Einführung in die strukturierte und objektorientierte 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 & tutorial

• 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.
ISBN 978-3-86490-099-0
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 do it right.
What is a program

program = data + commands
Data

- Set of address-able memory cells

\[
\begin{array}{ccc}
x & y & z \\
17 & 4 & 21 \\
\end{array}
\]

- Data is stored in binary format, e.g. \(17 = 10001\)
- Binary format is universal
  - numbers, text, image, audio, ...

- 1 Byte = 8 Bit
- 1 Wort = 4 Byte (typically)
### Commands

- **Operations on memory cells**

<table>
<thead>
<tr>
<th>Machine language</th>
<th>Programming Language</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>ACC ← x</code></td>
<td><code>z = x + y;</code></td>
</tr>
<tr>
<td><code>ACC ← ACC + y</code></td>
<td></td>
</tr>
<tr>
<td><code>z ← ACC</code></td>
<td></td>
</tr>
</tbody>
</table>

// load memory cell x
// add memory cell y
// store result in memory cell z
How to create a program?

1. **Problem**
2. **Spzefication**
3. **Recipe for solution**
4. **Requirements**
5. **Algorithm**
6. **Coded solution**
7. **Program**
8. **Machine Program**
9. **Compiler**
10. **Loader**
Algorithm

• Precise, step by step solution to a problem

  name

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

• program = specification of an algorithm in a programming language
Variables

• Variables are named container for values.

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
</tr>
</thead>
<tbody>
<tr>
<td>99</td>
<td>3</td>
</tr>
</tbody>
</table>

• Values can change

\[ x \leftarrow x + 1 \]

<table>
<thead>
<tr>
<th>x</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Type</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number</td>
<td>17, 54</td>
</tr>
<tr>
<td>Letter</td>
<td>‘a’, ‘x’</td>
</tr>
</tbody>
</table>
Statements

• Assignement

  1. compute value
  2. assign result to variable

• Sequence of statements

```
x ← 3
y ← 4
z ← x + y
```

Assertion

```
x = 3, y = 4, z = 7
```
Statements

- Condition / Choice
Statements

• Iterations, Loops
Example: swap values

proof of concept

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
<th>h</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td></td>
</tr>
</tbody>
</table>
Example: swap values

```java
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

Max (\(\downarrow a, \downarrow b, \downarrow c\), \(\uparrow \text{max}\))

\[ a > b \]

\[ a > b \? \]

\[ a > b \& a > c \]

\[ a > c \? \]

\[ c \geq a > b \]

\[ b \geq a \& b > c \]

\[ b > c \? \]

\[ a \leq b \leq c \]

\[ \text{max} \leftarrow a \]

\[ \text{max} \leftarrow c \]

\[ \text{max} \leftarrow b \]

\[ \text{max} \leftarrow c \]
Example: maximum of three numbers

```java
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

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
<th>rest</th>
</tr>
</thead>
<tbody>
<tr>
<td>28</td>
<td>26</td>
<td>8</td>
</tr>
<tr>
<td>26</td>
<td>8</td>
<td>4</td>
</tr>
<tr>
<td>8</td>
<td>4</td>
<td>0</td>
</tr>
</tbody>
</table>

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

```java
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

<table>
<thead>
<tr>
<th>x</th>
<th>root</th>
<th>a</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>5</td>
<td>2</td>
</tr>
<tr>
<td>3.5</td>
<td>3.5</td>
<td>2.85714</td>
</tr>
<tr>
<td>3.17857</td>
<td>3.14607</td>
<td></td>
</tr>
<tr>
<td>3.162222</td>
<td>3.162228</td>
<td></td>
</tr>
<tr>
<td>3.16228</td>
<td>3.16228</td>
<td></td>
</tr>
</tbody>
</table>
Example: square root

```java
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].
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
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

<table>
<thead>
<tr>
<th>Notation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Graphical or verbal notation</td>
<td></td>
</tr>
<tr>
<td>Higher programming languages (like Java)</td>
<td></td>
</tr>
<tr>
<td>Assembly languages</td>
<td></td>
</tr>
<tr>
<td>Machine code</td>
<td></td>
</tr>
<tr>
<td>Hardware, electric signals</td>
<td></td>
</tr>
</tbody>
</table>
Euclidian Algorithm \( \text{ggT}(A, B) \)

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

input of A and B

A > B

yes

A ↦ A - B

no

A < B

no

B ↦ B - A

yes

A = B

no

yes

Output of A
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

Eingabe von A und B

<table>
<thead>
<tr>
<th></th>
<th>ja</th>
<th>nein</th>
</tr>
</thead>
<tbody>
<tr>
<td>A &gt; B</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ja</td>
<td></td>
<td></td>
</tr>
<tr>
<td>A ≤ A - B</td>
<td>ja</td>
<td>nein</td>
</tr>
<tr>
<td>nein</td>
<td></td>
<td></td>
</tr>
<tr>
<td>B ≤ B - A</td>
<td></td>
<td></td>
</tr>
<tr>
<td>bis A = B</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ausgabe von A</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Pseudocode

- Semi-formal languages
- Examples:

  ```plaintext
  WHILE A not equal B
  IF A > B
  THEN subtract B from A
  ELSE
  subtract A from B
  ENDIF
  ENDWHILE
  ENDWHILE

  ggT := A
  ```
ESOP - Simple Programs

Assoc. Prof. Dr. Mathias Lux
ITEC / AAU
Agenda

• Symbols
• Variables, Constants
• Assignments
• Operators
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 escaped to \"

Examples
  - "a simple string"
  - "she said "Hallo""
Symbols: strings

• **String**
  – in Java not a base data type but an object.

• **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

<table>
<thead>
<tr>
<th></th>
<th>8 bit</th>
<th>2^7 .. 2^7-1</th>
<th>-128 .. 127</th>
</tr>
</thead>
<tbody>
<tr>
<td>byte</td>
<td>16 bit</td>
<td>2^15 .. 2^15-1</td>
<td>-32.768 .. 32.767</td>
</tr>
<tr>
<td>short</td>
<td>32 bit</td>
<td>2^31 .. 2^31-1</td>
<td>-2.147.483.648 ..</td>
</tr>
<tr>
<td>int</td>
<td>64 bit</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>long</td>
<td></td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

- **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 addresses, people names, licenses!
We've found 26,340 code results

- romankalib/PMap - main.js
  Last indexed on 3 Aug
  ```javascript
  debug("\"shit\"\";)
  ```

- matthewcv/nodestuff - mmcolors.js
  Last indexed on 31 Jul
  ```javascript
  alert("\"shit\"\";)
  ```

- lw8851206/HelloWorld - test.js
  Last indexed on 29 Jul
  ```javascript
  function shit()
  ```

- AC.SvsFMI/theDoctors - shit.js
  Last indexed on 25 Jul
  ```javascript
  alert("\"shit\"")
  ```

- nitirajathores/testrepo - lts.js
  Last indexed on 2 Aug
  ```javascript
  alert("\"fucking else shit\"")
  ```

- gpestania/legacy_slick.js - core_tests.js
  Last indexed on 1 Aug
  ```javascript
  /*tests and shit...*/
  ```

- bmelon11/myrepo - boo.js
  Last indexed on 28 Jul
  ```javascript
  console.log("eat shit")
  ```

- apiengine/apengine-client - page.js
  Last indexed on 23 Jul
  ```javascript
  some profile shit goes here
  ```

- AchintyaAshok/NYT-Intern-Project-Front-End - storyView.js
  Last indexed on 26 Jul
  ```javascript
  console.log("django is shit");
  ```

- JamieAppleseed/jamieappleseed.com - application.js
  Last indexed on 8 Aug
  ```javascript
  (function(win){
  // do shit
  })();
  ```
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 ..
Assignments

- 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

\[ x = y + 1; \]

1. compute statement
2. store in variable
Assignments

- Examples

```c
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. \(- x + 3 \times (y + 1)\)
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 binary 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.

<table>
<thead>
<tr>
<th>Multiplikation</th>
<th>Division</th>
</tr>
</thead>
<tbody>
<tr>
<td>x * 2</td>
<td>x &lt;&lt; 1</td>
</tr>
<tr>
<td>x * 4</td>
<td>x &lt;&lt; 2</td>
</tr>
<tr>
<td>x * 8</td>
<td>x &lt;&lt; 3</td>
</tr>
<tr>
<td>x * 16</td>
<td>x &lt;&lt; 4</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Divisor</th>
</tr>
</thead>
<tbody>
<tr>
<td>x / 2</td>
</tr>
<tr>
<td>x / 4</td>
</tr>
<tr>
<td>x / 8</td>
</tr>
<tr>
<td>x / 16</td>
</tr>
<tr>
<td>...</td>
</tr>
</tbody>
</table>

Division only works out for positive numbers.
The power of two (shifts)

Examples

\[
\begin{align*}
\text{x = 3;} & \quad 0000\ 0011 \\
\text{x = x << 2;} & \quad 0000\ 1100 \\
\text{x = x = -3;} & \quad 1111\ 1101 \\
\text{x = x << 1;} & \quad 1111\ 1010 \\
\text{x = x = 15;} & \quad 0000\ 1111 \\
\text{x = x >> 2;} & \quad 0000\ 0011
\end{align*}
\]
Assignment operators.

- arithmetic operations can be combined with assignments.

<table>
<thead>
<tr>
<th></th>
<th>short</th>
<th>long</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>+=</code></td>
<td><code>x += y;</code></td>
<td><code>x = x + y;</code></td>
</tr>
<tr>
<td><code>-=</code></td>
<td><code>x -= y;</code></td>
<td><code>x = x - y;</code></td>
</tr>
<tr>
<td><code>*=</code></td>
<td><code>x *= y;</code></td>
<td><code>x = x * y;</code></td>
</tr>
<tr>
<td><code>/=</code></td>
<td><code>x /= y;</code></td>
<td><code>x = x / y;</code></td>
</tr>
<tr>
<td><code>%=</code></td>
<td><code>x %= y;</code></td>
<td><code>x = x % y;</code></td>
</tr>
</tbody>
</table>

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;   // 00000011

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

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

Text has to be in file named ProgramName.java
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
If-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

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

Indentation

Best Practice: ```{...}``` for single statements too.
Indentations

• For readability
  – visualize structure

• 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

- 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

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

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

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>==</td>
<td>equal</td>
<td>x==3</td>
</tr>
<tr>
<td>!=</td>
<td>not equal</td>
<td>x!=y</td>
</tr>
<tr>
<td>&gt;</td>
<td>larger than</td>
<td>4&gt;3</td>
</tr>
<tr>
<td>&lt;</td>
<td>smaller than</td>
<td>x+1&lt;0</td>
</tr>
<tr>
<td>&gt;=</td>
<td>larger or equal</td>
<td>x&gt;=y</td>
</tr>
<tr>
<td>&lt;=</td>
<td>smaller or equal</td>
<td>x&lt;=y</td>
</tr>
</tbody>
</table>
Combining Comparisons

&& logic AND

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
<th>x &amp;&amp; y</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td>true</td>
<td>false</td>
<td>false</td>
</tr>
<tr>
<td>false</td>
<td>true</td>
<td>false</td>
</tr>
<tr>
<td>false</td>
<td>false</td>
<td>false</td>
</tr>
</tbody>
</table>

|| logic OR

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

! logic NOT

<table>
<thead>
<tr>
<th>x</th>
<th>!x</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>false</td>
</tr>
<tr>
<td>false</td>
<td>true</td>
</tr>
</tbody>
</table>
Boolean Operators

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

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

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

- Examples

```java
boolean p, q;
p = false;
qu = x > 0;
p = p || q && x < 10;
```
DeMorgan Rules

- $! (a \land b) \iff ! a \lor ! b$
- $! (a \lor b) \iff ! a \land ! b$

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

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

Statement = Assignment | IfStatement | SwitchStatement | WhileStatement | ... | Block.

WhileStatement = "while" "(" Expression ")" Statement .
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

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

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

<table>
<thead>
<tr>
<th>n</th>
<th>n % 10</th>
</tr>
</thead>
<tbody>
<tr>
<td>123</td>
<td>3</td>
</tr>
<tr>
<td>12</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
For Loop (Counting Loop)

- Used if number of iterations is known beforehand

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

.. is actually short for

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

1) Initialisation
2) Continuation condition
3) Update
### For Loop Examples

<table>
<thead>
<tr>
<th>Loop</th>
<th>Description</th>
<th>Iteration</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>for (i = 0; i &lt; n; i++)</code></td>
<td>i: 0, 1, 2, 3, ..., n-1</td>
<td></td>
</tr>
<tr>
<td><code>for (i = 10; i &gt; 0; i--)</code></td>
<td>i: 10, 9, 8, 7, ..., 1</td>
<td></td>
</tr>
<tr>
<td><code>for (int i = 0; i &lt;= n; i = i + 1)</code></td>
<td>i: 0, 1, 2, 3, ..., n</td>
<td></td>
</tr>
</tbody>
</table>
| `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

ForInit = Assignment {""," Assignment} | Type VarDecl {""," VarDecl}.
ForUpdate = Assignment {""," Assignment}. 
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*

```java
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

```java
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

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)
ESOP - Gleitkommazahlen, Methoden und Arrays

Assoc. Prof. Dr. Mathias Lux
ITEC / AAU
/**
 * 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);
        }
    }
}
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

```java
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 division)
  – 1.0 / i ... a double value
Float vs. Double

```java
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 smallest 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

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Default Value (for fields)</th>
</tr>
</thead>
<tbody>
<tr>
<td>byte</td>
<td>0</td>
</tr>
<tr>
<td>short</td>
<td>0</td>
</tr>
<tr>
<td>int</td>
<td>0</td>
</tr>
<tr>
<td>long</td>
<td>0L</td>
</tr>
<tr>
<td>float</td>
<td>0.0f</td>
</tr>
<tr>
<td>double</td>
<td>0.0d</td>
</tr>
<tr>
<td>char</td>
<td>'u0000'</td>
</tr>
<tr>
<td>String (or any object)</td>
<td>null</td>
</tr>
<tr>
<td>boolean</td>
<td>false</td>
</tr>
</tbody>
</table>
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, ...

```java
static void P() {
    ...
    Q();
    ...
}

static void Q() {
    ...
    R();
    ...
}

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

    public static void main(String[] args) {  // method call
        printRule();
        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**

```java
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.
- \( x = 100; \ y = 2 \times i; \)
  - actual parameters need to be type compatible with the formal parameters.

```java
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 \times i);
    }

}```
Functions

- Functions are methods that return a value.

```java
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 return type, e.g., `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**

```java
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;
    }
}
```

The binomial coefficient \( \binom{n}{k} \) can be calculated using the formula:

\[
\binom{n}{k} = \frac{n!}{k! \cdot (n-k)!}.
\]
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;
        }
    }
}
Prime Numbers

```java
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
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
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
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;
    }
}
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 its 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.

```java
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)

```java
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.

```java
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 arguments can be declared.

```java
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 a fixed length
  – which is given at the time of instantiation
• 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

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

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

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

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

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

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

```plaintext
a[3] = 0;
a[2*i+1] = a[i] * 3;
int len = a.length;
```
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);
    }
}
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“
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);
  }
}
Arrayzuweisung

```java
int[] a, b;
a = new int[3];
b = a;
a[0] = 17;
a = new int[4];
b = null;
```

array elements in Java are initialized with 0

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

changes b[0] too!

a now points to new array.

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

Can be solved in many different ways.

Cp. AlgoDat lesson!
Example: Eratosthenes‘ Sieve

```java
public class Sieve {
    public static void main(String[] args) {
        int maxPrime = 200000;
        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

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>a[0][0]</td>
<td>a[0][1]</td>
<td>a[0][2]</td>
</tr>
<tr>
<td>1</td>
<td>a[1][0]</td>
<td>a[1][1]</td>
<td>a[1][2]</td>
</tr>
<tr>
<td>2</td>
<td>a[2][0]</td>
<td>a[2][1]</td>
<td>a[2][2]</td>
</tr>
<tr>
<td>3</td>
<td>a[3][0]</td>
<td>a[3][1]</td>
<td>a[3][2]</td>
</tr>
</tbody>
</table>

• 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:

<table>
<thead>
<tr>
<th>Date</th>
<th>class name</th>
</tr>
</thead>
<tbody>
<tr>
<td>day : int</td>
<td>fields (class members)</td>
</tr>
<tr>
<td>month : String</td>
<td></td>
</tr>
<tr>
<td>year : int</td>
<td></td>
</tr>
</tbody>
</table>
Data Type Class

- Declaration
- Data type usage
- Access

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

```java
Date x, y;
```

```java
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 explicitly before use.
  – variable otherwise have the value null
Objects

Date x, y;

Instantiation
x = new Date();

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

reserves memory for the address
x,y have value null
creates a new Date object and assigns ist address to x.
Initial values are
0, null, false or \u0000
Assignments

```plaintext
y = x;

y.day = 20;
```

Reference / address assignment

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

\[ x == y \text{ returns true} \]
\[ x == y \text{ returns false} \]
C om pares actual values

• Has to be implemented by method.

```java
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

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

Compile
$> javac MainProgram.java

Multiple files

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

Compile
$> javac MainProgram.java C1.java C2.java
What can we do with classes?

class Point {
    double x, y;
}

class Polygon {
    Point[] points;
}
What can we do with classes?

- Classes can use other classes – and extend on that

```java
class Point {
    int x, y;
}
class Polygon {
    Point[][] pt;
    int color;
}
```
What can we do with classes?

- Implement methods with multiple return values

```java
public class Time {
    int h, m, s;
}
public 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

```java
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 its 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;
    }

    // [...]
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;
    }
}
<table>
<thead>
<tr>
<th>Fraction</th>
<th>class name</th>
</tr>
</thead>
<tbody>
<tr>
<td>int z</td>
<td>fields</td>
</tr>
<tr>
<td>int n</td>
<td></td>
</tr>
<tr>
<td>void mult(Fraction f)</td>
<td>methods</td>
</tr>
<tr>
<td>void add(Fraction f)</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Fraction</th>
<th>simple form</th>
</tr>
</thead>
<tbody>
<tr>
<td>z</td>
<td></td>
</tr>
<tr>
<td>n</td>
<td></td>
</tr>
<tr>
<td>mult(f)</td>
<td></td>
</tr>
<tr>
<td>add(f)</td>
<td></td>
</tr>
</tbody>
</table>
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
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) {
        ...
    }
}

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()
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 concatenation
  - `string1.concat(string2)`
  - "Hello ".concat("World!")
  - "Hello " + "World!"

- Note: The String class is immutable
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)
  – ...
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)
- ...
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
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;}
}
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
• Accessing static members by class name
  – `Window.border = ...; Window.setBorder(3);`
  – Static methods can access 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 support 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 $0 \leq x < 1$
  - sufficient for single numbers, not sequences

- Other value ranges
  - eg. `Math.random() * 10.0`
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
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:

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);
}
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;
    }
}

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

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

superclass

subclass

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

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

```java
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.
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

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

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

Only Article fields are accessible now.

runtime type test and cast

Now all fields are accessible.
Dynamic Binding

- Heterogeneous data structure

```java
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.

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

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

- Asset
  - load(\cdot)
  - unload(\cdot)
- Audio
  - load(\cdot)
  - unload(\cdot) from Asset
  - boolean: loop
  - play(\cdot)
  - pause(\cdot)
Example

Audio

Load into memory ... [Effect]

[Stream] ... Load into buffer

Implementing abstract methods

But re-using loop + Getter & Setter
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.
public class ShipExample {
    // actual position of the ship
    private int posX, posY;
    // maximum number for x and y
    private int maxX = 320, maxY = 640;

    public ShipExample() {
        this.posX = maxX / 2;
        this.posY = maxY / 2;
    }

    public void moveShip(int offsetX, int offsetY) {
        posX += offsetX;
        posY += offsetY;
        // check for violation of maximum
        if (posX > maxX)
            posX = maxX;
        if (posY > maxY)
            posY = 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 for implementation purposes should 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

**Reference 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.
  - \( m() \to m() \to m() \) directly recursive
  - \( m() \to n() \to m() \) 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
long fact (long n) {
    if (n == 1)
        return 1;
    else
        return fact(n-1) * n;
}

End of recursion when reaching 1!
Each activation of `fact(..)` has its 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);
}
```
Example: Recursive Binary Search

```java
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);
}
```

Example:
1. `elem = 17, low = 0, high = 7`
2. `elem = 17, low = 4, high = 7`
3. `elem = 17, low = 6, high = 7`

Diagram:
- Initial array: 0 1 2 3 4 5 6 7
- First search: 2 3 5 7 11 13 17 19
- Second search: 2 3 5 7 11 13 17 19
- Third search: 2 3 5 7 11 13 17 19

Result:
- First search: `m = 3`
- Second search: `m = 5`
- Third search: `m = 6`
Example: GCD

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

Iterative:
```java
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

- \( F_n = F_{n-1} + F_{n-2} \)

```java
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
java.lang

**Interface Iterable**<T>**

**All Known Subinterfaces:**
`BeanContext`, `BeanContextSupport`, `BlockingQueue<E>`, `Collection<E>`, `List<E>`, `Queue<E>`, `Set<E>`, `SortedSet<E>`

**All Known Implementing Classes:**
`AbstractCollection`, `AbstractList`, `AbstractQueue`, `AbstractSequentialList`, `AbstractSet`, `ArrayList`, `AttributeList`, `BeanContextSupport`, `BeanContextSupport`, `ConcurrentLinkedQueue`, `CopyOnWriteArrayList`, `CopyOnWriteArraySet`, `DelayQueue`, `EnumSet`, `HashSet`, `JobStateReasons`, `LinkedBlockingQueue`, `LinkedHashSet`, `LinkedList`, `PriorityBlockingQueue`, `PriorityQueue`, `RoleList`, `RoleList`, `RoleUnresolvedList`, `Stack`, `SynchronousQueue`, `TreeSet`, `Vector`

```java
public interface Iterable<T> {

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

**Method Summary**

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>Iterator&lt;T&gt; iterator()</code></td>
<td>Returns an iterator over a set of elements of type T.</td>
</tr>
</tbody>
</table>

**Method Detail**

**iterator**

```java
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?