Einführung in die Informatik: Programmierung und Softwaretechnik

Martin Hofmann

Institut für Informatik

LMU München

Wintersemester 2002/03

Organisatorisches

- Vorlesung: Mi 14 - 17 Uhr, Audi. max.
- Übungen:

  Do. 14-16, HS 214 Hauptgeb.

  Do. 16-18, HS 214 Hauptgeb.

  Fr. 10-12, HS 214 Hauptgeb. Fr. 16-18, HS 214 Hauptgeb. (Förderübung)
- Bücher zur Vorlesung:
  - Cay Horstmann, Computing Concepts with Java 2 Essentials. 2. Aufl. Wiley 2000. €55,-
  - H.-P. Gumm und M. Sommer. Einführung in die Informatik, 5. Aufl., Oldenbourg, 2002. €39,80

Organisatorisches

- Klausur: 6.2.2003, 14-17 Uhr, Audi. Max.

  Voraussetzung: Sinnvolle Bearbeitung der Programmieraufgaben.
- Programmieraufgaben ("Practicals"):
  - 4 Stück insgesamt.
  - Bearbeitungszeit jeweils ca. zwei Wochen.
  - Abgabe elektronisch über WWW-Seite.
  - Dürfen alleine, zu zweit, oder zu dritt bearbeitet und abgegeben werden. Practical 0 nur alleine.

Organisatorisches

- WWW-Seite: `http://www.tcs.informatik.uni-muenchen.de/lehre/WS02-03/PSE`
- Tutorenpräsenz im CIP-Pool:

  Mo. 9-13, 14-20

  Di. 9-13, 14:30-18:30

  Mi. 9-13

  Do. 12-14

  Fr. 11:30-15:30

Organisatorisches

- Rechner befinden sich in Kellerräumen der Oettingenstraße 67, sowie in den dortigen Baracken.
- Anmeldung für Rechnerkennung: Raum 003/Arktis. 16.10.-22.10. 18-19Uhr.

$$\left.\begin{array}{l}\text{Mittwoch (16.10.): S-Z}\\ \text{Donnerstag (17.10.): A-P}\\ \text{Freitag (18.10.): Q-Z}\\ \text{Montag (21.10.): A-M}\\ \text{Dienstag (22.10.): N-Z}\end{array}\right\}\text{Studentenausweis u. Lichtbildausweis mitbringen.}$$

- Programmieraufgaben können auch zuhause bearbeitet werden. Links zu Java-Umgebung etc. auf der WWW-Seite.

Was ist ein Computer

Maschine, die Rechenoperationen programmgesteuert abarbeitet.

Typische Operationen sind:

- Addiere zwei Zahlen
- Schreibe das Ergebnis der letzten Operation an eine bestimmte Speicherstelle
- Mache einen bestimmten Bildschirmpunkt rot
- Schicke den Buchstaben `A` an den Drucker
- Hole den Inhalt einer bestimmten Speicherstelle.
- Falls das Ergebnis der letzten Operation negativ war, so führe das Programm an dieser Stelle fort, andernfalls an jener.

Programmieren bedeutet, eine Befehlsfolge so festzulegen, dass eine bestimmte Aufgabe ausgeführt wird.

Aufbau eines Computers

Der Prozessor (Central Processing Unit, CPU) hat Zugriff auf Operanden und den Speicher (beim i386 2^{32} Speicherstellen).

Jede Speicherstelle enthält eine Zahl zwischen 0 und $2^{32}-1$.

Die CPU kann Werte aus dem Speicher holen, in den Speicher schreiben, durch elementare Rechenoperationen verarbeiten und den Programmablauf an einer bestimmten Speicherstelle fortführen.

Nach dem Einschalten holt die CPU den Inhalt der Speicherstelle 0, interpretiert ihn als Befehl und führt ihn aus. Sodann den Inhalt der Speicherstelle 1, usw.

Aufbau eines Computers

Physikalisch verbirgt sich hinter den Speicherstellen ROM (read-only memory), RAM (random access memory), Peripheriegeräte (Festplatte, Bildschirm, Tastatur, Drucker, etc.).

Die ersten Speicherstellen sind durch ROM belegt und enthalten ein festes Programm (BIOS).

Das BIOS liest u.a. den Bootsektor der Festplatte, schreibt das dort liegende Programm in den RAM und setzt die Ausführung dort fort.

Beispielprogramm

1. Hole den Inhalt der Speicherstelle 40
2. Hole den Wert 100
3. Falls der erste Wert größer als der zweite ist, dann fahre an Speicherstelle 240 fort (ansonsten an der nächsten Speicherstelle).

Diese Befehlssequenz sieht in der Java Virtual Machine so aus:

21 40 16 100 163 240

Im i386 Maschinencode:

41256 47972 14787 3983

Beim PowerPC wiederum anders.

Kurzbezeichnungen für Befehle

Kein Mensch kann und will solche Zahlenfolgen hinschreiben, darum gibt es Kurzbezeichnungen für die Befehle:

In der Java Virtual Machine:

```
iload        40
bipush      100
if_icmpgt   240
```

Im i386 Maschinencode:

```
mov 40,%eax
mov $100,%ebx
cmp %eax,%ebx
jg  240
```

Die Java Virtual Machine gibt es nicht wirklich. Sie wird vom jeweiligen Prozessor durch ein Programm simuliert. Dadurch sind Java Programme plattformunabhängig.

Symbolische Namen

Statt absolute Speicheradressen zu verwenden, kann man symbolische Namen benutzen, die dann durch konkrete Speicheradressen ersetzt werden.

Der Assembler nimmt diese Ersetzungen vor.

```
iload       intRate
bipush      100
if_icmpgt   intError
```


Höhere Programmiersprachen

Höhere Programmiersprachen erleichtern das Schreiben von Programmen weiter durch

- Mechanismen zur Wiederverwendung von Code
- Unterstützung von Modellen oberhalb der Maschinenebene (Funktionen, Objekte, . . .)
- Zugriff auf Programmbibliotheken
- Unterstützung der Programmwartung
- . . .

Beispielprogramm(fragment) in Java

```
if (intRate > 100) System.out.print("Interest rate error");
```

Der Compiler übersetzt das automatisch in Maschinensprache.

Andere Programmiersprachen

C

```
if (intRate > 100) printf("Interest rate error\n'');
```

Pascal

```
if intRate > 100 then writeln('Interest rate error')
```

OCAML

```
if intRate > 100 then print_string "Interest rate error\n";
```

Shellscript

```
if [ $intRate -gt 100 ] ;
then
echo interest Rate error
fi
```


Die Programmiersprache Java

Entstand Anfang der 1990 aus einem Projekt bei Sun Microsystems zur Programmierung elektronischer Geräte (Set top boxen, Waschmaschinen, etc.). Leiter: James Gosling.

Wurde dann zur plattformunabhängigen Ausführung von Programmen in Webseiten ("Applets") verwendet.

Seitdem stark expandiert, mittlerweile neben C++ die beliebteste Sprache.

Die Programmiersprache Java

Vorteile von Java

- Sicherheit
- Portierbarkeit (plattformunabhängig durch JVM)
- (Relativ) sauberes Sprachkonzept (Objektorientierung von Anfang an eingebaut)
- Verfügbarkeit großer Bibliotheken

Nachteile von Java

- Teilweise kompliziert und technisch, da nicht für Studenten entworfen (wie Pascal)
- Zu groß für ein Semester
- Ausführung relativ langsam und speicherplatzintensiv.

Das erste richtige Programm

```
public class Hello
{
    public static void main(String[] args)
    {
        System.out.println("Hello, World!");
    }
}
```

Dieses Programm gibt `Hello, World!` aus.

Das erste richtige Programm

Um es auszuführen müssen wir

- Uns "einloggen" (am Computer anmelden)
- Eine Datei `Hello.java` anlegen
- In die Datei den Programmtext schreiben
- Den Java Compiler mit dieser Datei aufrufen. Er erzeugt dann eine Datei `Hello.class`, die die entsprechenden JVM Befehle enthält.
- Diese Datei mit der JVM ausführen.

Anatomie unseres Programms

Einrückungen etc. spielen keine Rolle (nicht so in Shellscript oder Python). Wir können auch schreiben:

```
public class Hello{public static
void main(String[] args){System.out.println
("Hello, World!");}}
```

Der Anfang `public class Hello{···}` definiert eine Klasse mit Namen `Hello`.

Zwischen den Mengenklammern steht die Definition der Klasse.

Das Schlüsselwort `public` besagt, dass diese Klasse "öffentlich" sichtbar ist, im Gegensatz zu `private`.

Wir brauchen uns im Moment nur zu merken, dass ein Programm in so eine Klassendefinition eingeschlossen werden muss und dass Klassenname und Dateiname übereinstimmen müssen.

Anatomie unseres Programms

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

definiert eine Methode des Namens `main` in der Klasse `Hello`.

Zwischen den Mengenklammern steht die Definition der Methode.

Die Methode des Namens `main` wird automatisch beim Programmstart ausgeführt. Andere Methoden, werden innerhalb des Programms aufgerufen. Etwa `berechneZinsen`, `verschiebeRaumschiff`, `openConnection`, ...

Das Schlüsselwort `static` bedeutet, dass `main` keine Objekte der Klasse `Hello` verwendet.

Der Parameter `String[] args` erlaubt es, dem Programm beim Aufruf Daten, etwa einen Dateinamen mitzugeben.

Keine Angst!

All das erklären wir erst viel später. Für den Anfang merken wir uns, dass Programme immer so aussehen müssen:

```
public class Klassenname
{
  public static void main(String[] args)
  {
    Hier geht's los
  }
}
```

Statements

Die Methodendefinition (der Programmrumpf) besteht aus einer Folge von Statements (deutsch: “Befehlen”).

Hier haben wir nur ein Statement:

```
System.out.println("Hello, World!");
```

Statements enden immer mit Semikolon (Strichpunkt, `;`).

Dieses Statement ruft die eingebaute Methode `println` des Objektes `out` in der Klasse `System` mit dem Argument (Parameter) `"Hello, World!"` auf.

Mehrere Statements

```
System.out.println("Guten Tag.");
System.out.println("Urlaubsbeginn 18. Urlaubsende 31. Das macht");
System.out.println(31-18+1);
System.out.println("Urlaubstage.");
System.out.println("Auf Wiedersehen.");
```

Hier ist `31-18+1` ein arithmetischer Ausdruck. Sein Wert wird berechnet und von `println` ausgegeben.

Mehrere Statements

Will man keine Zeilenumbrüche kann man auch die Methode `print` verwenden.

```
System.out.println("Guten Tag.");
System.out.println("Urlaubsbeginn 18ter Urlaubsende 31ter.");
System.out.print("Das macht ");
System.out.print(31-18+1);
System.out.println(" Urlaubstage.");
System.out.println("Auf Wiedersehen.");
```

Ergebnis:

```
Guten Tag.
Urlaubsbeginn 18. Urlaubsende 31.
Das macht 14 Urlaubstage.
Auf Wiedersehen.
```


Escapesequenzen

Ein `"` beginnt und endet eine Zeichenkette. Will man das Symbol `"` selbst drucken, so muss man `\"` verwenden.

Das Symbol `\` selbst kriegt man durch `\\`.

Es gibt noch andere solche Escapesequenzen, z.B. `\n`: Zeilenumbruch.

```
System.out.println("Die Zeichen \" und \\ erhält man
durch Vorausstellen eines \\.\n Das war’s.");
```

Klassen und Objekte

Objekte enthalten Werte und Methoden (um diese Werte zu bearbeiten).

Klassen dienen als Muster für Objekte.

Die Klasse spezifiziert die Formate der Werte, und die definiert die Methoden.

Beispiel: die eingebaute Klasse `Rectangle`. Der Ausdruck

```
new Rectangle(5, 10, 20, 30)
```

erzeugt ein Objekt der Klasse `Rectangle` mit linker oberer Ecke (5,10) und Breite/Höhe 20/30.

Das Statement

```
System.out.println(new Rectangle(5,10,20,30));
```

gibt das Objekt aus:

```
java.awt.Rectangle[x=5,y=10,width=20,height=30]
```


Beispielprogramm

```
import java.awt.Rectangle;

public class Rechteck
{
    public static void main(String[] args)
    {
        System.out.println("Guten Tag.");
        System.out.println(new Rectangle(5,10,20,30));
    }
}
```

Die Deklaration `import java.awt.Rectangle;` importiert den Klassennamen aus der package `java.awt`.

Alternativ kann man auch `java.awt.Rectangle` schreiben.

Programmvariablen

Durch das Statement

```
Rectangle cornflakesPackung;
```

wird eine Programmvariable (kurz Variable) des Typs `Rectangle` deklariert.

Man kann der Variablen einen Wert zuweisen durch `=`. Z.B.

```
cornflakesPackung = new Rectangle(5,10,20,30);
```

Und dann ausgeben: `System.out.println(cornflakesPackung);`

Programmvariablen

Eine Programmvariable ist ein Verweis auf ein Objekt.

Schreiben wir

```
Rectangle knusperKram = cornflakesPackung;
```

(Beachte, Deklaration und Zuweisung gehen in einem.)

Dann ist `knusperKram` auch ein Verweis auf das erzeugte Objekt.

Es gibt aber nach wie vor nur eins!

Methoden

Die Klasse `Rectangle` enthält die Methode `translate` zum Verschieben eines Rechtecks. So verwenden wir sie:

```
cornflakesPackung.translate(15,25);
```

Geben wir jetzt `cornflakesPackung` aus, dann erhalten wir

```
java.awt.Rectangle[x=20,y=35,width=20,height=30]
```

Was kommt 'raus, wenn wir `knusperKram` ausgeben?

Antwort: Genau dasselbe, da ja `knusperKram` und `cornflakesPackung` auf dasselbe Objekt verweisen.