Objektorientierte Programmierung

Kapitel 1: Einführung

Stefan Brass

Martin-Luther-Universität Halle-Wittenberg

Wintersemester 2013/14

http://www.informatik.uni-halle.de/~brass/oop13/

Inhalt

* Computer: Minimal-Hardware (1) *

- Der Kern eines Computers, der die Programme ausführt, heißt CPU oder Prozessor.

  Z.B. Pentium 4, UltraSparc IV. CPU = Central Processing Unit.

- Die auszuführenden Befehle entnimmt der Prozessor dem Hauptspeicher (RAM).

  RAM = Random Access Memory. Der Hauptspeicher enthält sowohl Programme (Maschinenbefehle) als auch Daten (die von den Programmen verarbeitet werden).

- Der Hauptspeicher besteht aus vielen Speicherzellen, die über Adressen (z.B. von 0 bis 268 435 455 bei 256 MByte) angesprochen werden.

  In der Informatik steht "M"/"Mega" meist für 2^{20}, d.h. $1024 * 1024$, also etwas mehr als eine Million.

* Computer: Minimal-Hardware (2) *

- Jede Speicherzelle enthält ein Byte (bestehend aus 8 Bits, die jeweils 0 oder 1 sein können, dies ergibt 256 verschiedene Werte).

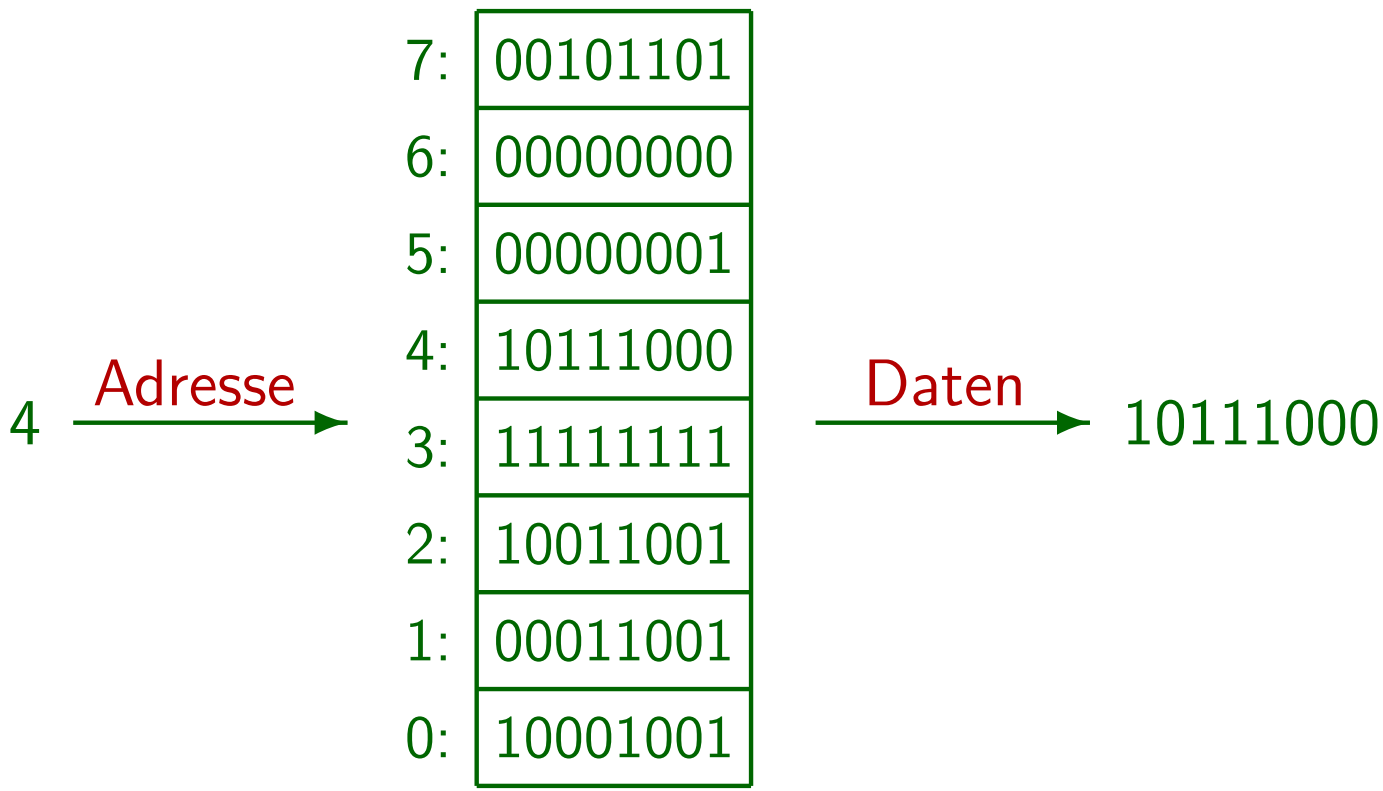
  Man kann Bytes zu größeren Einheiten (Worte) zusammenfassen, manche CPUs können auch einzelne Bits ansprechen. Deswegen kann man nicht genau sagen, was eine einzelne Speicherzelle ist.

- Man kann sich den Hauptspeicher also wie einen Schrank mit vielen Schubladen vorstellen. In jeder Schublade steckt eine Zahl zwischen 0 und 255.

  Informatiker beginnen häufig mit 0 zu zählen, da es ja eigentlich Folgen von Nullen und Einsen sind, und 00000000 einfach 0 entspricht. Das ist eine Interpretationsfrage. Z.B. auch möglich: -128 bis +127.

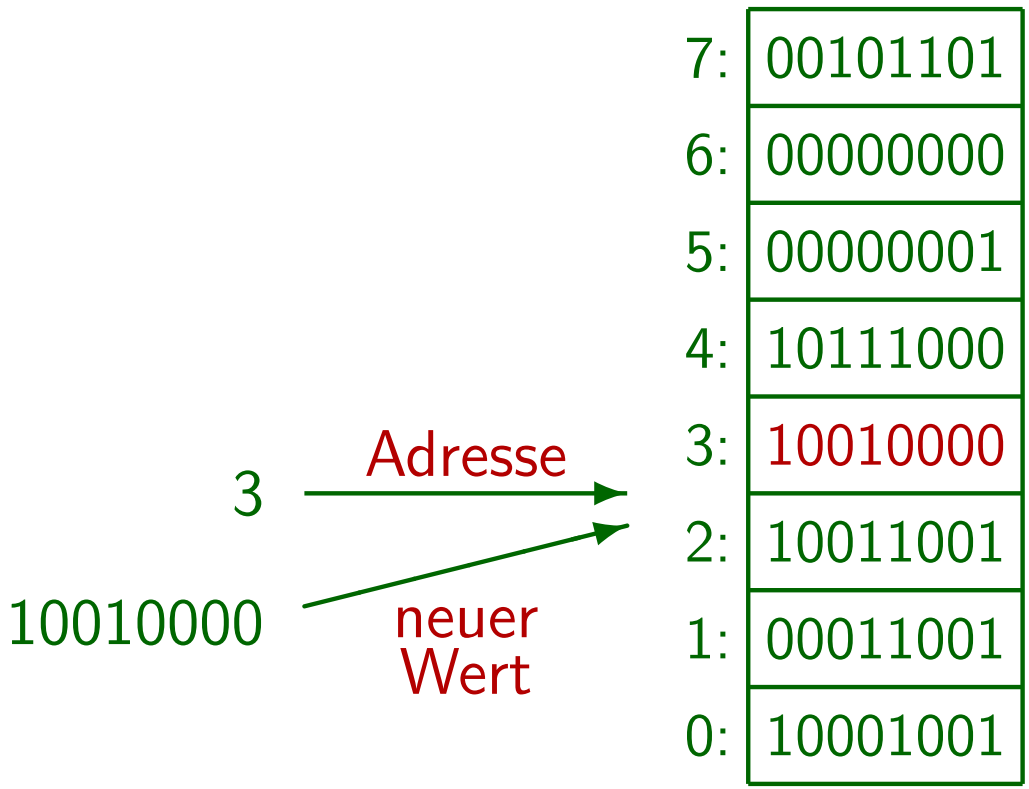
* Computer: Minimal-Hardware (3) *

Lesezugriff auf ein Byte des Hauptspeichers:



* Computer: Minimal-Hardware (4) *

Schreibzugriff auf ein Byte des Hauptspeichers:



* Maschinencode (1) *

- Die CPU enthält einen “Instruction Pointer” (oder “Program Counter”), der die Adresse des nächsten auszuführenden Befehls (in “Maschinencode”) enthält.
- Sie holt sich also den Wert aus dieser Speicherzelle.

  Eventuell auch die Werte aus einigen folgenden Speicherzellen: Viele Befehle sind länger als 1 Byte (z.B. 4–6 Byte). Typischerweise erkennt sie am ersten Byte des Befehls, wieviele Bytes noch nötig sind.
- Sie führt diesen Befehl aus, erhöht den Instruction Pointer, und holt sich den nächsten Befehl.

  Einige Befehle sind Sprungbefehle: Dann würde der Instruction Pointer auf einen neuen Wert gesetzt und nicht einfach der folgende Befehl geholt. Dies kann auch abhängig von Bedingungen geschehen.

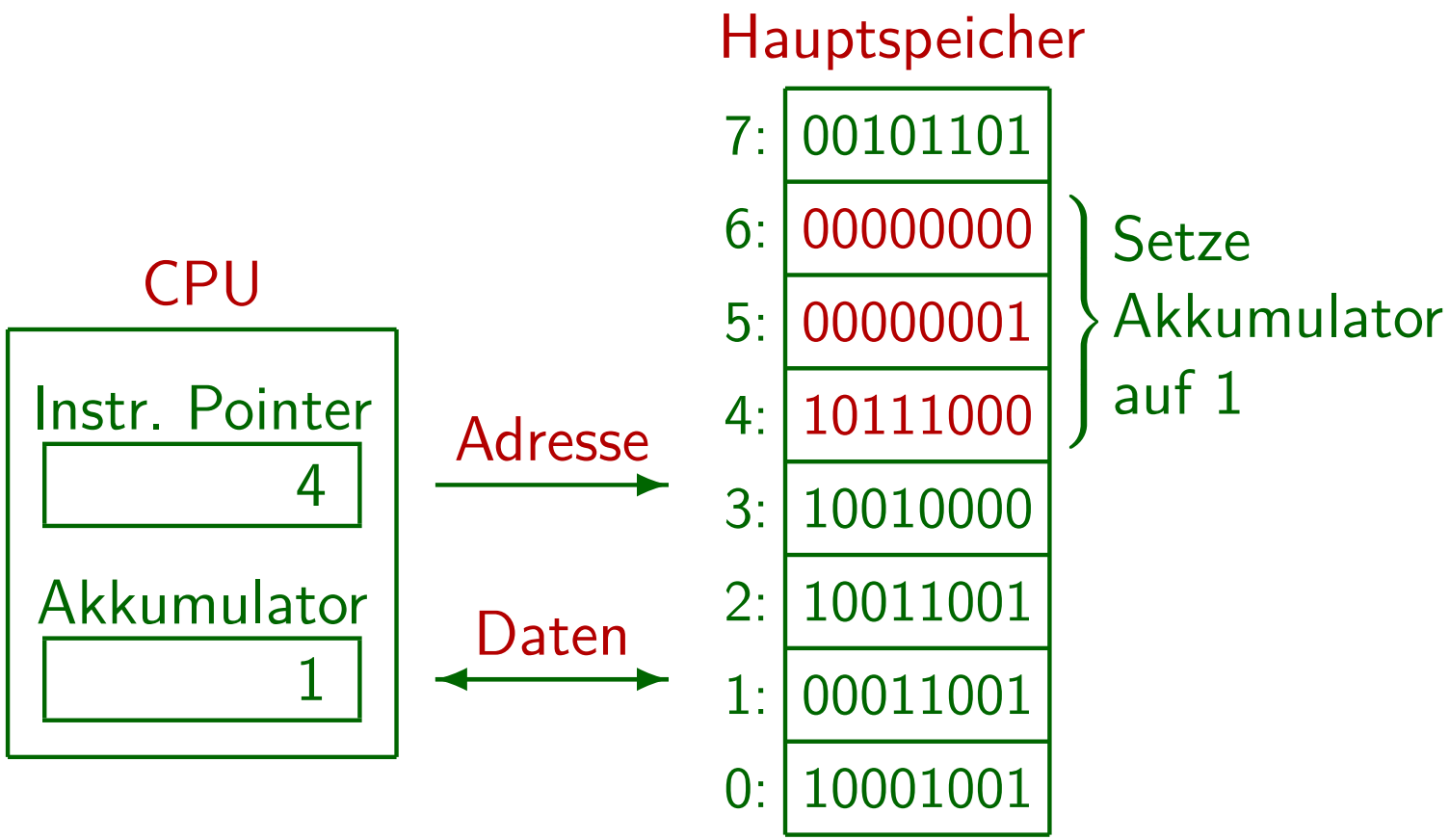
* Maschinencode (2) *

- Die einzelnen Befehle sind sehr einfach, z.B.
  - Lade den Inhalt von Speicherzelle X in eine spezielle Speicherzelle in der CPU ("Akkumulator").

    Viele CPUs haben mehr als eine solche Speicherzelle. Sie werden Register genannt. Es gibt aber normalerweise nur wenige Register. Die Register sind häufig größer als ein Byte, z.B. 4 Byte (32 Bit) oder 8 Byte (64 Bit).
  - Addiere den Inhalt von Speicherzelle X zum aktuellen Inhalt des Akkumulators hinzu.
  - Springe zu Speicherzelle X.

    Dies setzt also den Wert im Instruction Pointer, der auch ein spezielles Register der CPU ist. Der nächste Befehl wird dann von Speicherzelle X geholt.

* Maschinencode (3) *



Assembler

- Programme bestehen also letztendlich aus Folgen von Nullen und Einsen im Hauptspeicher.
- Anfangs mußte man tatsächlich in dieser Form programmieren (Maschinensprache).
- Dann wurden Assemblersprachen (kurz "Assembler") erfunden. Sie sind ein 1:1 Abbild der Maschinensprache, aber mit für den Menschen lesbaren Befehlen:

```
mov AX, 1
```

  Speichere den Wert 1 in das Register AX, den Akkumulator.

  `mov` steht kurz für "move": Bewege den Wert 1 nach AX.

- Der Assembler ist ein Programm, das solche Programme (Texte) in die internen Bitmuster für die Befehle übersetzt.

Texte, Interpretation von Bitmustern

- Die Texte können auch im Hauptspeicher des Rechners repräsentiert werden.
- Dazu interpretiert man die Bytes (Bitmuster) einfach als Buchstaben/Zeichen. Z.B. wäre ein "a" nach dem ASCII-Code das gleiche Bitmuster wie die Zahl 97.

  ASCII = American Standard Code for Information Interchange.

- Auf Maschinenebene kann das gleiche Bitmuster also ganz unterschiedlich interpretiert werden.

  Das Bitmuster in der Speicherzelle, auf die der "Instruction Pointer" zeigt, wird als Maschinenbefehl für die CPU interpretiert. Ansonsten hängt die Bedeutung an der Programmierung, wie man die Daten verarbeitet.

ASCII-Code

	0	1	2	3	4	5	6	7	8	9
0	NULL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT
10	LF	VT	FF	CR	SO	SI	DLE	DC1	DC2	DC3
20	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS
30	RS	US	␣	!	"	#	$	%	&	’
40	(	)	*	+	,	-	.	/	0	1
50	2	3	4	5	6	7	8	9	:	;
60	<	=	>	?	@	A	B	C	D	E
70	F	G	H	I	J	K	L	M	N	O
80	P	Q	R	S	T	U	V	W	X	Y
90	Z	[	\	]	^	_	‘	a	b	c
100	d	e	f	g	h	i	j	k	l	m
110	n	o	p	q	r	s	t	u	v	w
120	x	y	z	{	|	}	~	DEL		

Z.B. “A” hat den Code 65 (Zeile 60, Spalte 5).

Editor, Dateien

- Texte (z.B. ein Assembler-Programm) können mit einem weiteren Programm, dem Editor, eingegeben und geändert werden (über die Tastatur).
- Der Inhalt des Hauptspeichers geht verloren, wenn der Computer ausgeschaltet wird.
- Daher wird man den Text bzw. das Programm auf die Festplatte (oder einfach Platte, engl. Disk) abspeichern.
- Die Daten auf der Platte werden in Form von Dateien (Folgen von Bytes) verwaltet.

  Wenn Sie die Datei im Editor öffnen, kopiert er die Daten von der Platte in den Hauptspeicher. Wenn Sie auf "Speichern"/"Save" klicken, werden die ggf. veränderten Daten zurück in die Datei geschrieben.

Betriebssystem

- Die Verwaltung von Dateien ist eine der Funktionen des Betriebssystems (z.B. Windows, Linux).
- Das Betriebssystem
  - enthält eine Bibliothek von häufig verwendeten Funktionen (Programmcode),

    Programme können über einen Betriebssystemaufruf Daten aus einer Datei in den Hauptspeicher laden bzw. umgekehrt in die Datei schreiben. Sie brauchen nicht selbst Befehle zur Plattensteuerung zu enthalten.
  - ist eine Kontrollinstanz,

    Ein Rechner wird eventuell von mehreren Benutzern verwendet, dann darf man z.B. nicht beliebig auf Dateien anderer Benutzer zugreifen.
  - ermöglicht das Laden von Programmen aus Dateien (von der Platte) in den Hauptspeicher, um sie dort zu starten.

Dateien, Verzeichnisstruktur (1)

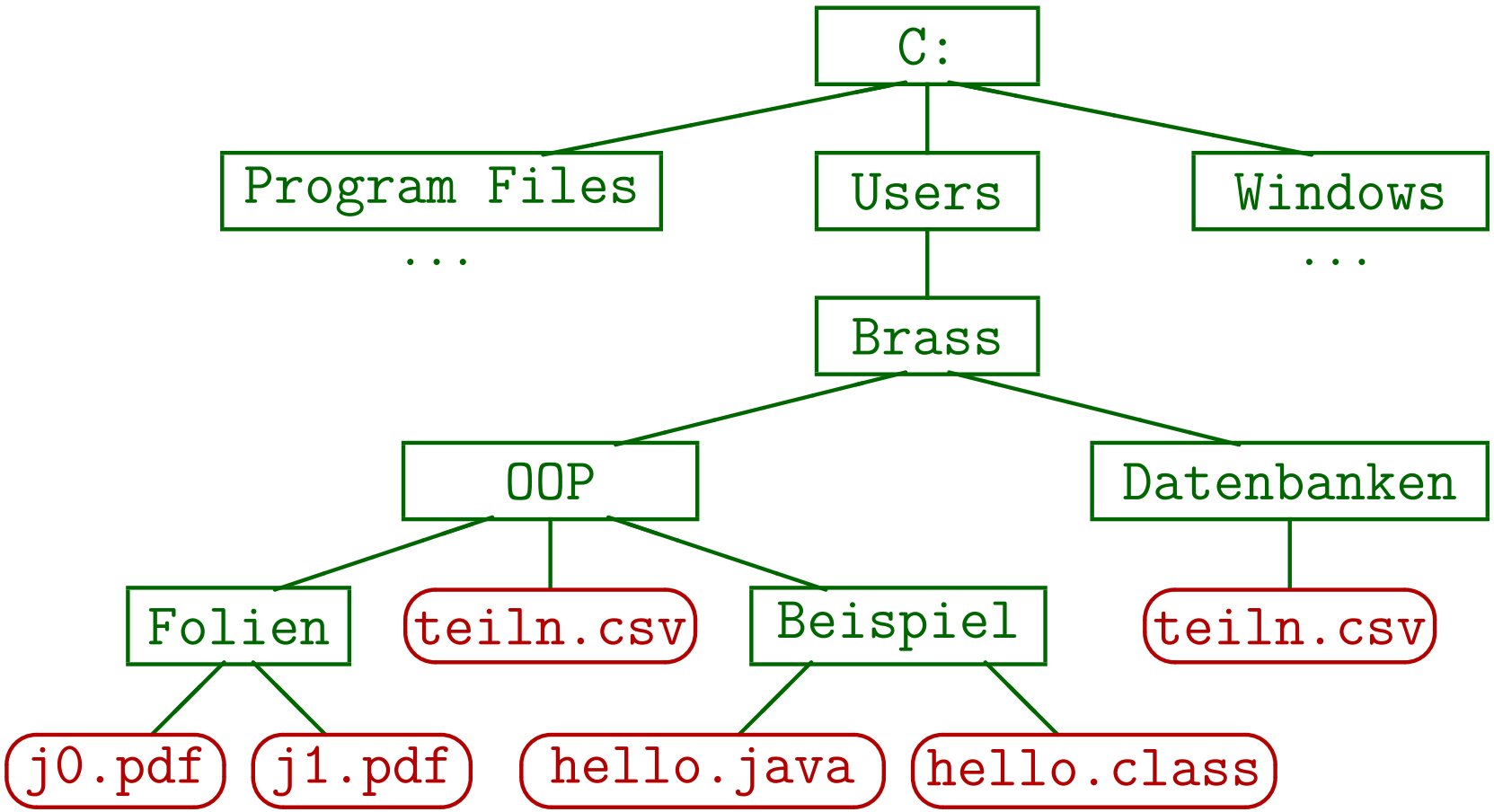
- Dateien haben einen Namen (Dateinamen).
- Dateien werden in Ordnern (Dateiverzeichnissen, Directories) strukturiert.

  Ordner und Dateiname identifizieren eindeutig die Datei.

- Ordner können selbst wieder Ordner enthalten, so dass eine hierarchische Struktur entsteht.

  In der Informatik ist so eine Datenstruktur als Baum bekannt. Allerdings wachsen in der Informatik die Bäume verkehrt herum: Die Wurzel (Hauptverzeichnis) wird oben dargestellt, die Verzweigung in Unterverzeichnisse und einzelne Dateien geschieht nach unten.
  Beim Betriebssystem Windows stehen auf oberster Ebene die Laufwerke, wie z.B. `C:`. Beim Betriebssystem UNIX (Linux, Solaris, ...) gibt es nur ein Hauptverzeichnis. Laufwerke können bei UNIX an beliebiger Stelle als Unterverzeichnisse in die Hierarchie integriert werden.

Dateien, Verzeichnisstruktur (2)



Dateien, Verzeichnisstruktur (3)

- Man kann Dateien über den vollständigen Namen (mit allen übergeordneten Ordnern) identifizieren:

  `C:\Users\Brass\OOP\Beispiel\Hello.java` (Windows)

  `/home/brass/OOP/Beispiel/Hello.java` (Unix)

  Solche Dateibezeichnungen heißen absolute Pfadnamen. Es gibt für jedes in Ausführung befindliche Programm ("Prozess") ein aktuelles Verzeichnis, von dem aus relative Pfadnamen möglich sind: Z.B. `Hello.java`, falls gerade `C:\Users\Brass\OOP\Beispiel` das aktuelle Verzeichnis ist, oder `..\Beispiel\Hello.java`, falls `C:\Users\Brass\OOP\Folien` akt. Verzeichnis.

- Es ist üblich, dass Dateinamen eine durch Punkt abgetrennte Endung haben ("Extension"), die die Art der Daten in dem Dokument anzeigt, z.B.:
  - `.pdf`: Textdokument (für Acrobat Reader / Evince).
  - `.java`: Java Programm (Eingabe für Compiler `javac`).

Inhalt

* Höhere Programmiersprachen (1) *

- Assembler-Sprachen hatten drei Nachteile:
  - Die Programme liefen nur mit dem CPU-Typ, für den sie geschrieben wurden (nicht portabel).
  - Die Befehle der CPUs sind sehr einfach, kompliziertere Programme also entsprechend lang.

    Es gibt Untersuchungen, nach denen Programme in der höheren Programmiersprache C dreimal kürzer sind als äquivalente Assembler-Programme, und auch entsprechend schneller entwickelt werden (d.h. Programmierer brauchen in diesen Sprachen die gleiche Zeit pro "Line of Code"). Die Produktivität verdreifachte sich also beim Schritt von Assembler zu der höheren Programmiersprache C.
  - Die Programme sind schlecht strukturiert und unübersichtlich, schwierig zu warten.

    Es geschehen auch leicht Fehler, da der Assembler alles zulässt.

* Höhere Programmiersprachen (2) *

- Daher wurden höhere Programmiersprachen erfunden, die erste erfolgreiche war Fortran (1954–57).

  Es hat auch etwas frühere Versuche gegeben.

  Fortran = FORmula TRANslator.

- Insbesondere konnte man jetzt die übliche mathematische Notation für Formeln verwenden, z.B.

```
X = 3 * Y + Z
```

  Dies entspricht einer Reihe von Maschinenbefehlen: Zuerst muß man den Wert von `Y` in den Akkumulator laden (der Variablen `Y` wird eine bestimmte Hauptspeicher-Adresse zugeordnet — man kann so mit Namen statt Adressen arbeiten). Dann muß man den Inhalt des Akkumulators mit 3 multiplizieren, anschliessend `Z` aufaddieren, und zum Schluß den aktuellen Inhalt des Akkumulators in die für `X` reservierte Speicherzelle schreiben.

* Höhere Programmiersprachen (3) *

- Java gehört zur Familie der C-ähnlichen Sprachen, und ist insbesondere von C++ beeinflusst.

  C ist seinerseits ein Ableger der Algol-Familie ("Algorithmic Language").

- C wurde 1969–1973 von Dennis Ritchie in den Bell Labs entwickelt (kleinere Änderungen 1977-1979), das wichtige Lehrbuch von Kernighan/Ritchie erschien 1978.

  Siehe: [http://cm.bell-labs.com/cm/cs/who/dmr/chist.html]

  C wurde parallel mit dem Betriebssystem UNIX entwickelt (zur Implementierung von UNIX). Die für UNIX zuerst benutzte PDP-7 hatte 8K 18-bit Worte RAM.

- C erreichte große Verbreitung.

  Es war eine kompakte/kleine Sprache, deren Befehle sehr direkt in Befehle der CPU übersetzt werden konnten (hardware-nah, effizient).

- Der ANSI-Standard für C erschien 1989 (ISO C90).

* Objektorientierte Sprachen (1) *

- Für große Programme hat sich eine objektorientierte Struktur als meistens sehr übersichtlich herausgestellt. Dies wird in C nicht unterstützt.

  Besonders graphische Benutzeroberflächen können objektorientiert gut entwickelt werden.

- Als erste objektorientierte Programmiersprache gilt heute Simula-67.

  Wie der Name schon sagt, war Simula für Simulationen gedacht, und wurde 1967 auf einer Konferenz vorgestellt (entwickelt von Ole-Johan Dahl and Kristen Nygaard in Oslo). Der Erfinder von C++, Bjarne Stroustrup, ist von Simula-67 wesentlich beeinflusst worden.

- Smalltalk-80 leiste einen wesentlichen Beitrag zur Verbreitung der Idee der Objektorientierung.

  Entwickelt in den 70er Jahren am XEROX PARC Forschungszentrum.

* Objektorientierte Sprachen (2) *

- Bjarne Stroustrup begann 1979, in den Bell Labs an einer objektorientierten Erweiterung von C zu arbeiten.

  Die Sprache hieß zuerst “new C”, dann “C with Classes”, und ab 1983 “C++”. Eine erste kommerzielle Implementierung erschien 1985. 1989 erschien Version 2.0, 1990 das “Annotated C++ Reference Manual”.

- Der ANSI-ISO Standard für C++ erschien 1998, eine neue Auflage 2003. Aktuell ist der Standard von 2011.

  Die Sprache C++ hat sich bis zum Erscheinen des ersten Standards wesentlich geändert, ältere Literatur ist heute kaum noch brauchbar.

- C++ ist eine große “Multi-Paradigma-Sprache”.

  Nicht nur objektorientiert. Java war anfangs deutlich schlanker/einfacher, ist aber im Laufe mehrerer Versionen auch gewachsen.

- Java hat viel von C++ übernommen, aber vereinfacht.

* Entwicklung von Java (1) *

- Sun Microsystems gründe 1990/91 eine Arbeitsgruppe, um neue Trends aufzuspüren und innovative Ideen zu entwickeln.

  Sun (gegr. 1982) ist Hersteller von UNIX-Workstations, Servern, dem Betriebssystem Solaris, dem Network File System NFS und Entwickler des SPARC Prozessors (eine RISC CPU). 2009/10 hat Oracle Sun übernommen.

- Das "Green Project" entwickelte einen Prototyp zur Steuerung und Vernetzung von interaktivem Fernsehen und anderen Geräten der Konsumelektronik.
- Hierfür entwickelte James Gosling die Programmiersprache "Oak" (1991/92), die später in Java umbenannt wurde.
- Die aus dem Forschungsprojekt entstandene Firma "First Person Inc." war nicht erfolgreich (fand keine Partner).

* Entwicklung von Java (2) *

- Das World Wide Web begann seinen Siegeszug 1993 mit dem Browser “Mosaic”.

  Im Januar 1993 gab es ungefähr 50 WWW/HTTP-Server, im Oktober über 200, im Juni 1994 über 1500.

- 1994 beginnen Sun-Entwickler (Patrick Naughton, Jonathan Payne) einen Browser “WebRunner” zu entwickeln, der in Java geschrieben ist (später in HotJava umbenannt).
- Die Sprache Java wurde nun verwendet, um Programmcode (“Applets”) über das Internet zu verteilen und im Browser auszuführen.
- Am 23.05.1995 stellt Sun “HotJava” auf der “SunWorld” vor. Netscape entschließt sich, Java auch in den “Netscape Navigator” (verbreiteter Browser) einzubauen.

* Entwicklung von Java (3) *

- JDK 1.0: 23.01.1996
- JDK 1.1: 19.02.1997

  Hier kamen u.a. Inner Classes hinzu. AWT Events überarbeitet. JDBC. Reflection (Introspection). Java Beans.

- J2SE 1.2: 08.12.1998 (Java 2, Standard Edition)
- J2SE 1.3: 08.05.2000
- J2SE 1.4: 06.02.2002
- J2SE 5.0: 30.09.2004 (interne Versionsnummer 1.5)

  Größere Änderungen, z.B. Generics, Annotations, Autoboxing, Enumerations, spezielle `for`-Schleife für Collections, static import, varargs.

- Java SE 6: 11.12.2006 (Version 1.6)
- Java SE 7: 28.07.2011 (Version 1.7)

Compiler

- Programme sind also spezielle Texte (Folgen von Zeichen).

  Diese Texte müssen den Regeln einer Programmiersprache (wie z.B. Java) folgen. Z.B. hat Java eine Grammatik (recht einfach und absolut präzise).

- Solche Texte können mit Hilfe eines Programms, des Compilers, in Maschinensprache übersetzt werden.

  Der erste Compiler mußte natürlich in Assembler geschrieben werden.
  "compile": zusammentragen, zusammenstellen (Maschinencode aus Mustern, Bibliotheken).

- Erst dadurch werden die Programme ausführbar:
  Die CPU selbst versteht ja nur Maschinenbefehle.

- Im Laufe der Zeit wurden viele Programmiersprachen vorgeschlagen (und Compiler für diese Sprachen entwickelt).

Begriffe

- Ein Algorithmus ist ein Verfahren, mit dem eine Aufgabe gelöst werden soll.

  Ein Algorithmus ist unabhängig von einer speziellen Programmiersprache.
  Z.B. wurden viele Algorithmen zum Sortieren vorgeschlagen.
  Kochrezepte, Bauanleitungen, Spieltaktiken sind ähnlich zu Algorithmen (in der Informatik aber eher nicht präzise genug).

- Einen Algorithmus kann man in einer Programmiersprache formal aufschreiben ("codieren").

- Quellcode (engl. "source code") ist die Eingabe für den Compiler (z.B. ein Programm in Java).

- Ziel der Übersetzung ist ein ausführbares Programm (Maschinenbefehle für die Ziel-Hardware).

Interpreter

- Eine weitere Möglichkeit, Programme eine höheren Programmiersprache auszuführen, sind Interpreter (z.B. typisch für die Sprachen Basic oder Lisp):
  - Sie laden den Programmtext in den Hauptspeicher,
  - verarbeiten ihn ggf. vor,
  - und führen es aus, ohne explizit Maschinensprache zu erzeugen.

    Die ausgeführten Maschinenbefehle sind Teil des Interpreter-Programms. Dies fragt jeweils den nächsten Befehl des gegebenen Programms ab und enthält entsprechende Fallunterscheidungen, in denen alle möglichen Befehle behandelt werden. Gewissermaßen simuliert es so die Ausführung des eigentlichen Programms.
- Unterschied zu Compiler: Keine getrennte Datei für Übersetzungsergebnis, man arbeitet nur mit Quellcode.

Aufgabe

- Angenommen, Sie haben ein Programm in einer höheren Programmiersprache geschrieben und es in ein ausführbares Programm übersetzt (mit einem Compiler).
- Sie möchten Ihr Programm mit verschiedenen Eingaben testen. Müssen Sie den Compiler jedesmal neu aufrufen?
- Ihr Programm ist so nützlich, dass Sie im Internet veröffentlichen wollen. Welche Vor- und Nachteile hat es, wenn Sie Quellcode oder das ausführbare Programm auf Ihre Webseite stellen?
- Welche Vor- und Nachteile haben Compiler und Interpreter?

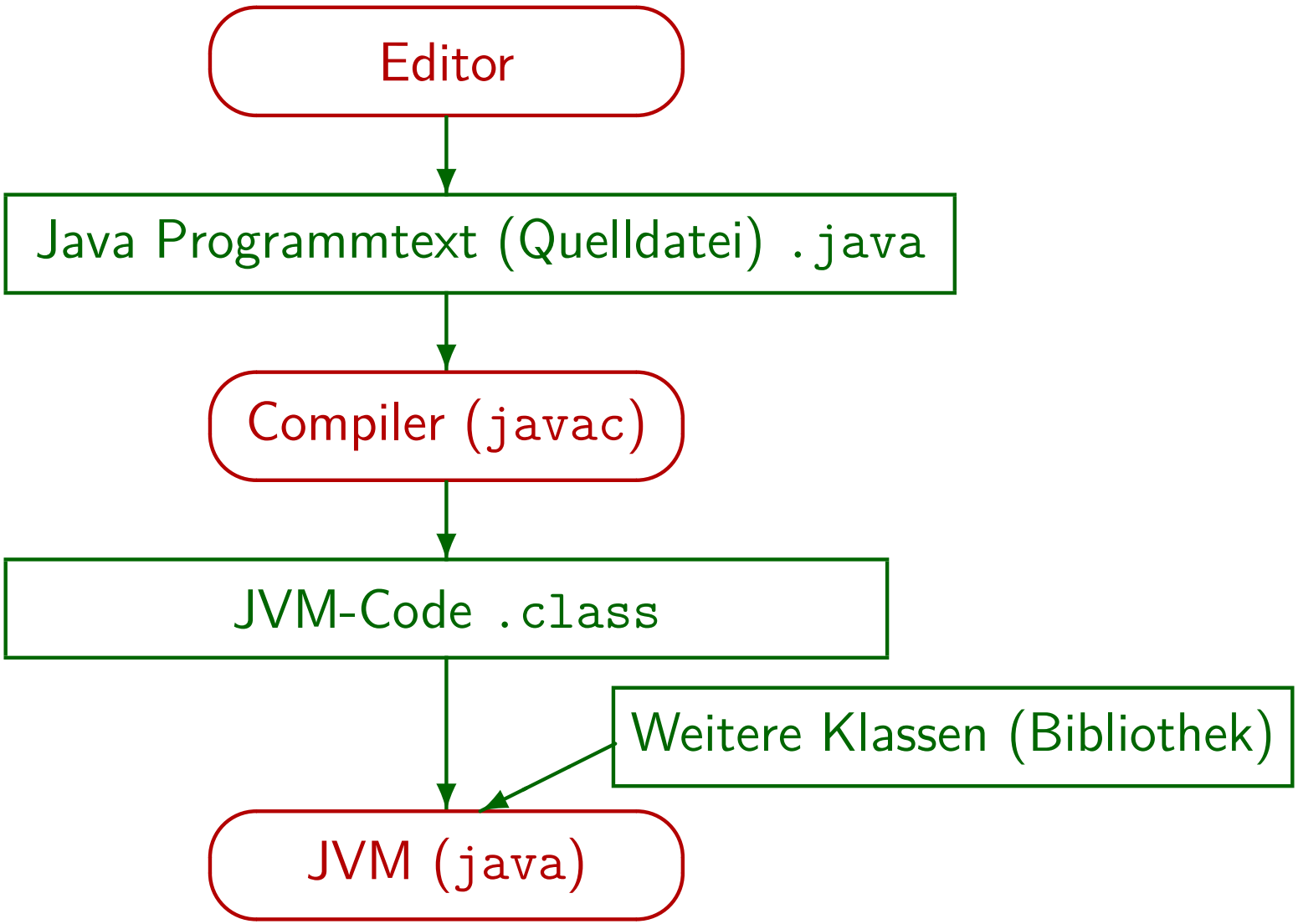
Java (1)

- Bei Java werden normalerweise Compiler und Interpreter kombiniert:
  - Der Compiler übersetzt nicht in Maschinencode, sondern in Instruktionen für die "Java Virtual Machine" (JVM). Dies wird auch "Java Bytecode" genannt.

    Das ist gewissermaßen schon Maschinencode, aber für eine gar nicht in Hardware (Elektronik) existierende Maschine.
  - Jeder unterstützte Rechner hat nun einen Interpreter für die JVM Instruktionen (das ist ja auch nur eine spezielle Programmiersprache).

    Der Interpreter muß natürlich in Maschinensprache vorliegen.
    Die Entwickler haben ihn in C++ geschrieben (250.000 Zeilen) und dann einen C++-Compiler benutzt.
    [http://openjdk.java.net/groups/hotspot/]

Java (2)


Editor
Java Programmtext (Quelldatei) .java
Compiler (javac)
JVM-Code .class
Weitere Klassen (Bibliothek)
JVM (java)

Java (3)

Vorteile dieses Verfahrens:

- Software-Anbieter muß Programm nicht für jede Platform (Hardware und Betriebssystem) einzeln bereitstellen: `class`-Dateien sind Plattform-unabhängig.

  Besonders wichtig für Ausführung in Browsern (Applets).

- Software-Anbieter muß Quellcode nicht offenlegen.

- Mehr Prüfungen möglich, weniger Sicherheitslücken.

  In Browsern würde man die Ausführung von beliebigem Maschinencode, der von irgendeiner Webseite heruntergeladen wurde, ja gar nicht erlauben.

- Bessere Fehlermeldungen als bei reiner Maschinensprache.

- Oft kompakter als reiner Maschinencode (Dateien kleiner).

Java (4)

Zur Effizienz:

- Schneller als gewöhnlicher Interpreter.

  Ein Teil des Interpretationsaufwands wird zur Ausführungszeit vermieden (z.B. syntaktische Analyse des Quellprogramms).

- Langsamer als Ausführung eines Programms, das schon fertig in Maschinencode vorliegt (erzeugt von Compiler).

- Moderne Implementierungen der Java Virtual Machine enthalten aber einen "Just in Time Compiler", der den Bytecode zur Ausführung doch in Maschinencode übersetzt.

  Wenn die Befehle häufig ausgeführt werden, fällt der Overhead für die Übersetzung nicht mehr ins Gewicht. Es gibt auch Optimierungen, die so leichter werden (gegenüber normalem Compiler). Dafür leichte Verzögerung beim Starten. Insgesamt Laufzeitunterschied nicht mehr groß.

Zur Einschätzung von Java (1)

- Java ist nicht nur eine Programmiersprache, es ist eine Programmier-Plattform.
- Die Bibliotheken, die mit Java mitgeliefert werden, sind mindestens so wichtig wie die Sprache selbst.

  Bibliotheken enthalten fertigen Programmcode für vielerlei Aufgaben, den man in eigenen Programmen aufrufen (mitbenutzen) kann.
- Die Anzahl von Klassen/Interfaces in der Bibliothek stieg von 211 in Version 1.0 auf 3777 in Version 6.

  Während in dieser Vorlesung die Sprache im Vordergrund steht, muß man als versierter Java-Programmierer einen nicht unwesentlichen Teil der Bibliothek kennen. Bei Sprachen wie C++ ist die Standard-Bibliothek im Vergleich zu Java eher klein, Erweiterungen (z.B. `Qt`, u.a. für portable GUI-Programmierung) sind halb-kommerziell und es gibt Konkurrenz.

* Zur Einschätzung von Java (2) *

- JavaScript ist eine ganz andere Sprache als Java (andere Basiskonzepte, aber auch C-ähnliche Syntax).

  Der Name ist eher aus Marketing-Gründen zu verstehen.

- Einige wichtige Unterschiede von Java zu C++:
  - C++ lässt bestimmte Dinge bewusst undefiniert, um dem Compiler mehr Optimierungsmöglichkeiten für die jeweilige Hardware zu lassen. Java enthält genaue Festlegungen.
  - In C++ kann man mit Hauptspeicheradressen rechnen und auf beliebige Speicherstellen unter Umgehung der Datentypen zugreifen (bewusst oder unbewusst).
  - In C++ muss/kann man die Speicherverwaltung für Objekte selbst programmieren, in Java gibt es nur dynamische Speicherverwaltung mit automatischer "Garbage Collection".

Inhalt

“Hello, World” Programm (1)

- Seit dem Buch von Kernighan und Ritchie über die Programmiersprache “C” ist es üblich, als erstes Programm zur Illustration einer Programmiersprache eines zu schreiben, das “`hello, world`” ausgibt.

  Sprache und Buch waren sehr einflussreich und sind bis heute verbreitet.

- Man kann so mit einem minimalen Beispiel erst einmal sehen, dass alles funktioniert.

  Kernighan/Ritchie schreiben: “This is the big hurdle; to leap over it, you have to be able to create the program text somewhere, compile it successfully, load it, run it, and find out where your output went. With these mechanical details mastered, everything else is comparatively easy.”

- Außerdem hat man eine Basis für Erweiterungen, und kann z.B. weitere Berechnungen in diesen Rahmen einbauen.

“Hello, World” Programm (2)

```
(1)  // Erstes Java-Programm
(2)
(3)  class Hello {
(4)      static public void main(String[] args) {
(5)          System.out.println("Hello, World!");
(6)      }
(7)  }
```

Die Zeilennummern links gehören nicht mit zum Programm. Sie machen es einfacher, sich auf bestimmte Zeilen zu beziehen. In der Datei `Hello.java` stehen nur die anderen (grün gedruckten) Zeichen.

Die folgende Erklärung zum Programm brauchen Sie nicht unbedingt vollständig zu verstehen, alles wird später noch systematisch und ausführlich erklärt.
Sie müssen aber lernen, das Programm mit einem Editor einzugeben (“abzutippen”), und mit Compiler und JVM auszuführen.

* “Hello, World” Programm (3) *

- Die erste Zeile

  ```
  // Erstes Java-Programm
  ```

  ist ein Kommentar für den menschlichen Leser.
  Der Compiler ignoriert sie.

  Genauer ignoriert der Compiler alles von // bis zum Ende der Zeile. Weil es der Compiler ignoriert, kann man dort beliebigen Text hinschreiben. Ansonsten könnte der Compiler normales Deutsch gar nicht verstehen.

- Man kann den Kommentar also auch weglassen, ohne dass sich etwas an der Funktionsweise des Programms ändert.

  Im allgemeinen sollen Programm aber möglichst verständlich sein, so dass z.B. auch andere Programmierer später eventuell notwendige Änderungen und Anpassungen vornehmen können. Dafür sind Kommentare ein wichtiges Mittel.

* "Hello, World" Programm (4) *

- Java ist eine formatfreie Sprache, d.h. es kommt auf die genaue Zeilenstruktur und die Einrückungen nicht an.

  Der Compiler betrachtet jede Folge von Leerzeichen, Zeilenumbrüchen und Tabulator-Zeichen als "Worttrennung". Sie hat die gleiche Wirkung wie ein einzelnes Leerzeichen (außer in Zeichenketten-Konstanten).

  Im Beispiel-Programm ist die Zeile (2), die Leerzeile, nicht wichtig.

- Wenn man will, kann man auch alles in eine Zeile schreiben.

  Außer den Kommentar, der erstreckt sich ja bis zum Zeilenende.

  Auch Java-Programmtext würde nach // ignoriert.

- Natürlich sollte das Programm möglichst übersichtlich aussehen, dazu hat sich eine Einrückung entsprechend der Klammerstruktur bewährt.

  Üblich ist eine Einrückung um jeweils 4 Leerzeichen, aber man kann auch Tabulator-Zeichen verwenden.

* “Hello, World” Programm (5) *

- In Zeile (3) bis (7) wird eine Klasse mit Namen “`Hello`” definiert.
- Klassen beschreiben in der objekt-orientierten Programmierung Mengen gleichartiger Programm-Objekte, die häufig Objekte der realen Welt abbilden, z.B. Studenten, Vorlesungen.

  Eine Klassendefinition ist eine Art Blaupause, mit der man Objekte des entsprechenden Typs erzeugen kann. Objekte enthalten Daten, bieten aber auch Methoden zur Verarbeitung dieser Daten.
- Die Klasse im Beispiel ist etwas untypisch, weil man gar nicht Objekte vom Typ “Hello” erzeugen möchte.

  Schon gar nicht mehrere. Was sollte das auch sein?

* “Hello, World” Programm (6) *

- Nicht alle Klassen sind zur Erzeugung von Objekten gedacht.
- Klassen werden (wie im Beispiel) auch als “Module” gebraucht, also als Einheiten zur Zusammenfassung eines Stück Programmtexts.

  Bei großen Programmen ist es wichtig, den Programmtext möglichst übersichtlich zu strukturieren. Man möchte auch Teile schaffen, die allgemein nützlich sind, und in anderen Programmen wiederverwendbar.

- Java ist eine rein objektorientierte Sprache, deshalb muss in Java fast alles innerhalb von Klassen-Definitionen stehen, insbesondere alle ausführbaren Programmteile (Methoden).

* "Hello, World" Programm (7) *

- Das Wort "`class`" ist ein Schlüsselwort von Java. Es hat eine feste Bedeutung (markiert Klassendefinitionen).

  Java hat insgesamt 50 Schlüsselworte. Die meisten werden Sie in dieser Vorlesung lernen müssen. Das ist aber kein stupides Auswendiglernen, sondern ergibt sich automatisch beim Erlernen und Verwenden der entsprechenden Sprachkonstrukte.

- Das Wort "`Hello`" ist ein Name (Bezeichner) der neu definierten Klasse. Sie können hier beliebige "Worte" wählen (Folgen von Buchstaben und einigen anderen Zeichen).

  Es muss also nicht ein deutsches oder englisches Wort sein, der Compiler würde die Bedeutung sowieso nicht verstehen. Damit das Programm für Menschen verständlich wird, sollte der Name der Klasse natürlich ein Hinweis auf ihren Zweck sein. Die 50 Schlüsselworte dürfen nicht verwendet werden, deswegen heissen sie auch "reservierte Worte".

* “Hello, World” Programm (8) *

- Der Klassennamen sollte dem Dateinamen entsprechen: Also Klasse “`Hello`” in der Datei “`Hello.java`”.

  Im allgemeinen kann eine `.java`-Datei mehrere Klassendefinitionen enthalten, jede wird in eine eigene `.class`-Datei übersetzt, die so wie die Klasse heißt. Das ist aber nur für kleine Java-Programme zu empfehlen, die ganz in eine `.java`-Datei geschrieben werden. Normalerweise bestehen Java-Programme aus vielen `.java`-Dateien, die jeweils nur eine Klasse enthalten. Falls eine Klasse als “`public`” deklariert wurde, muß die `.java`-Datei so heißen wie die Klasse.

- Es ist üblich, dass Klassennamen mit einem Großbuchstaben beginnen, das ist aber nur Konvention.

- Groß- und Kleinschreibung ist in Java wichtig, d.h. für den Compiler sind z.B. “`x`” und “`X`” unterschiedliche Namen.

  Entsprechend werden Schlüsselworte nur verstanden, wenn sie klein geschrieben werden.

* “Hello, World” Programm (9) *

- Die Bestandteile der Klasse stehen in geschweiften Klammern `{...}`.

  Dabei ist auf die korrekte Schachtelung zu achten. Da innerhalb der Klasse in Zeile (4) noch eine geschweifte Klammer auf geht, endet die Klassendefinition nicht bei der nächsten } in Zeile (6), sondern erst bei der übernächsten in Zeile (7). Die Klammer in Zeile (6) schließt die in Zeile (4). Viele Editoren heben die jeweils zugehörige Klammer farblich hervor, wenn sich der Cursor (aktuelle Position) auf einer Klammer befindet. Häufig kann man auch direkt zwischen dem Klammerpaar springen.

- Im Beispiel hat die Klasse nur einen Bestandteil (“member”), nämlich die Methode “`main`”.

  Statt “Methode” kann man auch “Funktion” oder “Prozedur” sagen. Methoden enthalten Anweisungen (“statements”), die die eigentliche Arbeit des Programms machen, wenn sie ausgeführt werden. Im Beispiel enthält die Methode eine Anweisung zum Drucken eines Textes.

* “Hello, World” Programm (10) *

- Die Methode “`main`” ist, wie der Name schon sagt, das Hauptprogramm.

  Wenn man mit dem Befehl “`java X`” ein Programm startet, beginnt die Ausführung in der Methode “`main`” der Klasse “`X`” (von bestimmten Initialisierungen abgesehen). Diese Methode kann dann wieder andere Methoden aufrufen, von der eigenen Klasse oder auch von anderen Klassen.

- Methoden, die nicht für ein bestimmtes Objekt aufgerufen werden, sondern für die Klasse selbst, werden mit dem Schlüsselwort “`static`” gekennzeichnet.

  Mindestens zum Programmstart gibt es noch keine Objekte der Klasse, im Beispiel werden auch nie welche angelegt. In dieser Situation kann man nur “statische Methoden” der Klasse aufrufen.

* "Hello, World" Programm (11) *

- Methoden, die beliebig von außerhalb der Klasse aufrufbar sind, werden mit dem Schlüsselwort "`public`" markiert.

  Im Gegensatz dazu werden Hilfsmethoden, die nur innerhalb der Klasse wichtig sind, mit dem Schlüsselwort "`private`" gekennzeichnet. Es gibt auch noch weitere Festlegungen zum Zugriffsschutz (wird später besprochen).

- Es ist ein wesentliches Prinzip der objektorientierten Programmierung, das man Methoden und Variablen (Daten) innerhalb einer Klasse verstecken kann.

  Die Teile der Klasse, die von außen zugreifbar sind, bilden die Schnittstelle der Klasse. Wer die Klasse in eigenen Programmen verwenden will, braucht nur die Schnittstelle zu kennen. Die übrigen Dinge können geändert werden, ohne dass sich die Schnittstelle der Klasse ändert. Wenn man z.B. einen schnelleren Algorithmus gefunden hat, um die Aufgabe der Klasse zu erfüllen, müssen die Verwender der Klasse ihre Programme nicht anpassen. Die Daten werden so auch vor ungeprüften Änderungen geschützt.

* “Hello, World” Programm (12) *

- Methoden liefern dem Aufrufer im Normalfall einen Ergebniswert zurück.

  Methoden sind mit mathematischen Funktionen vergleichbar, wie z.B. der Sinus-Funktion. Sie haben Argumentwerte als Eingabe (bei Sinus einen Winkel), und liefern einen Funktionswert (bei Sinus eine reelle Zahl zwischen -1 und $+1$). Im Unterschied zu mathematischen Funktionen können Methoden allerdings noch weitere Veränderungen im Berechnungszustand vornehmen, z.B. eine Ausgabe machen. Deswegen ist nicht garantiert, dass, wenn man die gleiche Methode mit dem gleichen Argumentwert mehrfach aufruft, man immer das gleiche Ergebnis bekommt.

- Die Methode “`main`” hat keinen Ergebniswert. Deswegen wird der spezielle, leere Datentyp “`void`” angegeben.

  Bei der Sinus-Funktion würde dort entsprechend “`double`” stehen (für Fliesskommazahlen doppelter Genauigkeit, die zur Approximation reeller Zahlen verwendet werden).

* "Hello, World" Programm (13) *

- Die Methode "`main`" hat aber eine Eingabe, nämlich die Argumente von der Kommandozeile.
- Beim Aufruf kann man weitere Daten angeben, z.B.

```
java Hello my Computer
```

- Die Methode "`main`" kann diese Werte abfragen, sie stehen in der Variable "`args`" die als Parameter der Methode angegeben ist.

  Der Parameter hat den Datentyp "`String[]`", das ist ein Array (Vektor, Feld, Liste) von Zeichenketten. Beim Beispielaufruf steht in `args[0]` der Wert "`my`" und in `args[1]` der Wert "`Computer`".

- Das Beispielprogramm benutzt diese Werte aber nicht.

  Dennoch muss ein Parameter vom Typ "`String[]`" angegeben werden, sonst wird das Hauptprogramm nicht erkannt. Der Parameter-Name ist egal.

* “Hello, World” Programm (14) *

- Die Methode “`main`” enthält nur eine einzige Anweisung:

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

- Hierdurch wird die Methode “`println`” (“print line”) aufgerufen, und zwar für das Objekt “`out`”, das ein Attribut der Klasse “`System`” ist.

- Diese Klasse ist Bestandteil der Bibliothek, die mit Java mitgeliefert wird.

  Man sagt auch, es ist Teil der “API” (“Application Program Interface”) der Java Platform. Die API ist unter [http://docs.oracle.com/javase/7/docs/api/] dokumentiert. Sie können die Klasse “`System`” in der Auswahlbox links unten finden. Wenn Sie darauf klicken, bekommen Sie die Beschreibung der Klasse. Oben finden Sie unter “Field Summary”, dass “`out`” ein Objekt der Klasse “`PrintStream`” ist. Wenn Sie darauf klicken, finden Sie unter “Method Summary” die Methode “`println`” für ein `String`-Argument.

* “Hello, World” Programm (15) *

- `"Hello, World!"` (der Eingabewert für die Methode `println`) ist eine Zeichenketten-Konstante, also ein Text.

  Man sagt auch auf “Computer-Deutsch” einfach “String”. Man muss sich das wohl so vorstellen, dass die Buchstaben an einem Bindfaden/Band aufgereiht sind.

- Zeichenketten müssen in doppelte Anführungszeichen `"` eingeschlossen werden, nicht etwa Apostroph-Zeichen `'`.

  Die Apostroph-Zeichen werden für einzelne Zeichen verwendet, nicht für Folgen von Zeichen (Texte). Solche Feinheiten sind für den Compiler wichtig. Falls eine Zeichenkette ein Anführungszeichen enthalten soll, muß man einen Rückwärtsschrägstrich \ voranstellen, sonst hält der Compiler das Anführungszeichen schon für das Ende der Zeichenkette und kann mit dem folgenden Text dann nichts mehr anfangen. Zeilenumbrüche sind auch nicht in String-Konstanten erlaubt.

Ausführung des Programms (1)

- Schreiben Sie das obige Programm in die Datei "`Hello.java`".

  Natürlich ohne die Zeilennummern. Sie benötigen einen Editor. Minimale Lösung wäre `notepad`, aber es gibt natürlich viel mächtigere Editoren.

- Rufen Sie den Compiler mit folgendem Befehl auf:

  ```
  javac Hello.java
  ```

  Sie müssen den Befehl in der Eingabeaufforderung, Konsole, "Command Prompt" eingeben (in Windows unter "Zubehör"). Natürlich müssen Sie mit `cd` in das Verzeichnis wechseln, in dem "`Hello.java`" gespeichert ist.

- Führen Sie das Programm aus mit

  ```
  java Hello
  ```

  Beachten Sie, dass Sie die Endung "`.class`" weglassen müssen.

Ausführung des Programms (2)

- Das erste, was schiefgehen könnte, ist, das der Compiler `javac` nicht gefunden wird.

  UNIX würde Ihnen folgende Meldung geben: "`javac: Command not found`''. Windows: "`'javac' is not recognized as an internal or external command, operable program or batch file.`"

- Prüfen Sie den Suchpfad.

  Unter UNIX (Linux etc.) geben Sie "`echo $PATH`" ein, unter Windows "`echo %path%`". In der Liste der Verzeichnisse (unter UNIX durch "`:`" getrennt, unter Windows durch "`;`") muß das Verzeichnis gelistet werden, in dem das Programm `javac` steht, das ist das Unterverzeichnis "`bin`" des JDK-Verzeichnisses.

Syntaxfehler (1)

- Falls Sie beim Abtippen des Programms einen Fehler gemacht haben, so dass der Compiler die Eingabe nicht verstehen kann, erhalten Sie eine Fehlermeldung.

  Im positiven Fall sagt der Compiler dagegen nichts und erstellt die Datei `Hello.class`.

- Wenn Sie z.B. das Semikolon in Zeile `(5)` weglassen, macht der Compiler folgende Ausgabe:

```
Hello.java:5: ';' expected
        System.out.println("Hello, World!")
                                           ^
1 error
```

  Mit etwas Erfahrung wird man das für eine sehr klare Fehlermeldung halten: Der Compiler hat genau an der richtigen Stelle im Programm eine Meldung ausgegeben, die deutlich sagt, was zu tun ist, nämlich dort ein Semikolon einzufügen (und den Compiler dann erneut aufzurufen).

Syntaxfehler (2)

- Leider sind die Fehlermeldungen nicht immer so klar. Schreibt man z.B. `static` groß, so erhält man:

```
Hello.java:4: <identifier> expected
    Static public void main(String[] args) {
           ^
1 error
```

Der Compiler liest das Programm von vorne nach hinten und gibt die Fehlermeldung an der ersten Stelle aus, an der keine gültige Fortsetzung mehr möglich ist. Das Wort "`Static`" wäre als Name einer Klasse möglich, und diese Klasse könnte als Datentyp eines Attributs oder Ergebnis-Datentyp einer Methode angegeben sein, deren Name (Bezeichner, "identifier") hier fehlt. Fügt man aber einen solchen Bezeichner ein (z.B. "`X`"), beschwert sich der Compiler über ein fehlendes Semikolon. Fügt man auch das ein, merkt er endlich, dass er gar keine Klasse "`Static`" kennt: Er analysiert zunächst die reine Syntax und schaut erst später die Bedeutung der Namen nach, dazu kommt er im Fehlerfall nicht mehr.

Syntaxfehler (3)

- Es ist also möglich, dass der Fehler nicht an genau der Stelle liegt, wo er gemeldet wird, sondern davor.

  Bei modernen Compilern kann er nicht dahinter liegen, da das Programm von vorne nach hinten verarbeitet wird, und der Fehler an der ersten Stelle gemeldet wird, wo keine gültige Fortsetzung mehr möglich ist.

- Meist ist er nur kurz davor.

  Theoretisch könnte er beliebig weit davor liegen, aber es muss dann doch eine Beziehung geben, z.B. eine Variable, die man hier verwendet, und die man vorher falsch deklariert hat; oder eine Klammer, die man hier schließt, und vorher vergessen hat, zu öffnen.

Syntaxfehler (4)

- Da der Compiler bei einem Fehler nicht aufhört, sondern auch den Rest des Programms noch analysiert, kann es "Folgefehler" geben: Wenn man den ersten Fehler beseitigt hat, verschwinden sie automatisch.
- Beispiel: Schlüsselwort "`class`" vergessen:

```
Hello.java:3: class, interface, or enum expected
Hello {
^
Hello.java:4: class, interface, or enum expected
    static public void main(String[] args) {
                       ^
Hello.java:6: class, interface, or enum expected
    }
    ^
```

Syntaxfehler (5)

- Vergisst man z.B. die Zeichenketten-Konstante in Zeile (5) wieder mit " zu schliessen, so erhält man:

```
Hello.java:5: unclosed string literal
        System.out.println("Hello, World!);
                           ^
Hello.java:5: ';' expected
        System.out.println("Hello, World!);
                                           ^
Hello.java:7: reached end of file while parsing
}
 ^
3 errors
```

Die erste Meldung trifft zu und ist hilfreich, die anderen sind Folgefehler.
Bei der Erholung von einem Fehler werden Teile des Programms übersprungen, offenbar gehörte dazu auch die } in Zeile (6), so dass er dann das Dateiende erreicht, wenn noch eine { offen ist.

Syntaxfehler (6)

- Schreibt man "`System`" falsch, z.B. klein, so erhält man folgende Meldung:

```
Hello.java:5: package system does not exist
        system.out.println("Hello, World!");
               ^
1 error
```

  Diese Meldung ist nicht ganz zutreffend, eigentlich sollte "`system`" ein Klassenname sein. Das weiss der Compiler aber nicht. Pakete ("packages") sind Zusammenfassungen von Klassen (ähnlich wie Verzeichnisse zur Strukturierung von Dateien benutzt werden). Für den Zugriff auf die Klassen in einem Paket (und Unterpakete) wird auch die "`.`"-Notation benutzt. Insofern ist es nicht abwegig, dass der Compiler hier vermutet, "`system`" könnte ein Paket sein.

Syntaxfehler (7)

- Schreibt man "`out`" falsch, z.B. groß:

```
Hello.java:5: cannot find symbol
symbol : variable Out
location: class java.lang.System
        System.Out.println("Hello, World!");
               ^
1 error
```

  In der Klassendeklaration der Bibliotheksklasse "`System`" gibt es keine Variable (kein Attribut, Datenelement) "`Out`". Groß-/Kleinschreibung ist wichtig. Der Compiler weist nicht darauf hin, dass es "`out`" gibt. Manche IDEs machen das (Fehlermeldungen sind auch vom Compiler abhängig).

  Die Klasse "`System`" gehört zum Paket "`java.lang`", deswegen lautet ihr voller Name "`java.lang.System`". Man kann sie aber auch einfach mit "`System`" ansprechen (ebenso wie alle anderen Klassen in diesem speziellen Paket).

Syntaxfehler (8)

- Es ist wichtig, Fehler wirklich aufzuklären: Wenn Sie eine unverständliche Fehlermeldung erhalten, kopieren Sie sich das Eingabe-Programm, damit Sie später ggf. den Dozenten fragen können.

  Ansonsten ist eine übliche Methode, wenn man eine Fehlermeldung nicht versteht, dass man das Programm modifiziert. Die Hoffnung dabei ist, dass entweder der Fehler verschwindet, oder man beim modifizierten Programm eine Fehlermeldung bekommt, die man besser versteht. Es kann dann aber passieren, dass man am Ende nicht mehr weiss, was genau das ursprüngliche Programm war. Dann wird es natürlich schwierig, den Fehler aufzuklären.

- Häufig hat man nur eine Kleinigkeit an einem Programm verändert, das vorher funktionierte. Dann muss der Fehler natürlich an der Änderung liegen.

  Auch dann wäre es nützlich, wenn man das alte Programm noch hätte.

Syntaxfehler (9)

- Ein Editor, der die Java-Syntax kennt, kann auch helfen:
  - Verschiedene syntaktische Kategorien werden unterschiedlich eingefärbt, z.B. Schlüsselworte anders als normale Bezeichner, und Kommentare auch speziell.

    Mit //-Kommentaren kann man kaum etwas falsch machen, aber Java hat noch eine zweite Syntax für Kommentare: von /* bis */. Hat man hier einen Fehler gemacht, überliest der Compiler einen großen Teil des Programms. Der Editor zeigt diesen Teil aber in einer speziellen Farbe an, so dass es ins Auge springt, dass etwas nicht stimmt.
  - Bei manchen Editoren kann man mit `{...}` geklammerte Abschnitte auch zuklappen.
  - Viele Editoren schlagen automatisch eine Einrückung entsprechend der Klammerstruktur vor, so dass man leicht erkennt, wenn man eine Klammer vergessen hat.

Ausführen des Programms (1)

- Nehmen wir an, die `.class`-Datei (Bytecode-Datei) wurde erfolgreich erzeugt. Nun muss man sie ausführen.
- Das geschieht mit dem Befehl

  ```
  java Hello
  ```

  Wieder einzugeben in der Windows Eingabeaufforderung (Command Prompt) oder der UNIX Shell.
- Falls man beim `java`-Aufruf die Endung "`.class`" mit angibt, erhält man folgende Meldung:

  ```
  Error: Could not find or load main class
         Hello.class
  ```

  Beim Open JDK ist die Meldung etwas anders und deutlich länger (u.a. Exception `java.lang.NoClassDefFoundError`).

Ausführen des Programms (2)

- Falls die `class`-Datei nicht gefunden wird, obwohl sie im aktuellen Verzeichnis steht, prüfe man den `CLASSPATH`.

  `CLASSPATH` ist eine Umgebungsvariable, die Verzeichnisse enthält, in denen das Programm `java` nach `class`-Dateien sucht. Man kann ihn unter Windows mit `echo %CLASSPATH%` und unter UNIX mit `echo $CLASSPATH` ausgeben. Es ist in Ordnung, wenn er nicht definiert ist, dann sucht `java` auch im aktuellen Verzeichnis. Falls er aber definiert ist, schauen Sie ob "`.`" (das aktuelle Verzeichnis) ein Bestandteil ist und hängen Sie ggf. "`;.`" an. Man kann den `CLASSPATH` auch beim `java`-Aufruf setzen: "`java -cp . Hello`".

- Hat man die `main`-Methode anders genannt, bekommt man folgende Meldung:

```
Error: Main method not found in class Hello,
    please define the main method as:
    public static void main(String[] args)
```

Ausführen des Programms (3)

- Die `class`-Datei ist eine Binärdatei, keine Text-Datei.
- Wenn man versucht, sie mit einem Texteditor zu öffnen, wird ein Wirrwar von merkwürdigen Zeichen angezeigt.

  Eventuell erkennt der Editor auch, dass es keine Textdatei ist, und weigert sich, sie zu öffnen.
- Dies zeigt den Effekt der unterschiedlichen Interpretation einer Folge von Bytes: Für die Java Virtual Machine (im Programm `java`) sind es sinnvolle Instruktionen.

  Falls man den Inhalt er Datei in lesbarer Form ausgegeben haben will, kann man den Java Disassembler verwenden: Rufen Sie "`javap Hello`" auf, um die Klasse und von außen zugreifbare Komponenten der Klasse anzuzeigen, mit "`javap -c Hello`" werden zusätzlich die Maschinenbefehle der JVM angezeigt (nur für Experten interessant). Der Name "`javap`" kommt wohl von "print java class files". Disassembler: Umkehrabbildung zu Assembler.

Schlussbemerkung

- Machen Sie sich keine Sorgen, wenn Sie noch nicht alles voll verstanden haben.
- Ziel war es, einen ersten Eindruck von Java-Programmen zu geben.

  Und Sie in die Lage zu versetzen, den Compiler und den Bytecode-Interpreter zu verwenden, also Ihre Programme auszuführen.
- Es wird später alles noch ausführlicher und präziser erklärt.

  Das braucht dann aber einige Zeit, und würde vermutlich vielen Teilnehmern wenig Spaß machen, wenn man nicht schon vorher einen Eindruck hat, was man damit anfangen kann.
- Stellen Sie aber Fragen, und helfen Sie damit auch, diese Folien noch weiter zu verbessern.