Objektorientierte Programmierung

Kapitel 1: Einführung

Prof. Dr. Stefan Brass

Martin-Luther-Universität Halle-Wittenberg

Wintersemester 2018/19

http://www.informatik.uni-halle.de/~brass/oop18/

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 (älterer PC, heute typisch 4 GByte).

  In der Informatik steht "M"/"Mega" meist für 2^{20}, d.h. $1024 * 1024$, also etwas mehr als eine Million. "G"/"Giga" entsprechend für 2^{30} (ca. 1.07 Milliarden).

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).

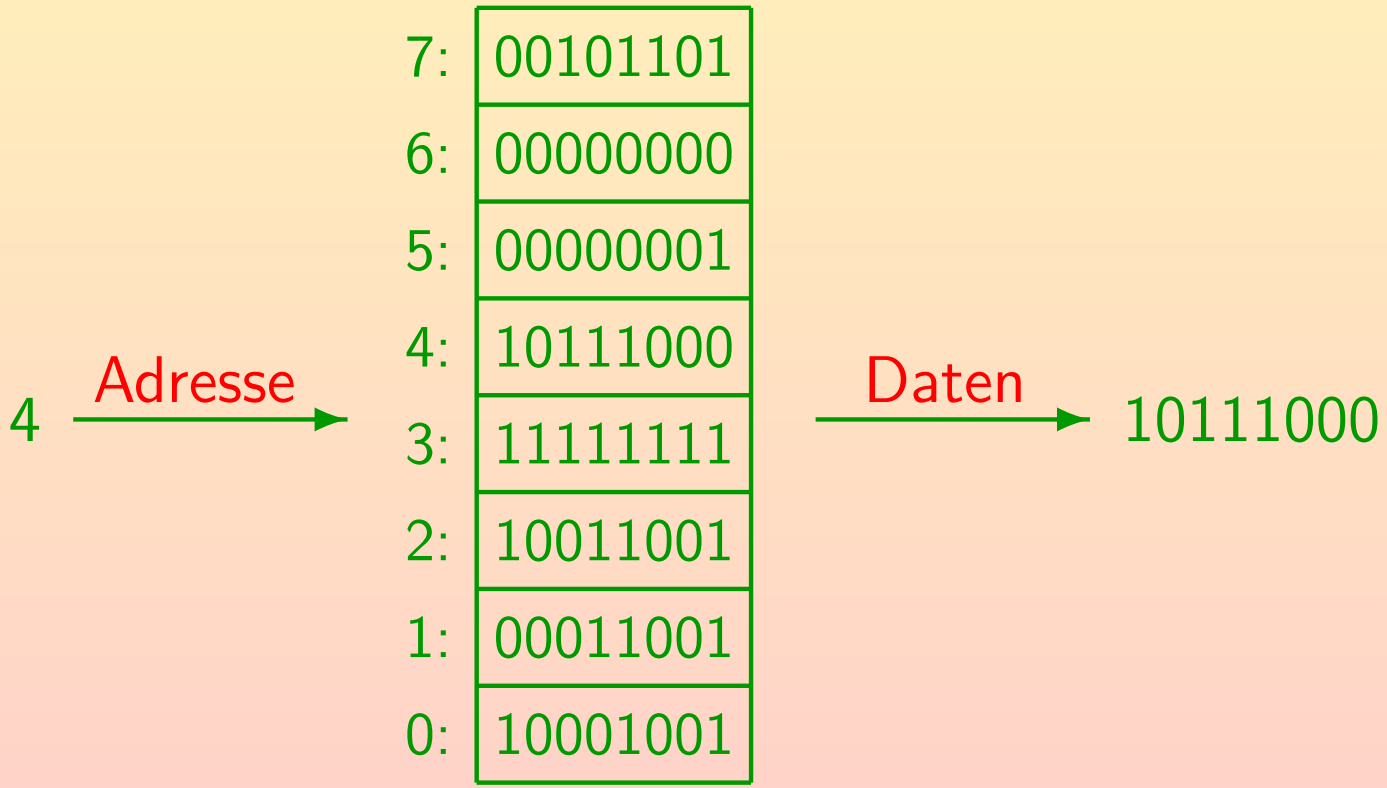
  Man kann Bytes zu größeren Einheiten (Worte) zusammenfassen, manche CPUs können auch einzelne Bits ansprechen. Deswegen kann man nicht genau sagen, was eine einzelne Speicherzelle ist.

- 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.
  Man kann die Bitmuster auch ganz anders interpretieren, z.B. als Buchstaben oder Maschinenbefehle (s.u.).

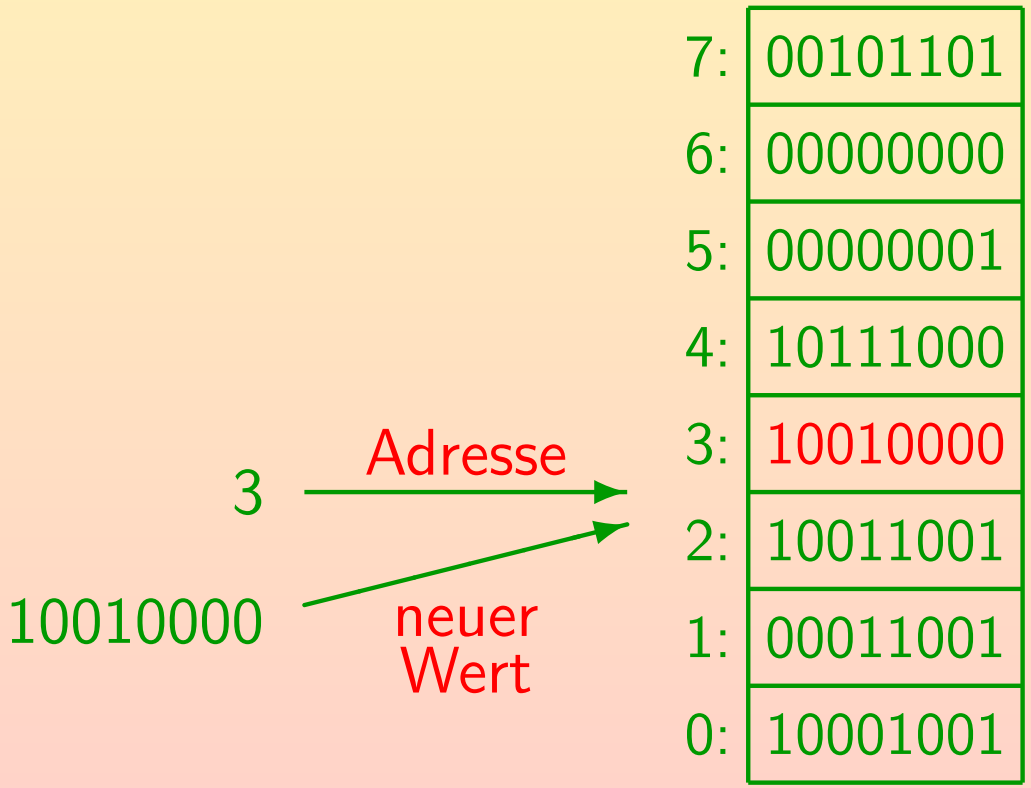
Computer: Minimal-Hardware (3)

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.

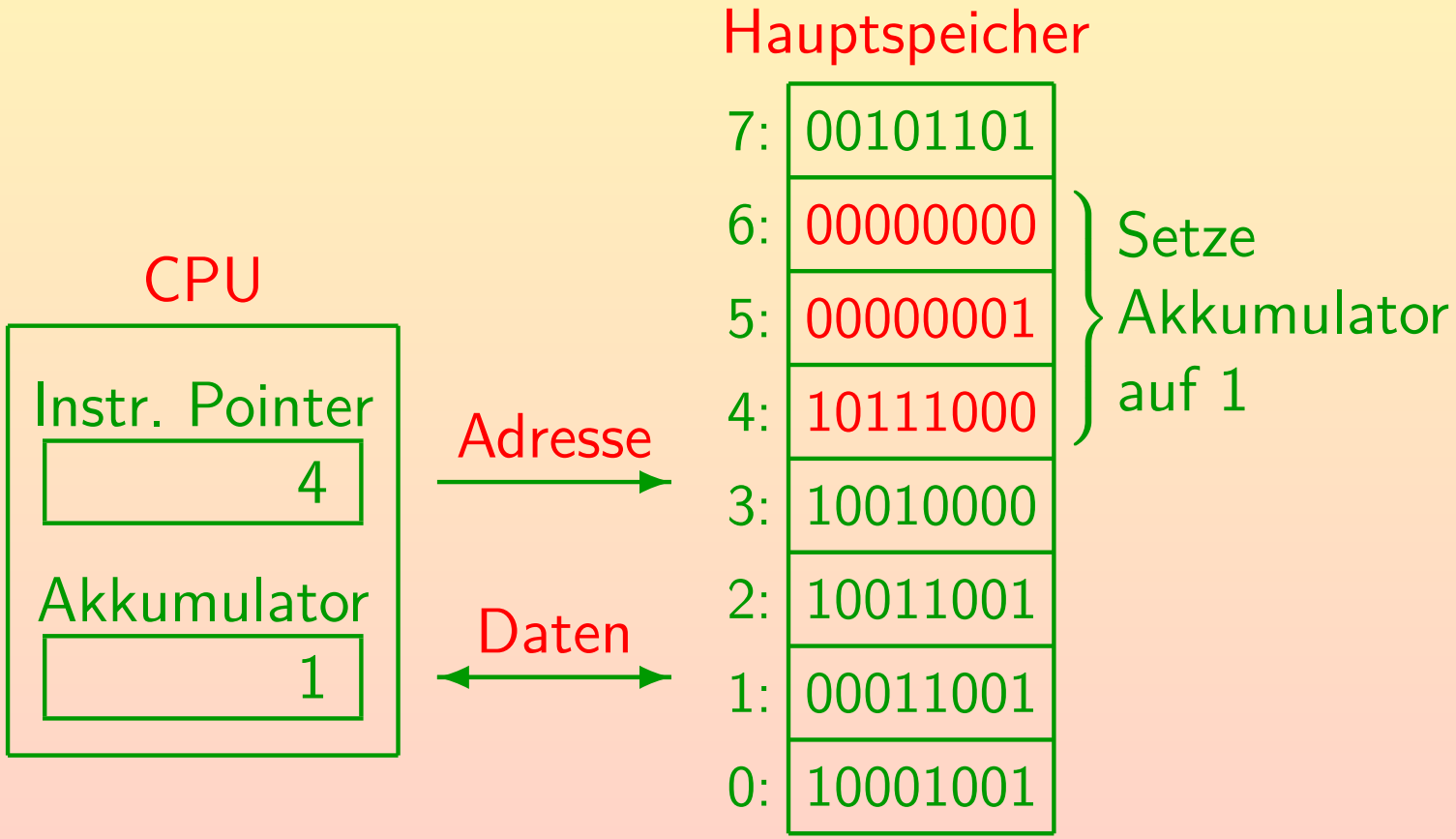
Maschinencode (2)

- 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.

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.

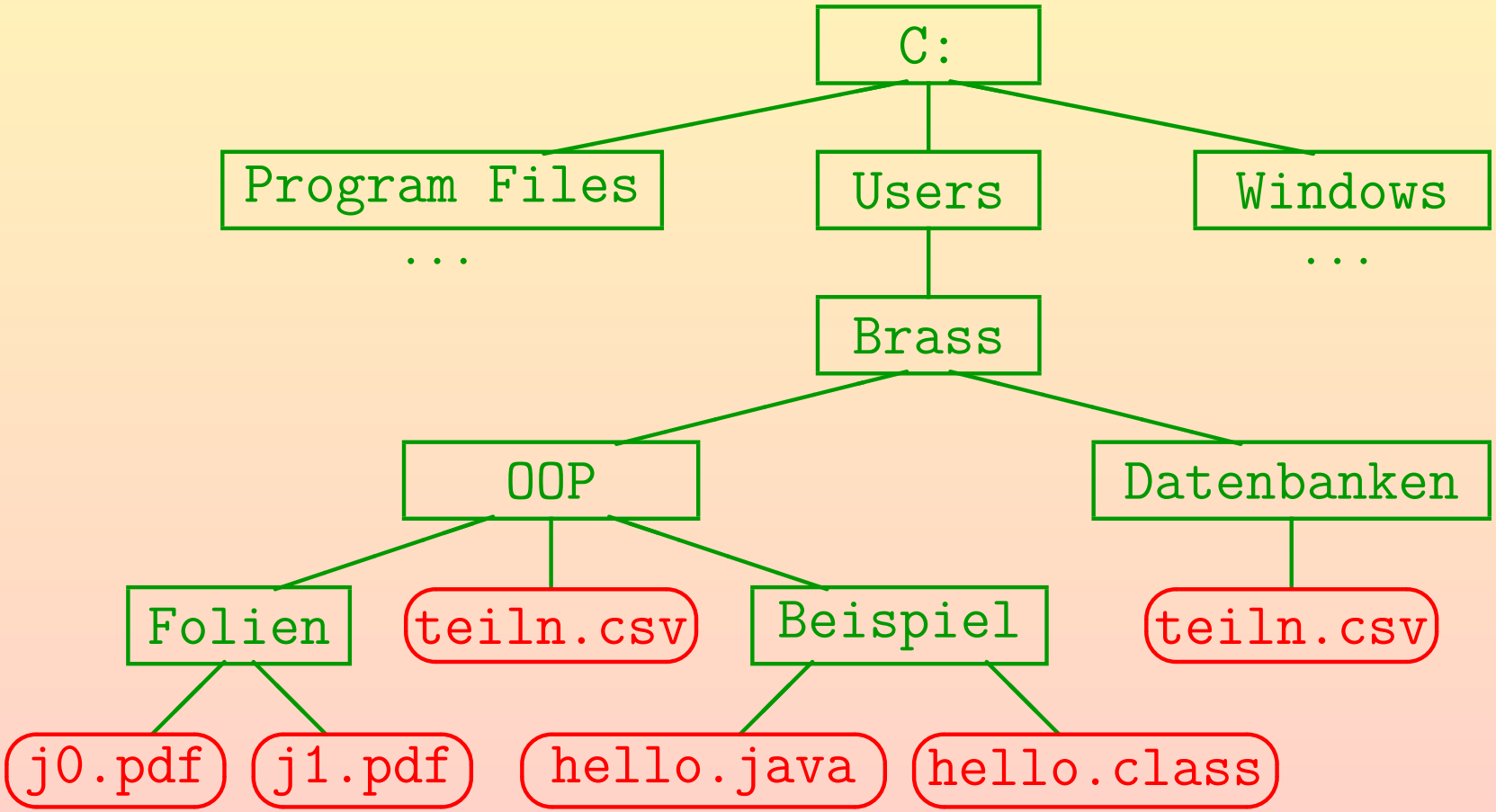
Dateien, Verzeichnisstruktur (1)

- 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)
- Java SE 8: 18.03.2014

  U.a. Lambda-Ausdrücke, Default Methods, bessere Typ-Inferenz.

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?

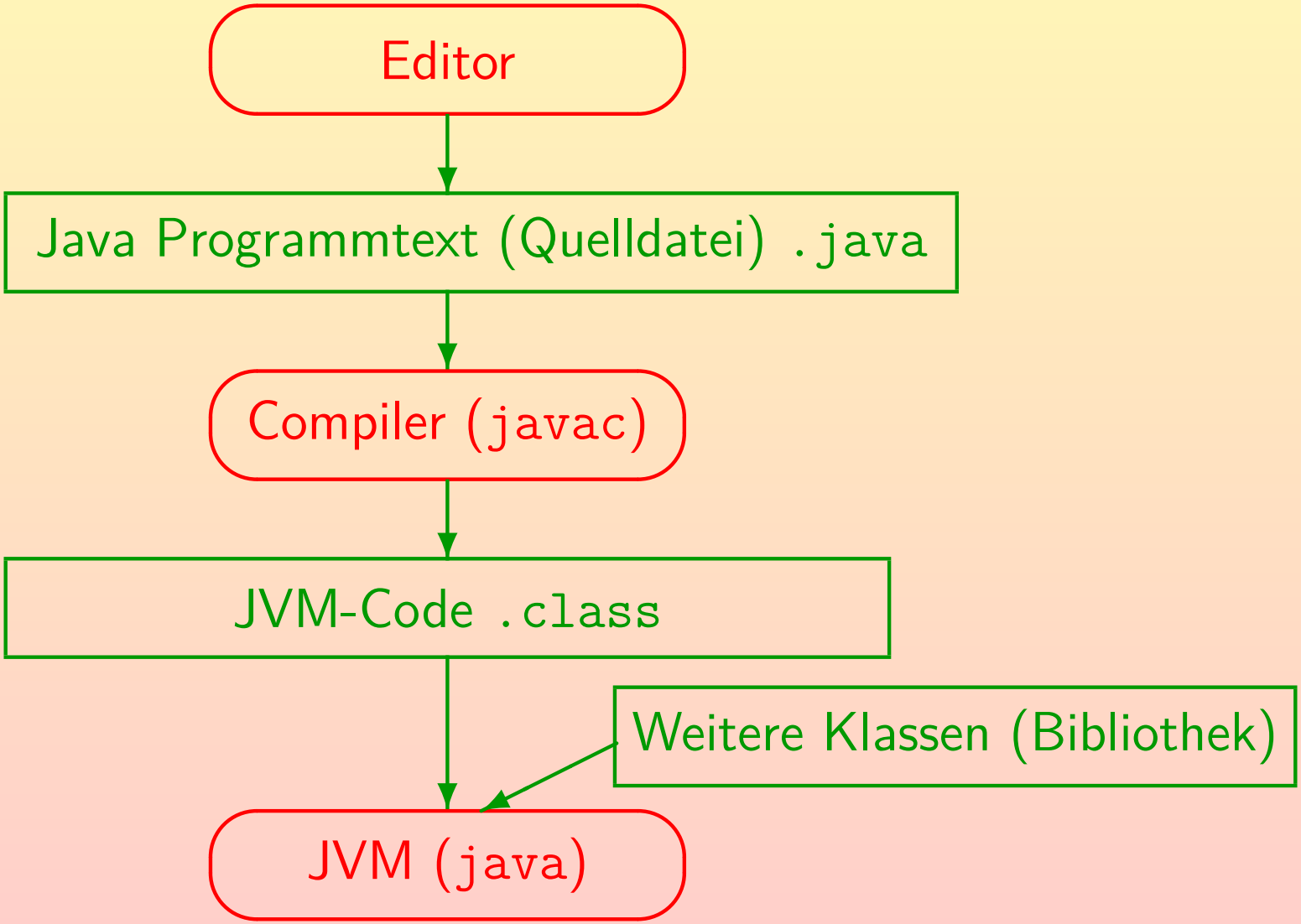
Java (1)

- 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:

- Der Software-Anbieter muss das Programm nicht für jede Plattform (Hardware + Betriebssystem) einzeln bereitstellen: `class`-Dateien sind Plattform-unabhängig.

  Besonders wichtig für Ausführung in Browsern (Applets).

- Der Software-Anbieter muss den 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.

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 genaue Einrückung mit beliebig vielen Leerzeichen oder Tabulator-Zeichen ist egal, die Zeilenaufteilung (größtenteils) auch.

Sie brauchen das Programm jetzt noch nicht 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. (Die erweiterte Version der Folien enthält einige Erklärungen für Neugierige.)

Weitere Beispiel-Programme

- Das erste Beispiel war eine “Konsolen-Anwendung”.
- Sie wirkt etwas antiquiert, weil sie keine eigene graphische Benutzeroberfläche (GUI: “Graphical User Interface”) hat, sondern nur Zeichen auf ein Terminal-Fenster ausgibt.

  Früher kommunizierte man mit Computern über solche Terminals in Hardware.
- Selbstverständlich kann man auch Programme mit graphischen Benutzeroberflächen in Java schreiben.

  Darin ist Java sogar besonders stark, weil es eine Bibliothek zur portablen Programmierung von Benutzeroberflächen enthält. Allerdings muss man mehr von Java wissen, und größere Bibliotheken kennen, um solche Programme wirklich zu verstehen. Es soll ja nicht (lange) beim stupiden “Abtippen” bleiben.
- Java ist auch die Grundlage der “App”-Programmierung auf Android Smartphones.

Inhalt

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.

- Außerdem könnte es Syntaxfehler geben, falls Sie beim Abtippen des Programms einen Fehler gemacht haben. Diese werden unten noch ausführlich diskutiert.

Ausführung des Programms (3)

- 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. Offiziell heißt der Befehl "Java Program Launcher".

- 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`).

Ausführung des Programms (4)

- 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)
```

Format der `class`-Datei

- 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 der 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.

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.
- Der Compiler versteht das Programm nur, wenn Sie die Regeln der Java-Syntax exakt befolgen.

  Es kommt auf jedes Zeichen an. Während ein Mensch den Sinn eines Textes trotz einigen Tippfehlern und Ungenaugigkeiten verstehen kann, ist der Compiler eine Maschine ohne Verstand. Er verarbeitet den Text, solange dieser genau den Vorgaben entspricht. Sonst verweigert er die Arbeit.
  Der Vorteil dabei ist, dass es keine Interpretations-Spielräume bleiben.
- Im positiven Fall sagt der Compiler dagegen nichts und erstellt die Datei `Hello.class`.

  Dies ist das Übersetzungsergebnis (die JVM-Maschinenbefehle).
  Später wird erklärt, wie diese dann ausgeführt werden.

Syntaxfehler (2)

- 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 "`;`" einzufügen.

- Korrigieren Sie den Fehler in der Quelldatei "`Hello.java`" und rufen Sie den Compiler erneut auf.

  Sie können das Editor-Fenster offen lassen, aber vergessen Sie nicht, abzuspeichern. Der Compiler liest das Programm ja aus der Datei.

  Gerade als Anfänger braucht man oft mehrere Anläufe, bis ein Programm fehlerfrei durch den Compiler läuft. Mit der Zeit macht man weniger Fehler.

Syntaxfehler (3)

- 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 (4)

- 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 (Programm-Bestandteil), die man hier verwendet, und die man vorher falsch deklariert hat; oder eine Klammer, die man hier schließt, und vorher vergessen hat, zu öffnen.

- Es ist möglich, dass nach der Beseitigung eines Fehlers neue Fehler gemeldet werden.

  Eventuell auch weiter vorne, weil der Compiler den Text in mehreren Durchgängen ("Phasen") bearbeitet.

Syntaxfehler (5)

- 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.

  Damit der Compiler den Rest des Textes weiter analysieren konnte, musste er Annahmen darüber machen, was Sie gemeint haben könnten.
  Wenn diese Annahmen falsch waren, führen sie zu weiteren Fehlermeldungen, weil der Rest des Textes nicht dazu passt.

- Kümmern Sie sich zuerst um den ersten angezeigten Fehler und ignorieren Sie den Rest.

  Früher dauerte ein Compilerlauf lange, eventuell musste man bei Großrechnern auch Stunden warten, bis der Programmlauf ausgeführt wurde.
  Da war es wichtig, nicht nur einen Fehler gemeldet zu bekommen.

Syntaxfehler (6)

- 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
    }
    ^
```

- Hier gibt es nur einen Fehler, aber drei Fehlermeldungen.

  Lassen Sie sich also von einer langen Liste von Fehlermeldungen nicht entmutigen. Lange Programme sollten Sie in mehreren Ausbaustufen eingeben, und jeweils die Compilierbarkeit testen. So vermeiden Sie, dass Sie wirklich viele Fehler auf einmal haben.

Syntaxfehler: Weitere Beispiele (1)

- 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: Weitere Beispiele (2)

- 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: Weitere Beispiele (3)

- 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).

Aus Fehlern lernen!

- Es ist wichtig, Fehler wirklich aufzuklären: Geben Sie sich nicht damit zufrieden, dass es zufällig funktioniert.

  Wenn Sie den Fehler nicht verstanden haben, werden Sie ihn wieder machen. Außerdem funktioniert das Programm vielleicht nur für die eine Eingabe, die Sie ausprobiert haben. Es bleibt auch ein Gefühl der Unsicherheit.

- Wenn Sie eine unverständliche Fehlermeldung erhalten, kopieren Sie sich das Programm (um ggf. später zu fragen).

  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.

Umgang mit Fehlermeldungen (1)

- Was machen Sie, wenn der Compiler

  "`Hello.java:23: error: Hrmpf!`"

  meldet (Sie verstehen es nicht)?

  Das kommt nicht häufig vor, meist sind die Fehlermeldungen verständlich.

  Im Laufe der Zeit lernt man auch immer mehr Fehlermeldungen kennen.

  - Schauen Sie sich die Zeile 23 an, und auch die vorangehende Zeile: Vielleicht sehen Sie ja, was nicht stimmt.
  - Kopieren Sie die Java-Quelldatei mit dem Fehler: Spätestens in der Übung soll es aufgeklärt werden.
  - Überlegen Sie, was Sie zuletzt geändert haben.

    Wenn Sie die letzte Version der Datei noch haben (die noch durch den Compiler lief), muss der Fehler ja im neuen Teil stecken (unter Linux können Sie zwei Dateien mit dem Befehl `diff` vergleichen).

Umgang mit Fehlermeldungen (2)

- Maßnahmen bei unverständlichen Fehlermeldungen, Forts.:
  - Benutzen Sie einen Editor, der Kommentare, Schlüsselworte und Zeichenketten farbig markiert. Prüfen Sie die Farben in der Umgebung des Fehlers.

    Besonders die Verwendung von Schlüsselworten als normale Bezeichner könnte zu unverständlichen Fehlermeldungen führen.
  - Prüfen Sie die Klammerstruktur.

    Die meisten Editoren haben einen Befehl, mit dem Sie von einer Klammer zur zugehörigen Klammer springen können. Sie können häufig auch die Einrückung der Programm-Zeilen automatisch machen lassen: Bei Eclipse wählen Sie zuerst den ganzen Programmtext aus durch drücken von `Ctrl+A` (`Edit→Select All`). Anschließend drücken Sie `Ctrl+I` (`Source→Correct Indentation`), oder wählen `Source→Format` für noch mehr automatische Formatierung. Sie können dann besser sehen, wo etwas nicht passt.

Umgang mit Fehlermeldungen (3)

- Maßnahmen bei unverständlichen Fehlermeldungen, Forts.:
  - Prüfung der Klammerstruktur, Forts.:

    Wenn Sie die Formatierung Ihres Programms nicht automatisch machen können oder wollen, können Sie auch probehalber am Dateiende eine "}" hinzufügen, und zur passenden Klammer { (A) springen. Wenn das funktioniert, haben Sie eine Klammer, die nicht zu geht. Wechseln Sie dann zu der Klammer } (B), von der Sie annehmen, dass sie eigentlich die Klammer { (A) schließen müßte. Springen Sie nun von Klammer } (B) wieder zur zugehörigen Klammer { (C). Vermutlich ist dies die Klammer, die nicht zugeht. Notfalls müssen Sie die Sache nochmal iterieren: Eigentlich beabsichtigte zugehörige Klammer finden, und zur tatsächlich zugehörigen Klammer springen. Der Suchbereich wird immer kleiner. Statt einer fehlenden schließenden Klammer könnte es natürlich auch eine vergessene öffnende Klammer geben. Das kann man prüfen, indem man am Dateianfang eine "{" einfügt, u.s.w.

Umgang mit Fehlermeldungen (4)

- Maßnahmen bei unverständlichen Fehlermeldungen, Forts.:
  - Haben Sie abgespeichert?

    Bezieht sich die Fehlermeldung überhaupt auf die Datei, die Sie anschauen?
  - Markieren Sie die Zeile als Kommentar (durch Voranstellen von //: “auskommentieren”). Sie wird dann vom Compiler ignoriert. Tritt der Fehler noch immer auf?

    Sie können so nach und nach immer mehr Programmteile entfernen. Wenn nach Entfernung von Zeile X der Fehler nicht mehr auftritt, wird Zeile X wohl etwas mit dem Fehler zu tun haben. Sie müssen natürlich darauf achten, dass die Klammerstruktur noch stimmt: Wenn Sie eine Zeile mit einer “{” auskommentieren, beschwert sich der Compiler anschließend wegen der zugehörigen “}”.

Umgang mit Fehlermeldungen (5)

- Maßnahmen bei unverständlichen Fehlermeldungen, Forts.:
  - Suchen Sie mit Google nach "`Java error Hrmpf`".

    Vielleicht hat schon jemand diese Fehlermeldung gehabt, und jemand anders hat ihm einen Tipp gegeben, was sie bedeutet.
  - Posten Sie die Fehlermeldung und das zugehörige Programmstück im StudIP-Forum zu dieser Vorlesung.
  - Falls Sie eine IDE benutzen (z.B. Eclipse), schauen Sie, ob ein "Quick Fix" vorgeschlagen wird.

    Diese Korrektur muss nicht immer funktionieren, aber wahrscheinlich bekommen sie anschließend wenigstens eine andere Fehlermeldung.
  - Probieren Sie einen alternativen Java-Compiler.
  - Fragen Sie den Professor!

Entwicklungsumgebungen

- Alternative zu Einzelwerkzeugen (Editor, Compiler, ...): IDE ("Integrated Development Environment").
- Vorteile IDE:
  - Wirkt moderner, macht mehr automatisch, enthält Hilfen.
- Nachteile IDE:
  - Funktionsumfang kann zu Beginn erschlagen.

    Es gibt viel, was man am Anfang nicht unbedingt braucht, und was eher im Wege steht (Konzentration auf das Wesentliche fällt schwerer).
  - Man muss auch ohne die Hilfen programmieren können.

    Die Hilfen dienen zur Produktivitätssteigerung für den Profi. Wenn man damit fehlendes Wissen über die Programmiersprache ausgleicht, erwirbt man dieses Wissen eventuell langsamer. Klausur ohne solche Hilfe!
  - Man kann nicht mit ggf. bekanntem Editor arbeiten.

Eclipse (1)

- Eclipse ist eine bekannte IDE (eigentlich nur Rahmen für Plugins, aber Verwendung als Java-IDE ist Standardfall).

  Eclipse können Sie bei [http://eclipse.org/downloads/] herunterladen.

  Wählen Sie unter "Package Solutions" die "Eclipse IDE for Java Developers".

  Die Installation beschränkt sich normalerweise auf das Entpacken des Archivs.

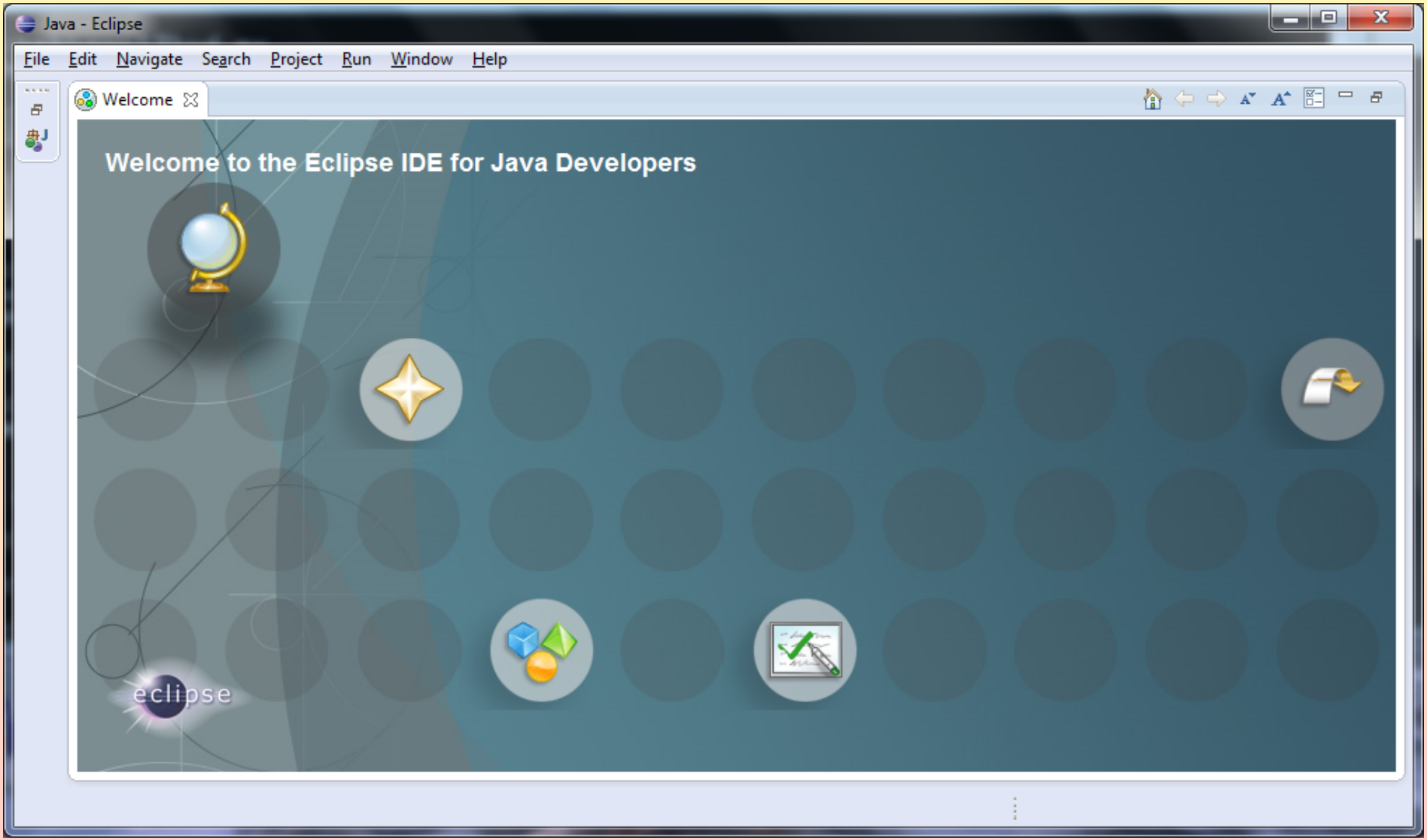
  Eclipse bringt einen eigenen Compiler mit, aber das "Java Runtime Environment" (JRE) muss schon auf Ihrem Rechner installiert sein, ggf. von [http://www.oracle.com/technetwork/java/javase/downloads/] holen.

- Rufen Sie die Anwendung `eclipse` im Ordner `eclipse` auf.

- Beim ersten Start werden Sie gefragt, wo der Workspace abgelegt werden soll, das ist ein Verzeichnis für die Dateien Ihres Projektes (z.B. `.java` und `.class`-Dateien).

  Auf dem Welcome-Bildschirm gibt es Links zu Tutorials und anderer Information. Sie können später jederzeit unter `Help`→`Welcome` wieder öffnen.

Eclipse (2)


Java - Eclipse
File Edit Navigate Search Project Run Window Help
Welcome
Welcome to the Eclipse IDE for Java Developers
eclipse

Eclipse (3)

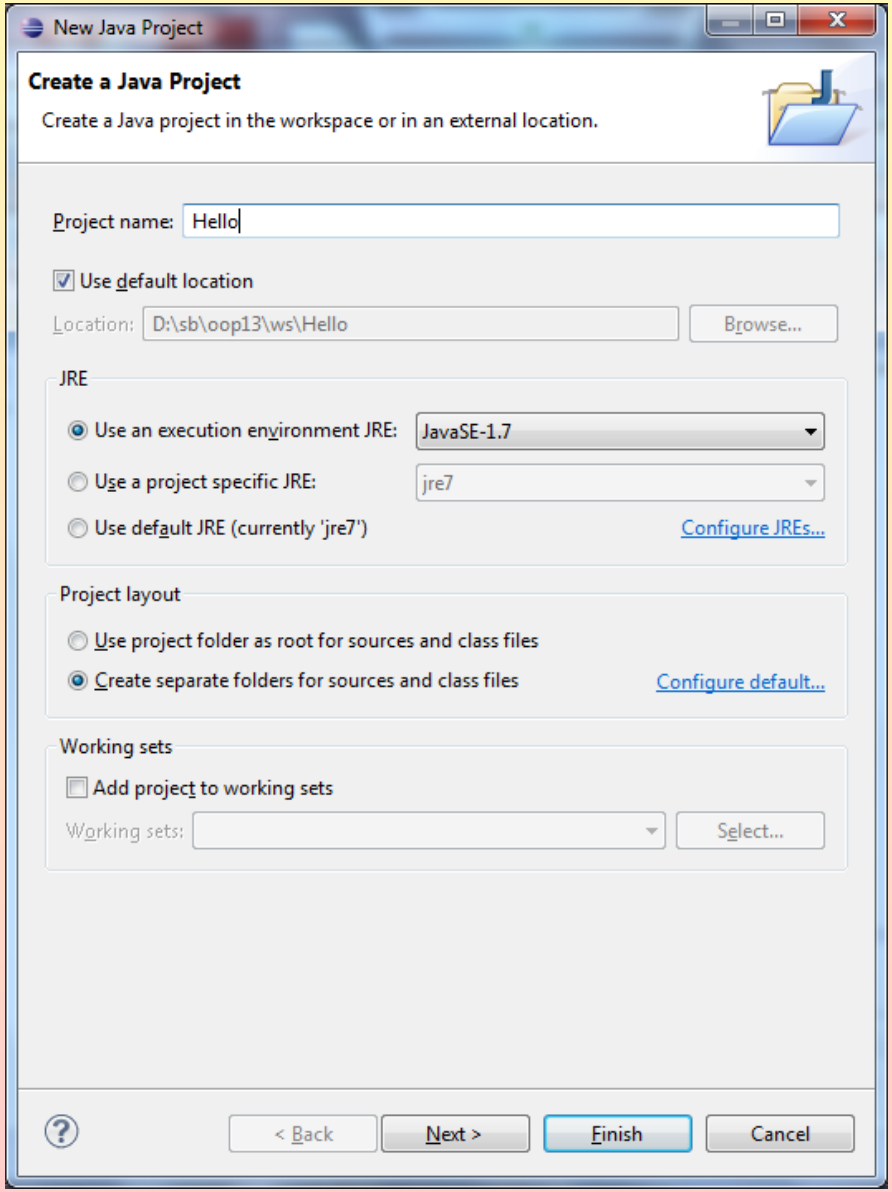
- Legen Sie ein neues Projekt an: `File→New→Java Project`.

  Sie müssen einen Projektnamen (z.B. "`Hello`") eingeben und können dann auf "`Finish`" klicken.

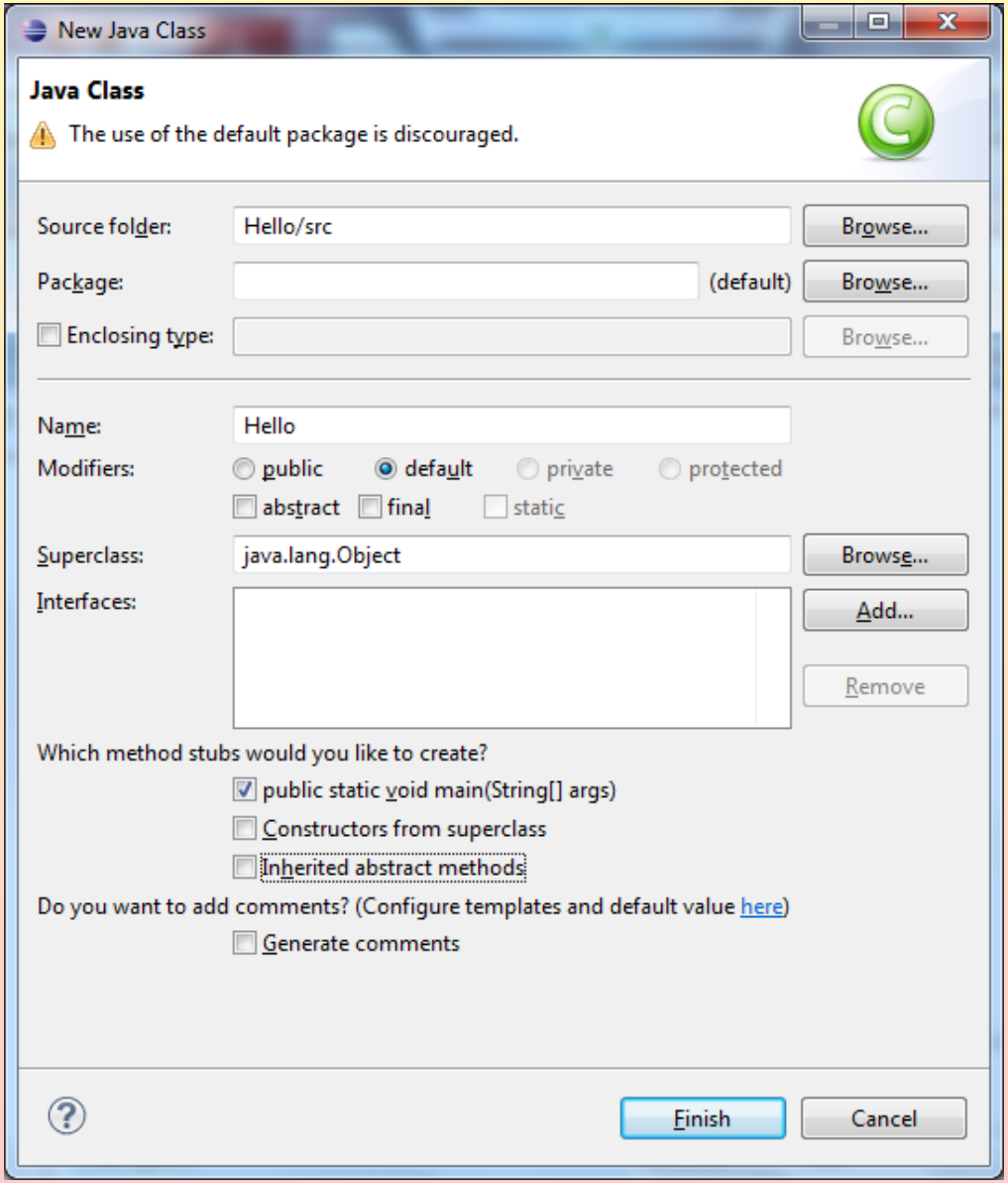
- Legen Sie unter `File→New→Class` eine Klasse an.

  Sie müssen nur den Namen der Klasse eingeben, den Rest können Sie so lassen. Wenn Sie wollen können Sie bei "Modifiers" "`default`" statt der Voreinstellung "`public`" wählen (ist im Moment egal). Wenn Sie sich Tipp-Arbeit ersparen wollen, können Sie bei "Which method stubs would you like to create?" "`public static void main(String[] args)`" auswählen. Eclipse legt nun eine Datei mit dem Klassennamen und der Endung `.java` an. Es ist wichtig, dass die Datei unter "Workspace"/"Projekt"/"src", ggf. noch "(default package)" (das ist aber kein eigenes Verzeichnis) angelegt wird. Wenn Sie z.B. direkt unter "Projekt" steht, bekommen Sie später den Fehler "Editor does not contain a main type". Sie können die Datei aber im "Package Explorer" rechts verschieben.

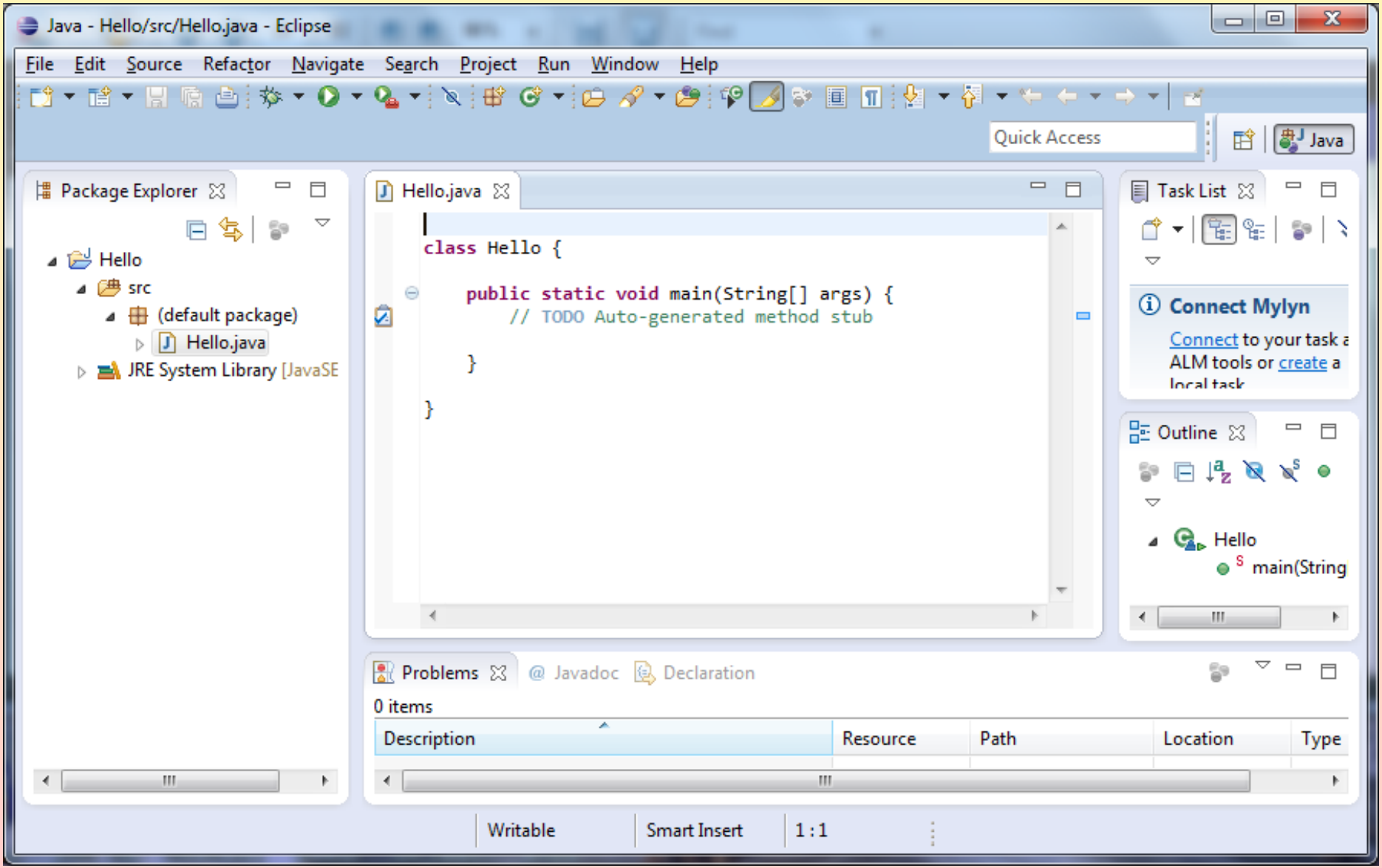
Eclipse (4)


New Java Project
Create a Java Project
Create a Java project in the workspace or in an external location.
Project name: Hello
Use default location
Location: D:\sb\oop13\ws\Hello
Browse...
JRE
Use an execution environment JRE: JavaSE-1.7
Use a project specific JRE: jre7
Use default JRE (currently 'jre7')
Configure JREs...
Project layout
Use project folder as root for sources and class files
Create separate folders for sources and class files
Configure default...
Working sets
Add project to working sets
Working sets:
Select...
< Back
Next >
Finish
Cancel

Eclipse (5)


New Java Class
Java Class
The use of the default package is discouraged.
Source folder:
Hello/src
Browse...
Package:
(default)
Browse...
Enclosing type:
Browse...
Name:
Hello
Modifiers:
public
default
private
protected
abstract
final
static
Superclass:
java.lang.Object
Browse...
Interfaces:
Add...
Remove
Which method stubs would you like to create?
public static void main(String[] args)
Constructors from superclass
Inherited abstract methods
Do you want to add comments? (Configure templates and default value here)
Generate comments
Finish
Cancel

Eclipse (6)


Java - Hello/src/Hello.java - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help
Quick Access
Java
Package Explorer
Hello
src
(default package)
Hello.java
JRE System Library [JavaSE
Hello.java
class Hello {
public static void main(String[] args) {
// TODO Auto-generated method stub
}
}
Task List
Connect Mylyn
Connect to your task a
ALM tools or create a
local task
Outline
Hello
main(String
Problems
Javadoc
Declaration
0 items
Description
Resource
Path
Location
Type
Writable
Smart Insert
1 : 1

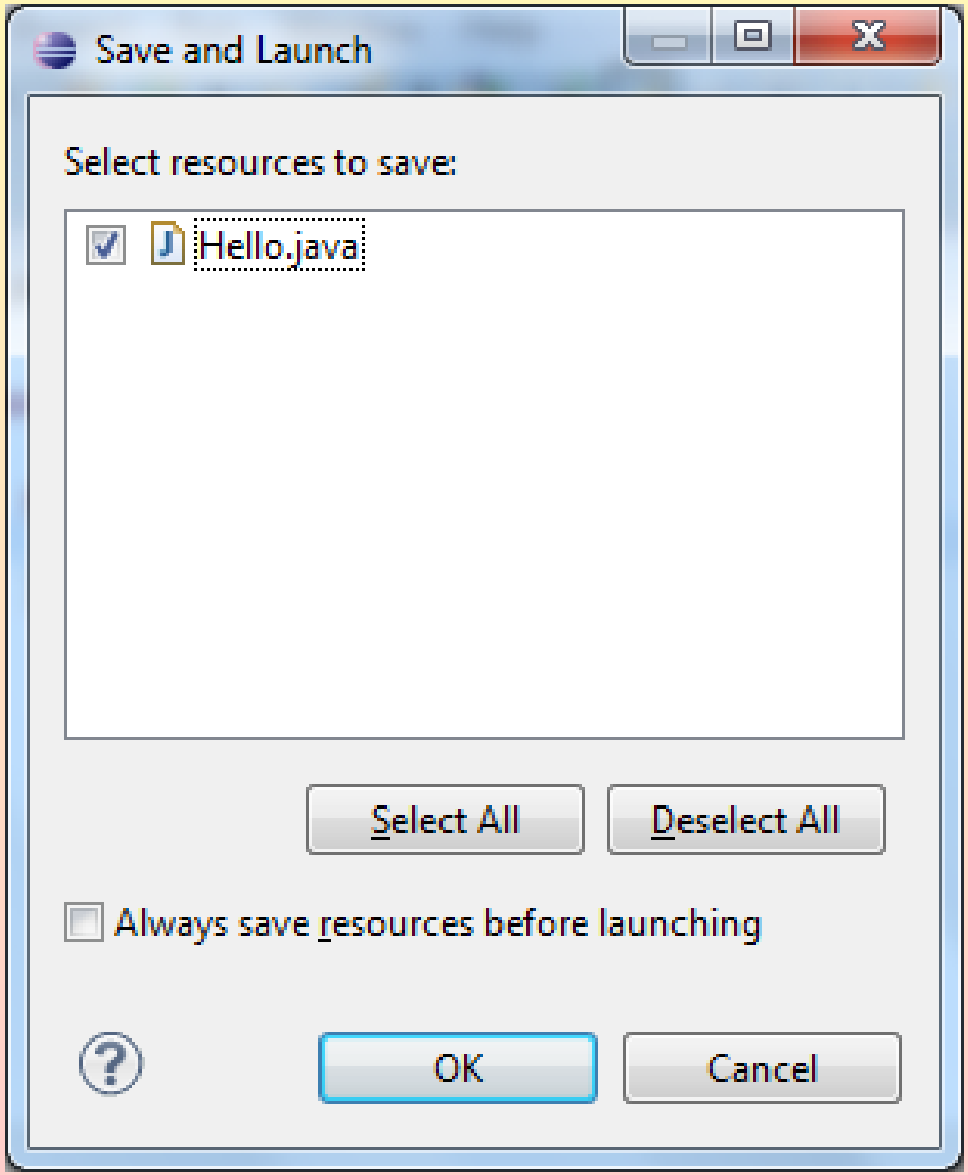
Eclipse (7)

- Vervollständigen Sie den Programmtext.

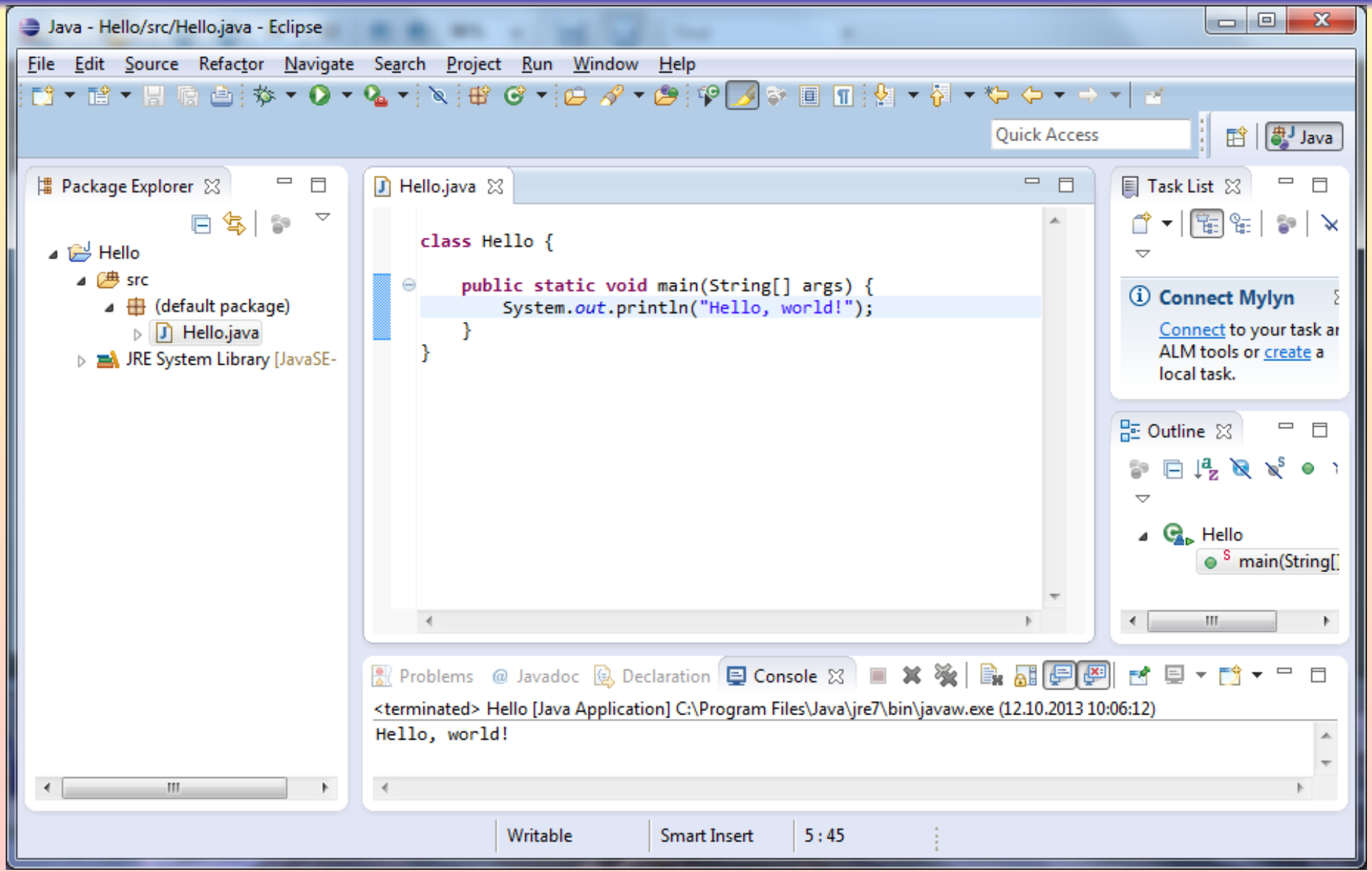
  Nachdem Sie z.B. "`System.`" getippt haben, öffnet sich ein Pop-up Menü, das die Komponenten dieser Klasse anzeigt. Sie können `out` auswählen. Einmal klicken zeigt eine Information, doppelt klicken oder "Enter" drücken übernimmt es. Sie können aber auch einfach weiter tippen. Wenn Sie nach `System.out.println` die "`(`" tippen, wird automatisch "`);`" eingefügt und es werden Ihnen Vorschläge für den Text zwischen den Klammern gemacht. Sie können aber einfach weitertippen (`"Hello ...`). Solange der Text nicht syntaktisch korrekt ist (wenn Sie das schließende Anführungszeichen " noch nicht getippt haben), findet sich oben rechts, vorn vor der Zeile und hinten im Scrollbar eine rote Markierung.

- Speichern Sie unter `File→Save` ab.
- Rufen Sie dann `Run→Run` auf, um die Datei zu compilieren und auszuführen.

Eclipse (8)


Save and Launch
Select resources to save:
Hello.java
Select All
Deselect All
Always save resources before launching
OK
Cancel

Eclipse (9)


Java - Hello/src/Hello.java - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help
Quick Access
Java
Package Explorer
Hello
src
(default package)
Hello.java
JRE System Library [JavaSE-
Hello.java
class Hello {
public static void main(String[] args) {
System.out.println("Hello, world!");
}
}
Task List
Connect Mylyn
Connect to your task a
ALM tools or create a
local task.
Outline
Hello
main(String[
Problems Javadoc Declaration Console
<terminated> Hello [Java Application] C:\Program Files\Java\jre7\bin\javaw.exe (12.10.2013 10:06:12)
Hello, world!
Writable
Smart Insert
5 : 45

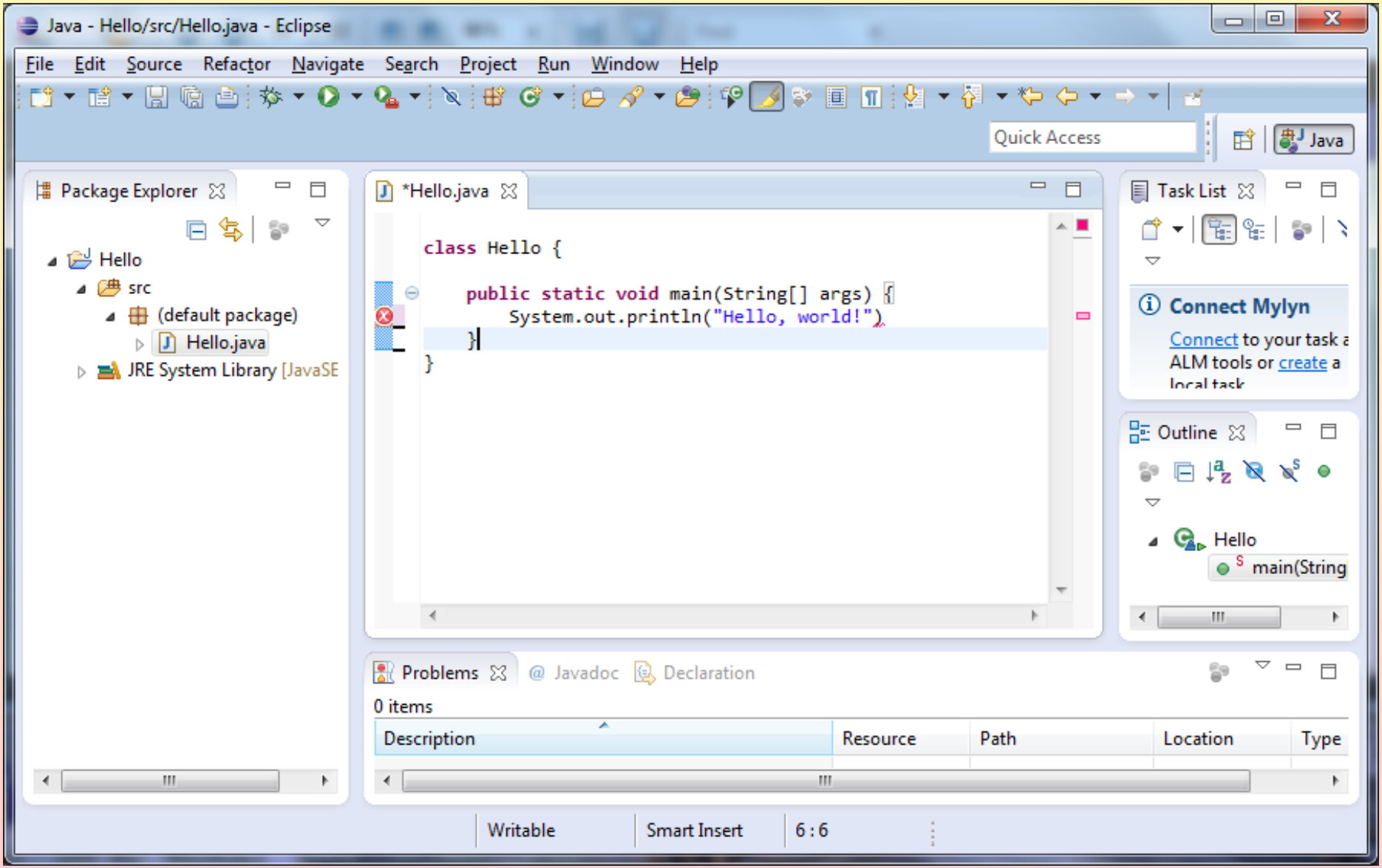
Eclipse (10)

- Falls die Quelldatei vor dem "Run→Run" nicht abgespeichert wurde, wird Ihnen automatisch angeboten, das zu tun.
- Die Ausgabe des Programm steht in dem Teilfenster ("View") unten (rechts). Sie müssen ggf. den Tab "Console" auswählen.

  Wenn Sie die Konsole geschlossen haben, können Sie sie mit dem Menupunkt "Window→Show View→Console" wieder bekommen.
- Falls bei der Übersetzung Syntaxfehler festgestellt wurden, stehen die Fehlermeldungen unten rechts unter "Problems".

  Das ist ein weiterer Tab in dem Bereich, in dem auch die Konsole angezeigt wird. Die Fehlermeldung erscheint aber auch, wenn der Mauszeiger über der roten Markierung vorn in der Zeile oder im Scrollbar steht.
  Falls Syntaxfehler festgestellt wurden, wird gefragt, ob der "Launch" dennoch fortgesetzt werden soll, aber das bringt in diesem Fall nichts (man bekommt doch den Fehler).

Eclipse (11)


Java - Hello/src/Hello.java - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help
Quick Access
Java
Package Explorer
Hello
src
(default package)
Hello.java
JRE System Library [JavaSE
*Hello.java
class Hello {
public static void main(String[] args) {
System.out.println("Hello, world!");
}
}
Task List
Connect Mylyn
Connect to your task a
ALM tools or create a
local task
Outline
Hello
main(String
Problems
Javadoc
Declaration
0 items
Description
Resource
Path
Location
Type
Writable
Smart Insert
6 : 6

Eclipse (12)

- Wenn der Mauszeiger über dem fehlerhaften Bereich ist (oder mit `Edit→Quick Fix`) werden mögliche Korrekturen angeboten.

  Die muss natürlich nicht das sein, was Sie beabsichtigt haben und kann auch zu neuen Problem führen. Außerdem gibt es nicht immer einen Korrekturvorschlag.

- Falls Sie sich die Aufteilung der Teilbereiche des Fensters durcheinander gebracht haben, können Sie mit "`Window→Reset Perspective...`" die Standard-Aufteilung wieder herstellen.

  Eventuell haben Sie auch eine andere Ansicht ("Perspective") gewählt, dann gehen Sie zu `Window→Open Perspective→Java`.

- Eclipse beenden können Sie mit `File→Exit`.

Schlussbemerkung

- Machen Sie sich keine Sorgen, wenn Sie noch nicht alles voll verstanden haben.
- Ziel war es, Sie in die Lage zu versetzen, Programme auszuführen. Dazu brauchen Sie:
  - Betriebssystem (z.B. Windows oder Linux mit Konsole)
  - Editor (z.B. `notepad`, `notepad++`, `gedit`)
  - Compiler/Übersetzer (`javac`)
  - Bytecode-Interpreter: Java Virtual Machine (`java`)
  - Optional alles in einer IDE (`Eclipse`, `Netbeans`, `BlueJ`)

  Am Ende sollten Sie beide Möglichkeiten beherrschen.
- Außerdem: Erster Eindruck von Java-Programmen.

  Es wird später alles noch ausführlicher und präziser erklärt.