Informatik I

Vorlesung am D-BAUG der ETH Zürich

Hermann Lehner, Felix Friedrich
ETH Zürich

HS 2017

1. Einführung

Willkommen zur Vorlesung !

Material

Vorlesungshomepage:

http://lec.inf.ethz.ch/baug/informatik1

Das Team

Dozenten	Hermann Lehner	
	Felix Friedrich	
Chef-Assistent	Andrea Lattuada	
Assistenten	Malte Schwerhoff	
	Rafael Wampfler	Gökcen Cimen
	Kai Sandbrink	Patrick Gruntz
	Irfan Bunjaku	Simon Guldimann
	Clemens Bachmann	Frederic Vogel
	Sander Staal	Philipp Schimmelfennig
	Nihat Isik	

Programmieren und Problemlösen

In diesem Kurs “lernen” Sie programmieren in Java

- Die Software Entwicklung ist ein *Handwerk*.
- Vergleich: Erlernen eines Musikinstruments.
- **Das Problem:** Es ist noch keiner vom Zuhören Pianist geworden.

Deshalb bietet dieser Kurs Ihnen viele Möglichkeiten, zu üben.
Nutzen Sie dies aus!

Programmieren und Problemlösen

In diesem Kurs *lernen* Sie Problemlösen mit ausgewählten Algorithmen und Datenstrukturen.

- Sprach-übergreifendes *Grundlagenwissen*
- Vergleich: Rhythmus-Lehre, Tonleitern, Noten-Lesen.
- **Das Problem:** Ohne Musikinstrument macht dies kein Spass.

Deshalb kombinieren wir das Problemlösen mit dem Erlernen von Java.

Inhalte der Vorlesung

Programmieren mit Java

Einführung
Anweisungen und Ausdrücke
Zahlendarstellungen
Kontrollfluss
Arrays
Methoden und Rekursion
Typen, Klassen und Objekte
Vererbung und Polymorphie

Matlab

Einführung
Anwendungsszenarien

Ziel der *heutigen* Vorlesung

- Einführung *Computermodell* und Algorithmus
- Allgemeine Informationen zur Vorlesung
- Das *erste Programm* schreiben

1.1 Informatik und Algorithmus

Informatik, der Euklidische Algorithmus

Was ist Informatik?

- Die Wissenschaft der **systematischen Verarbeitung von Informationen**,. . .
- . . . insbesondere der automatischen Verarbeitung mit Hilfe von Digitalrechnern.

(Wikipedia, nach dem “Duden Informatik”)

Informatik $\neq$ EDV-Kenntnisse

EDV-Kenntnisse: *Anwenderwissen*

- Umgang mit dem Computer
- Bedienung von Computerprogrammen (für Texterfassung, Email, Präsentationen,. . .)

Informatik: *Grundlagenwissen*

- Wie funktioniert ein Computer?
- Wie schreibt man ein Computerprogramm?

Inhalt dieser Vorlesung

- Systematisches Problemlösen mit Algorithmen und der Programmiersprache Java.
- Also:
  nicht nur,
  aber auch Programmierkurs.

Algorithmus: Kernbegriff der Informatik

Algorithmus:

- Handlungsanweisung zur schrittweisen Lösung eines Problems
- Ausführung erfordert keine Intelligenz, nur Genauigkeit (sogar Computer können es)
- nach *Muhammed al-Chwarizmi*, Autor eines arabischen Rechen-Lehrbuchs (um 825)

"Dixit algorizmi..." (lateinische Übersetzung)

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

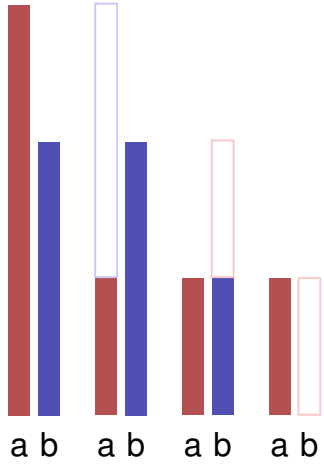
Der älteste nichttriviale Algorithmus

Euklidischer Algorithmus (aus Euklids *Elementen*, 3. Jh. v. Chr.)

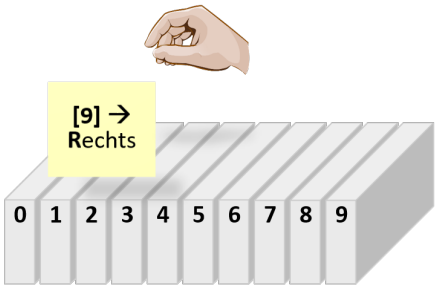
- Eingabe: ganze Zahlen $a > 0$, $b > 0$
- Ausgabe: ggT von a und b

Solange $b \neq 0$
  Wenn $a > b$ dann
    $a \leftarrow a - b$
  Sonst:
    $b \leftarrow b - a$

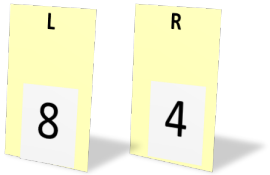
Ergebnis: a.



Live Demo: Turing Maschine



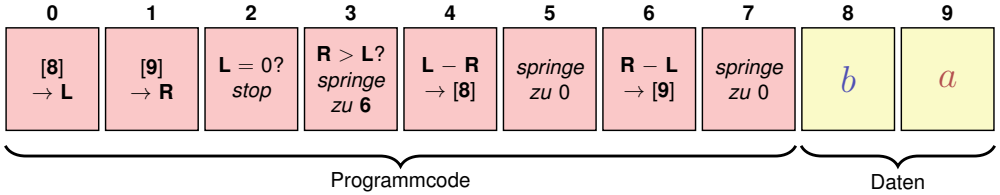
Speicher

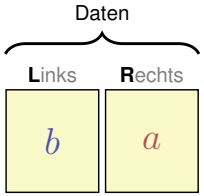


Register

Euklid in der Box

Speicher





Register

Solange $b \neq 0$
  Wenn $a > b$ dann
    $a \leftarrow a - b$
  Sonst:
    $b \leftarrow b - a$

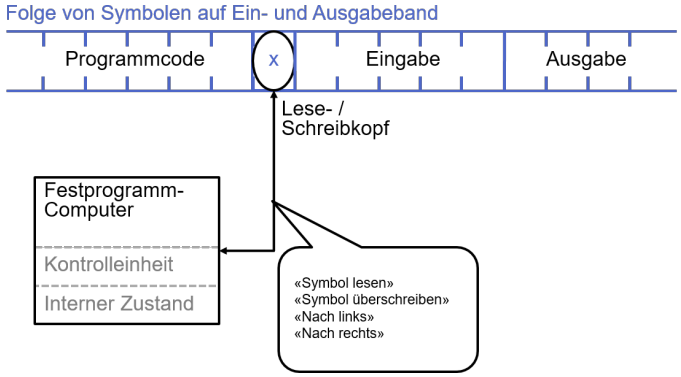
Ergebnis: a.

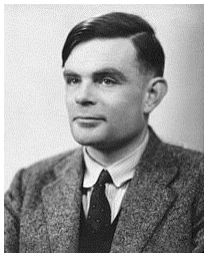
1.3 Computermodell

Turing Maschine, Von Neumann Architektur

Computer – Konzept

Eine geniale Idee: Universelle Turingmaschine (Alan Turing, 1936)





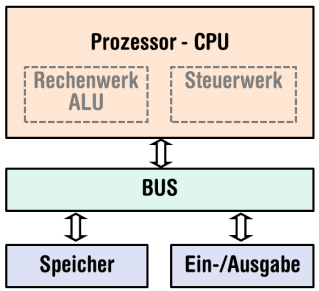
Alan Turing

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

Computer – Umsetzung

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

Von Neumann Architektur



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

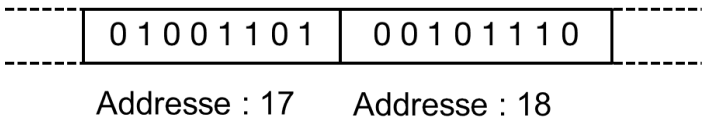
Computer

Zutaten der *Von Neumann Architektur:*

- Hauptspeicher (RAM) für Programme *und* Daten
- Prozessor (CPU) zur Verarbeitung der Programme und Daten
- I/O Komponenten zur Kommunikation mit der Aussenwelt

Speicher für Daten *und* Programm

- Folge von Bits aus $\{0, 1\}$.
- Programmzustand: Werte aller Bits.
- Zusammenfassung von Bits zu Speicherzellen (oft: 8 Bits = 1 Byte).
- Jede Speicherzelle hat eine Adresse.
- Random Access: Zugriffszeit auf Speicherzelle (nahezu) unabhängig von ihrer Adresse.



Prozessor

Der Prozessor (CPU)

- führt Befehle in Maschinensprache aus
- hat eigenen "schnellen" Speicher (Register)
- kann vom Hauptspeicher lesen und in ihn schreiben
- beherrscht eine Menge einfachster Operationen (z.B. Addieren zweier Registerinhalte)

Rechengeschwindigkeit

In der mittleren Zeit, die der Schall von mir zu Ihnen unterwegs ist...

30 m $\widehat{=}$ mehr als $100.000.000$ Instruktionen

arbeitet ein heutiger Desktop-PC mehr als 100 Millionen Instruktionen ab.[1]

[1] Uniprozessor Computer bei 1GHz

Programmieren

- Mit Hilfe einer *Programmiersprache* wird dem Computer eine Folge von Befehlen erteilt, damit er genau das macht, was wir wollen.
- Die Folge von Befehlen ist das *(Computer)-Programm*.

The Harvard Computers, Menschliche Berufsrechner, ca.1890

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

Warum Programmieren?

- Da hätte ich ja gleich Informatik studieren können ...
- Es gibt doch schon für alles Programme ...
- Programmieren interessiert mich nicht ...
- Weil Informatik hier leider ein Pflichtfach ist ...
- ...

Mathematik war früher die Lingua franca der Naturwissenschaften an allen Hochschulen. Und heute ist dies die Informatik.

Lino Guzzella, Präsident der ETH Zürich, NZZ Online, 1.9.2017

Darum Programmieren!

- Jedes Verständnis moderner Technologie erfordert Wissen über die grundlegende Funktionsweise eines Computers.
- Programmieren (mit dem Werkzeug Computer) wird zu einer Kulturtechnik wie Lesen und Schreiben (mit den Werkzeugen Papier und Bleistift)
- Die meisten qualifizierten Jobs benötigen zumindest elementare Programmierkenntnisse.
- Programmieren macht Spass!

Dieser Kurs ist für Sie

- Sie erlernen das *Fundament* – die Grundlagen der Informatik und des Programmierens – bei uns auf einem anspruchsvollen Niveau.
- Sie müssen die erlernten *Prinzipien* später in einem anderen Kontext – unter anderem für andere Programmiersprachen (C++, Python ,Matlab ,R) – *anwenden können*.
- Das ist keine Vorgabe von uns – wir wissen das von *Ihnen* (= Ihrem Departement).

Programmiersprachen

- Sprache, die der Computer "versteht", ist sehr primitiv (Maschinensprache).
- Einfache Operationen müssen in viele Einzelschritte aufgeteilt werden.
- Sprache variiert von Computer zu Computer.

Höhere Programmiersprachen

darstellbar als Programmtext, der

- von Menschen *verstanden* werden kann
- vom Computermodell *unabhängig* ist
  $\rightarrow$ Abstraktion!

Java

- Basiert auf einer *virtuellen Maschine* (mit von-Neumann Architektur)
  - Programmcode wird in Zwischencode übersetzt
  - Zwischencode läuft in einer simulierten Rechnerumgebung, Interpretation des Zwischencodes durch einen Interpreter
  - Optimierung: Just-In-Time (JIT) Kompilation von häufig genutztem Code: virtuelle Maschine $\rightarrow$ physikalische Maschine
- Folgerung, und erklärtes Ziel der Entwickler von Java: Portabilität

  write once – run anywhere

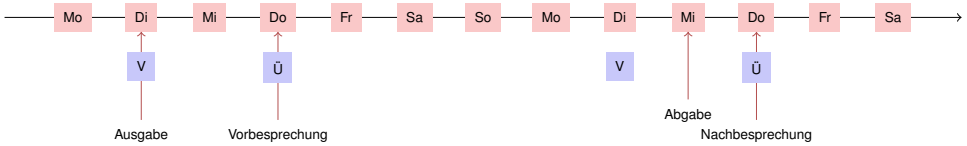
1.5 Allgemeine Informationen zur Vorlesung

Organisatorisches, Tools, Übungen, Prüfung

Einschreibung in Übungsgruppen

- Gruppeneinteilung selbstständig via Webseite `http://echo.ethz.ch`
- Funktioniert nach Belegung dieser Vorlesung in myStudies.
- Die gezeigten Räume und Termine abhängig vom Studiengang.

Übungsbetrieb



- Übungsblattausgabe zur Vorlesung (online).
- Vorbesprechung in der folgenden Übung.
- Bearbeitung der Übung bis spätestens am Tag vor der nächsten Übungsstunde (23:59h).
- Nachbesprechung der Übung in der nächsten Übungsstunde. Feeback zu den Abgaben innerhalb einer Woche nach Nachbesprechung.

Zu den Übungen

- An der ETH ist (seit HS 2013) für die Prüfungszulassung kein Testat erforderlich.
- Bearbeitung der wöchentlichen Uebungsserien ist also freiwillig, wird aber *dringend* empfohlen!

Fehlende Resourcen sind keine Entschuldigung!

Für die Übungen verwenden wir eine Online-Entwicklungsumgebung, benötigt lediglich einen Browser, Internetverbindung und Ihr ETH Login.

Falls Sie keinen Zugang zu einem Computer haben: in der ETH stehen an vielen Orten öffentlich Computer bereit.

Tutorial

In der ersten Woche bearbeiten Sie selbständig unser *Java-Tutorial*

- Einfacher Einstieg in Java, kein Vorwissen nötig!
- Zeitbedarf: ca. zwei Stunden
- In der zweiten Woche gibt's ein *Self Assessment* zum Tutorial
- Auch das Tutorial ist basierend auf **Codeboard.io**

→ Das ist gut investierte Zeit!

Tutorial - Url

Java Tutorial

Hier finden Sie das Tutorial

`https://frontend-1.et.ethz.ch/sc/WKrEKYAuHvaeTqLzr`

Buch zur Vorlesung

Sprechen Sie Java?

Hanspeter Mössenböck

dpunkt.verlag

- Gut aufgebautes Lernmaterial
- Vertiefte Diskussion der Themen
- Übungsaufgaben mit Lösungen

- *An der Prüfung werden wir 1-2 Aufgaben aus dem Buch bringen*

Relevantes für die Prüfung

Prüfungsstoff für die Endprüfung (in der Prüfungssession 2018) schliesst ein

- Vorlesungsinhalt (Vorlesung, Handout) und
- Übungsinhalte (Übungsstunden, Übungsaufgaben).

Prüfung ist schriftlich

Es wird sowohl praktisches Wissen (Programmierfähigkeit[2]) als auch theoretisches Wissen (Hintergründe, Systematik) geprüft.

[2] soweit in schriftlicher Prüfung möglich

Unser Angebot

- Ihre Programmierübungen werden (halb)automatisch bewertet. Durch Bearbeitung der wöchentlichen Übungsserien kann ein Bonus von maximal 0.25 Notenpunkten erarbeitet werden, der an die Prüfung mitgenommen wird.
- Der Bonus ist proportional zur erreichten Punktzahl über alle Serien, wobei volle Punktzahl einem Bonus von 0.25 entspricht.

Akademische Lauterkeit

Regel: Sie geben nur eigene Lösungen ab, welche Sie selbst verfasst und verstanden haben.

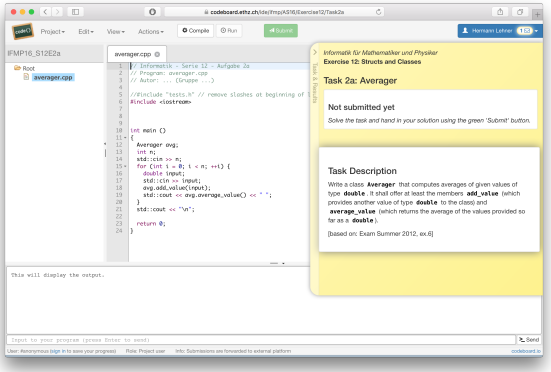
Wir prüfen das (zum Teil automatisiert) nach und behalten uns insbesondere mündliche Prüfgespräche vor.

Sollten Sie zu einem Gespräche eingeladen werden: geraten Sie nicht in Panik. Es gilt primär die Unschuldsvermutung. Wir wollen wissen, ob Sie verstanden haben, was Sie abgegeben haben.

Codeboard

Codeboard ist eine Online-IDE: Programmieren im Browser!

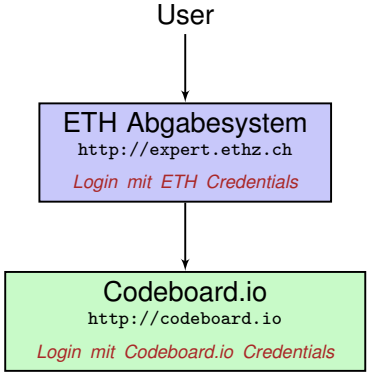
- Falls vorhanden, bringen Sie Ihren Laptop/Tablet/... mit in den Unterricht.
- Sie können direkt in der Vorlesung Beispiele ausprobieren, ohne dass Sie irgendwelche Tools installieren müssen.



Expert

Unser Übungssystem besteht aus zwei unabhängigen Systemen, die miteinander kommunizieren:

- **Das ETH Abgabesystem:** Ermöglicht es uns, Ihre Aufgaben zu bewerten
- **Die Online IDE:** Die Programmierumgebung



Übungseinschreibung

Codeboard.io Registrierung

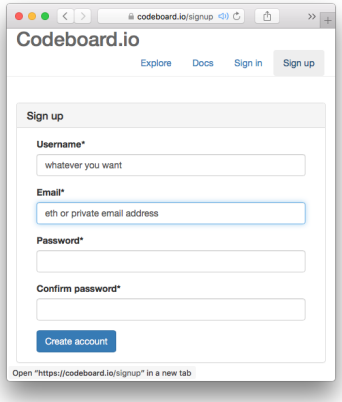
Gehen Sie auf `http://codeboard.io` und erstellen Sie dort ein Konto, bleiben Sie am besten eingeloggt.

Einschreibung in Übungsgruppen

Gehen Sie auf `http://expert.ethz.ch/baugi1_2017e00t01` und schreiben Sie sich dort in eine Übungsgruppe ein.

Codeboard.io Registrierung

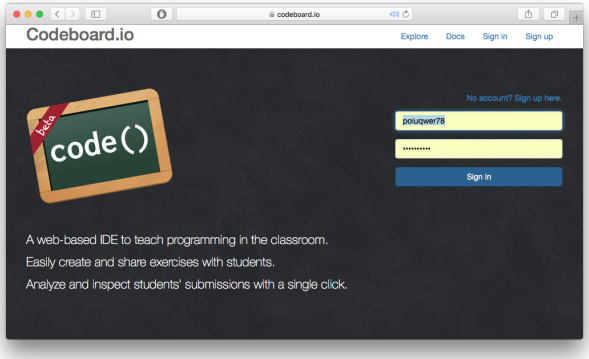
Falls Sie noch keinen **Codeboard.io** Account haben ...



- Wir verwenden die Online IDE **Codeboard.io**
- Erstellen Sie dort einen Account, um Ihren Fortschritt abzuspeichern und später Submissions anzuschauen
- Anmeldedaten können beliebig gewählt werden! *Verwenden Sie nicht das ETH Passwort.*

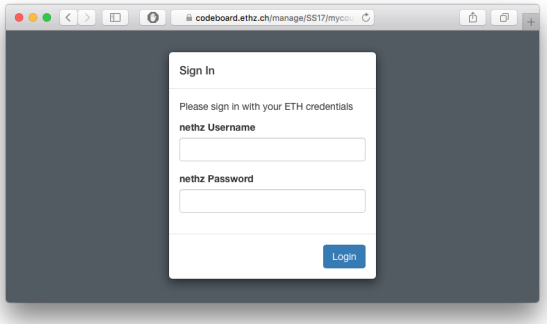
Codeboard.io Login

Falls Sie schon einen Account haben, loggen Sie sich ein:



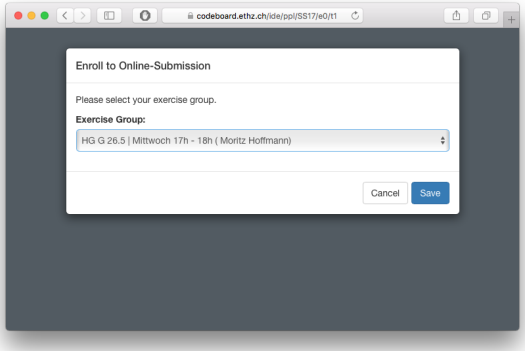
Einschreibung in Übungsgruppen - I

- Besuchen Sie `http://expert.ethz.ch/baugi1_2017e00t01`
- Loggen Sie sich mit Ihrem nethz Account ein.



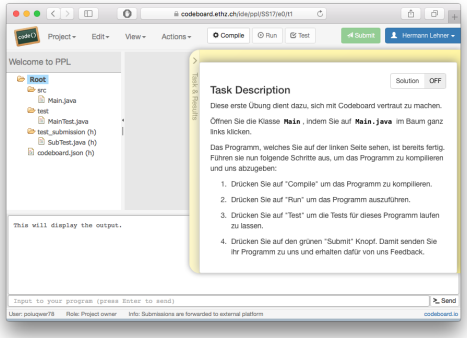
Einschreibung in Übungsgruppen - II

Schreiben Sie sich in diesem Dialog in eine Übungsgruppe ein.



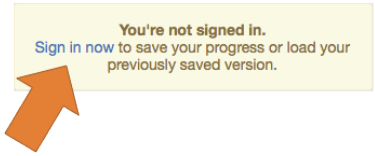
Die erste Übung

Sie sind nun eingeschrieben und die erste Übung ist geladen. Folgen Sie den Anweisungen in der gelben Box.



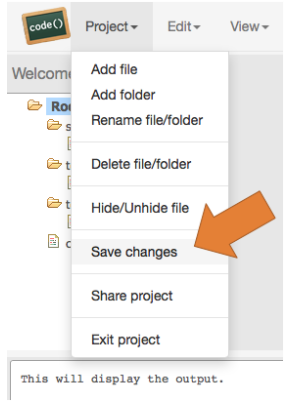
Die erste Übung - Codeboard.io Login

Achtung! Falls Sie diese Nachricht sehen, klicken Sie auf Sign in now und melden Sie sich dort mit Ihrem **Codeboard.io** Account ein.



Die erste Übung - Fortschritt speichern!

Achtung! Speichern Sie Ihren Fortschritt regelmässig ab. So können Sie jederzeit an einem anderen Ort weiterarbeiten.



2. Java Einführung

Programmieren – Ein erstes Java Programm

Was braucht es zum Programmieren?

- **Editor:** Programm zum Ändern, Erfassen und Speichern vom Java-Programmtext
- **Compiler:** Programm zum Übersetzen des Programmtexts in Maschinensprache
- **Computer:** Gerät zum Ausführen von Programmen in Maschinensprache
- **Betriebssystem:** Programm zur Organisation all dieser Abläufe (Dateiverwaltung, Editor-, Compiler- und Programmaufruf)

Deutsch vs. Programmiersprache

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 und Semantik

- Programme müssen, wie unsere Sprache, nach gewissen Regeln geformt werden.
  - Syntax: Zusammenfügungsregeln für elementare Zeichen (Buchstaben).
  - Semantik: Interpretationsregeln für zusammengefügte Zeichen.
- Entsprechende Regeln für ein Computerprogramm sind einfacher, aber auch strenger, denn Computer sind vergleichsweise dumm.

Syntax und Semantik von Java

Syntax

- Was *ist* ein Java Programm?
- Ist es *grammatikalisch* korrekt?

Semantik

- Was *bedeutet* ein Programm?
- Welchen Algorithmus realisiert ein Programm?

Erstes Java Programm

```
// 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);
  }
}
```

Klasse: Ein Programm

Methode: benannte Folge von Anweisungen.

Java Klassen

Ein Java-Programm besteht aus mindestens einer Klasse mit einer main-Methode. Die Folge von Anweisungen in dieser Methode wird ausgeführt, wenn das Programm startet.

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

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


Verhalten eines Programmes

Zur Compilationszeit:

- vom Compiler akzeptiertes Programm (syntaktisch korrektes Java)
- Compiler-Fehler

Zur Laufzeit:

- korrektes Resultat
- inkorrektes Resultat
- Programmabsturz
- Programm *terminiert* nicht (Endlosschleife)

Kommentare

```
// 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

Kommentare und Layout

Kommentare

- hat jedes gute Programm,
- dokumentieren, *was* das Programm *wie* macht und wie man es verwendet und
- werden vom Compiler ignoriert.
- Syntax: “Doppelslash” `// bis Zeilenende.`

Ignoriert werden vom Compiler ausserdem

- Leerzeilen, Leerzeichen,
- Einrückungen, die die Programmlogik widerspiegeln (sollten)

Kommentare und Layout

Dem Compiler ist’s egal...

```
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);}}
```

... uns aber nicht!

Anweisungen (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

Anweisungen (statements)

- Bausteine eines Java Programms
- werden (sequenziell) *ausgeführt*
- enden mit einem Semikolon
- Jede Anweisung hat (potenziell) einen *Effekt*.

Ausdrucksanweisungen

- haben die Form

  expr;

  wobei *expr* ein Ausdruck ist
- Effekt ist der Effekt von *expr*, der Wert von *expr* wird ignoriert.

Beispiel: `b = b*b;`

Anweisungen – Werte und Effekte

```
// 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

Werte und Effekte

- bestimmen, was das Programm macht,
- sind rein semantische Konzepte:
  - Zeichen 0 bedeutet Wert $0 \in \mathbb{Z}$
  - `a = In.readInt();` bedeutet Effekt "Einlesen einer Zahl"
- hängen vom Programmzustand (Speicherinhalte / Eingaben) ab

Anweisungen – Variablendefinitionen

```
// 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

Deklarationsanweisungen

- führen neue Namen im Programm ein,
- bestehen aus Deklaration + Semikolon

  Beispiel: `int a;`
- können Variablen auch initialisieren

  Beispiel: `int b = a * a;`

Typen und Funktionalität

`int`:

- Java Typ für ganze Zahlen,
- entspricht $(\mathbb{Z}, +, \times)$ in der Mathematik

In Java hat jeder Typ einen Namen sowie

- Wertebereich (z.B. ganze Zahlen)
- Funktionalität (z.B. Addition/Multiplikation)

Fundamentaltypen

Java enthält fundamentale Typen für

- Ganze Zahlen (`int`)
- Reelle Zahlen (`float`, `double`)
- Wahrheitswerte (`boolean`)
- ...

Literale

- repräsentieren konstante Werte,
- haben festen *Typ* und *Wert*
- sind "syntaktische Werte".

Beispiele:

- `0` hat Typ `int`, Wert 0.
- `1.2e5` hat Typ `double`, transWertvalue $1.2 \cdot 10^5$.

Variablen

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

Beispiel

`int a;` definiert Variable mit

- Name: `a`
- Typ: `int`
- Wert: (vorerst) undefiniert
- Addresse: durch Compiler bestimmt

Objekte

- repräsentieren Werte im Hauptspeicher
- haben *Typ*, *Adresse* und *Wert* (Speicherinhalt an der Adresse),
- können benannt werden (Variable) ...
- ... aber auch anonym sein.

Anmerkung

Ein Programm hat eine *feste* Anzahl von Variablen. Um eine variable Anzahl von Werten behandeln zu können, braucht es "anonyme" Adressen, die über temporäre Namen angesprochen werden können.

Bezeichner und Namen

(Variablen-)Namen sind Bezeichner:

- erlaubt: A,...,Z; a,...,z; 0,...,9;_
- erstes Zeichen ist Buchstabe.

Es gibt noch andere Namen:

- `Out.println` (qualifizierter Name)

Ausdrücke (Expressions)

- repräsentieren *Berechnungen*
- sind entweder primär (`b`)
- oder zusammengesetzt (`b*b`)...
- ...aus anderen Ausdrücken, mit Hilfe von Operatoren

Analogie: Baukasten

Ausdrücke (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| ); ||
```


Ausdrücke (Expressions)

- repräsentieren *Berechnungen*,
- sind *primär* oder *zusammengesetzt* (aus anderen Ausdrücken und Operationen)

`a * a`
zusammengesetzt aus
Variablenname, Operatorsymbol, Variablenname
Variablenname: primärer Ausdruck

- können geklammert werden

`a * a` ist äquivalent zu `(a * a)`

Ausdrücke (Expressions)

haben *Typ*, *Wert* und *Effekt* (potenziell).

Beispiel

`a * a`

- Typ: `int` (Typ der Operanden)
- Wert: Produkt von `a` und `a`
- Effekt: keiner.

Beispiel

`b = b * b`

- Typ: `int` (Typ der Operanden)
- Wert: Produkt von `b` und `b`
- Effekt: Weise `b` diesen Wert zu.

Typ eines Ausdrucks ist fest, aber Wert und Effekt werden erst durch die *Auswertung* des Ausdrucks bestimmt.

Operatoren und Operanden

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

Linker Operand (Variable)

Rechter Operand (Ausdruck)

Zuweisungsoperator

Multiplikationsoperator

Operatoren

Operatoren

- machen aus Ausdrücken (*Operanden*) neue zusammengesetzte Ausdrücke
- spezifizieren für die Operanden und das Ergebnis die Typen
- haben eine Stelligkeit

Multiplikationsoperator *

- erwartet zwei R-Werte vom gleichen Typ als Operanden (Stelligkeit 2)
- "gibt Produkt als Wert des gleichen Typs zurück", das heisst formal:
  - Der zusammengesetzte Ausdruck repräsentiert den Wert des Produktes der Werte der beiden Operanden

Beispiele: `a * a` und `b * b`

Zuweisungsoperator =

- Weist linkem Operanden den Wert des rechten Operanden zu und gibt den linken Operanden zurück

Beispiele: `b = b * b` und `a = b`

Vorsicht, Falle!

Der Operator = entspricht dem Zuweisungsoperator in der Mathematik (:=), nicht dem Vergleichsoperator (=).