ETH zürich

Hermann Lehner, Felix Friedrich

Computer Science I

Course at D-BAUG at ETH Zurich

Autumn 2019

1. Introduction

Welcome to the Lecture Series!

`https://www.mentimeter.com/s/54775dbcef2827005cfcaa8e80bff221`

Mathematics used to be the lingua franca of the natural sciences on all universities. Today this is computer science.

Lino Guzzella, president of ETH Zurich 2015-2018, NZZ Online, 1.9.2017

((BTW: Lino Guzzella is not a computer scientist, he is a mechanical engineer and prof. for thermotronics ☺)

Programming and Problem Solving

In this course you learn how to program using Java

- Software development is a handicraft
- Analogy: learn to play a musical instrument
- **The problem:** nobody has become a pianist from listening to music.

Hence this course offers several possibilities, to train. Make use of it!

Programming and **problem solving**

In this course you learn to solve problems with selected algorithms and data structures

- **Fundamental knowledge** independent of the language
- Comparison: musical scale, read music, rythm skills.
- **The problem:** without musical instrument this is no fun.

Hence we combine learning problem solving with learning the programming language Java.

Course Content

Programming using Java

introduction
statements and expressions
number representations
control flow
arrays
methods and recursion
types, classes and objects
inheritance and polymorphy

Algorithmen

Searching and Sorting

Goal of *today's* Lecture

- Introduction of computer model and algorithms
- Writing a **first program**
- General informations to the course

1.1 Computer Science and Algorithms

Computer Science, Euclidean Algorithm

Algorithm: Fundamental Notion of Computer Science

Algorithm:

- Instructions to solve a problem step by step
- Execution does not require any intelligence, but precision (even computers can do it)
- according to *Muhammed al-Chwarizmi*, author of an arabic computation textbook (about 825)

"Dixit algorizmi..." (Latin translation)

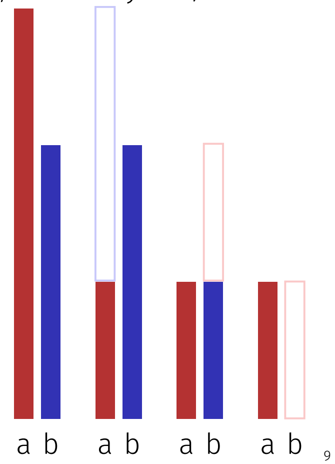
http://de.wikipedia.org/wiki/Algorithmus

Oldest Nontrivial Algorithm

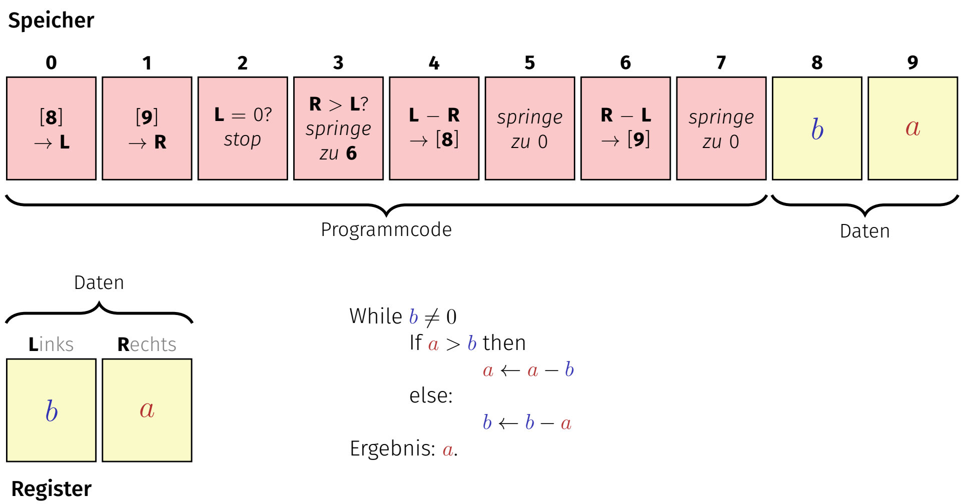
Euclidean algorithm (from the *elements* from Euklid, 3. century B.C.)

- Input: integers $a > 0$, $b > 0$
- Output: gcd of a und b

While $b \neq 0$
  If $a > b$ then
    $a \leftarrow a - b$
  else:
    $b \leftarrow b - a$
Result: a.



Euklid in the Box



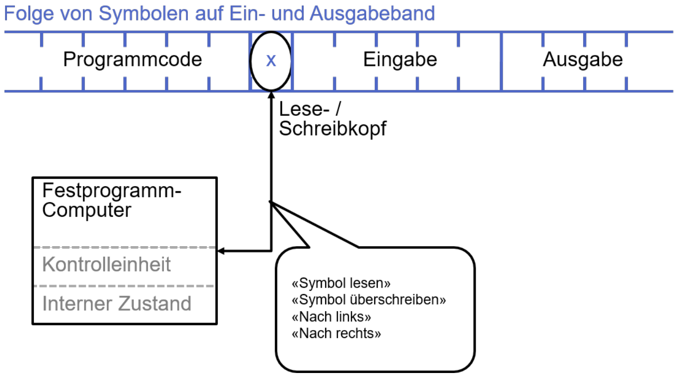
While $b \neq 0$
  If $a > b$ then
    $a \leftarrow a - b$
  else:
    $b \leftarrow b - a$
Ergebnis: a.

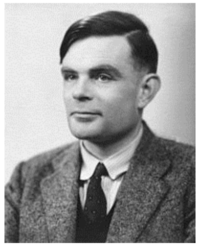
1.2 Computer Model

Turing Machine, Von Neumann Architecture

Computers – Concept

An bright idea: universal Turing machine (Alan Turing, 1936)



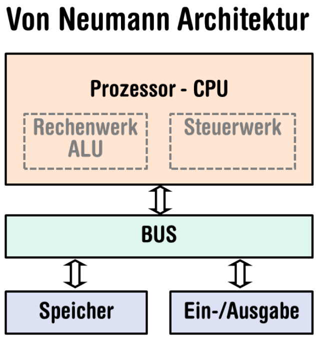


Alan Turing

http://en.wikipedia.org/wiki/Alan_Turing

Computer – Implementation

- Z1 – Konrad Zuse (1938)
- ENIAC – John Von Neumann (1945)



Konrad Zuse

John von Neumann

http://www.hs.uni-hamburg.de/DE/GNT/hh/biogr/zuse.htm
http://commons.wikimedia.org/wiki/File:John_von_Neumann.jpg

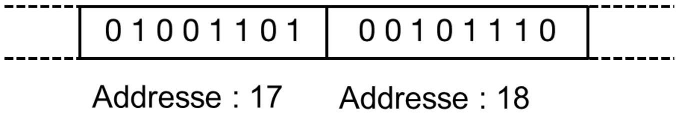
Computer

Ingredients of a *Von Neumann Architecture*

- Memory (RAM) for programs **and** data
- Processor (CPU) to process programs and data
- I/O components to communicate with the world

Memory for data *and* program

- Sequence of bits from $\{0, 1\}$.
- Program state: value of all bits.
- Aggregation of bits to memory cells (often: 8 Bits = 1 Byte)
- Every memory cell has an address.
- Random access: access time to the memory cell is (nearly) independent of its address.



Processor

The processor (CPU)

- executes instructions in machine language
- has an own "fast" memory (registers)
- can read from and write to main memory
- features a set of simplest operations = instructions (e.g. adding to register values)

Computing speed

In the time, onaverage, that the sound takes to travel from from my mouth to you ...

30 m ≙ more than 100.000.000 instructions

a contemporary desktop PC can process more than 100 millions instructions [1]

[1] Uniprocessor computer at 1 GHz.

Programming

- With a **programming language** we issue commands to a computer such that it does exactly what we want.
- The sequence of instructions is the **(computer) program**

The Harvard Computers, human computers, ca.1890

http://en.wikipedia.org/wiki/Harvard_Computers

Programming Languages

- The language that the computer can understand (machine language) is very primitive.
- Simple operations have to be disassembled into many single steps
- The machine language varies between computers.

Higher Programming Languages

can be represented as program text that

- can be *understood* by humans
- is *independent* of the computer model

  → Abstraction!

Java

- is based on a **virtual machine** (with von-Neumann architecture)
  - Program code is translated into intermediate code
  - Intermediate code runs in a simulated computing envrionment, the intermediate code is executed by an interpreted
  - Optimisation: Just-In-Time (JIT) compilation of frequently used code: virtual machine → physical machine
- Consequence, and manifested goal of the Java developers: **portability**

  write once – run anywhere

2. Introduction to Java

Programming – a first Java Program

Programming Tools

- **Editor:** Program to modify, edit and store Java program texts
- **Compiler:** program to translate a program text into machine language
- **Computer:** machine to execute machine language programs
- **Operating System:** program to organize all procedures such as file handling, editor-, compiler- and program execution.

German vs. Programming Language

Deutsch

Es ist nicht genug zu wissen,
man muss auch anwenden.
(Johann Wolfgang von Goethe)

Java / C / C++

```
// computation
int b = a * a; // b = a^2
b = b * b;     // b = a^4
```


Syntax and Semantics

- Like our language, programs have to be formed according to certain rules.
  - **Syntax**: Connection rules for elementary symbols (characters)
  - **Semantics**: interpretation rules for connected symbols.
- Corresponding rules for a computer program are simpler but also more strict because computers are relatively stupid.

Syntax and Semantics of Java

Syntax

- What *is* a Java program?
- Is it *grammatically* correct?

Semantics

- What does a program *mean*?
- What kind of algorithm does a program implement?

First Java Program

```
// Program to raise a number to the eighth power
public class Main {
  public static void main(String[] args) {
    // input
    Out.print("Compute a^8 for a= ?");
    int a;
    a = In.readInt();
    // computation
    int b = a * a; // b = a^2
    b = b * b; // b = a^4
    // output b * b, i.e. a^8
    Out.println(a + "^8 = " + b * b);
  }
}
```

← **Class**: a program (at `public class Main {`)

← **Method**: named sequence of statements. (at `public static void main(String[] args) {`)

Java Classes

A Java program comprises at least one class with main-method. The sequence of statements in this method is executed when the program starts.

```
public class Main{
  // Potentiell weiterer Code und Daten

  public static void main(String[] args) {
    // Hier beginnt die Ausfuehrung
    ...
  }

}
```


Behavior of a Program

At compile time:

- program accepted by the compiler (syntactically correct)
- Compiler error

During runtime:

- correct result
- incorrect result
- program crashes
- program does not terminate (endless loop)

Comments

```
// Program to raise a number to the eighth power
public class Main {
  public static void main(String[] args) {
    // input
    Out.print("Compute a^8 for a= ?");
    int a;
    a = In.readInt();
    // computation
    int b = a * a;  // b = a^2
    b = b * b;  // b = a^4
    // output b * b, i.e. a^8
    Out.println(a + "^8 = " + b * b);
  }
}
```

Kommentare

Comments and Layout

Comments

- are contained in every good program.
- document *what* and *how* a program does something and how it should be used,
- are ignored by the compiler
- Syntax: “double slash” `// until the line ends.`

The compiler *ignores* additionally

- Empty lines, spaces,
- Indendations that should reflect the program logic

Comments and Layout

The compiler does not care...

```
public class Main{public static void main(String[] args){Out.print
("Compute a^8 for a= ?");int a;a = In.readInt();int b = a*a;b =
b * b;Out.println(a + "^8 = " + b * b);}}
```

... but we do!

Statements

```
// Program to raise a number to the eighth power
public class Main {
  public static void main(String[] args) {
    // input
    Out.print("Compute a^8 for a= ?");
    int a;
    a = In.readInt();
    // computation
    int b = a * a;  // b = a^2
    b = b * b; // b = a^4
    // output b * b, i.e. a^8
    Out.println(a + "^8 = " + b * b);
  }
}
```

Ausdrucksanweisungen

Statements

- building blocks of a Java program
- are *executed* (sequentially)
- end with a semicolon
- Any statement provide an **effect** (potentially)

Expression Statements

- have the following form:

  expr;

  where *expr* is an expression
- Effect is the effect of *expr*, the value of *expr* is ignored.

```
b = b * b;
```

Statements – Values and Effects

```
// Program to raise a number to the eighth power
public class Main {
  public static void main(String[] args) {
    // input
    Out.print("Compute a^8 for a= ?");
    int a;
    a = In.readInt();
    // computation
    int b = a * a; // b = a^2
    b = b * b; // b = a^4
    // output b * b, i.e. a^8
    Out.println(a + "^8 = " + b * b);
  }
}
```

Effekt: Ausgabe des Strings `Compute ...`

Effekt: Eingabe einer Zahl und Speichern in a

Effekt: Speichern des berechneten **Wertes** von a*a in b

Effekt: Speichern des berechneten **Wertes** von b * b in b

Effekt: Ausgabe des **Wertes** von a und des berechneten **Wertes** von b * b

Values and Effects

- determine what a program does,
- are purely semantical concepts:
  - Symbol `0` means Value $0 \in \mathbb{Z}$
  - `a = In.readInt();` means effect "read in a number"
- depend on the program state (memory content, inputs)

Variable Definitions

```
// Program to raise a number to the eighth power
public class Main {
  public static void main(String[] args) {
    // input
    Out.print("Compute a^8 for a= ?");
    int a;
    a = In.readInt();
    // computation
    int b = a * a; // b = a^2
    b = b * b;  // b = a^4
    // output b * b, i.e. a^8
    Out.println(a + "^8 = " + b * b);
  }
}
```

Deklarationsanweisungen

Typ-namen

Declaration Statements

- introduce new names in the program,
- consist of declaration and semicolon

```
int a;
```

- can initialize variables

```
int b = a * a;
```

Types and Functionality

`int`:

- Java integer type
- corresponds to $(\mathbb{Z}, +, \times)$ in math

In Java each type has a name and

- a domain (e.g. integers)
- functionality (e.g. addition/multiplication)

Fundamental Types

Java comprises fundamental types for

- integers (`int`)
- real numbers (`float`, `double`)
- boolean values (`boolean`)
- ...

Literals

- represent constant values
- have a fixed **type** and **value**
- are "syntactical values".

- `0` has type `int`, value 0.
- `1.2e5` has type `double`, value $1.2 \cdot 10^5$.

Variables

- represent (varying) values,
- have
  - **name**
  - **type**
  - **value**
  - **address**
- are "visible" in the program context.

`int a;` defines a variable with

- name: `a`
- type: `int`
- value: (initially) undefined
- Address: determined by compiler

Objects

- represent values in main memory
- have **type**, **address** and **value** (memory content at the address)
- can be named (variable) ...
- ... but also anonymous.

Remarks

A program has a *fixed* number of variables. In order to be able to deal with a variable number of value, it requires "anonymous" addresses that can be address via temporary names.

Identifiers and Names

(Variable-)names are identifiers

- allowed: A,...,Z; a,...,z; 0,...,9;_
- First symbol needs to be a character.

There are more names:

- `Out.println` (Qualified identifier)

Expressions

- represent *Computations*
- are either **primary** (`b`)
- or **composed** (`b * b`)...
- ...from different expressions by **operators**

Analogy: building blocks

Expressions

```
// input
Out.print("Compute a^8 for a= ?");
int a;
a = In.readInt();

// computation
int b = a * a; // b = a^2
b = b * b;    // b = a^4

// output b * b, i.e. a^8
Out.println(a + "^8 = " + b * b );
```

Expressions

- represent *computations*
- are *primary* or *composite* (by other expressions and operations)

> `a * a`
> composed of
> variable name, operator symbol,variable name
> variable name: primary expression

- can be put into parantheses

> `a * a` is equivalent to `(a * a)`

Expressions

have **type**, **value** und **effect** (potentially).

Example `a * a`

- type: `int` (type of the operands)
- Value: product of `a` and `a`
- Effect: none.

Example `b = b * b`

- type: `int` (Typ der Operanden)
- Value: product of `b` and `b`
- effect: assignment of the product value to `b`

The type of an expression is fixed but the value and effect are only determined by the *evaluation* of the expression

Operators and Operands

```
// input
Out.print("Compute a^8 for a= ?");
int a;
a = In.readInt();

// computation
int b = a * a; // b = a^2
b = b * b;     // b = a^4

// ou          8
Out.println(a + "^8 = " + b * b );
```

left operand (variable)

right operand (expression)

assignment operator

multiplication operator

Operators

Operators

- make expressions (*operands*) into new composed expressions
- specify the required and resulting types for the operands and the result
- have an arity

Multiplication Operator *

- expects to R-values of the same type as operands (arity 2)
- "returns the product as value of the same type", that means formally:
  - The composite expression is value of the product of the value of the two operands

Examples: `a * a` and `b * b`

Assignment Operator =

- Assigns to the left operand the value of the right operand and returns the left operand

Examples: `b = b * b` and `a = b`

Attention, Trap!

The operator `=` corresponds to the assignment operator of mathematics (:=), not to the comparison operator (=).