Objektorientierte Programmierung

Kapitel 2: Erste Schritte in der Programmierung

Stefan Brass

Martin-Luther-Universität Halle-Wittenberg

Wintersemester 2013/14

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

Inhalt

Wiederholung (1)

- Am "Hello, World!" Beispiel haben wir gesehen, wie ein Programm ausprobiert werden kann:
  - Man benutzt einen Editor, um das Programm (den Quelltext in der Sprache Java) in eine Datei einzugeben, z.B. die Datei "`Hello.java`".

    Man kann Notepad benutzen, aber das ist wenig komfortabel. `Notepad++` wird häufig empfohlen, bzw. `gedit` unter Linux. Andere mächtige Editoren sind z.B. `gvim` und `Emacs`. `Word` eignet sich nicht für Programmcode, da es nicht nur den reinen Text, sondern auch zusätzliche Angaben zur Formatierung in der Datei speichert.
  - Dann benutzt man einen Compiler, um diesen Quelltext in Befehle für die Ziel-Maschine zu übersetzten, in unseren Fall die "Java Virtual Machine" (JVM).

    ```
    javac Hello.java
    ```

Wiederholung (2)

- Ausführung des "Hello, World" Beispiels, Forts.:
  - Der Compiler erzeugt eine Datei "`Hello.class`", die die JVM-Befehle (Java Bytecode) enthält.

    Sie ist binär (keine Textdatei). Wenn Sie sie in den Editor laden, sehen Sie nur ein großes Durcheinander von Zeichen, weil die Bits, die eigentlich JVM-Befehle sind, als Zeichencodes interpretiert werden. Wenn Sie die JVM-Befehle sehen wollen: "`javap -c Hello.class`".
  - Weil das Java-Programm nicht direkt in Maschinencode für die eigentliche CPU unseres Rechners übersetzt wurde, benutzen wir jetzt das Programm "`java`", das die JVM in Software simuliert:

    ```
    java Hello
    ```
  - Bei der Ausführung wird der Text "Hello, World!" auf den Bildschirm (Console, Terminal) ausgegeben.

Wiederholung (3)

- Die Verwendung von Editor, Compiler, und Interpreter für Java-Bytecode (JVM) wird Ihnen auch in der Übung gezeigt.

  Wenn Sie damit Probleme haben sollten, fragen Sie nach bzw. holen Sie sich Hilfe: Sie brauchen die Möglichkeit, Java-Programme ausprobieren zu können.

- Unter [http://ideone.com/] gibt es das auch im Web, das ist aber nur für einfache Programme zu Anfang geeignet.

- Mit dem Programm PuTTY [http://www.putty.org/] können Sie sich auf `anubis.informatik.uni-halle.de` einloggen, und dort `javac` und `java` nutzen.

  Editoren ohne graphische Schnittstelle, die Sie in diesem Terminal-Fenster verwenden können, sind z.B. z.B. `nano` [http://wiki.ubuntuusers.de/Nano], `ne` [http://ne.dsi.unimi.it/], `vi`, `emacs`. Dateien können Sie mit dem Programm `pscp` (kommt mit `PuTTY`) zwischen `anubis` und Ihrem PC übertragen.

* Eclipse (1) *

- Alternativ können Sie auch eine IDE ("Integrated Development Environment") wie Eclipse verwenden.

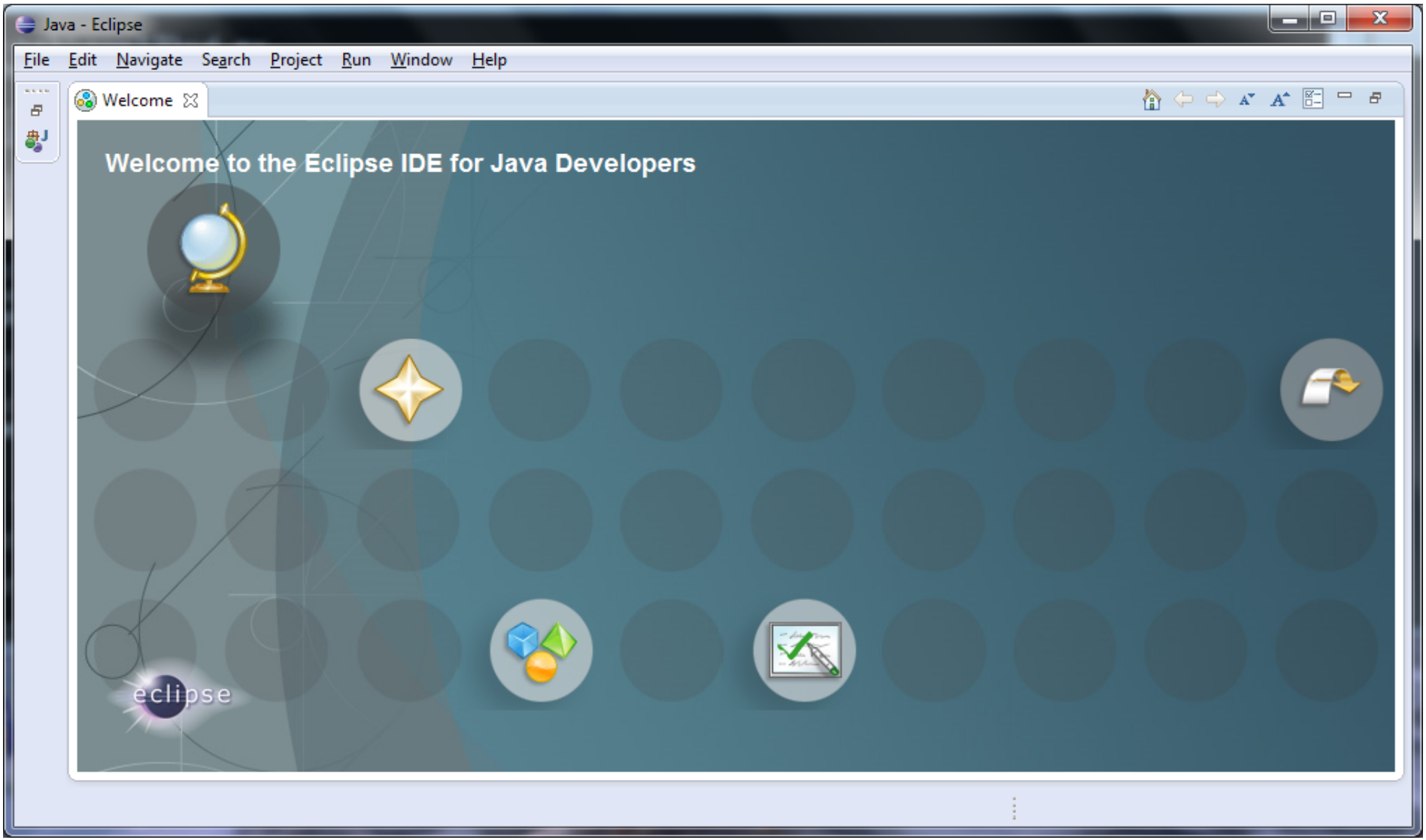
  Eclipse können Sie bei [http://eclipse.org/downloads/] herunterladen. Wählen Sie unter "Package Solutions" die "Eclipse IDE for Java Developers". Die Installation beschränkt sich normalerweise auf das Entpacken des Archivs. Eclipse bringt einen eigenen Compiler mit, aber das "Java Runtime Environment" (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) *

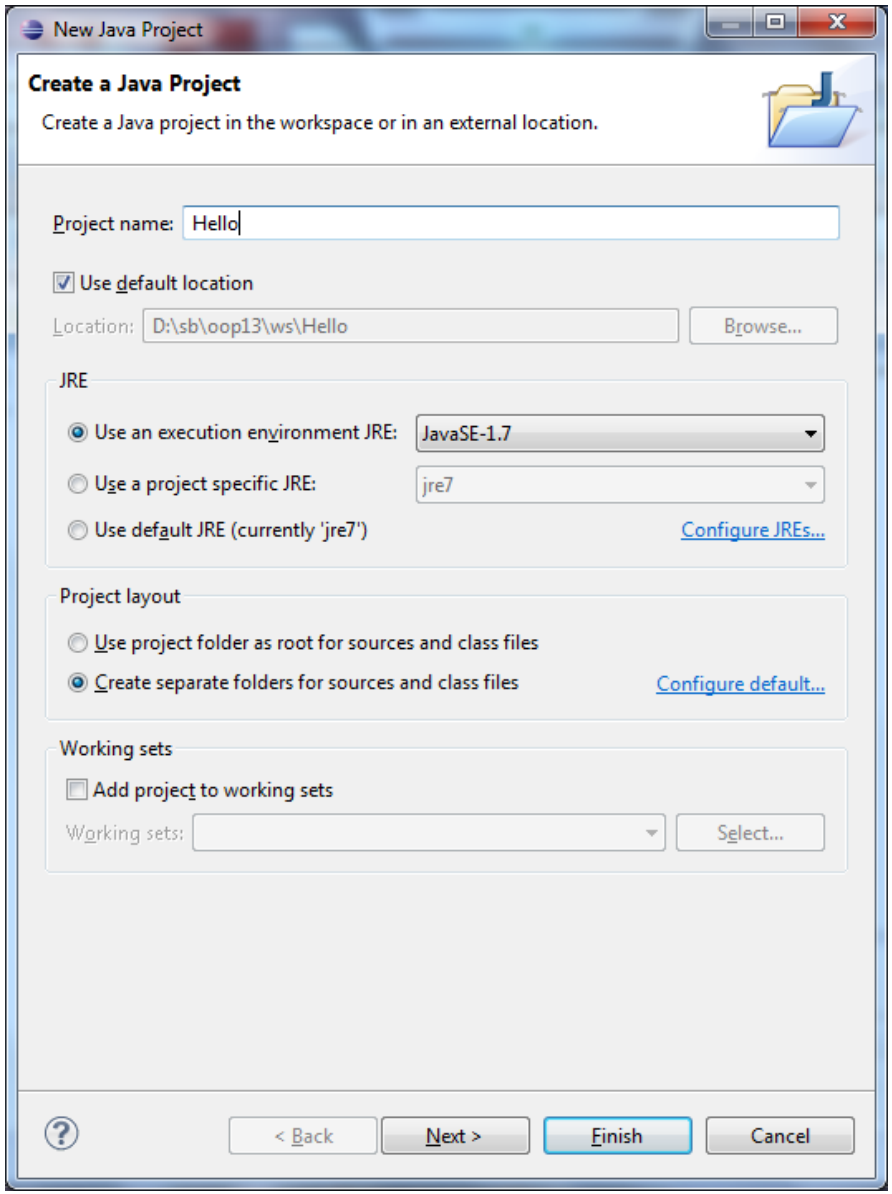
- Legen Sie ein neues Projekt an: `File→New→Java Project`.

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

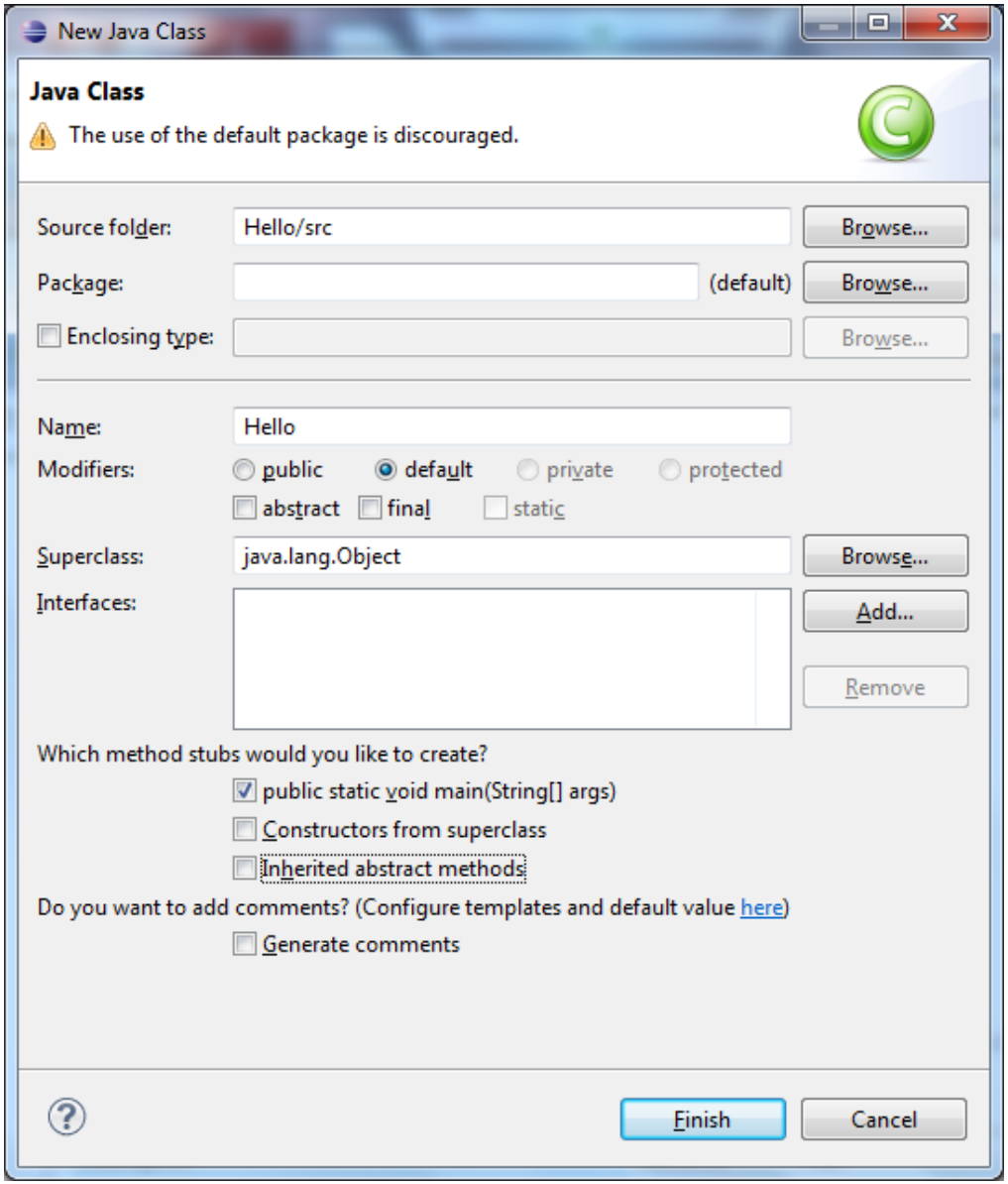
- Legen Sie unter `File→New→Class` eine Klasse an.

  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.

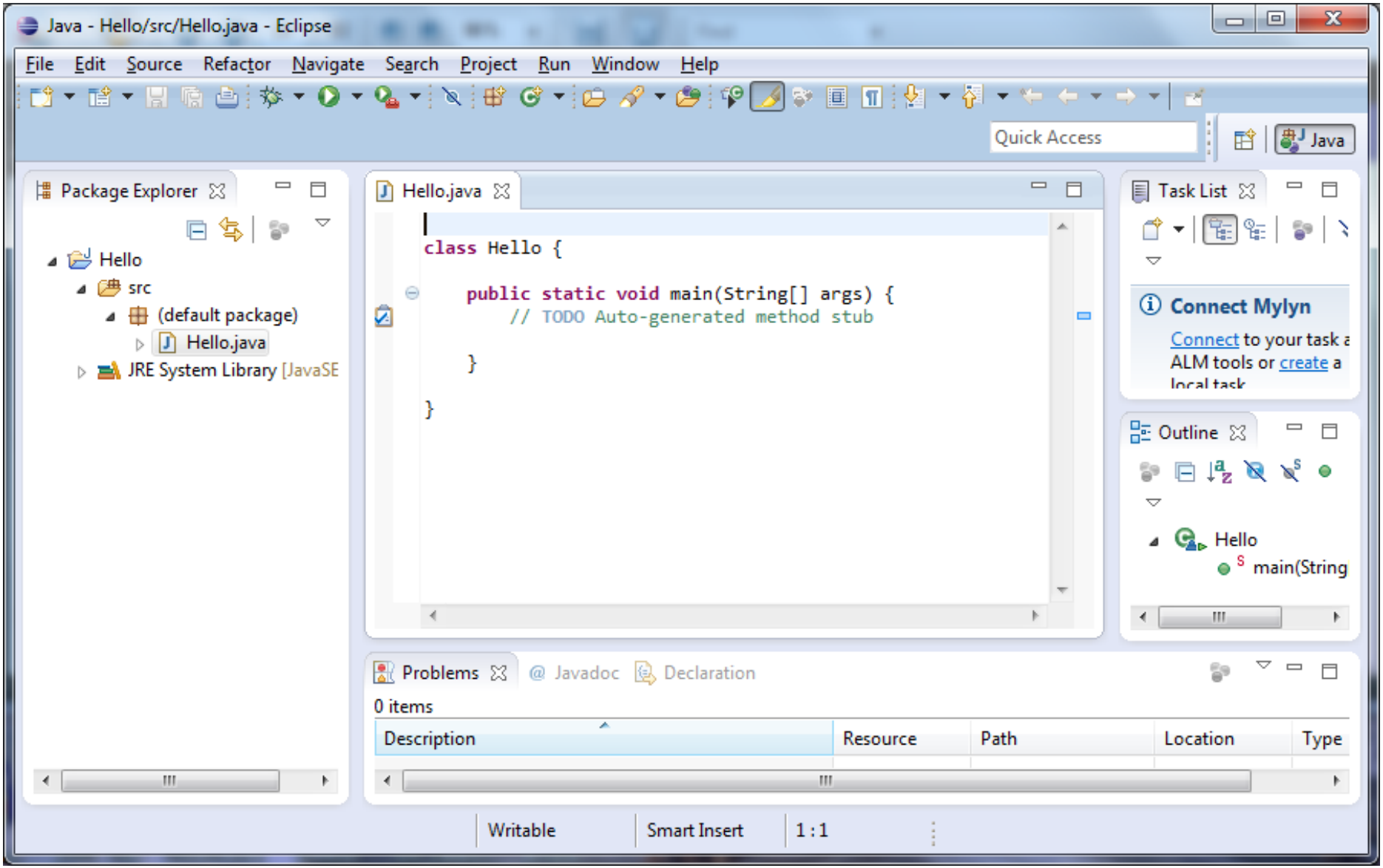
* Eclipse (4) *


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) *

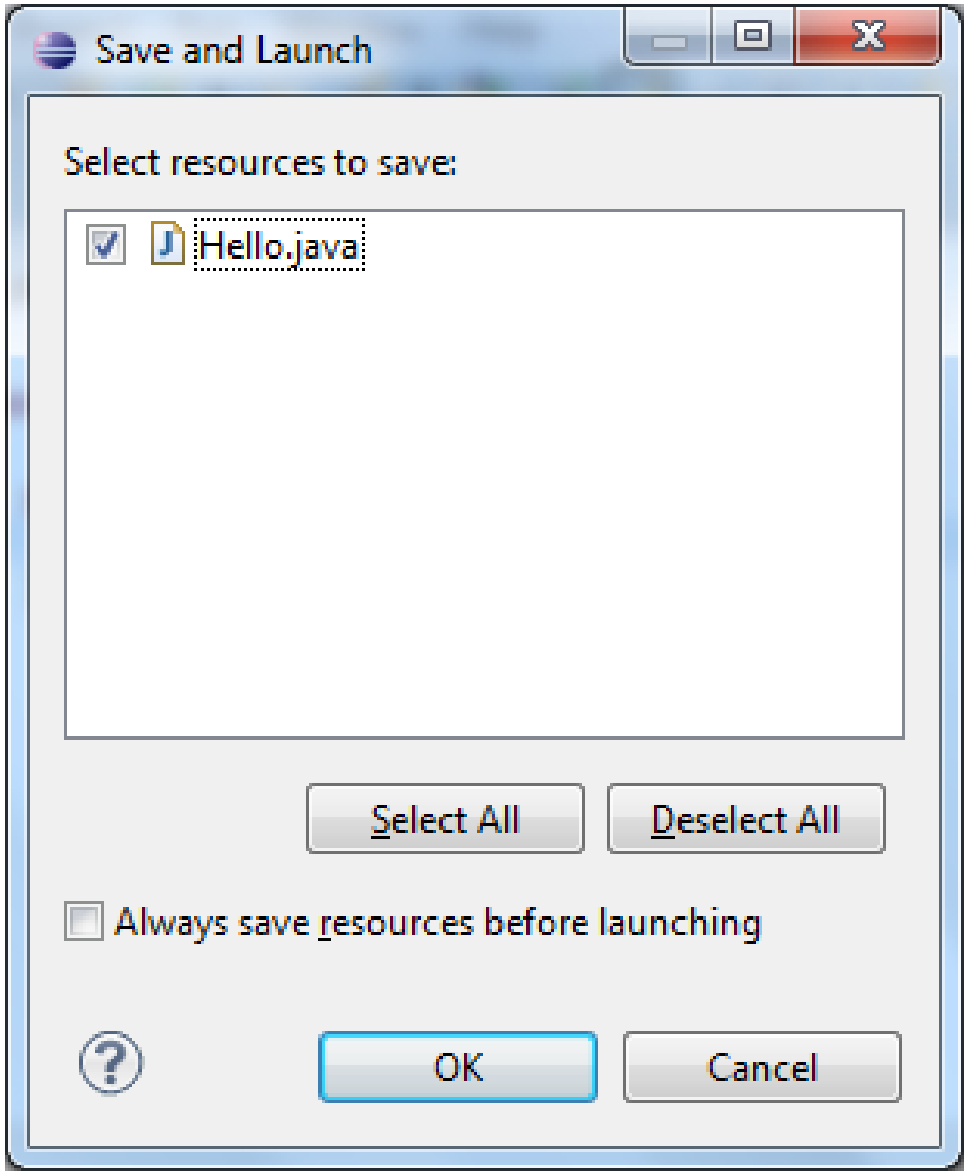
- Vervollständigen Sie den Programmtext.

  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.

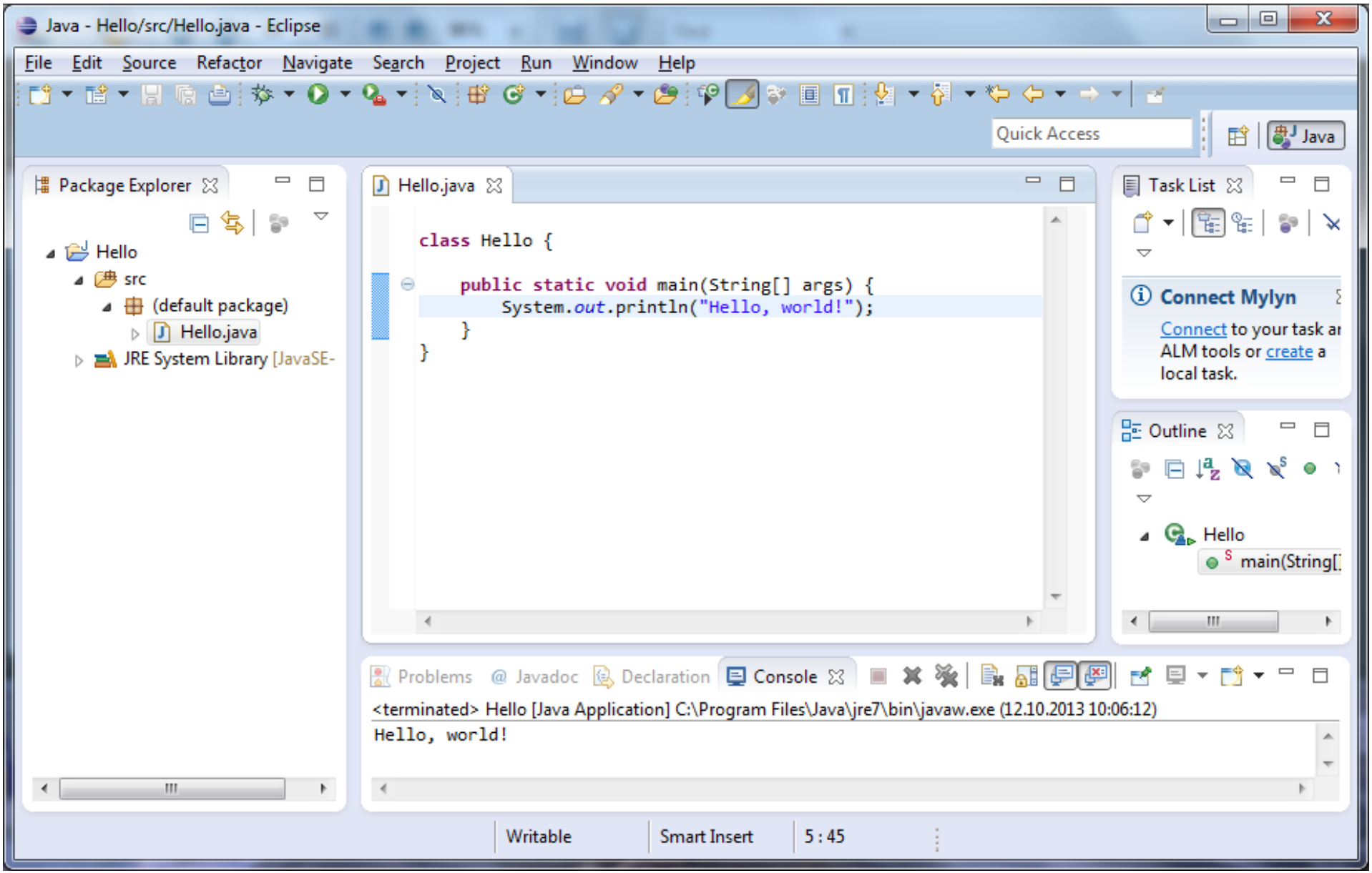
- Speichern Sie unter `File`→`Save` ab.

- 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

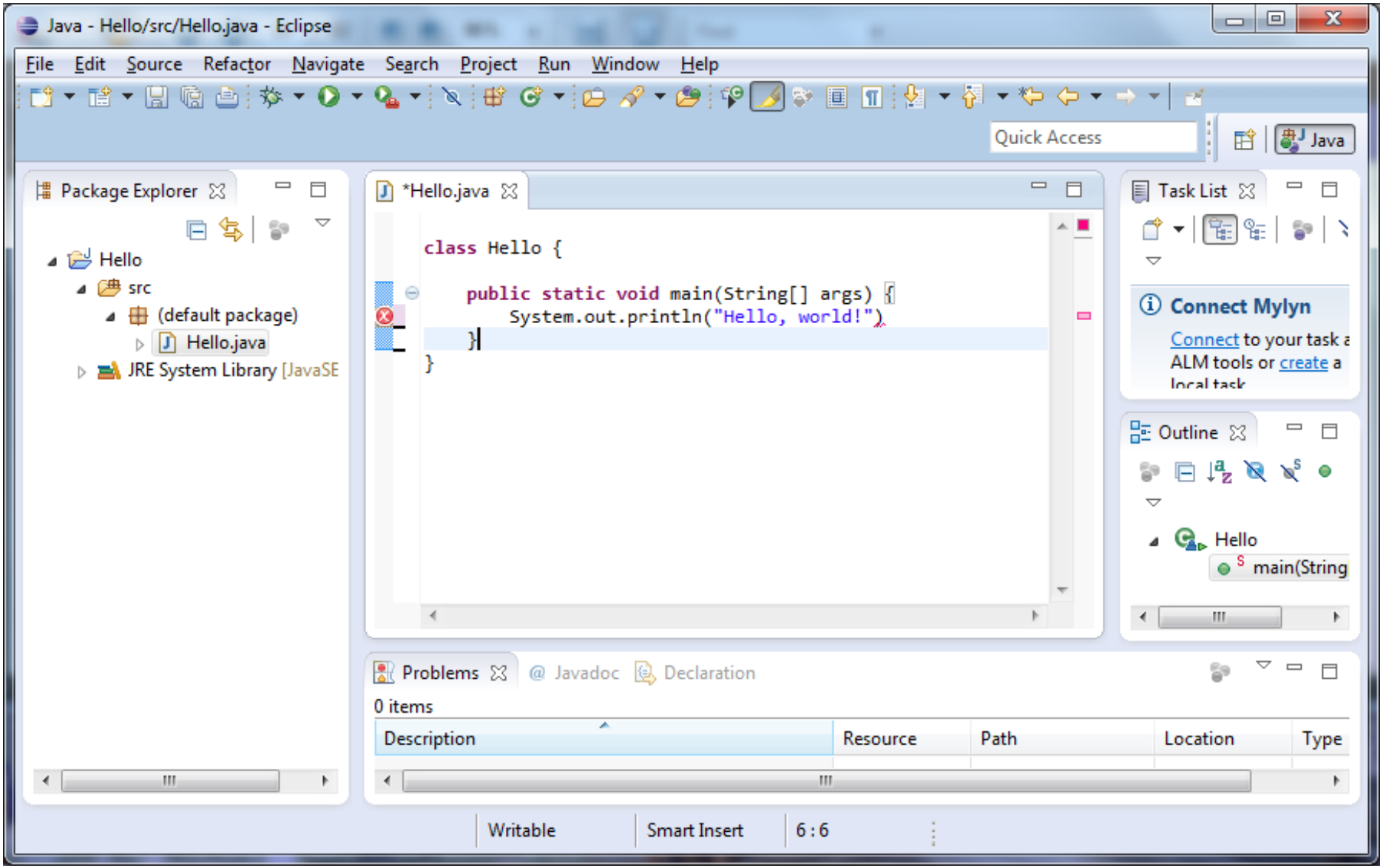
* Eclipse (10) *

- 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`.

Vergleich: Einzelwerkzeuge vs. IDE

- 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.
- Sie werden beides lernen müssen (und mehrere IDEs).

Umgang mit Fehlermeldungen (1)

- 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, gibt der Compiler eine Fehlermeldung aus, und weigert sich, das Programm zu übersetzen, solange der Fehler nicht korrigiert ist. Der Vorteil dabei ist, dass es keine Interpretations-Spielräume bleiben.

- Es ist normal, dass man mehrere Anläufe braucht, bis ein Programm fehlerfrei durch den Compiler läuft.

  Natürlich macht ein erfahrener Programmierer weniger Fehler, aber er schreibt auch längere Programme.

- Man sollte gerade als Anfänger die Compilierbarkeit des Programms öfters testen und Fehler entfernen.

  Nicht ein langes Programm am Stück schreiben und dann erstmals compilieren.

Umgang mit Fehlermeldungen (2)

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

  Vielleicht sind es Folgefehler: Wenn Sie den ersten Fehler korrigieren, verschwinden mit etwas Glück auch die anderen Fehlermeldungen. (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.)

- Die Fehlermeldung enthält eine Zeilennummer: Der Fehler wird sich in oder vor dieser Zeile befinden.

  Wenn Sie z.B. in der Zeile davor ein Semikolon vergessen haben, wird das erst in der nächsten Zeile gemeldet, weil es dort ja noch stehen könnte (Java erlaubt beliebige Zeilenumbrüche, auch vor dem Semikolon). Erst wenn der Compiler dort etwas sieht, was nicht passt (z.B. "}"), meldet er den Fehler.

Umgang mit Fehlermeldungen (3)

- 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 (4)

- 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 (5)

- 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 (6)

- 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 (7)

- 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.
  - Schauen Sie, ob Eclipse vielleicht einen "Quick Fix" vorschlägt.

    Der muss nicht immer funktionieren, aber wahrscheinlich bekommen sie anschließend wenigstens eine andere Fehlermeldung.
  - Probieren Sie einen alternativen Java-Compiler.

Erstes Beispiel: Hello, World!

- Inhalt der Datei "`Hello.java`":

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

- Java ist eine formatfreie Sprache: Zwischen den Worten und Symbolen kann man beliebig Leerzeichen, Tabulatoren und Zeilenumbrüche einfügen.

  Man könnte das ganze Programm also auch in eine Zeile schreiben. Die Zeilenstruktur und Einrückungen helfen nur, es dem menschlichen Leser leichter zu machen — dem Compiler ist es egal.

- Die Groß/Klein-Schreibung ist dagegen wichtig.

Klassendeklarationen

- Ein Java-Programm besteht aus Klassendeklarationen.

  Normalerweise sind Klassen Blaupausen zur Erzeugung von Objekten.
  Z.B. könnte es eine Klasse "`Kunde`" geben, und jedes einzelne Objekt repräsentiert einen Kunden, enthält also `Name`, `Vorname`, `KdNr`, u.s.w.
  Im Beispiel dient die Klasse nur als "Modul", also als Einheit zur Strukturierung von Programmcode. Es werden keine Objekte der Klasse erzeugt.

- Eine Klassendeklaration besteht also (etwas vereinfacht) aus
  - dem Schlüsselwort "`class`",
  - dem Namen der neuen Klasse (ein Bezeichner, s.u.),
  - einer "`{`" (geschweifte Klammer auf),
  - Deklarationen der Bestandteile der Klasse, insbesondere Variablen und Methoden (s.u.),
  - einer "`}`" (geschweifte Klammer zu).

Bezeichner (1)

- Klassen, Variablen, Methoden, u.s.w. benötigen Namen (engl. “identifier”: Bezeichner).
- Solche Namen können aus Buchstaben und Ziffern bestehen, wobei das erste Zeichen ein Buchstabe sein muß.

  Die Zeichen “`_`” und “`$`” zählen als Buchstabe, wobei “`$`” aber nur in von Programmen erzeugtem Programmcode verwendet werden soll.

  Umlaute sind grundsätzlich möglich, führen aber gerade bei Klassennamen möglicherweise zu Problemen, weil die Klassennamen auch als Dateinamen verwendet werden, und nicht jedes Betriebssystem Umlaute gleich codiert.
- Klassennamen beginnen üblicherweise mit einem Großbuchstaben. Bei mehreren Worten wählt man die “Kamelhöcker-Schreibweise”, z.B. `LagerPosition`.

  Das ist nur Konvention und Empfehlung, dem Compiler ist es egal.

  Leerzeichen in Namen wären aber nicht möglich.

Bezeichner (2)

- Bezeichner müssen in einem gewissen Kontext eindeutig sein, z.B. kann man nicht zwei Klassen mit gleichem Namen deklarieren (innerhalb eines "Paketes", s.u.).

  Wenn man später ein Objekt der Klasse anlegen will, muß der Compiler ja wissen, von welcher Klasse, d.h. was die zugehörige Klassendeklaration ist.

- Die Schlüsselwörter (oder "reservierten Wörter") können auch nicht als Namen verwendet werden. Man kann z.B. keine Klasse deklarieren, die "`class`" heißt.

  Die Schlüsselworte sind: `abstract`, `assert`, `boolean`, `break`, `byte`, `case`, `catch`, `char`, `class`, `const`, `continue`, `default`, `do`, `double`, `else`, `enum`, `extends`, `final`, `finally`, `float`, `for`, `if`, `goto`, `implements`, `import`, `instanceof`, `int`, `interface`, `long`, `native`, `new`, `package`, `private`, `protected`, `public`, `return`, `short`, `static`, `strictfp`, `super`, `switch`, `synchronized`, `this`, `throw`, `throws`, `transient`, `try`, `void`, `volatile`, `while`.

Methoden

- Prozeduren/Funktionen/Methoden (grob gesprochen alles dasselbe) enthalten die eigentlich auszuführenden Befehle.
- Eine Methodendeklaration besteht aus:
  - "Modifizierern", im Beispiel `public` und `static`,
  - einem Ergebnis- oder Rückgabetyp, im Beispiel `void`,
  - dem Namen der Methode, in Beispiel `main`,
  - einer Parameterliste, eingeschlossen in `(` und `)`,

    Dies sind die Argument- oder Eingabewerte der Funktion. Bei der speziellen Methode "`main`" (Hauptprogramm, Start der Ausführung) ist vorgesehen, dass Kommandozeilen-Argumente übergeben werden. Sie werden hier nicht verwendet, müssen aber deklariert werden.
  - die eigentlich auszuführenden Befehle, eingeschlossen in `{` und `}` (Methoden-Rumpf).

Rahmenprogramm (1)

- Man muss das jetzt noch nicht alles verstehen.

  Klassen und Methoden werden später noch sehr ausführlich behandelt.

- Man kann das Rahmenprogramm auch erstmal als gegeben annehmen.

- Für uns ist zunächst nur der Rumpf der Methode "`main`" interessant (Hauptprogramm):

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

Rahmenprogramm (2)

- Außerdem wird man natürlich einen eigenen Klassennamen wählen:

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

- Wenn man die Klasse `XYZ` nennt, sollte sie in der Quelldatei `XYZ.java` deklariert werden.

  Für Klassen, die nicht als "`public`" deklariert sind, ist das nicht unbedingt nötig. Das Ergebnis des Compiler-Aufrufs steht aber auf jeden Fall in `XYZ.class`. Es ist übersichtlicher, wenn beides zusammen passt.

Inhalt

Ausgabebefehle (1)

- Der Rumpf einer Methode (z.B. von `main`) enthält eine Folge von Anweisungen (Befehlen, engl. “Statements”).
- Ein Beispiel ist die Anweisung zur Ausgabe eines Textes:

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

- Das “`ln`” steht für “line”. Es gibt auch eine Variante, die nicht einen Zeilenumbruch nach dem Text ausgibt:

```
System.out.print("Hello, world!");
```

- Das ist eventuell interessant, wenn man mehrere Textstücke hintereinander ausgeben will.

  Alternativ kann man Textstücke und andere auszugebene Daten auch mit dem Operator “+” zusammenfügen, das wird unten noch gezeigt.

* Ausgabebefehle (2) *

- Muss der Befehl "`System.out.println`" so lang sein?

  Das ist eine berechtigte Frage. In C war es einfach: `printf("...");`

- "`println`" ist eine Methode. Praktisch alle auszuführenden Befehle stehen in Java innerhalb von Methoden.

  Die Methode `println` der Klasse `PrintStream` wurde von den Entwicklern der Standard Java Bibliothek (Java Platform SE API) programmiert.

- `System` ist eine Klasse (Zusammenfassung nützlicher Objekte und Methoden zur Interaktion mit der Laufzeitumgebung).

  [http://docs.oracle.com/javase/7/docs/api/java/lang/System.html]

- Sie enthält im Feld/Attribut "`out`" ein Objekt für den Standard-Ausgabestrom (Ausgabe auf Konsole/Terminal).

- Für dieses Objekt ruft man die Methode `println` auf.

Methodenaufruf (1)

- Ein Methodenaufruf (wie der Druckbefehl) besteht aus
  - Dem Namen einer Klasse, oder einem Ausdruck, der zu einem Objekt auswertbar ist.

    Es gibt zwei verschiedene Arten von Methoden: Die "statischen" Methoden brauchen eine Klasse, die "normalen" Methoden ein Objekt. Dies wird alles später noch ausführlich behandelt. Im Beispiel ist "`System.out`" ein Ausdruck, der ein Objekt liefert.
  - einem Punkt "`.`",
  - dem Namen der Methode (im Beispiel "`println`"),
  - einer öffnenden runden Klammer "`(`",
  - einer Liste von Argumenten,

    Im Beispiel gibt es nur ein Argument, nämlich den zu druckenden Text.
  - einer schließenden runden Klammer "`)`".

Methodenaufruf (2)

- Das Semikolon “;” am Ende gehört nicht mehr zum Methodenaufruf, sondern markiert das Ende der Anweisung.

  In diesem Fall besteht die Anweisung aus einem einzelnen Methodenaufruf.

- Auch mathematische Funktionen sind in Java Methoden.
- Zum Beispiel kann man so die $\sqrt{2}$ berechnen:

```
System.out.println(Math.sqrt(2.0));
```

- Die Zahl 2.0 ist die Eingabe für die Funktion `sqrt` (“square root”: Quadratwurzel).

  Diese Funktion ist in der Klasse `Math` definiert.

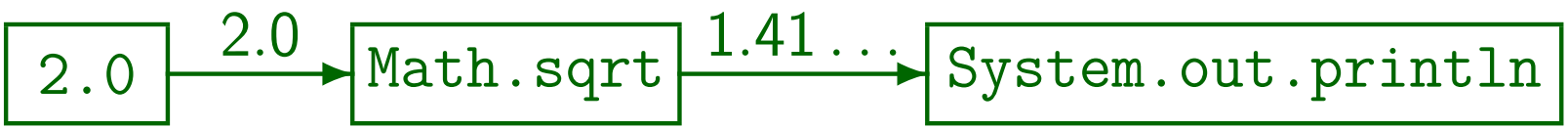
  [http://docs.oracle.com/javase/7/docs/api/java/lang/Math.html]

- Der berechnete Wert wird dann weiter an die Methode `println` gegeben, um ihn auszudrucken.

Methodenaufruf (3)

- Die Struktur des Ausdrucks ist also so ähnlich wie ein geschachtelter Funktionsaufruf $f(g(x))$ in der Mathematik, nur dass die Funktionsnamen etwas länger sind.

  [Man könnte das Objekt für den Ausgabestrom als zusätzliches Argument der Druck-Funktion sehen, aber das wird später noch besprochen.]



- Die Methode `println` unterscheidet sich von einer mathematischen Funktion in zwei Punkten:
  - Sie liefert keinen Ergebniswert, man kann sie also nicht weiter in einen anderen Methodenaufruf schachteln.
  - Sie bewirkt eine Änderung des Berechnungszustandes, nämlich eine Ausgabe.

Konstanten/Literale: Zeichenketten (1)

- Mit Konstanten (auch Literale genannt) kann man Datenwerte aufschreiben.
- Das “Hello, world!”-Programm enthält z.B. einen Text (Folge von Zeichen, “Zeichenkette”, engl. “String”):

```
"Hello, world!"
```

  Das englische Wort “String” bedeutet u.a. Schnur, vielleicht soll man sich die Zeichen wie an einer Perlenkette aufgefädelt vorstellen. In der Informatik übersetzt man “String” mit “Zeichenkette”. Man kann solche Fachworte aber auch unübersetzt lassen. In Java sind Zeichenketten Objekte der Klasse `String`.

- Die Anführungszeichen am Anfang und Ende dienen nur der Markierung/Begrenzung der Zeichenkette, sie werden nicht mit ausgegeben.

  Das erste Zeichen der Zeichenkette ist “`H`”, das letzte “`!`”.

Konstanten/Literale: Zeichenketten (2)

- Zeichenkettenkonstanten bestehen aus
  - Einem Anführungszeichen " (Anfangs-Markierung)
  - fast beliebigen Zeichen und einigen Escapesequenzen (s.u.)
  - einem Anführungszeichen " (Ende-Markierung).
- "Fast beliebige Zeichen" bedeutet alle Zeichen außer
  - Anführungszeichen "

    Dann würde der Compiler ja denken, die Zeichenkettenkonstante ist zu Ende.
  - Rückwärtsschrägstrich \

    Dieses Zeichen wird zur Einleitung von Escapesequenzen gebraucht, s.u.
  - Zeilenumbrüchen ("Carriage Return", "Linefeed").

Konstanten/Literale: Zeichenketten (3)

- In Zeichenkettenkonstanten können verschiedene Escape-Sequenzen verwendet werden, z.B.
  - `\"` für ein Anführungszeichen,

    Durch die Markierung mit dem vorangestellten \ hält der Compiler dieses Anführungszeichen nicht für das Ende der Zeichenkette, sondern für ein Anführungszeichen.
  - `\\` für den Rückwärtsschrägstrich,

    Da der Rückwärtsschrägstrich jetzt eine Spezialbedeutung hat, braucht man auch eine Möglichkeit, ihn einzugeben.
  - `\n` für einen Zeilenumbruch.

    Eine String-Konstante muss auf derselben Zeile geschlossen werden, in der sie geöffnet wurde (falls man das schließende " vergisst, bekommt man so die Fehlermeldung am Ende der Zeile, statt am Ende der Datei). Bei Bedarf kann man mit "`\n`" einen Zeilenumbruch codieren.

Konstanten/Literale: Zeichenketten (4)

- Um Escape-Sequenzen zu verstehen, muss man Folgendes unterscheiden:
  - die Darstellung des Wertes im Java-Quellcode (Konstante)
  - den repräsentierten Wert (bei Ausführung des Programms).
- Z.B. ist `"\"\\"` eine Zeichenkette aus zwei Zeichen: einem Anführungszeichen und einem Rückwärtsschrägstrich.
  - `"\"\\"`: Erstes Zeichen: `"`.
  - `"\"\\"`: Zweites Zeichen: `\`.

    Solche unübersichtlichen Zeichenketten sind natürlich selten, meist steht eine einzelne Escape-Squenz zwischen normalen Zeichen, so dass man sie leicht auseinander halten kann. Merken Sie sich nur, dass ein Rückwärtsschrägstich \ in Zeichenketten eine besondere Bedeutung hat, und " im Innern (ohne \) nicht erlaubt ist.

Konstanten/Literale: Zahlen (1)

- Im Beispiel mit der Berechnung der $\sqrt{2}$ stand eine Konstante für eine reelle Zahl:

  `2.0`

- Solche Zahlkonstanten bestehen (vereinfacht) aus:
  - eine Folge von Ziffern vor dem Punkt,
  - einem Punkt "`.`" (Dezimalpunkt),
  - einer Folge von Ziffern nach dem Punkt.

    Es reicht, wenn vor oder nach dem "`.`" eine Ziffer steht (also `.3` oder `3.` wären ok). Außerdem gibt es noch die wissenschaftliche Schreibweise mit Zehnerpotenz und eine Hexadezimalschreibweise. Später ausführlich.

  - Selbstverständlich ist auch ein Vorzeichen möglich, aber das zählt formal schon als Rechenoperation (s.u.).

Konstanten/Literale: Zahlen (2)

- Selbstverständlich gibt es auch ganze Zahlen, sie werden einfach als Folge von Ziffern geschrieben, z.B.:

  100

  Wieder kann man effektiv ein Vorzeichen voranstellen. Weil das als eigene Operation zählt, wäre zwischen Vorzeichen und Zahl ein Leerzeichen möglich (aber nicht nötig), während man innerhalb der eigentlichen Zahlkonstante natürlich keine Leerzeichen verwenden kann (dann wären es zwei Zahlkonstanten hintereinander, das würde einen Syntaxfehler geben).

- Man muss führende Nullen vermeiden, z.B. ist 010 nicht die Zahl 10, sondern 8!

  Aus historischen Gründen (Kompatibilität zu C) schaltet Java auf Oktalschreibweise (zur Basis 8, nicht zur Basis 10) um, wenn die ganze Zahl mit 0 beginnt. Für eine einzelne 0 ist es egal, die bedeutet immer Null, egal, was die Basis ist.

Datentypen (1)

- Ein Datentyp ist eine Menge von Werten gleicher Art (zusammen mit den zugehörigen Operationen, s.u.).
- Java unterscheidet deutlich zwischen
  - reellen Zahlen wie `2.0`, diese haben den Datentyp "`double`",

    "`double`" kommt von "doppelte Genauigkeit": Intern kann man nur eine gewisse Anzahl Stellen darstellen, nicht beliebige reelle Zahlen.
    Es gibt auch einen Typ "`float`" für Zahlen mit einfacher Genauigkeit, aber den verwendet man kaum noch.
  - ganzen Zahlen wie `2`, diese haben den Datentyp "`int`",

    Von Englisch "integer": ganze Zahl.
  - Zeichenketten wie `"2"`, diese haben den Datentyp "`String`".

Datentypen (2)

- Die interne Repräsentation dieser Werte im Speicher des Rechners ist ganz unterschiedlich.

  Ein `int`-Wert braucht 32 Bit. Ein `double`-Wert braucht 64 Bit, und ist intern mit Mantisse und Exponent dargestellt (wie in "wissenschaftlicher Notation"). Ein `String`-Wert ist die Hauptspeicher-Adresse eines Objektes, dieses enthält in der OpenJDK-Implementierung ein Zeichen-Array (Speicherbereich für Zeichen) und drei ganze Zahlen, und ist damit (je nach Rechner) mindestens 128 Bit groß (plus den Speicherbereich für das Array).

- Datentypen helfen, Fehler zu vermeiden. Beispiel:

  ```
  System.out.println(Math.sqrt("abc"));
  ```

  Der Compiler gibt hier eine Fehlermeldung aus:

  "`method sqrt in class Math cannot be applied to given types; required: double, found: String`"

Datentypen (3)

- Eine Umwandlung von `int` nach `double` würde der Compiler bei Bedarf automatisch einfügen, z.B. würde

  ```
  System.out.println(Math.sqrt(2));
  ```

  funktionieren.

  Tatsächlich wird 2.0 an die `sqrt`-Funktion übergeben ("widening conversion").

- Von manchen Funktionen gibt es mehrere Versionen für unterschiedliche Datentypen, z.B. gibt es von der `println`-Funktion Varianten für die Typen `int`, `double` und `String` (und noch weitere Typen).

  Die übergebenen Bits müssen ja richtig interpretiert werden. Es handelt sich hier um unterschiedliche Methoden, die zwar den gleichen Namen haben, aber aufgrund der Datentypen des übergebenen Wertes auseinander gehalten werden können ("überladene Methoden").

Rechenoperationen (1)

- Selbstverständlich kennt Java die vier Grundrechenarten. Zum Beispiel gibt der Befehl

  ```
  System.out.println(1 + 1);
  ```

  den Wert 2 aus.

  Die Leerzeichen links und rechts vom "+" sind optional (kann man weglassen).

- Java kennt die Regel "Punktrechnung vor Strichrechnung":

  ```
  System.out.println(1 + 2 * 3);
  ```

  gibt den Wert 7 aus, und nicht etwa 9.

- Wenn man möchte, kann man Klammern setzen:

  ```
  System.out.println(1 + (2 * 3));
  ```

  Dies wäre die implizite Klammerung, hier wären die Klammern überflüssig.

  Will man aber die Addition vor der Multiplikation, wären die Klammern wichtig.

Rechenoperationen (2)

- Bei der Division ist zu beachten, dass bei `int`-Werten links und rechts die ganzzahlige Division (Division mit Rest) verwendet wird.
- Zum Beispiel druckt

  ```
  System.out.println(14 / 5);
  ```

  den Wert 2.

  Es wird immer in Richtung auf 0 gerundet, d.h. abgerundet (bei Ergebnis > 0).
- Den zugehörigen Rest erhält man mit dem Prozent-Operator:

  ```
  System.out.println(14 % 5);
  ```

  gibt 4 aus.

  Die Division ergibt 2 Rest 4. Die Bestimmung des Rests wird in der Mathematik auch mit dem Modulo-Operator `mod` geschrieben.

Rechenoperationen (3)

- Wenn die Operanden vom Divisions-Operator / `double`-Werte sind, findet die normale Division statt, und das Ergebnis ist ein `double`-Wert.
- Zum Beispiel liefert

  ```
  System.out.println(14.0 / 5.0);
  ```

  den Wert `2.8`.
- Wenn nur einer der Operanden ein `double`-Wert ist, und der andere ein `int`-Wert, so wird der `int`-Wert automatisch in einen `double`-Wert umgewandelt, bevor die Division ausgeführt wird.

  Zum Beispiel liefert "`System.out.println(14.0 / 5);`" auch `2.8` (weil eigentlich `14.0 / 5.0` gerechnet wird).

Rechenoperationen (4)

- Der Operator `/` ist also auch überladen (wie `println`).
  Es gibt zwei verschiedene Versionen:
  - Die eine hat zwei Operanden vom Typ `int`, und liefert ein `int` (ganzzahlige Division).
  - Die andere hat zwei Operanden vom Typ `double`, und liefert ein `double`.

    Es gibt es noch Varianten für andere Typen (`float` und `long`): später.
- Tatsächlich gibt das für alle arithmetischen Operationen, aber bei der Division ist es am offensichtlichsten.
- Weitere Kombinationen von Eingabetypen werden behandelt, indem vor der Operation der Operand mit dem kleineren Typ `int` in den größeren Typ `double` umgewandelt wird.

Rechenoperationen (5)

- Die beiden Varianten der Division unterscheiden sich auch im Verhalten beim Fehler "Division durch Null":
  - Bei der ganzzahligen Dvision wird das Programm mit dem Fehler "`ArithmeticException`" abgebrochen.

    Der Fehler wird erst zur Laufzeit bemerkt, wenn das Programm ausgeführt wird. Der Compiler übersetzt das Programm normal. Ein Grund ist, dass der Fehler auch von der Eingabe abhängen kann, und nicht immer auftreten muss.
  - Bei der Division von `double`-Werten wird ein spezieller Wert "`Infinity`" ($+\infty$) geliefert, der zum Wertebereich von `double` gehört. Das Programm läuft weiter.

    Begründung: Man kann nicht alle reellen Zahlen mit `double`-Werten exakt darstellen. Es kann zu Rundungsfehlern kommen. Die Zahl, durch die dividiert wird, könnte also auch nicht 0, sondern eine sehr kleine Zahl sein. Das Ergebnis müsste dann sehr groß sein.

Mathematische Funktionen (1)

- Damit Sie Java wie einen Taschenrechner benutzen können, hier noch einige mathematische Funktionen:
  - `Math.sqrt(`x`)`: $\sqrt{x}$
  - `Math.exp(`x`)`: e^x
  - `Math.pow(`x`,`y`)`: x^y
  - `Math.log(`x`)`: $\ln(x)$ (Logarithmus zur Basis e)
  - `Math.log10(`x`)`: $\log(x)$ (Logarithmus zur Basis 10)
  - `Math.sin(`x`)`: $\sin(x)$ (Argument in Bogenmaß/`rad`)
  - `Math.cos(`x`)`: $\cos(x)$ (Argument in Bogenmaß/`rad`)
- Die vollständige Liste finden Sie in der Dokumentation zur Java Platform SE 7 API (Klasse `Math` im Paket `java.lang`).

[http://docs.oracle.com/javase/7/docs/api/java/lang/Math.html]

Mathematische Funktionen (2)

- Die Klasse `Math` enthält außerdem noch zwei Konstanten:
  - `Math.PI`: $\pi = 3.14159\ldots$
  - `Math.E`: $e = 2.71828\ldots$ (Eulersche Zahl)
- Wenn man den Sinus von 30° berechnen will, kann man das z.B. so tun (180° entspricht π Radiant):

```
System.out.println(Math.sin(30*Math.PI/180));
```

  Das Ergebnis ist leider nicht ganz exakt: `0.4999999999999994`

- Es gibt aber auch eine eigene Funktion für die Umrechnung:

```
System.out.println(Math.sin(Math.toRadians(30)));
```

- Außerdem enthält die Klasse `Math` noch eine Funktion zur Berechnung von Zufallswerten:
  - `Math.random()`: Zufallswert im Intervall $[0, 1)$

Konkatenation von Zeichenketten (1)

- Der Operator `+` ist nicht nur für `int`- und `double`-Werte definiert, sondern man kann ihn auch auf Zeichenketten (Werte vom Typ `String`) anwenden.
- In diesem Fall fügt er die Zeichenketten zusammen (konkateniert sie):

  ```
  System.out.println("abc" + "def");
  ```

  druckt: `abcdef`.
- Auch hier macht der Compiler bei Bedarf eine automatische Typanpassung: Wendet man `+` auf eine Zeichenkette und eine Zahl an, so wird die Zahl in eine Zeichenkette umgewandelt, und dann wird konkateniert.

Konkatenation von Zeichenketten (2)

- Wenn man z.B. nicht nur das "nackte" Ergebnis drucken will, kann man das so machen:

```
System.out.println("Die Wurzel aus 2 ist: " +
                   Math.sqrt(2));
```

- Aufgabe: Warum sind hier die Klammern wichtig?

```
System.out.println("1+1 = " + (1+1));
```

Hinweis: Wenn man keine Klammern setzt, wird implizit von links geklammert.

- Es gibt noch viele weitere Funktionen für Zeichenketten, die später in einem eigenen Kapitel behandelt werden.

[http://docs.oracle.com/javase/7/docs/api/java/lang/String.html]

Anweisungsfolge (1)

- Im Methodenrumpf kann man eine Folge von Anweisungen schreiben, z.B. drei Ausgabe-Anweisungen hintereinander:

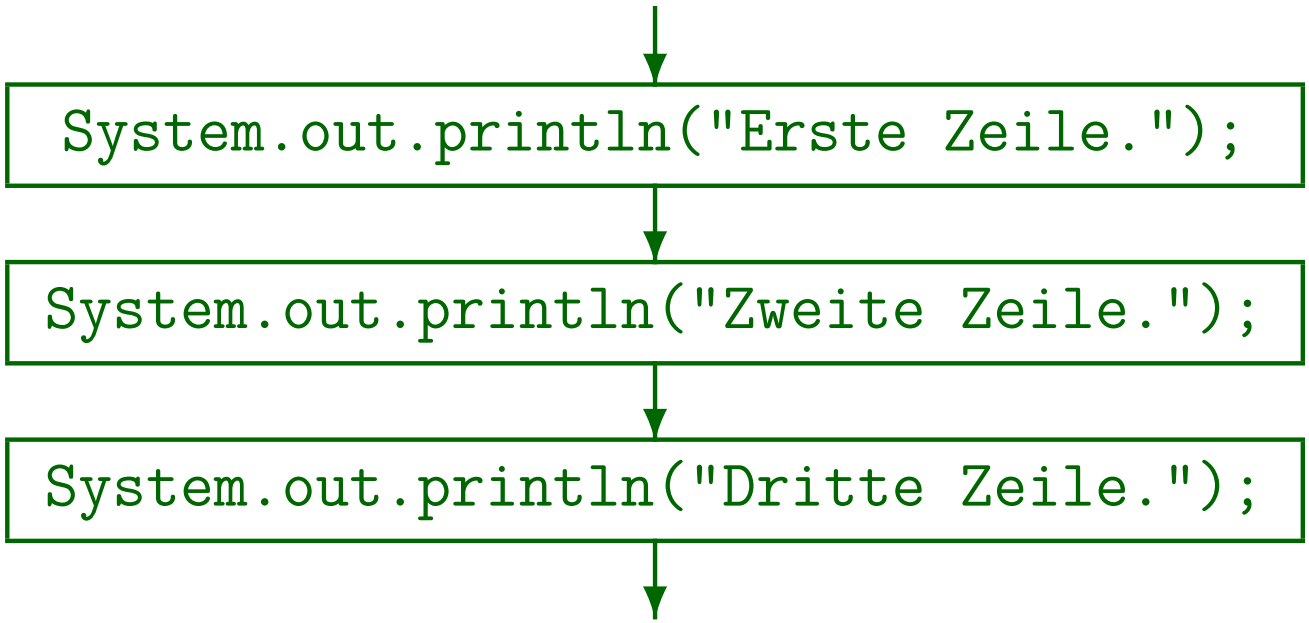
```
class Hello2 {
    public static void main(String[] args) {
        System.out.println("Erste Zeile.");
        System.out.println("Zweite Zeile.");
        System.out.println("Dritte Zeile.");
    }
}
```

- Die Ausgabe des Programms ist dann:

```
Erste Zeile.
Zweite Zeile.
Dritte Zeile.
```

Anweisungsfolge (2)

- Die Anweisungen werden in der Reihenfolge ausgeführt, in der man sie aufschreibt:



- Jede Anweisung modifiziert den Berechnungszustand, der nach der vorangegangenen Anweisung erreicht ist.

Anweisungsfolge (3)

- Wenn man die Anweisungen Schritt für Schritt ausführt, steht nach der ersten Anweisung in der Ausgabe:

```
Erste Zeile.
```

- Der zweite Befehl fügt seinen Text ans Ende der Ausgabe an:

```
Erste Zeile.
Zweite Zeile.
```

- Und genauso die dritte Anweisung:

```
Erste Zeile.
Zweite Zeile.
Dritte Zeile.
```

- So eine schrittweise Ausführung eines Programms ist möglich mithilfe eines Debuggers (wird später erläutert).

Inhalt

Variablen (1)

- Variablen sind benannte Speicherstellen.
- Man kann in der Variablen einen Wert ablegen, z.B. eine ganze Zahl (`int`-Wert), eine reelle Zahl (`double`-Wert) oder eine Zeichenkette (`String`-Wert).
- Den gespeicherten Wert kann man später in Wertausdrücken (Formeln) verwenden (ihn also wieder auslesen).
- Jede Variable kann nur Werte eines bestimmten Datentyps speichern.
- Man muss dem Compiler mitteilen, wie die Variablen heißen, die man verwenden will, und welchen Datentyp jede hat (Variablen müssen deklariert werden).

Variablen (2)

- Man kann eine Variable mit einer Schublade einer Kommode vergleichen, in die man einen Datenwert hineintun kann.
- Dabei gibt es aber auch Unterschiede:
  - Eine Variable kann immer nur einen Datenwert enthalten. Wenn man einen neuen hineintut, verschwindet der alte, er wird "überschrieben".
  - Eine Variable kann niemals leer sein.

    Die Bits im Hauptspeicher sind aus technischen Gründen immer 0 oder 1. Es gibt keinen dritten Wert "leer". Der Java Compiler verhindert aber, dass man einen undefinierten Wert abfragt.
  - Wenn man den Wert abfragt, nimmt man ihn nicht heraus: Er bleibt solange in der Variable, bis ein neuer Wert explizit hineingespeichert wird.

Variablen (3)

- Sie kennen Variablen aus der Mathematik.
- Auch dort sind sie ein Platzhalter in einer Formel, für den man später unterschiedliche Werte einsetzen kann, z.B.

$$f(x) = x^2 + 2 * x + 1$$

- Java löst allerdings keine Gleichungen für Sie, um unbekannte Werte zu ermitteln.

  Selbstverständlich können Sie selbst einen Gleichungslöser programmieren.

- Außerdem kann man in eine Java Variable nacheinander verschiedene Werte speichern. In der Mathematik kann der Wert zwar unbekannt oder beliebig sein, ist aber fest.

  Der zeitliche Ablauf spielt in der Mathematik keine Rolle, in Java schon.

  Dafür ist der Wert niemals wirklich unbekannt: Man hat ja vorher eine Wert in die Variable hineingespeichert.

Variablen: Motivation (1)

- Kompliziertere Ausdrücke mit tief geschachtelten Funktionsaufrufen können unübersichtlich werden.
- Dann ist es besser, zunächst nur ein Zwischenergebnis zu berechnen, und sich dies in einer Variablen zu merken:

```
class Zins1 {
    public static void main(String[] args) {
        double betrag = 500.0;
        double zinssatz = 2.0;
        double zinsen = betrag * (zinssatz/100);
        System.out.println("Zinsen: " + zinsen);
    }
}
```

Variablen: Motivation (2)

- Variablen werden spätestens dann benötigt, wenn das Programm Werte von der Tastatur einliest.

  Oder über das Netz oder von einer Datei oder mittels Maus-Klicks ...

- Dann sind die tatsächlichen Werte ja noch gar nicht bekannt, wenn man das Programm schreibt.

- Sie sind erst bei Ausführung des Programms bekannt, und können bei jeder Ausführung des Programms anders sein.

- Man beachte, dass das Programm nur einmal compiliert wird. Der entstandene Bytecode kann mehrfach ausgeführt werden.

  D.h. zur Compilezeit sind die Werte auch noch nicht bekannt, erst zur Laufzeit. Der Compiler sorgt also dafür, dass die eingelesenen Werte zuerst im Speicher des Rechners abgelegt werden — in einer Variablen.

Beispiel: Eingabe (1)

- Beispiel mit Eingabe des Geldbetrags (`int`-Wert):

```
import java.util.Scanner;
class Zins2 {
    public static void main(String[] args) {
        System.out.print("Betrag (ganze Euro): ");
        Scanner input = new Scanner(System.in);
        double betrag = input.nextInt();
        double zinssatz = 2.0;
        double zinsen = betrag * (zinssatz/100);
        System.out.println("Zinsen: " + zinsen);
    }
}
```

Beispiel: Eingabe (2)

- Es gibt in Java mehrere Möglichkeiten, wie man die Eingabe von der Konsole (Tastatur) durchführen kann.

  Alle sehen zunächst relativ kompliziert aus, aber man kann sich daran gewöhnen.

  Wenn man mehr über Klassen und Objekte weiss, wird es klarer.

- Eine Möglichkeit ist die Verwendung der `Scanner`-Klasse.

  Sie gehört seit Version 1.5 zum Java-Funktionsumfang.

- Weil sie nicht zum Java Kern gehört, braucht man eine `import`-Anweisung, um dem Compiler mitzuteilen, dass man diese Klasse verwenden will.

- In der Klasse `System` gibt es eine Variable `in`, die den Standard-Eingabestrom (normalerweise Tastatur) darstellt.

- Der Eingabestrom würde nur Folgen von Bytes liefern.

Beispiel: Eingabe (3)

- Es wird nun ein neues Objekt `input` der Klasse `Scanner` angelegt, das seine Eingabe aus dem Standard-Eingabestrom bezieht.

  `input` ist auch eine Variable, die aber keine ganze oder reelle Zahl enthält, sondern ein Objekt der Klasse `Scanner` (genauer eine Referenz auf so ein Objekt — das wird später noch sehr ausführlich diskutiert).

- Dieses Objekt kümmert sich darum, die Bytes zu Worten zusammenzufassen, und in den gewünschten Datentyp umzuwandeln.

- Die Klasse `Scanner` hat z.B. eine Methode `nextInt()`, die die nächste ganze Zahl aus der Eingabe liefert.

  Es gibt auch eine Methode `nextDouble()`, dann muss man aber bei den normalen Einstellungen für Deutschland ein Komma "`,`" statt dem Dezimalpunkt verwenden, also z.B. `100,00`.

Beispiel: Eingabe (4)

- Wenn man z.B. “`abc`” statt einer Zahl eingibt, bekommt man eine “`java.util.InputMismatchException`”, also einen Fehler.
- Das Programm wird abgebrochen, und die Fehlermeldung ist nur für Programmierer geeignet.

  Nicht für den Nutzer, der Zinsen berechnen möchte.
- Wir besprechen später, wie man solche Fehler abfängt, und darauf reagieren kann, z.B.
  - eine verständliche Fehlermeldung ausgibt,
  - dem Benutzer eine neue Chance gibt, einen korrekten Zahlwert einzugeben.

Variablen-Deklarationen (1)

- Eine Variable wird mit einer Variablen-Deklaration angelegt. Diese besteht (etwas vereinfacht) aus:
  - Dem Datentyp der Variable, z.B. `int` oder `double`,
  - dem Namen der neuen Variable (Bezeichner),
  - optional einem Gleichheitszeichen "`=`" und einem Wert,

    Der Wert kann auch mit einem Ausdruck berechnet werden.
  - einem Semikolon "`;`".
- Wenn man nicht gleich einen Wert in die Variable einträgt, achtet der Compiler darauf, dass man auch keinen Wert abfragen kann, bis man nicht einen eingetragen hat.

  Er ist da gelegentlich etwas zu pedantisch. Oft ist es das Beste, die Variable gleich zu "initialisieren" (einen ersten Wert einzutragen).

Variablen-Deklarationen (2)

- Es ist üblich, dass Namen von Variablen mit einem Kleinbuchstaben beginnen, und bei mehreren Worten auch die Kamelhöckerschreibweise gewählt wird.

  Das ist nur Konvention, und stammt natürlich aus dem englischsprachigen Bereich, wo Kleinschreibung von Nomen kein Problem ist. Falls Sie es nicht mögen, könnten Sie natürlich gegen diese Konvention verstoßen. Auf die Dauer empfiehlt es sich aber wohl, Programmtexte ganz in Englisch zu schreiben. Aufgrund der Schlüsselworte bekommen Sie sonst ein Deutsch-Englisches-Mischmasch, und die internationale Zusammenarbeit ist Ihnen auch verwehrt.

- Meist haben Variablen in Methoden ("lokale Variablen") aber eher kurze Namen (nur ein Wort).

  Nicht selten auch nur ein Zeichen, wie `i`, `n`, `s`, `x`. Die Verständlichkeit des Programms darf darunter aber nicht leiden.

Variablen-Deklarationen (3)

- Innerhalb einer Methode (wie z.B. `main`) kann man nicht zwei Variablen mit gleichem Namen deklarieren.
- Der Compiler muss ja wissen, auf welche Variable man sich bezieht, wenn man den Namen verwendet.
- Auch der Name "`args`" des Parameters geht nicht.

  Dies ist auch eine Variable, die ihren Wert automatisch beim Aufruf der Methode erhält.
- Selbstverständlich sind auch die reservierten Worte (Schlüsselworte) wie `class` ausgeschlossen.

  Liste siehe Folie 29.

Zuweisungen (1)

- Man kann den Wert einer Variablen während eines Programmlaufs ändern.
- Dies geschieht mittels einer "Zuweisung" das ist eine Anweisung folgender Art:

```
zinssatz = 3.0;
```

- Eine Zuweisung besteht also (etwas vereinfacht) aus:
  - Name der Variablen,
  - einem Gleichheitszeichen "=",
  - einem Ausdruck, der einen Wert liefert, also z.B. einer Konstanten oder einem komplizierteren Ausdruck mit Variablen und Rechenoperationen.
  - und einem Semikolon ";".

Zuweisungen (2)

- Der Typ des Wertes auf der rechten Seite muß zum Datentyp der Variable auf der linken Seite passen.

  Z.B. kann man nicht `"abc"` (einen String) eine eine Variable speichern, die nach der Deklaration den Typ `int` (ganze Zahl) hat.

- Man beachte, dass die Variable `zinssatz` jetzt an verschiedenen Stellen im Programm einen unterschiedlichen Wert hat:

  - Nach der Deklaration mit der Initialisierung hat sie den Wert `2.0`.
  - Diese Zuweisung ändert den Wert in `3.0`.

Zuweisungen (3)

```
class Zins3 {
    public static void main(String[] args) {
        double betrag = 500.0;
        double zinssatz = 2.0;
        System.out.println("Zinssatz: " +
                           zinssatz);
        double zinsen = betrag * (zinssatz/100);
        System.out.println("Zinsen: " + zinsen);
        zinssatz = 3.0;
        System.out.println("Neuer Zinssatz: " +
                           zinssatz);
        zinsen = betrag * (zinssatz/100);
        System.out.println("Zinsen: " + zinsen);
    }
}
```

Zuweisungen (4)

- Das Programm gibt folgendes aus:

```
Zinssatz: 2.0
Zinsen: 10.0
Neuer Zinssatz: 3.0
Zinsen: 15.0
```

- Falls Sie die Geldbeträge mit zwei Nachkommastellen ausgeben wollen, verwenden Sie z.B.

```
System.out.println("Zinsen: " +
                   String.format("%.2f", zinsen));
```

[http://docs.oracle.com/javase/7/docs/api/java/util/Formatter.html]

- Natürlich werden wir später besprechen, wie man vermeiden kann, dass man die Berechnung der Zinsen zwei Mal aufschreiben muss (mit altem und neuen Zinsatz).