~	1
0	

```
Statement = Assignment | IfStatement | WhileStatement |
   DoWhileStatement | ... | Block.
DoWhileStatement = "do" Statement "while" "(" Expression ")" ";".
```

For Loop (Counting Loop)

- Used if number of iterations is known beforehand

```
sum = 0;
for ( i = 1 ; i <= n ; i++ )
   sum = sum + i;
```

1) Initialisation
2) Continuation condition
3) Update

.. is actually short for

```
sum = 0;
i = 1;
while ( i <= n ) {
   sum = sum + i;
   i++;
}
```

For Loop Examples

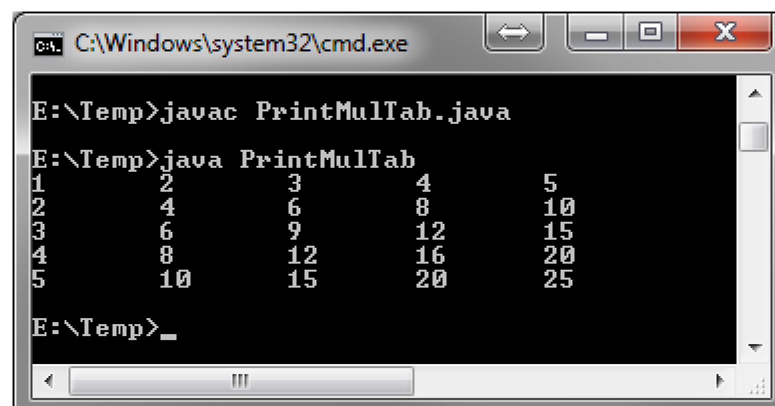
for (i = 0; i < n; i++)	i: 0, 1, 2, 3, ..., n-1
for (i = 10; i > 0; i--)	i: 10, 9, 8, 7, ..., 1
for (int i = 0; i <= n; i = i + 1)	i: 0, 1, 2, 3, ..., n
for (int i = 0, j = 0; i < n && j < m; i = i + 1, j = j + 2)	i: 0, 1, 2, 3, ... j: 0, 2, 4, 6, ...
for (;;) ...	Endless loop

For Loop Definition

```
ForStatement = "for" "(" [ForInit] ";" [Expression] ";"
   [ForUpdate] ")" Statement.
ForInit = Assignment {"," Assignment} | Type VarDecl {","
   VarDecl}.
ForUpdate = Assignment {"," Assignment}.
```

For Loop Example

```
class PrintMulTab {
  public static void main (String[] arg) {
      int n = 5;
      for (int i = 1; i <= n; i++) {
            for (int j = 1; j <= n; j++) {
                  System.out.print(i * j + "\t");
            }
            System.out.println();
      }
  }
}
```



Termination of Loops

- Terminate with keyword *break*

```
while (In.done()) {
    sum = sum + x;
    if (sum > 1000) {
      Out.println("zu gross");
      break;
    }
    x = In.nextNumber();
}
```

- However, it's better to use the continuation condition

```
while (In.done() && sum < 1000) {
    sum = sum + x;
    x = In.nextNumber();
}
if (sum > 1000)
    Out.println("zu gross");
```

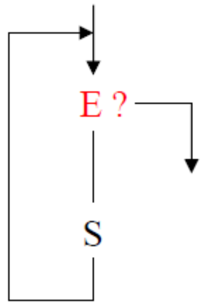
Termination of Outer Loops

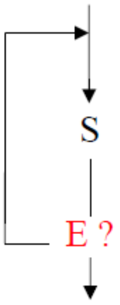
```
outer: // Label!
for (;;) { // endless loop!
   for (;;) {
         ...
         if (...) break;       // terminates inner loop
         else break outer;    // terminates outer loop
         ...
   }
}
```

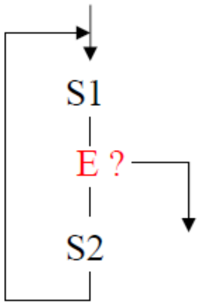
Loop Termination

- When to use `break`
  - on errors (performance!)
  - multiple exit points within a loops
  - real endless loops (eg. in real time systems)

Types of Loops







while (E)
  S

for (I; E; U)
  S

do
  S
while (E)

for (;;) {
  S1;
  if (E) **break**;
  S2;
}

Which Type of Loop When?

- Selection based on “Convenience”
  - counting, condition at begin or end ..
- Selection based on performance
  - (s.u. für Javascript, http://jsperf.com/fun-with-for-loops/8)

Test runner

Done. Ready to run again.

Run again

Testing in Chrome 37.0.2062.124 32-bit on Windows Server 2008 R2 / 7 64-bit

	Test	Ops/sec
FOR standard	`for (var i; i < a.length; i++) { n++; }`	329,591,795 ±0.23% fastest
FOR optimized	`for (var i, imax = a.length; i < imax; i++) { n++; }`	329,708,498 ±0.43% 0.16% slower
While Counting Down	`var i = a.length + 1; while(--i) { n++; }`	29,620,863 ±19.14% 92% slower

ESOP – Gleitkommazahlen, Methoden und Arrays

Assoc. Prof. Dr. Mathias Lux

ITEC / AAU

Repeat ..

```java
/**
 * Check for primes, simple version ...
 */
public class Primes {
    public static void main(String[] args) {
        int maxPrime = 1000;
        // iterate candidates
        for (int candidate = 3; candidate <= maxPrime; candidate++) {
            boolean isPrime = true;
            // iterate potential dividers
            for (int divider = 2; divider < candidate; divider++) {
                // check for division without rest
                if (candidate % divider == 0) {
                    isPrime = false;
                }
            }
            if (isPrime)
                System.out.println("prime = " + candidate);
        }
    }
}
```

- Find prime numbers < maxPrime

Floating Point Numbers

- Two data types
  - float … 32 Bit precision (24/8 in Java 8)
  - double … 64 bit precision (53/11 in Java 8)
- Syntax

```
FloatConstant = [Digits] "." [Digits] [Exponent]
   [FloatSuffix].
Digits = Digit {Digit}.
Exponent = ("e" | "E") ["+" | "-"] Digits.
FloatSuffix = "f" | "F" | "d" | "D".
```

Floating Point Numbers

- **Variables**
  - float x, y;
  - double z;
- **Constants**
  - 3.14 // type double
  - 3.14f // type float
  - 3.14E0 // $3.14 * 10^0$
  - 0.314E1 // $0.314 * 10^1$
  - 31.4E-1 // $31.4 * 10^{-1}$
  - .23
  - 1.E2 // 100

Harmonic Series

```
public class HarmonicSequence {
    public static void main (String[] arg) {
        float sum = 0;
        int n = 10;
        for (int i = n; i > 0; i--)
            sum += 1.0f / i;
        System.out.println("sum = " + sum);
    }
}
```

- Exchanging 1.0f / i what would happen?
  - 1 / i ... 0 (integer divison)
  - 1.0 / i ... a double value

Float vs. Double

```
public class HarmonicSequence {
    public static void main (String[] arg) {
        float sum = 0;
        int n = 10;
        for (int i = n; i > 0; i--)
            sum += 1.0f / i;
        System.out.println("sum = " + sum);
    }
}
```

D:\Java\JDK\jdk1.6.0_45\bin\java …
sum = 2.9289684

Process finished with exit code 0

```
public class HarmonicSequence {
    public static void main (String[] arg) {
        double sum = 0;
        int n = 10;
        for (int i = n; i > 0; i--)
            sum += 1.0d / i;
        System.out.println("sum = " + sum);
    }
}
```

D:\Java\JDK\jdk1.6.0_45\bin\java …
sum = 2.9289682539682538

Process finished with exit code 0

Assignments and Operations

- Type compatibility
  - double $\supseteq$ float $\supseteq$ long $\supseteq$ int $\supseteq$ short $\supseteq$ byte
- Operators possible
  - Arithmetic operators (+,-,*,/)
  - Comparison (==, !=, <, <=, >, >=)
    Note! Do not check floating point values for equality!

Assignments and Casts

```
float f; int i;
f = i;          // works
i = f;          // does not work
i = (int) f;    // works, but cuts after comma;
                // too large or too small lead to
                // Integer.MAX_VALUE, Integer.MIN_VALUE
f = 1.0;        // does not work, 1.0 is type double
```

Review: Data Types

- Ganzzahlige Typen: `byte`, `char`, `short`, `int`, `long`
- Gleitkommazahlen: `float`, `double`
- Zeichenketten: `String`
- Boolesche Variablen: `boolean`

See also https://docs.oracle.com/javase/tutorial/java/nutsandbolts/datatypes.html

Review: Data Types

- Integer expressions are of type `int`
  - are fitted into smalles container,
  - ie. byte, short, …
- Floating point and scientific number expressions are type `double`
- Explicit type with suffix
  - „L“ or „l“ -> `long`
  - „d“ -> `double`
  - „f“ -> `float`

Review: Data Types

Data Type	Default Value (for fields)
byte	0
short	0
int	0
long	0L
float	0.0f
double	0.0d
char	'\u0000'
String (or any object)	null
boolean	false

Review: Data Types

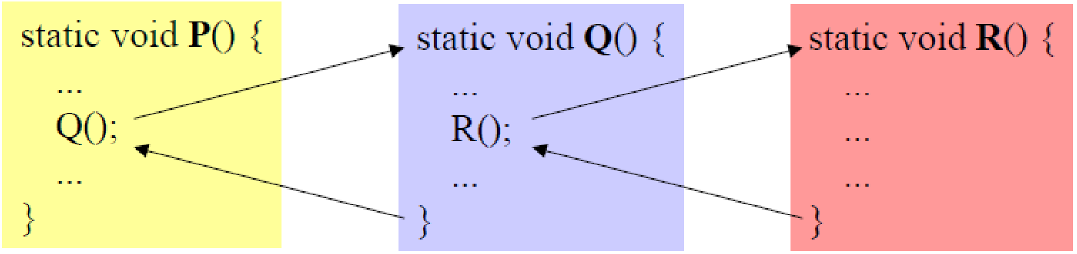
- IDEA – DIE supports you by pointing out problems and types
- Suffix for explicit type
  - `120L // that‘s a long`

Methods

- Cp. functional programming languages
  - subroutines, functions, …
- Goal is to re-use code
  - Code that would otherwise show up more than once.
- All in all less to write
  - less lines of code, less work
  - easier to find errors and maintain.

Methods in Java

- We first introduce methods as subroutines.
  - .. that‘s a non-default interpretation.
- Name conventions for methods
  - start with verb and lower case letter
  - examples:
    - printHeader, findMaximum, traverseList, …



Methods in Java

```
public class SubroutineExample {
    private static void printRule() {        // method head
        System.out.println("-------------");  // method body
    }

    public static void main(String[] args) {
        printRule();                          // method call
        System.out.println("Header 1");
        printRule();
    }

}
```

D:\Java\JDK\jdk1.6.0_45\bin\java ...

Header 1

Process finished with exit code 0

Parameters

- Input of values supported by methods

```
class Sample {

  static void printMax (int x, int y) {
    if (x > y) Out.print(x); else Out.print(y);
  }

  public static void main (String[] arg) {
    ...
    printMax(100, 2 * i);
  }
}
```

formal parameters

- in the method head
- are the variables in the method body

actual parameters

- in the method call
- can be expressions

Parameters

- Actual parameters are stored in the variables defined by the formal parameters.
- `x = 100; y = 2 * i;`
  - actual parameters need to be type compatible with the formal parameters.

```
class Sample {

  static void printMax (int x, int y) {
     if (x > y) Out.print(x); else Out.print(y);
  }

  public static void main (String[] arg) {
     ...
     printMax(100, 2 * i);
  }
}
```

Functions

- Functions are methods that return a value.

```
class Sample {
  static int max (int x, int y) {
    if (x > y) return x; else return y;
  }

  public static void main (String[] arg) {
    ...
    int result = 3 * max(100, i + j) + 1;
    ...
  }
}
```

- They have a <u>return type</u>, eg. `int` instead of `void`
- They use the `return` keyword to exit
- Can be used in expressions

Functions vs. Procedures

- Functions
  - methods with return values
  - `static int max (int x, int y) {...}`
- Procedures
  - methods without return values
  - `static void printMax (int x, int y) {...}`

Example

```
public class BinomialCoefficient {
    public static void main(String[] args) {
        int n = 5, k = 3;
        int result = factorial(n) /
                (factorial(k) * factorial(n - k));
        System.out.println("result = " + result);
    }

    public static int factorial(int k) {
        int result = 1;
        for (int i = 2; i <= k; i++) {
            result *= i;
        }
        return result;
    }
}
```

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

Return & Rekursion

```
public class BinomialCoefficient {
    static int n = 5, k = 3;

    public static void main(String[] args) {
        int result = factorial(n) /
                (factorial(k) * factorial(n - k));
        System.out.println("result = " + result);
    }

    public static int factorial(int k) {
        if (k>1) {
            return factorial(k-1)*k;
        }
        else {
            return 1;
        }
    }
}
```

- Return ends method
- Can be called at any place
- Method calling itself -> direct recursion

Prime Numbers

```java
/**
 * Primes based on function.
 */
public class PrimesWithMethod {
    public static void main(String[] args) {
        int maxPrime = 1000;
        // iterate candidates
        for (int candidate = 3; candidate <= maxPrime;
candidate++) {
            if (isPrime(candidate))
                System.out.println("prime = " + candidate);
        }
    }

    public static boolean isPrime(int candidate) {
        boolean isPrime = true;
        // iterate potential dividers
        for (int divider = 2; divider < candidate; divider++) {
            // check for division without rest
            if (candidate % divider == 0) {
                isPrime = false;
            }
        }
        return isPrime;
    }
}
```

```java
/**
 * Check for primes, simple version ...
 */
public class Primes {
    public static void main(String[] args) {
        int maxPrime = 1000;
        // iterate candidates
        for (int candidate = 3; candidate <= maxPrime;
candidate++) {
            boolean isPrime = true;
            // iterate potential dividers
            for (int divider = 2; divider < candidate; divider++) {
                // check for division without rest
                if (candidate % divider == 0) {
                    isPrime = false;
                }
            }
            if (isPrime)
                System.out.println("prime = " + candidate);
        }
    }
}
```

Scope of Variables

- Based on groups of statements -> blocks
  - { … },
  - for (int i; …) {…}
- A variable defined in a block is not known outside

Example

```
public class BinomialCoefficient {
    public static void main(String[] args) {
        int n = 5, k = 3;
        int result = factorial(n) /
                (factorial(k) * factorial(n - k));
        System.out.println("result = " + result);
    }

    public static int factorial(int k) {
        int result = 1;
        for (int i = 2; i <= k; i++) {
            result *= i;
        }
        return result;
    }
}
```

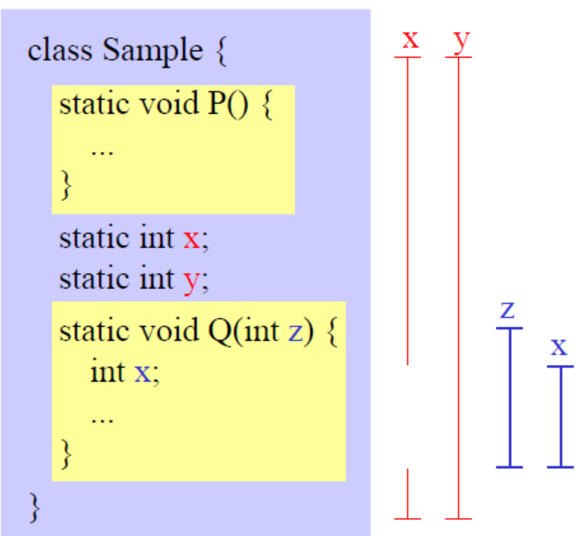
Different variables with different scope

Example: Scope

```java
public class BinomialCoefficient {
    static int n = 5, k = 3;

     public static void main(String[] args) {
         int result = factorial(n) /
                 (factorial(k) * factorial(n - k));
         System.out.println("result = " + result);
     }

     public static int factorial(int k) {
         int result = 1;
         for (int i = 2; i <= k; i++) {
             result *= i;
         }
         return result;
     }
}
```

Smallest scope is the the actual one.

Visibility of Names: Local Variables



Regeln

1. A name can only be declared once within a scope.
2. locale names are prioritized over class scope names.
3. Visibility of a local name starts with ist declaration and ends with the method.
4. Variables in class scope are visible in all methods.

Local & Static

Static Variables

- Are initialized at program start
- Are released upon program termination

Local Variables

- Are initialized at each method call
- Are released upon termination of method.

```
class C {
    static int a, b;
    static void P() {
        int x, y;
        ...
    }
    ...
}
```

Static variables: declared with static at class level; also visible in methods.

Local variables: declared in a method; local, only visible there.

Locality

Best Practice: declare variables as local as possible. Don‘t use static unless there is no other way.

Benefits:

- Clarity: bring together declaration and usage
- Security: Local variables can not be overwritten by other methods
- Efficiency: access to local variable is often faster

Method Overloading

- Methods can be declared multiple times with different sets of formal parameters (difference in type, not names)

```
static void write (int i) {...}
static void write (float f) {...}
static void write (int i, int width) {...}
```

- At call time method implementation fitting to actual parameters is chosen.

```
write(100);        ⇒  write (int i)
write(3.14f);      ⇒  write (float f)
write(100, 5);     ⇒  write (int i, int width)
short s = 17;
write(s);          ⇒  write (int i);
```

Varargs

- In Java methods with an arbitrary number of argumewnts can be declared.

```
public class VarargExample {
    public static void main(String[] args) {
        printList("one", "two", "three");
    }

    public static void printList(String... list) {
        System.out.println("list[0] = " + list[0]);
        System.out.println("list[1] = " + list[1]);
        System.out.println("list[2] = " + list[2]);
    }
}
```

Arrays

- Combination of data of the same type
- Arrays have afixed length
  - which is given at the time of instatiation
- Array variables are references
  - In Java! cp. int, float, etc. -> base types
- Access uses index values
  - first element at index 0

One-Dimensional Arrays

- Name a for the whole array
- elements are accessed by their index
- indexing starts with 0
- elements are „nameless“ variables

	a[0]	a[1]	a[2]	a[3]	...
a					

Declaration

- declares array with name and type
- length is not (yet) known

```
int[] a;
float[] b;
```

Instantiation

- creates a new int array with 5 elements
- assigns adress a

```
a = new int[5];
b = new float[10];
```

Accessing Arrays

- array elemts are just like variables
- index can be expression
- run time error if array is not instantiated
- run time error if index < 0 oder >= length

```
a[3] = 0;
a[2*i+1] = a[i] * 3;
```

- *length* is pre-defined operator
- returns number of elements

```
int len = a.length;
```

Example

```
public class ArrayExample {
    public static void main(String[] args) {
        int[] myArray = new int[5];
        // initialisiere Werte in Array: {1, 2, 3, 4, 5}
        for (int i = 0; i < myArray.length; i++) {
            myArray[i] = i+1;
        }
        // Berechne Durchschnitt:
        float sum = 0;
        for (int i = 0; i < myArray.length; i++) {
            sum += myArray[i];
        }
        System.out.println(sum/myArray.length);
    }
}
```

- Computes mean
- implicit cast to float!

Example: While, For Each

```java
public class ArrayExample {
    public static void main(String[] args) {
        int[] myArray = new int[5];
        // initialisiere Werte in Array: {1, 2, 3, 4, 5}
        int i = 0;
        while (i < myArray.length) { // while
            myArray[i] = i+1;
            i++;
        }
        // Berechne Durchschnitt:
        float sum = 0;
        for (int myInt : myArray) { // for each
            sum += myInt;
        }
        System.out.println(sum/myArray.length);
    }
}
```

- Other loop constructs
- „for each“

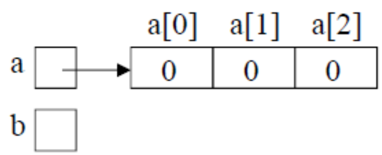
Example: Instantiation

```
public class ArrayExample {
    public static void main(String[] args) {
        // initialisiere Werte in Array: {1, 2, 3, 4, 5}
        int[] myArray = {1, 2, 3, 4, 5};
        // Berechne Durchschnitt:
        float sum = 0;
        for (int myInt : myArray) { // for each
            sum += myInt;
        }
        System.out.println(sum/myArray.length);
    }
}
```

- Different way to create!

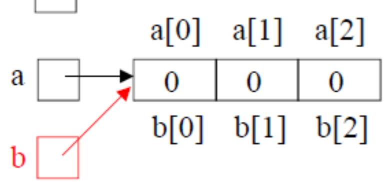
Arrayzuweisung

`int[] a, b;`
`a = new int[3];`



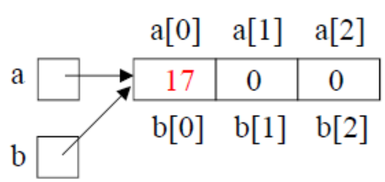
array elements in Java are initialized with 0

`b = a;`



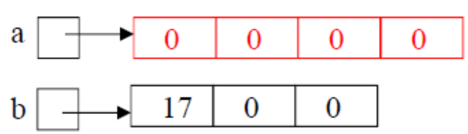
b gets the same value as `a`. It‘s a reference!!!

`a[0] = 17;`



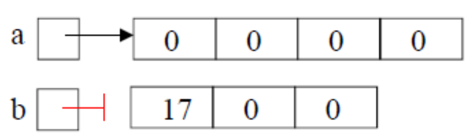
changes `b[0]` too!

`a = new int[4];`



`a` now points to new array.

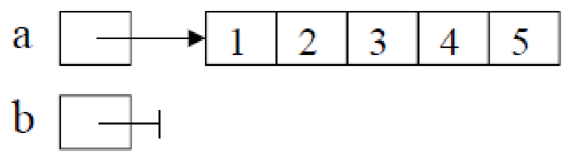
`b = null;`



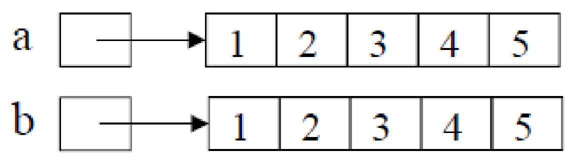
`null` is a special value, which can be assigned to all reference data type variables.

Copying Arrays

```
int[] a = {1, 2, 3, 4, 5};
int[] b;
```




```
b = (int[]) a.clone();
```



- Cast necessary, `a.clone()` returns type `Object[]`

Command Line Parameters

- Calling a program with parameters
  – java <program> par1 par2 par3 …
- Parameters are in a String-Array
  – main(String[] args) method of the program.

Command Line Parameters

```
public class ArrayExample {
    public static void main(String[] args) {
        for (int i = 0; i < args.length; i++) {
            String arg = args[i];
            System.out.println(arg);
        }
    }
}
```

```
$> java ArrayExample one two three
one
two
three
```

Example: Linear Search

```
public class ArrayExample {
    public static void main(String[] args) {
        int[] myArray = {12, 2, 32, 74, 26, 42, 53, 22};
        int query = 22;
        for (int i = 0; i < myArray.length; i++) {
            if (query == myArray[i]) {
                System.out.println("Found at position " + i);
            }
        }
    }
}
```

- Each element is touch -> linear
- Needs *n* steps – What is the size of *n*?

Example: Sorting

- How does one sort an array *a*?
- Naive approach:
  1. Create array *b* of the same size and type.
  2. Move minimum of a to next free position of b
  3. If a is not empty start over with step 2.

Example: Sorting

```
public class ArrayExample {
    public static void main(String[] args) {
        // o.b.d.A. a[k] > 0 & a[k] < 100
        int[] a = {12, 2, 32, 74, 26, 42, 53, 22};
        // create result array
        int[] b = new int[a.length];
        for (int i = 0; i < b.length; i++) { // set each item of b
            int minimum = 100;
            int pos = 0;
            for (int j = 0; j < a.length; j++) { // find minimum
                if (a[j] < minimum) {
                    minimum = a[j];
                    pos = j;
                }
            }
            b[i] = minimum;
            a[pos] = 100; // set visited.
        }

        for (int i = 0; i < b.length; i++) {
            System.out.print(b[i] + ", ");
        }
    }
}
```

- Can be solved in many different ways.
- Cp. AlgoDat lesson!

Example: Eratosthenes‘ Sieve

```
public class Sieve {
    public static void main(String[] args) {
        int maxPrime = 200 000;
        boolean[] sieve = new boolean[maxPrime];
        // init array
        for (int i = 0; i < sieve.length; i++) {
            sieve[i] = true;
        }

        // mark the non-primes
        for (int i = 2; i < Math.sqrt(sieve.length); i++) {
            if (sieve[i] == true) { // if it is a prime
                int k = 2;
                while (k*i < sieve.length) {
                    sieve[k*i] = false;
                    k++;
                }
            }
        }

        // print results
        for (int i = 2; i < sieve.length; i++) {
            if (sieve[i]) System.out.println(i);
        }
    }
}
```

Viel schneller!

ESOP – Classes and Objects

Assoc. Prof. Dr. Mathias Lux

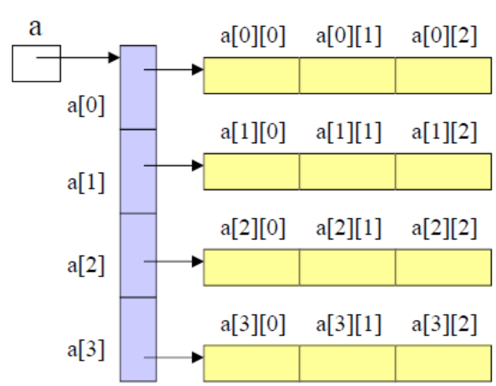
ITEC / AAU

Multidimensional Arrays

- 2-dimensional arrays == matrix

	0	1	2
0	a[0][0]	a[0][1]	a[0][2]
1	a[1][0]	a[1][1]	a[1][2]
2	a[2][0]	a[2][1]	a[2][2]
3	a[3][0]	a[3][1]	a[3][2]

- In Java: arrays of arrays



Declaration and instantiation

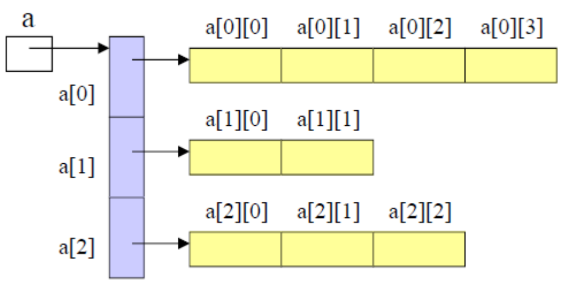
```
int[][] a;
a = new int[4][3];
```

Access

```
a[i][j] = a[i][j+1];
```

Multidimensional Arrays

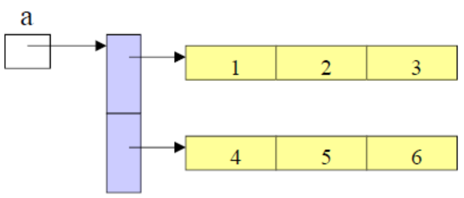
- Rows can be of arbitrary length




```
int[][] a = new int[3][];
a[0] = new int[4];
a[1] = new int[2];
a[2] = new int[3];
```

- Initialisation

```
int[][] a = {{1, 2, 3},{4, 5, 6}};
```



Looking back ..

- Scalar data types
  - „basic data types“ int, byte, short, int, long, float, double, boolean, char
  - Variable contains value
- Aggregated data types
  - More than a single basic data organized through a single name
  - cp. arrays …

Looking back …

- Reference data type
  - variable stores reference / address
  - not a value

- In Java
  - basic data type -> by value
  - everything else -> by reference

About „everything else“ …

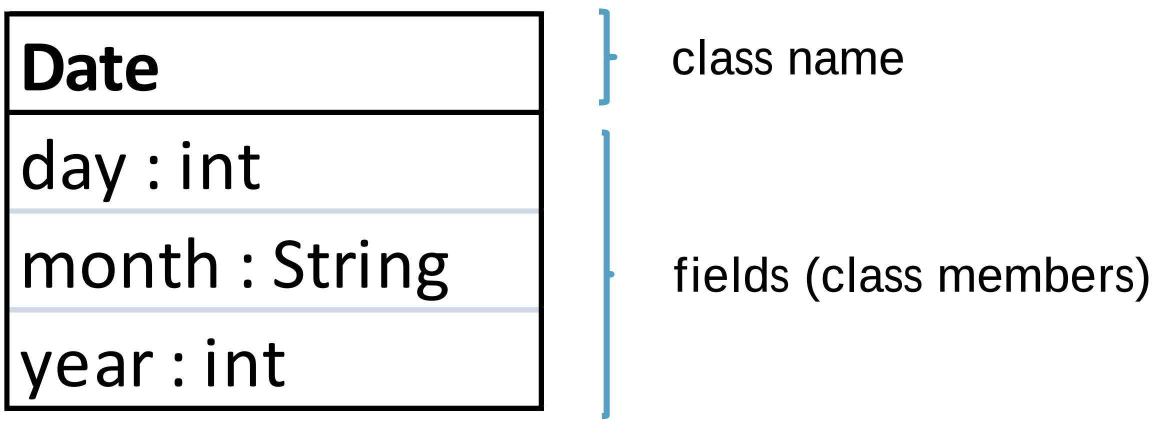
- Basically a combination …
  - of fundamentalen Datentypen
  - in a (sometimes) complex structure
- Different concepts in different languages
  - Pascal: Record
  - C: struct
  - Java / Python: class

Java Classes

- Example: Store a data in a single structure.
  - day, month, year, …
- Basic data types not practical …
  - storing more than one
  - return values of functions
  - comparing to other dates

Java Classes

- Combine all necessary variables in one structure:



Data Type Class

- Declaration

```
class Date {
    int day;
    String month;
    int year;
}
```

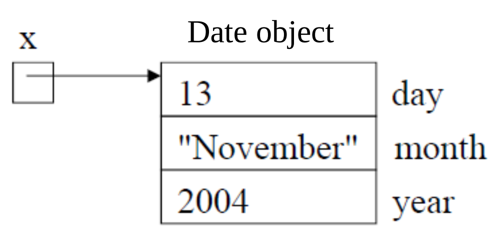
fields of class date

- Data type usage

```
Date x, y;
```

- Access

```
x.day = 13;
x.month = "November";
x.year = 2004;
```



Date variables are references / addresses to objects.

Objects

- Class is like a template
  - from which instances (objects) are created
- Objects (instances) of a class have to be created explicitely before use.
  - variable otherwise have the value `null`

Objects

```
Date x, y;
```

reserves memory for the address

x,y have value null

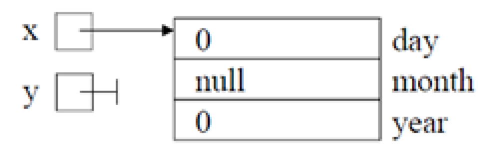
Instantiation

```
x = new Date();
```

creates a new Date object and assigns ist address to x.

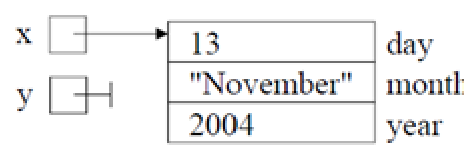
Initial values are

0, null, false or ‚'\u0000'



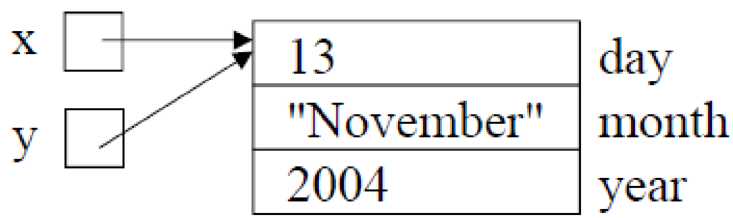
Usage

```
x.day = 13;
x.month = „November“
x.year = 2004;
```



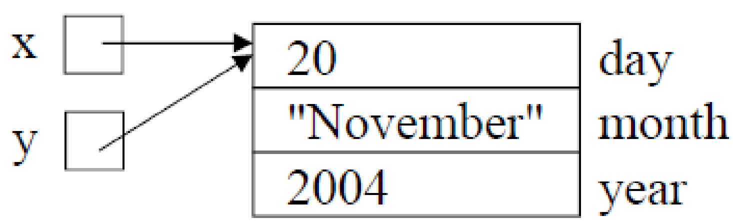
Assignments

y = x;



Reference / address assignment

y.day = 20;



changes x.day too!

Assignments

```
class Date {
    int day;
    String month;
    int year;
}
```

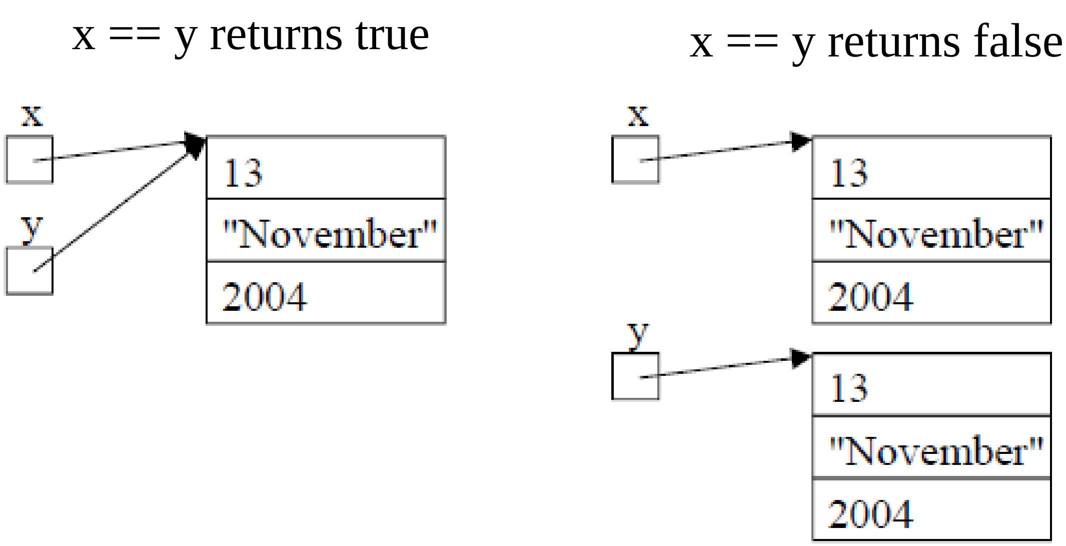
```
class Address {
    int number;
    String street;
    int zipCode;
}
```

```
Date d1, d2 = new Date();
Address a1, a2 = new Address();
```

```
d1 = d2;   // ok, same type
a1 = a2;   // ok, same type
d1 = a2;   // not ok, different type (although structure is the same)
```

Comparing references

- `x == y und x != y` ... compares references
- `<, <=, >, >=` ... not applicable



Compares actual values

- Has to be implemented by method.

```
static boolean equalDate (Date x, Date y) {
   return x.day == y.day &&
    x.month.equals(y.month) &&
    x.year == y.year;
}
```

Declaration of Classes

Single file

```java
class C1 {
    ...
}
class C2 {
    ...
}
class MainProgram {
    public static void
            main (String[] arg) {
            ...
    }
}
```

MainProgram.java

Compile
$> javac MainProgram.java

Multiple files

```java
class C1 {
    ...
}
class C2 {
    ...
}
class MainProgram {
    public static void
            main (String[] arg) {
            ...
    }
}
```

C1.java

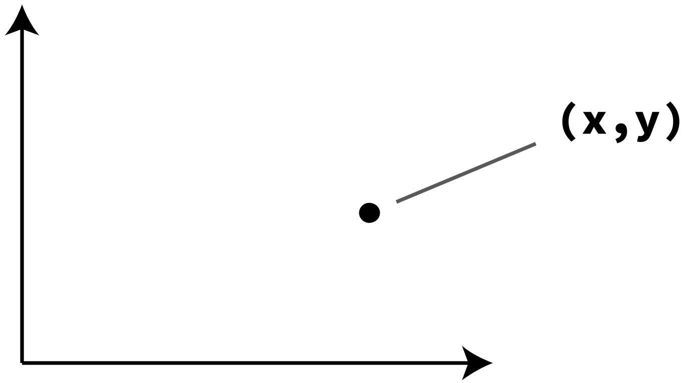
C2.java

MainProgram.java

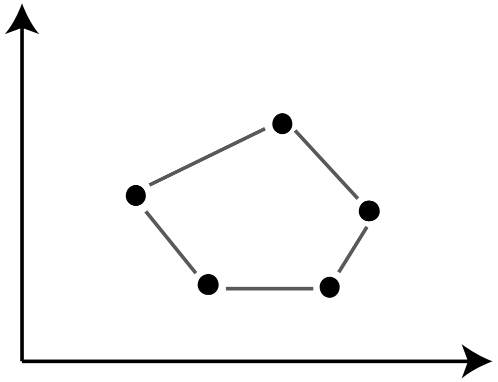
Compile
$> javac MainProgram.java C1.java C2.java

What can we do with classes?

```
class Point {
  double x,y;
}
```




```
class Polygon {
  Point[] points;
}
```



ALPEN-ADRIA UNIVERSITÄT KLAGENFURT | WIEN GRAZ

What can we do with classes?

- Classes can use other classes
  - and extend on that

```
class Point {
   int x, y;
}
class Polygon {
   Point[] pt;
   int color;
}
```

What can we do with classes?

- Implement methods with multiple return values

```
class Time {
    int h, m, s;
}
class Program {
    static Time convert (int sec) {
      Time t = new Time();
      t.h = sec / 3600; t.m = (sec % 3600) / 60; t.s = sec % 60;
      return t;
    }
    public static void main (String[] arg) {
      Time t = convert(10000);
      System.out.println(t.h + ":" + t.m + ":" + t.s);
    }
}
```

What can we do with classes?

- Combination of classes and arrays

```
class Person {
    String name, phoneNumber;
}

class Phonebook {
    Person[] entries;
}

class Program {
    public static void main (String[] arg) {
      Phonebook phonebook = new Phonebook();
      phonebook.entries = new Person[10];
      phonebook.entries[0].name = "Mathias Lux"
      phonebook.entries[0].phoneNumber = "+43 463 2700 3615"
      // ...
    }
}
```

Object Oriented Programming

- What we assumed up to now
  - classes combine data types to structures
  - works with base data types, arrays and other classes.
- Object oriented programming
  - class = data + methods

Example: Position Class

```
class Position {
    private int x;
    private int y;

    void goLeft()  { x = x - 1; }
    void goRight() { x = x + 1; }
}

// … Usage
Position pos1 = new Position();
pos1.goLeft();
Position pos2 = new Position();
pos2.goRight();
```

- Methods are defined locally
  - without `static`
- Each object has ist own state
  - pos1 = new Position()
  - pos2 = new Position()
  - …

Example: Position Class

```
class Position {
    private int x;
    private int y;

    // Methoden mit Parametern
    void goLeft(int n)  {
        x = x - n;
    }

    // [...]
}
```

- Usage of Parameters in methods ..
- .. and return values

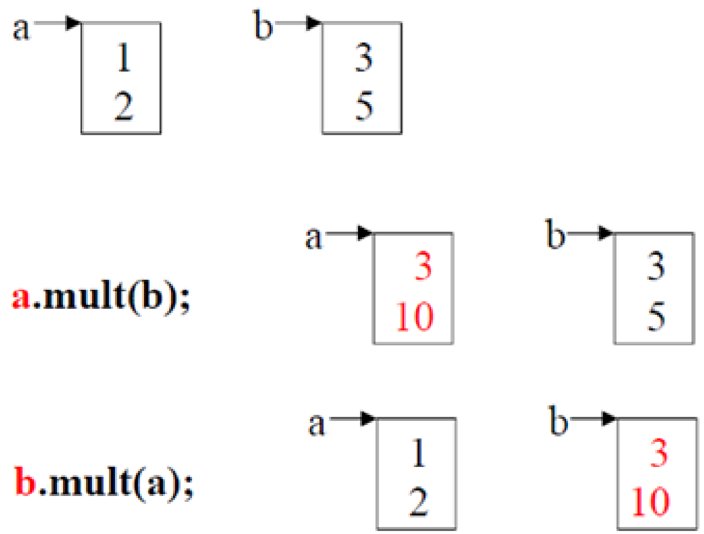
Example: Position Class

```
class Position {
    private int x;
    private int y;

    // Keyword "this"
    void goLeft(int x)  {
        this.x = this.x - x;
    }

    // [...]
}
```

- `this` is used to access fields of the object (object scope)
- Without `this` the local variable would be used.

Example: Fraction Class

```java
public class Fraction {
    int n;  // numerator
    int d;  // denominator

    /**
     * Multiply this fraction with another one.
     *
     * @param f the second factor
     */
    void mult(Fraction f) {
        n = f.n * n;
        d = f.d * d;
    }

    /**
     * Add a fraction to this one.
     *
     * @param f the fraction to add to this one.
     */
    void add(Fraction f) {
        d = f.d * d;  // bring to same denominator
        n = f.n * d + n * f.d;
    }
}
```



Only one object changes!

UML Notation

Fraction	
int z int n	*fields*
void mult(Fraction f) void add(Fraction f)	*methods*

class name: Fraction

Fraction	*simple form*
z n	
mult(f) add(f)	

Constructors

- Special methods
  - are called upon object creation
  - used for initialisation of values
  - have the same name as the class
  - without function type or `void`
  - can have parameters
  - can be overloaded

Constructors

```java
public class ExtendedFraction {
    int n;  // numerator
    int d;  // denominator

    /**
     * Constructor for the fraction class.
     * @param n
     * @param d
     */
    public ExtendedFraction(int n, int d) {
        this.n = n;
        this.d = d;
    }

    public ExtendedFraction() {
        n = 0;
        d = 1; // make sure denominator is not 0.
    }

    /**
     * Multiply this fraction with another one.
     *
     * @param f the second factor
     */
    void mult(ExtendedFraction f) {
        ...
```

```java
ExtendedFraction f = new ExtendedFraction();
ExtendedFraction g = new ExtendedFraction(3 , 5);
```

- calls matching constructors

Constructors…

- Example: time class

- Example: position class

Class example: java.lang.String

- Char-Array vs. Strings
  - `char[] helloArray = { 'h', 'e', 'l', 'l', 'o', '.' };`
  - `String helloString = new String(helloArray);`
  - `System.out.println(helloString);`
- Length of a String-Object
  - `helloString.length()`
- Reading chars from Strings
  - `helloString.charAt(2) // result: 'l,`
  - `helloString.getChars(...)`
  - `helloString.toCharArray()`

Example: Reverse String

```
public class ReverseString {
    public static void main(String[] args) {
        // input String
        String myString = new String("FTW");
        // data structures for reversing
        char[] tmpCharsIn = new char[myString.length()];
        char[] tmpCharsOut = new char[myString.length()];
        // getting the input data to an array:
        myString.getChars(0, myString.length(), tmpCharsIn, 0);
        // iterating output and setting chars:
        for (int i = 0; i < tmpCharsOut.length; i++) {
            tmpCharsOut[i] = tmpCharsIn[myString.length()-1-i];
        }
        // print result:
        System.out.println(new String(tmpCharsOut));
    }
}
```

Java String

- String concatination
  - `string1.concat(string2)`
  - `"Hello ".concat("World!")`
  - `"Hello " + "World!„`

- Note: The String class is <u>immutable</u>

Strings ⇄ Numbers

- String to number
  - `float a = (Float.valueOf(“3.14”)).floatValue();`
  - `float a = Float.parseFloat(“3.14”);`
  - Entsprechend für die anderen numerischen Typen
- Number to String
  - `String s = Double.toString(42.0);`

String - Manipulation

- Substring
  - `String substring(int beginIndex, int endIndex)`
  - `String substring(int beginIndex)`
- Lower and upper case
  - `String toLowerCase()`
  - `String toUpperCase()`
- trim white space at the end of a String
  - `String trim()`

String - Search

- Search for `char` or `String` in Strings
  - `int indexOf(int ch)`
  - `int lastIndexOf(int ch)`
  - `int indexOf(int ch, int fromIndex)`
  - `int lastIndexOf(int ch, int fromIndex)`
- With `String` as argument
  - `int indexOf(String str)`
  - `...`

Example

```java
public static void main(String[] args) {
    // input
    String myFileName = "paper.pdf";
    // find the position of the last dot
    int dotIndex = myFileName.lastIndexOf('.');
    // take substring and add new suffix
    String newFileName = myFileName.substring(0, dotIndex) + ".doc";
    // print result:
    System.out.println("newFileName = " + newFileName);
}
```

String – Add. Methods

- `boolean endsWith(String suffix)`
- `boolean startsWith(String prefix)`
- `int compareTo(String anotherString)`
- `boolean equals(Object anObject)`
- `...`

more information:
https://docs.oracle.com/javase/8/docs/api/java/lang/String.html

CharSequence

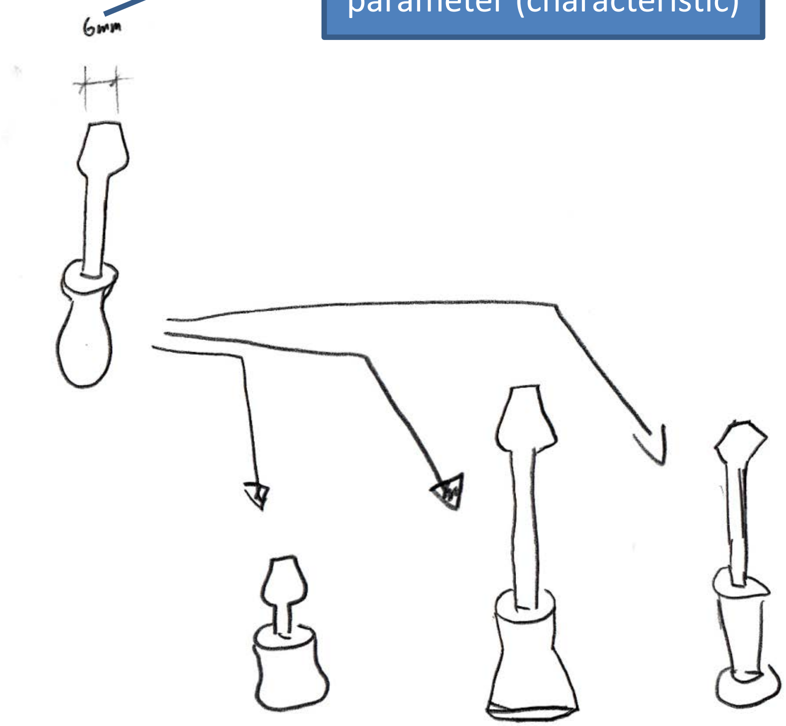
- `String` is immutable
  - Manipulations are expensive
- `CharSequence` is Interface String-like classes
  - `StringBuilder`
  - `StringBuffer`

more Information:
https://docs.oracle.com/javase/8/docs/api/java/lang/CharSequence.html

static

static variable, class parameter (characteristic)

Class



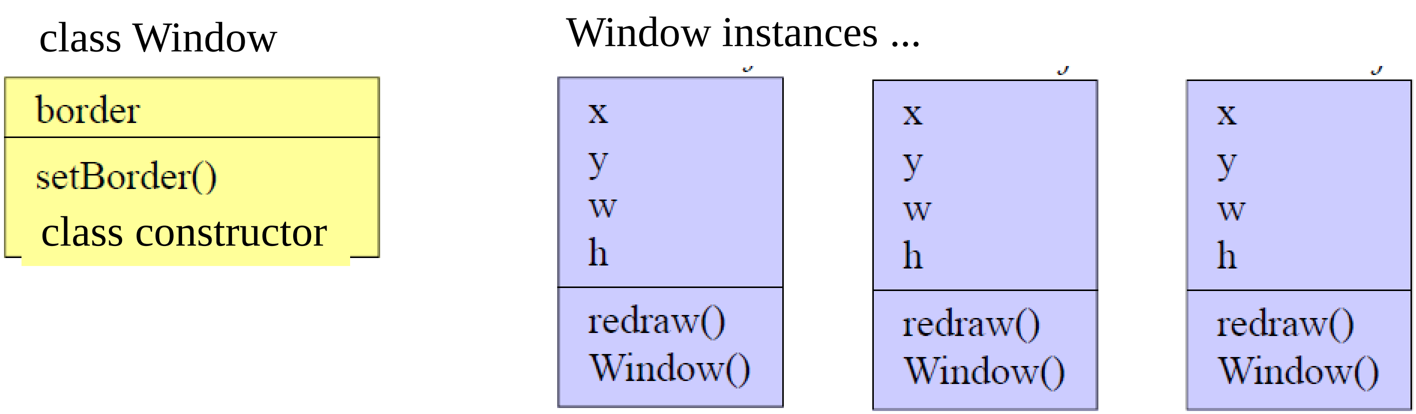
Instances

static

```
class Window {
    int x, y, w, h;        // object fields (in each object different)
    static int border;     // static (class) field (only once per class)

    // constructor (initialisation of the object)
    Window(int x, int y, int w, int h) {...}

    // class constructor (initialisation of the class)
    static {
        border = 3;
    }

    // method of the object (instance)
    void redraw () {...}

    // static (class) method, operates on class level, not object
    static void setBorder (int n) {border = n;}
}
```

static

- Object methods can access static (class) fields
  - `redraw()` can access `border`
- Static (class) methods can't access object fields
  - `setBorder()` can't access x



static

Order of execution

- Loading of class Window
  - class fields are created - `border`
  - class constructor is called
- At instantiantion time - `new Window(...)`
  - object fields are created - `x, y, w, h`
  - object constructor is called

static

- Accessing static members by class name
  - `Window.border = ...; Window.setBorder(3);`
  - Static methods can acces them directly
    `border = ...; setBorder(3);`
- Non static members: instance variable
  - `Window win = new Window(100, 50);`
    `win.x = ...; win.redraw();`
  - Non static methods can access object variables directly
    `x = ...; redraw();`

static

- Note: static fields will not be collected by the garbage collection.
- Therefore, prioritize locality of data!

- Cp. later lessons (object oriented programming, software engineering)

Example for static: java.lang.ath

- Java provides additional mathematical supprt in the class `Math`
- Each method in Math is static
  - optional static import
  - `import static java.lang.Math.*;`
  - method calls like global functions, eg. cos(x)

Java Math Constants

- Math.E
  - Euler‘s number e
- Math.PI
  - π

Java Math Basics

- absolute values
  - `int Math.abs(int value)`
  - also for `double`, `long`, `float`
- rounding up and down
  - `double Math.ceil(double value)`
  - `double Math.floor(double value)`
- rounding
  - `long Math.round(double value)`
  - `int Math.round(float value)`

Java Math Basics

- Minimum of two values
  - `double Math.min(double arg1, double arg2)`
  - also for `float, long, int`
- Maximum of two values
  - `double Math.max(double arg1, double arg2)`
  - also for `float, long, int`

Java Math Exp & Log

- Exponential function and logarithm
  - `double Math.log(double value)`
  - `double Math.exp(double value)`
- Power and root
  - `double Math.pow(double base, double exp)`
  - `double Math.sqrt(double value)`

Java Math Trigonometrie

- trigonometric functions
  - `double Math.sin(double value)`
  - auch für `cos, tan, asin, acos, atan`
- angle of a vector (polar coordinates)
  - `double Math.atan2(double x, double y)`

Example: ASCII sine wave

```java
public static void main(String[] args) {
   for (double d = 0d; d < 10; d+=0.1) {
         double x = 60*(Math.sin(d) + 1);
         x = Math.round(x);
         for (int i = 0; i< x; i++) System.out.print(' ');
         System.out.println('*');
   }
}
```

Java Math - Random

- `double Math.random()`
  - generates pseudo random number `0<=x<1`
  - sufficient for single numbers, not sequences
- Other value ranges
  - eg. `Math.random() * 10.0`

Example: Randome Names

```
public class SimpleNameGenerator {
    public static void main(String[] args) {
        char[] v = new char[]{'a', 'e', 'i', 'o', 'u', 'y'};
        char[] c = new String("bcdfghjklmnpqrstvwxz").toCharArray();
        System.out.print(getRandomChar(v));
        System.out.print(getRandomChar(c));
        System.out.print(getRandomChar(v));
        System.out.print(getRandomChar(c));
        System.out.print(getRandomChar(c));
        System.out.print(getRandomChar(v));
        System.out.print(getRandomChar(c));
    }

    public static char getRandomChar(char[] c) {
        int randomIndex = (int) Math.floor(c.length * Math.random());
        return c[randomIndex];
    }
}
```

More Math

- JavaDoc
  - https://docs.oracle.com/javase/8/docs/api/java/lang/Math.html
- BigInteger
  - for arbitrarily big integers
- BigDecimal
  - for arbitrarily precise decimal numbers

Example: Stack & Queue

- Stack
  - push(x) … puts on top of the stack
  - pop() … removes and returns topmost element
  - LIFO data structure == last in first out
- Queue (buffer)
  - put(x) … adds x at the end of the queue
  - get() … removes and returns first element
  - FIFO data structure == first in first out

Stack ...

```java
public class Stack {
    int[] data;
    int top;

    Stack(int size) {
        data = new int[size];
        top = -1;
    }

    void push(int x) {
        if (top == data.length - 1)
            System.out.println("-- overflow");
        else
            data[++top] = x;
    }

    int pop() {
        if (top < 0) {
            System.out.println("-- underflow");
            return 0;
        } else
            return data[top--];
    }
}
```

Usage:

```java
public static void main(String[] args) {
    Stack s = new Stack(10);
    s.push(3);
    s.push(5);
    int x = s.pop() - s.pop();
    System.out.println("x = " + x);
}
```

Queue

```java
public class Queue {
    int[] data;
    int head, tail, length;

    Queue(int size) {
        data = new int[size];
        head = 0;
        tail = 0;
        length = 0;
    }

    void put(int x) {
        if (length == data.length)
             System.out.println("-- overflow");
        else {
            data[tail] = x;
            length++;
            tail = (tail + 1) % data.length;
        }
    }

    int get() {
        int x;
        if (length <= 0) {
            System.out.println("-- underflow");
            return 0;
        } else  x = data[head];
        length--;
        head = (head + 1) % data.length;
        return x;
    }
}
```

Usage:

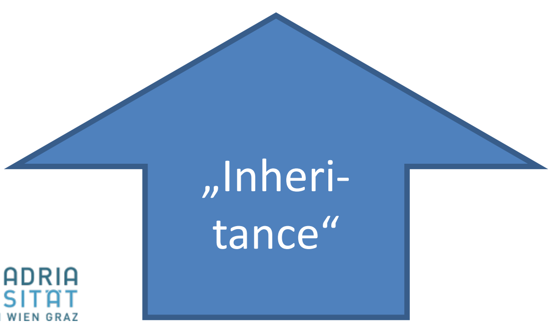
```java
Queue q = new Queue(10);
q.put(3);
q.put(6);
int x = q.get(); // x == 3
int y = q.get(); // y == 6
```

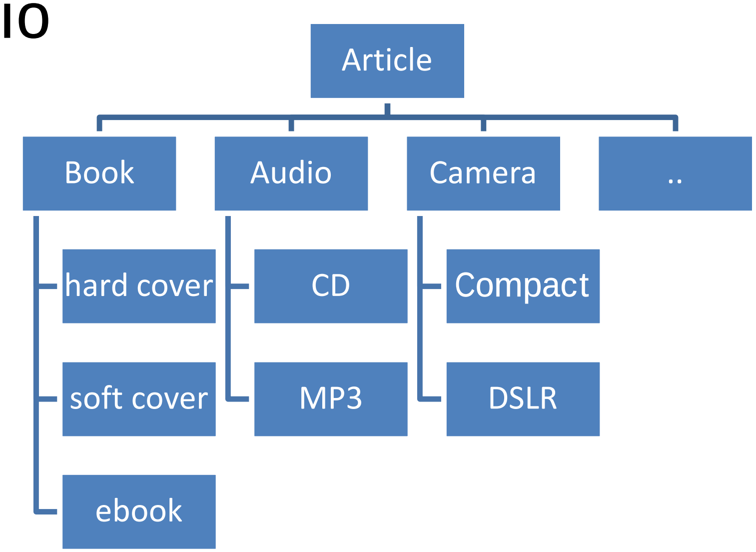
Classification

Real world concepts can often be ordered in a hierarchy

Example:

- ebook has all characteristics of a book
  ebook has all characteristics of an article
- CD and MP3 both are of type Audio
- Book, Audio and Camera are
  of type Article

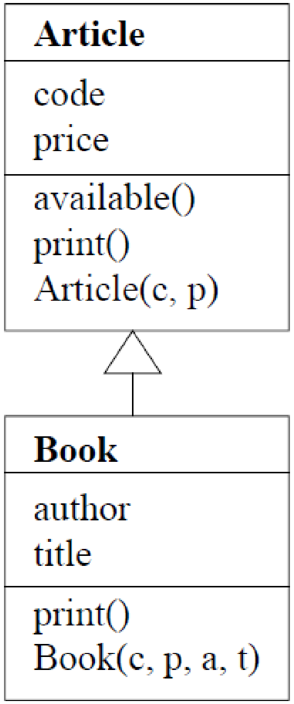




Inheritance

```
class Article {
    int code;
    int price;
    boolean available() {...}
    void print() {...}
    Article(int c, int p) {...}
}
```

superclass




```
class Book extends Article {
    String author;
    String title;
    void print() {...}
    Book(int c, int p,
        String a, String t) {...}
}
```

subclass

inherits: code, price, available, print
adds: author title, constructor
overrides: print

All classes extend Object, even if no superclass is given.

Overriding methods

```
class Article {
  ...
  void print() {
    Out.print(code + " " + price);
  }
  Article(int c, int p) {
    code = c; price = p;
  }
}
```

```
class Book extends Article {
  ...
  void print() {
    super.print();
    Out.print(" " + author + ": " + title);
  }
  Book(int c, int p, String a, String t) {
    super(c, p);
    author = a; title = t;
  }
}
```

```
Book book =
   new Book(code, price, author, title);
```

➔ creates Book object

➔ Book constructor

- ➔ Article constructor
- ➔ set Book fields

```
book.print();
```

➔ print() from Book object

- ➔ print() from Article
- ➔ Out.print(…)

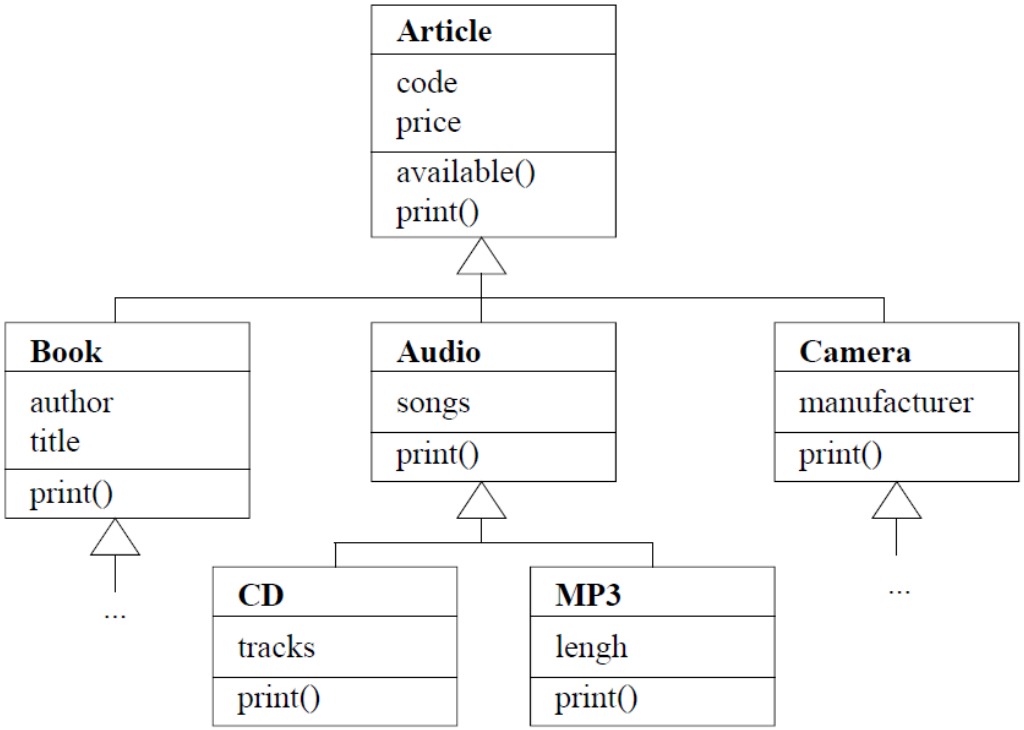
Addendum

`super` can only access the direct super class.

- Otherwise the principle of inheritance is violated
  - by ignoring the super class.

ALPEN-ADRIA UNIVERSITÄT KLAGENFURT | WIEN GRAZ

Class Hierarchies



Each book is an Article, but not each Article is a book

Inter-Class Compatibility

- Subclasses are specializations of superclasses
- `Book` objects can be assigned to `Article` variables

```
Article a = new Book(code, price, author, title);
```

a → | code, price | author, title |

a.code
a.price

Only Article fields are accessible now.

```
if (a instanceof Book)
   Book b = (Book) a;
```

runtime type test and cast

b → | code, price, author, title |

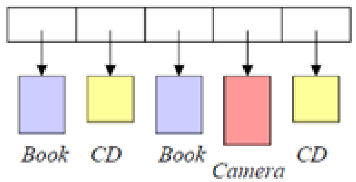
b.code
b.price
b.author
b.title

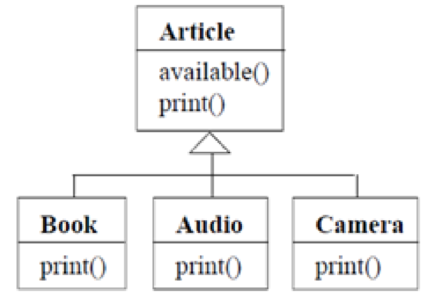
Now all fields are accessible.

Dynamic Binding

- Heterogeneous data structure

Article[] a;





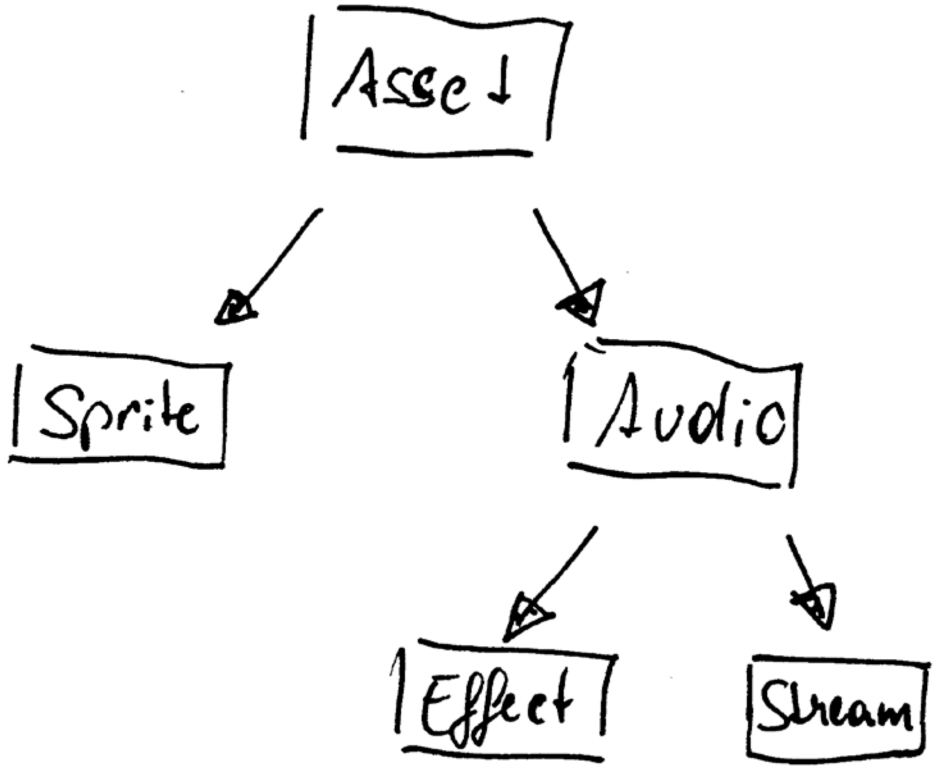
- All instances are of type Article and can be used as such:

```
void printArticles() {
    for (int i = 0; i < a.length; i++) {
        if (a[i].available()) {
            a[i].print();
        }
    }
}
```

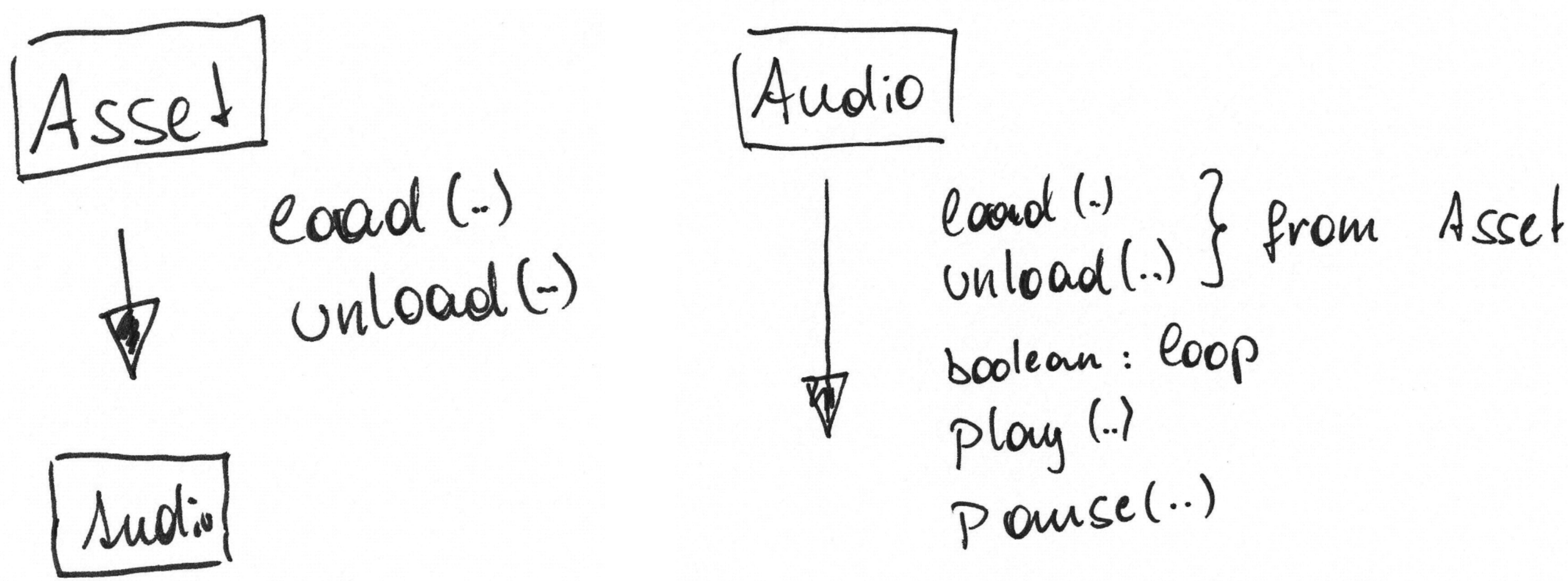
available() from the Article class
print() from Book, Audio or Camera.

- Dynamic binding: obj.print() calls the method of the actual instance.

Example …

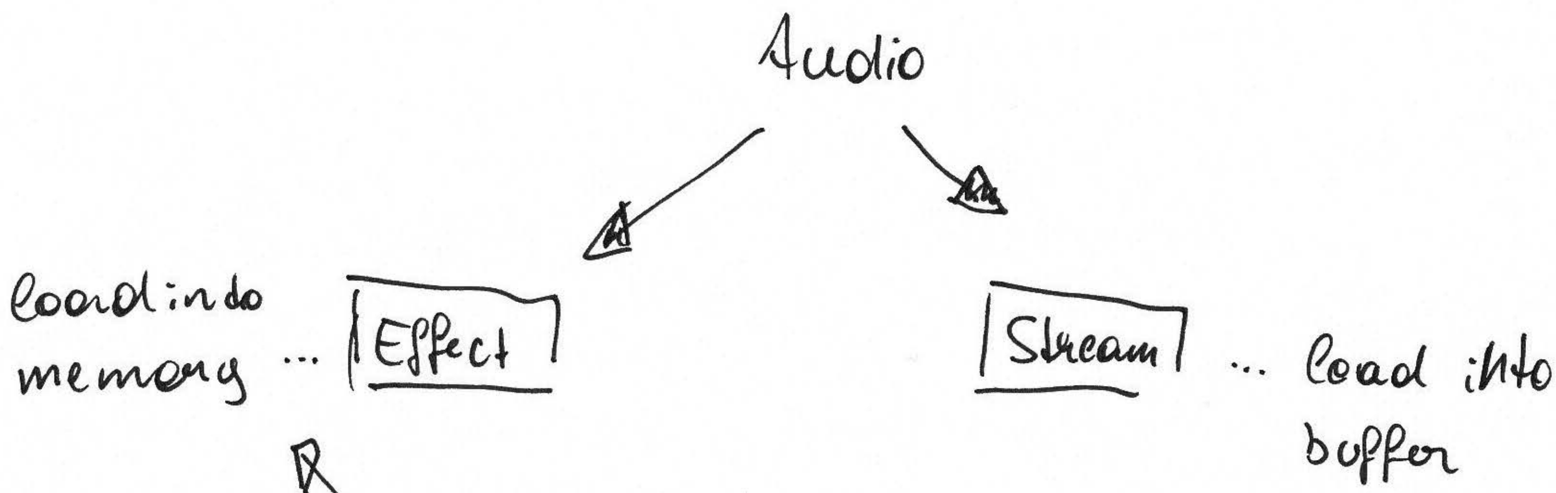


Example



Asset
load(..)
unload(..)
Audio
Audio
load(..)
unload(..)
from Asset
boolean : loop
play(..)
pause(..)

Example



Implementing abstract Methods

But re-using loop + Getter & Setter

Additional Concepts

Keyword **abstract**

- defines that each subclass has such a member,
- but does not implement / provide it
  - it has to be implemented by the subclass

ESOP – Information Hiding

Assoc. Prof. Dr. Mathias Lux

ITEC / AAU

Encapsulation

In big software projects the globally available names (classes, fields and methods) need to be structured and organized

- We distinguish between public and hidden identifiers.

Example

```
public class ShipExample {
    // actual position of the ship
    private int positionX, positionY;
    // maximum number for x and y
    private int maxX = 320, maxY = 640;

    public ShipExample() {
        this.positionX = maxX/2;
        this.positionY = maxY/2;
    }

    public void moveShip(int offSetX, int offsetY) {
        positionX += offSetX;
        positionY += offsetY;
        // check for violation of maximum
        if (positionX > maxX)
            positionX = maxX;
        if (positionY > maxY)
            positionY = maxY;
    }
}
```

Encapsulation

- Clients can only access specified classes, fields and methods.
- A critical part cannot be accessed or overwritten from external sources.

Encapsulation

- Identifier from the specification of an abstract data type should be public.
- Identifier, that are only needed fro implementation purposes shoüuld be hidden.

All in all ...

- Never put more out into the public than you actually need there.

Example: Too Public

```
Stack myStack = new Stack();
myStack.push(1);
myStack.push(2);
myStack.push(3);
myStack.top = 0; // 2 and 3 are „deleted“
int drei = myStack.pop();
```

ESOP – Recursion / Interface / Math

Assoc. Prof. Dr. Mathias Lux
ITEC / AAU

Let‘s recall …

Base Data Types

Signed, two-complement integers

- long – 64 bit
- int – 32 bit
- short – 16 bit
- byte – 8 bit

Floating point numbers

- float – 32 bit
- double – 64 bit

Others

- char - 16-bit Unicode character
- boolean – true / false

Referecne Data Types

Everything with „new“

- Arrays
- Objects

Wrapper Classes

- `Byte`, `Short`, `Integer`, `Long`, `Float`, `Double`
  - wrap base data types
- Wrapper classes are reference data types
  - no base data types!
- Wrapping is partially automated
  - Autoboxing & Unboxing
- Cp. class `Boolean`

Recursion

- A method **m()** is called *recursive*, if it calls itself.
  - m() -> m() -> m()   directly recursive
  - m() -> n() -> m()   indirectly recursive

Recursion: Factorial n!

- Definition factorial
  - n! = (n-1)! * n
  - 1! = 1

- Example
  - 4! = 4*3! = 4*3*2! = 4*3*2*1! = 4*3*2*1

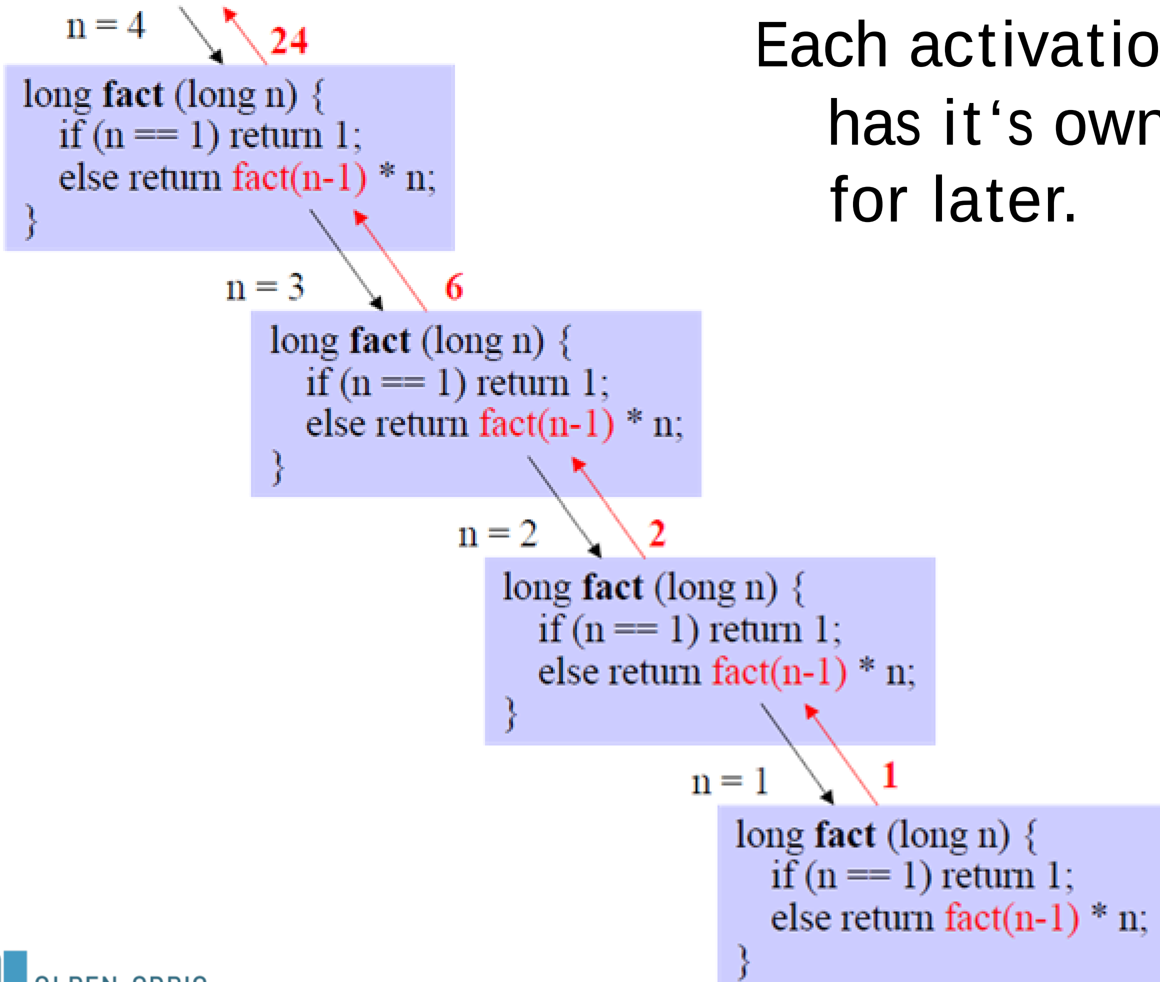
Recursion: Factorial n!

```
long fact (long n) {
  if (n == 1)
      return 1;
  else
      return fact(n-1) * n;
}
```

End of recursion
when reaching 1!

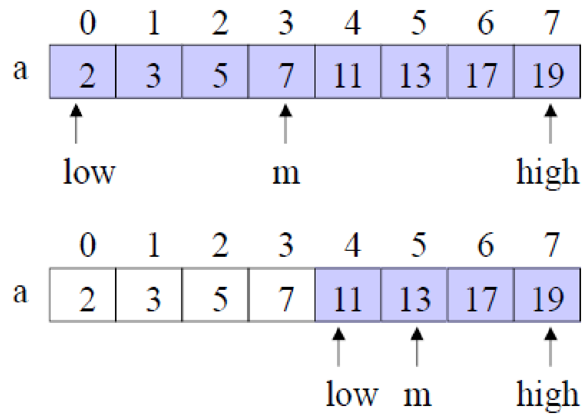
Recursive Process

Each activation of `fact(..)` has it's own n and stores it for later.



Example: Recursive Binary Search

Array has to be sorted!



- Find index m of the element in the middle
- 17 > a[m] -> search in right side of the array

```
static int search (int elem, int[] a, int low, int high) {
   if (low > high) return -1; // empty
   int m = (low + high) / 2;
   if (elem == a[m]) return m;
   if (elem < a[m]) return search(elem, a, low, m-1);
   return search(elem, a, m+1, high);
}
```

non-recursive part

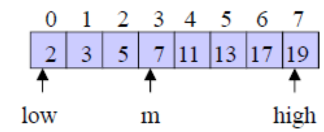
recursion

Example: Recursive Binary Search

elem = 17, low = 0, high = 7 ↑ 6

```
static int search (int elem, int[] a, int low, int high) {
    if (low > high) return -1;
    int m = (low + high) / 2;
    if (elem == a[m]) return m;
    if (elem < a[m]) return search(elem, a, low, m-1);
    return search(elem, a, m+1, high);
}
```

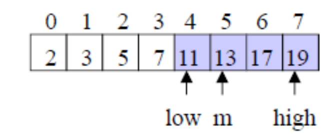
m = 3



low = 4, high = 7 ↓ ↑ 6

```
static int search (int elem, int[] a, int low, int high) {
    if (low > high) return -1;
    int m = (low + high) / 2;
    if (elem == a[m]) return m;
    if (elem < a[m]) return search(elem, a, low, m-1);
    return search(elem, a, m+1, high);
}
```

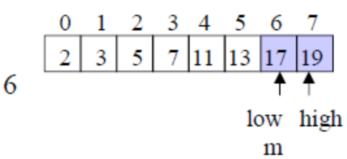
m = 5



low = 6, high = 7 ↓ ↑ 6

```
static int search (int elem, int[] a, int low, int high) {
    if (low > high) return -1;
    int m = (low + high) / 2;
    if (elem == a[m]) return m;
    if (elem < a[m]) return search(elem, a, low, m-1);
    return search(elem, a, m+1, high);
}
```

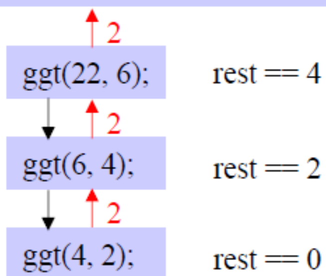
m = 6



Example: GCD

recursive

```
static int ggt (int x, int y) {
    int rest = x % y;
    if (rest == 0) return y;
    else return ggt(y, rest);
}
```



iterative

```
static int ggt (int x, int y) {
    int rest = x % y;
    while (rest != 0){
        x = y; y = rest;
        rest = x % y;
    }
    return y;
}
```

- Recursive algorithms can be implemented in an iterative way
  - recursive: often smaller program
  - iterative: often faster
- Recursion is extremely useful with some data structures (trees, graphs)

Example: Fibonacci Numbers

- $F_n = F_{n-1} + F_{n-2}$

```
public static int get(int number) {
  if (number <= 2)
     return 1;
  return get(number-1) + get(number-2);
}
```

Interfaces

- Class-like mechanism
  - for the definition of behaviour only.
- Allows for separation between definition and implementation
  - abstract data type

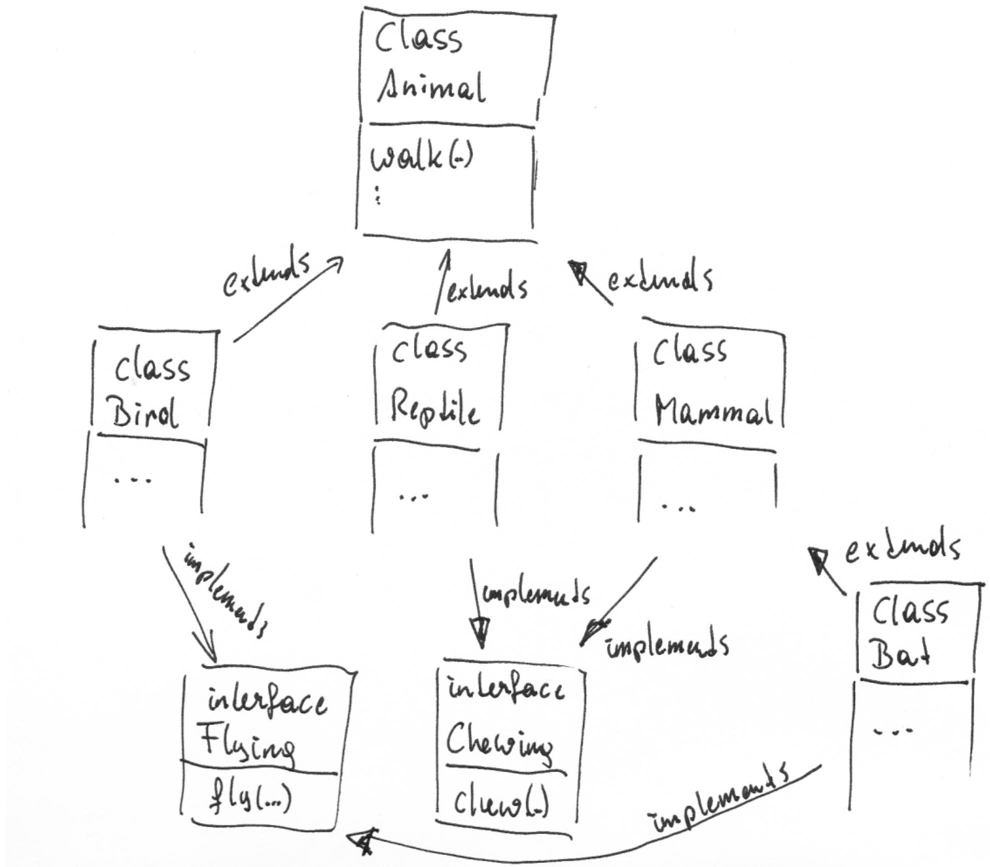
Interfaces

- Specification via `interface` keyword
- Method specifications
  - describe how to handle the implementing object.
  - without method body, just the head
- No object variables
  - Aber evt. Konstante

Interfaces

- The name of the interface can be used as a data type in Java.
- Implementation of an interface via `class`
  - implementing methods
  - having instance variables

Interface Example I



Interface Example II

Java™ 2 Platform Standard Ed. 5.0

java.lang

Interface Iterable<T>

All Known Subinterfaces:

BeanContext, BeanContextServices, BlockingQueue<E>, Collection<E>, List<E>, Queue<E>, Set<E>, SortedSet<E>

All Known Implementing Classes:

AbstractCollection, AbstractList, AbstractQueue, AbstractSequentialList, AbstractSet, ArrayBlockingQueue, ArrayList, AttributeList, BeanContextServicesSupport, BeanContextSupport, ConcurrentLinkedQueue, CopyOnWriteArrayList, CopyOnWriteArraySet, DelayQueue, EnumSet, HashSet, JobStateReasons, LinkedBlockingQueue, LinkedHashSet, LinkedList, PriorityBlockingQueue, PriorityQueue, RoleList, RoleUnresolvedList, Stack, SynchronousQueue, TreeSet, Vector

```
public interface Iterable<T>
```

Implementing this interface allows an object to be the target of the "foreach" statement.

Method Summary	
`Iterator<T>`	`iterator()` Returns an iterator over a set of elements of type T.

Method Detail

iterator

```
Iterator<T> iterator()
```

Returns an iterator over a set of elements of type T.

When to use Interfaces?

- Making minimal functionality of an abstract data type visible
- Multiple inheritance
  - Graph, nicht Baum

Interface Examples

- Java Interfaces `Iterable`, `Comparable` und `Serializable`

NameGenerator

- How can a name generator be programmed with interfaces & inheritance?