Einführung in die Programmierung mit Java

Teil 1: Grundlagen

Martin Hofmann Steffen Jost

LFE Theoretische Informatik, Institut für Informatik,
Ludwig-Maximilians Universität, München

17. Oktober 2017

LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

Teil 1: Grundlagen

Vorlesung

"**Einführung in die Programmierung**" (9 ECTS)
versus "**Einführung in die Informatik**" (9 oder 6 ECTS):

- Studierende mit integriertem Nebenfach müssen dies mit Ihrem eigenem Prüfungsamt klären!
- Studierende mit regulärem Nebenfach "Informatik":
  - Sollten "Einführung in die Informatik" hören.
  - Freiwillig darf stattdessen "Einführung in die Programmierung" eingebracht werden (empfohlen für Studierende mit mathematischem Hintergrund).
  - Studierende mit Nebenfach Informatik, welche 12=9+3 ECTS benötigen, müssen zusätzlich besuchen:
    Kurs "**Java für Anfänger**", 3 ECTS!
    http://www.mobile.ifi.lmu.de/lehrveranstaltungen/java-fuer-anfaenger-ws1718/
  - Freiwillige EIP-Hörer, welche nur "Einführung in die Informatik" für 6 ECTS benötigen, bekommen zusätzliche ECTS nicht angerechnet.

Übungen

Begleitend zur Vorlesung finden jede Woche Übungen in kleinen Gruppen statt. Zur Teilnahme an einer Übung ist eine Anmeldung bei UniWorX verpflichtend:

http://uniworx.ifi.lmu.de

Anmeldung ab Freitag, 20.10., 13 Uhr möglich. Login bei UniWorX entweder mit @campus.lmu.de oder @cip.ifi.lmu.de
Anmeldung zu Übungsgruppen werden nur überprüft, falls der Raum zu voll sein sollte. Abwesende werden wieder abgemeldet!

Hausübungen Jedes Übungsblatt beinhaltet speziell gekennzeichnete Hausübungen. Diese können per UniWorX abgegeben werden. Dazu wird keine Anmeldung zu einer Übungsgruppe benötigt.
Hausübung sind eine gute Vorbereitung auf die Klausur. Durch Hausübung können Bonuspunkte für die Klausur erzielt werden; nähere Details siehe Vorlesungshomepage

http://www.tcs.ifi.lmu.de/lehre/ws-2017-18/eip

Ressourcen

Literatur Die Vorlesung richtet sich primär nach *C. Horstmann: Big Java, 2007* und *Big Java Early Objects, 2016*.

Weitere Literatur Empfehlungen und die Folien (z.B. zur Verwendung als Notizbuch), finden Sie auf der Vorlesungshomepage:

`http://www.tcs.ifi.lmu.de/lehre/ws-2017-18/eip`

Java In der Vorlesung verwenden wir **Java JDK9**.

Hinweise zur Installation und Empfehlungen zu optionalen Entwicklungsumgebungen finden Sie ebenfalls auf der Vorlesungshomepage (ein beliebiger Editor reicht auch).

Wer Probleme bei der Installation hat oder keinen eigenen Rechner besitzt, kann die vorinstallierten Geräte im **CIP-Pool der Informatik** nutzen.

Inhalt der Vorlesung

- Einführung: Informatik, Java, das erste Programm. in Java: Datentypen, Kontrollstrukturen
- Objektorientierte Programmierung: Klassen, Objekte, Vererbung, Schnittstellen
- Das Java Typsystem, generische Typen.
- Algorithmen für Listen und Bäume, insbes. Balancierung von binären Suchbäumen
- Nebenläufigkeit und Threads
- Entwicklung von grafischen Benutzeroberflächen
- Softwaretechnik: Testen, Modellierung mit UML, Entwurfsmuster, Verifikation mit Hoare Logik

Was ist Informatik?

Von franz. *informatique* (= *information* + *mathématiques*).
Engl.: *computer science*, neuerdings auch *informatics*.

- DUDEN Informatik: Wissenschaft von der systematischen Verarbeitung von Informationen, besonders der automatischen Verarbeitung mit Computern.
- Gesellschaft f. Inf. (GI): Wissenschaft, Technik und Anwendung der maschinellen Verarbeitung und Übermittlung von Informationen.
- Association vor Computing Machinery (ACM): Systematic study of algorithms and data structures.

Teilbereiche der Informatik

- Technische Informatik
  Aufbau und Wirkungsweise von Computern
- Praktische Informatik
  Konstruktion von Informationsverarbeitungssystemen sowie deren Realisierung auf Computern
- Theoretische Informatik
  Theoretische und verallgemeinerte Behandlungen von Fragen und Konzepten der Informatik
- Angewandte Informatik
  Verbindung von Informatik mit anderen Wissenschaften

Typische Arbeitsgebiete

- Algorithmen und Komplexität
- Betriebssysteme
- Bioinformatik
- Datenanalyse, maschinelles Lernen
- Grafik
- Medieninformatik
- Programmiersprachen und Compiler
- Rechnerarchitektur und Rechnernetze
- Robotik
- Simulation
- Softwareentwicklung
- Spezifikation, Verifikation, Modellierung
- Wirtschaftsinformatik

Algorithmusbegriff

Muḥammad ibn Mūsā al-Khwārizmī schreibt um 830 das Lehrbuch *al-Kitāb al-muḫtaṣar fī ḥisāb al-ğabr wa-ʾl-muqābala* (Handbuch des Rechnens durch Ergänzen und Ausgleichen)

Al Chwarizmi

Quelle: Wikimedia, gemeinfrei

Algorithmen und Programmiersprachen

Algorithmus:
Systematische, schematisch ausführbare Verarbeitungsvorschrift.

Alltagsalgorithmen:
Kochrezepte, Spielregeln, schriftliches Rechnen.

Bedeutende Algorithmen:
MP3-Komprimierung, RSA Verschlüsselung, Textsuche, Tomographie,. . .

Programmiersprachen:
erlauben eine ausführbare Beschreibung von Algorithmen. Sie unterstützen Wiederverwendung und Kapselung und erlauben dadurch die Erstellung großer Softwaresysteme.

Die Programmiersprache Java

Entstand Anfang der 1990 aus einem Projekt bei Sun Microsystems zur Programmierung elektronischer Geräte (Set top boxen, Waschmaschinen, etc.). Leiter: James Gosling.
Wurde dann zur plattformunabhängigen Ausführung von Programmen *in Webseiten* ("Applets") verwendet.
Seitdem stark expandiert, mittlerweile neben C und C++ die beliebteste Sprache. Außerdem Hauptsprache für Android-Anwendungen (seit 2008).
Weiterentwicklungen: C#, Scala.
Außerdem Tendenz zu Skriptsprachen (JavaScript, Python, PHP, etc)

Vorteile von Java

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

Nachteile von Java

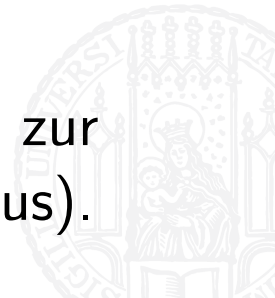
- Teilweise kompliziert und technisch, da nicht für Studenten entworfen (wie Pascal)
- Zu groß für ein Semester
- Ausführung relativ langsam und speicherplatzintensiv.
- Alternativen: C#, Scala, Python, Haskell

Die Java Virtual Machine

Java Programme werden vom Java Compiler übersetzt in eine Folge von elementaren Befehlen (*Bytecodes*), die von einer *virtuellen Maschine* (JVM) ausgeführt werden.
Die JVM verwendet einen Stapel (*stack*) zur Speicherung von Zwischenergebnissen.
Beispiel:

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

1. Lege den Inhalt der Speicherstelle 40 auf den Stapel.
2. Lege den Wert 100 auf den Stapel.
3. Falls der erste Wert größer als der zweite ist, dann springe zur Speicherstelle 240 (ansonsten führe den nächsten Befehl aus).



Plattformunabhängigkeit

Die JVM, die den Bytecode ausführt, ist auf unterschiedlichen Plattformen (Windows, Unix, Mac, Smartphone (Android)) implementiert.
Damit kann Java Bytecode auf all diesen Plattformen ausgeführt werden.
Eine Windows `.exe` Datei kann dagegen nur unter Windows ausgeführt werden.

Das erste Java Programm

```
public class Hello
{
    public static void main(String[] args)
    {
        /* Hier findet die Ausgabe statt */
        System.out.println("Hello, World!");
    }
}
```

Durchführung am Rechner

Um es auszuführen müssen wir

- Eine **Datei** `Hello.java` anlegen
- In die Datei den Programmtext schreiben
- Den Java Compiler mit dieser Datei aufrufen. Er erzeugt dann eine Datei `Hello.class`, die die entsprechenden JVM Befehle enthält. Bei Unix: `javac Hello.java`.
- Diese Datei mit der JVM ausführen. Bei Unix: `java Hello`.

Anatomie unseres Programms

- Einrückungen etc. spielen keine Rolle.
- Kommentare werden in `/*...*/` eingeschlossen. Sie werden von `javac` ignoriert.
  Alternativ kann ein einzeiliger Kommentar auch mit `//` begonnen werden, dieser Kommentar geht nur bis zum Ende der Zeile und benötgt kein schliessenden Zeichen.
- Zwischen den Mengenklammern steht die Definition der Klasse.
- Das Schlüsselwort `public` besagt, dass diese Klasse "öffentlich" sichtbar ist, im Gegensatz zu `private`.

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

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

- definiert eine **Methode** des Namens `main` in der Klasse `Hello`.
- Zwischen den Mengenklammern steht die Definition der Methode.
- Die Methode des Namens `main` wird automatisch beim Programmstart ausgeführt. Andere Methoden, werden innerhalb des Programms aufgerufen. Etwa `berechneZinsen`, `verschiebeRaumschiff`, `openConnection`, ...
- Das Schlüsselwort `static` bedeutet, dass `main` im Prinzip eine Funktion (und keine "richtige" Methode) ist. Mehr dazu später.
- Der **Parameter** `String[] args` erlaubt es, dem Programm beim Aufruf Daten, etwa einen Dateinamen mitzugeben. Man darf ihn nicht weglassen.

Keine Angst

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

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

und in einer Datei *Klassenname*`.java` stehen müssen.

Statements

Die Methodendefinition (der Programmrumpf) besteht aus einer Folge von **Statements** (deutsch: "Befehlen").
Hier haben wir nur ein Statement:

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

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

Mehrere Statements

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

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

Mehrere Statements

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

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

Ergebnis:

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

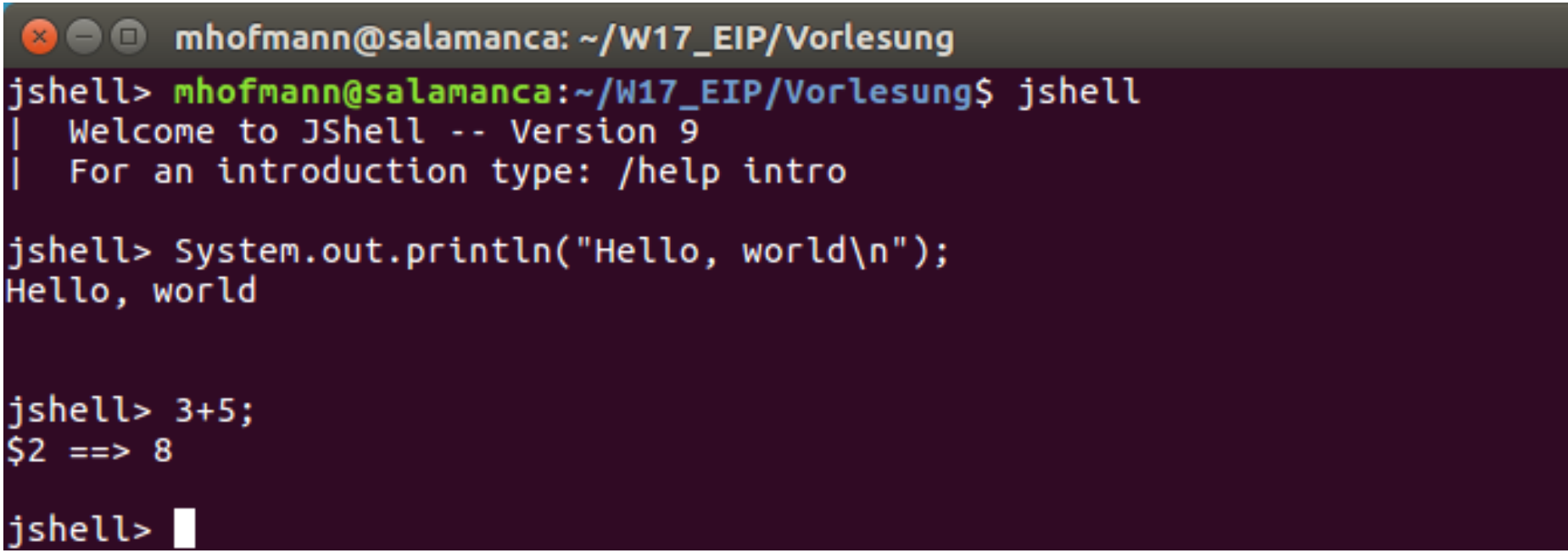
Escapesequenzen

Ein " beginnt und endet eine Zeichenkette. Will man das Symbol " selbst drucken, so muss man \" verwenden.
Das Symbol \ selbst kriegt man durch \\.
Es gibt noch andere solche **Escapesequenzen**, z.B. \n: Zeilenumbruch.

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

Die Java Shell

Seit 2017 kann man auch interaktiv Java-Befehle (*snippets*) auswerten. Das ist zum Lernen und Ausprobieren nützlich.



Klassen und Objekte

Objekte enthalten Werte und Methoden (um aus diesen Werten Ergebnisse zu berechnen *und* um diese Werte zu verändern).
Klassen dienen als Muster für Objekte.
Die Klasse spezifiziert die Formate der Werte, und definiert die Methoden.
Beispiel: die eingebaute Klasse `Rectangle`. Der Ausdruck

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

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

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

gibt das Objekt aus:

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

Beispielprogramm

```
import java.awt.Rectangle;

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

Die Deklaration `import java.awt.Rectangle;` importiert den Klassennamen aus der **package** `java.awt`.
Alternativ kann man auch `java.awt.Rectangle` schreiben.

Programmvariablen

Durch das Statement

```
Rectangle cornflakesPackung;
```

wird eine **Programmvariable** (kurz **Variable**) des **Typs** `Rectangle` deklariert.
Man kann der Variablen einen Wert zuweisen durch `=`. Z.B.

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

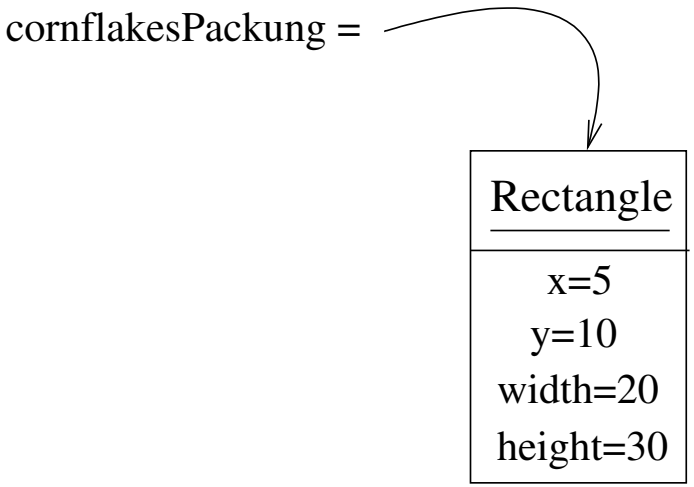
Und dann ausgeben:

```
System.out.println(cornflakesPackung);
```

Liefert:

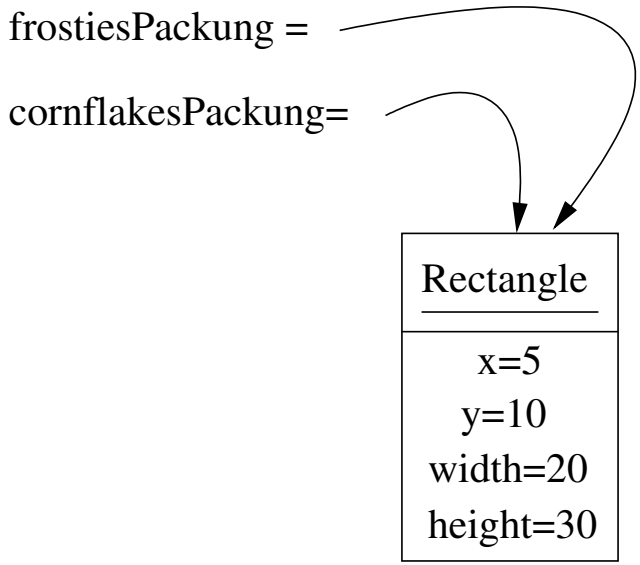
```
java.awt.Rectangle[x=5,y=10,width=20,height=30]
```

Die Programmvariable enthält einen **Verweis** auf ein Objekt.



Schreiben wir

```
Rectangle frostiesPackung =
    cornflakesPackung;
```



(Beachte, Deklaration und Zuweisung gehen in einem.)
Dann ist `frostiesPackung` auch ein Verweis auf das erzeugte Objekt.
Es gibt aber nach wie vor nur eins!

Methoden

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

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

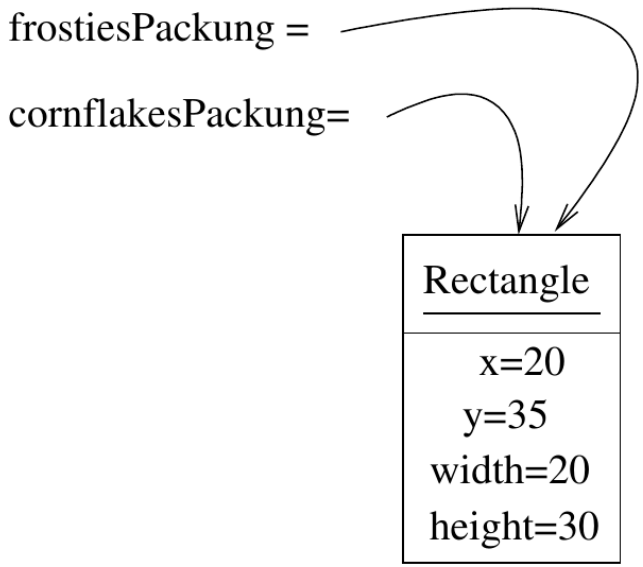
Geben wir jetzt `cornflakesPackung` aus, dann erhalten wir

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

Was kommt heraus, wenn wir `frostiesPackung` ausgeben?

Antwort:

Genau dasselbe, da ja `frostiesPackung` und `cornflakesPackung` auf **dasselbe** Objekt verweisen.



Dokumentation mit Javadoc

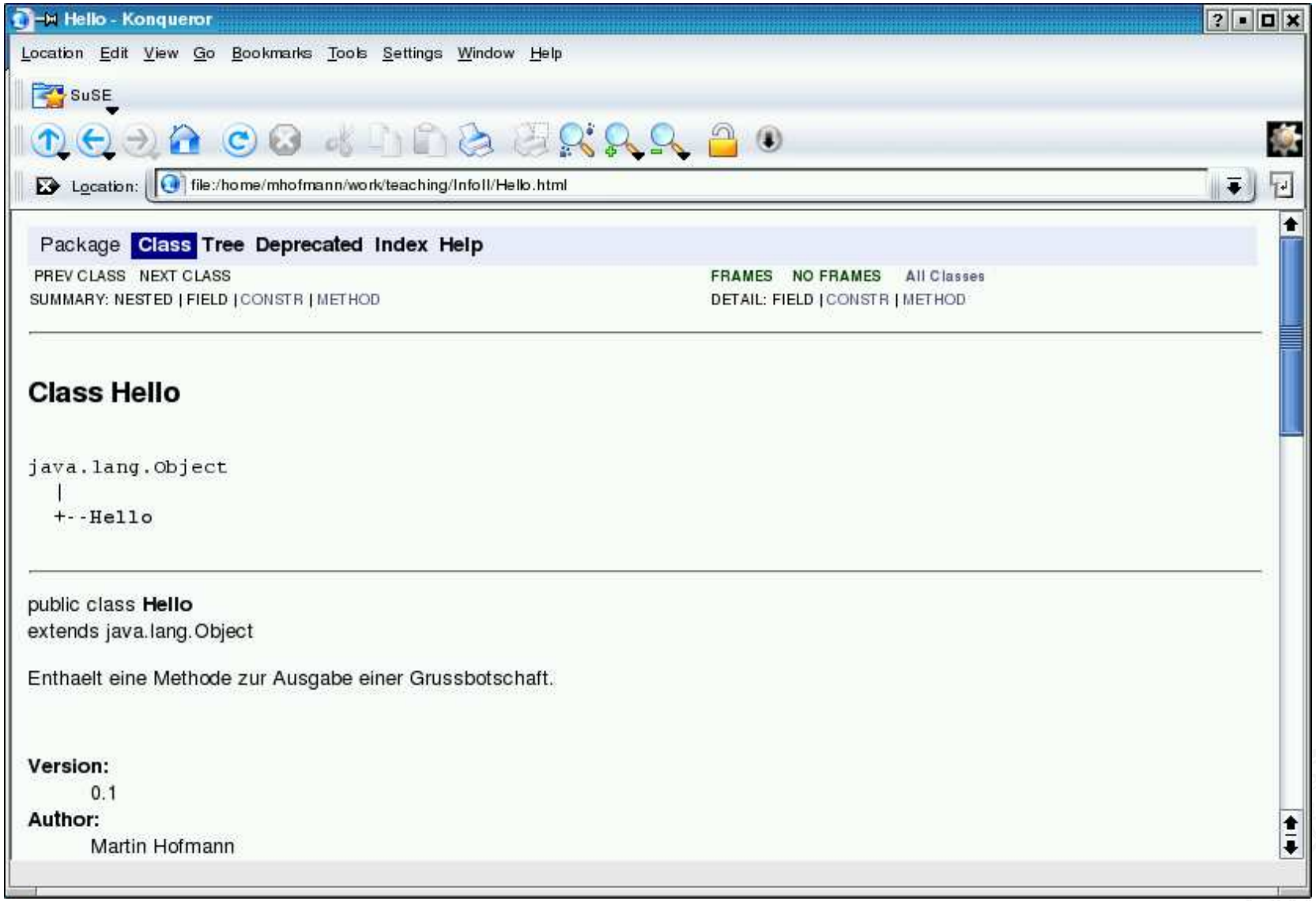
Mit `javadoc` können bestimmte Kommentare zur Erzeugung von HTML (mit Internet Browser lesbarer) Dokumentation verwendet werden:

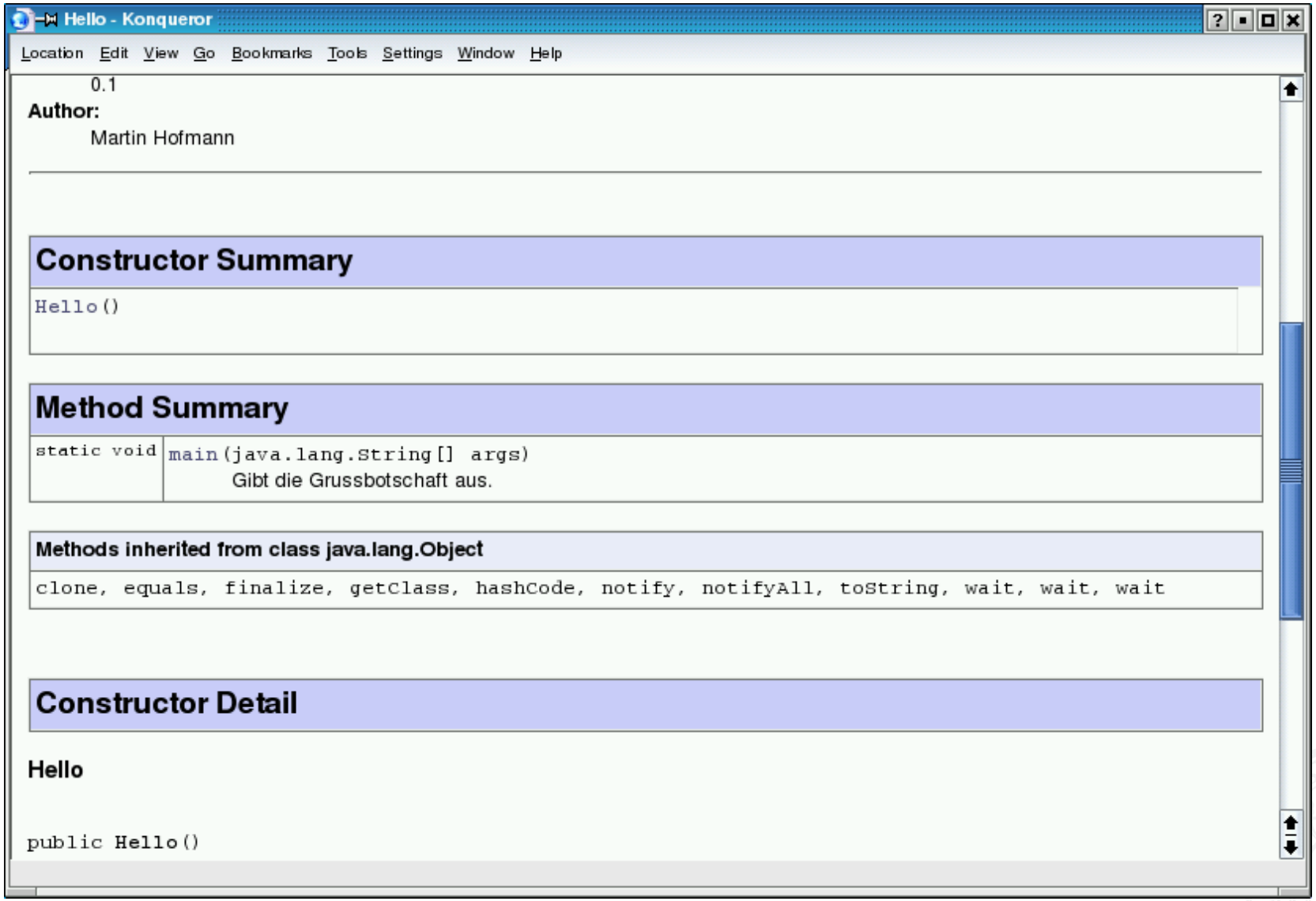
- Ein Javadoc Kommentar beginnt immer mit `/**` und endet mit `*/`.
- Zeilen innerhalb eines Javadoc Kommentars dürfen mit `*` beginnen.
- Ein Javadoc Kommentar soll immer vor einer Deklaration stehen (Klasse, Methode, ...)
- weitere Regeln: siehe Beispiele und `man javadoc`.

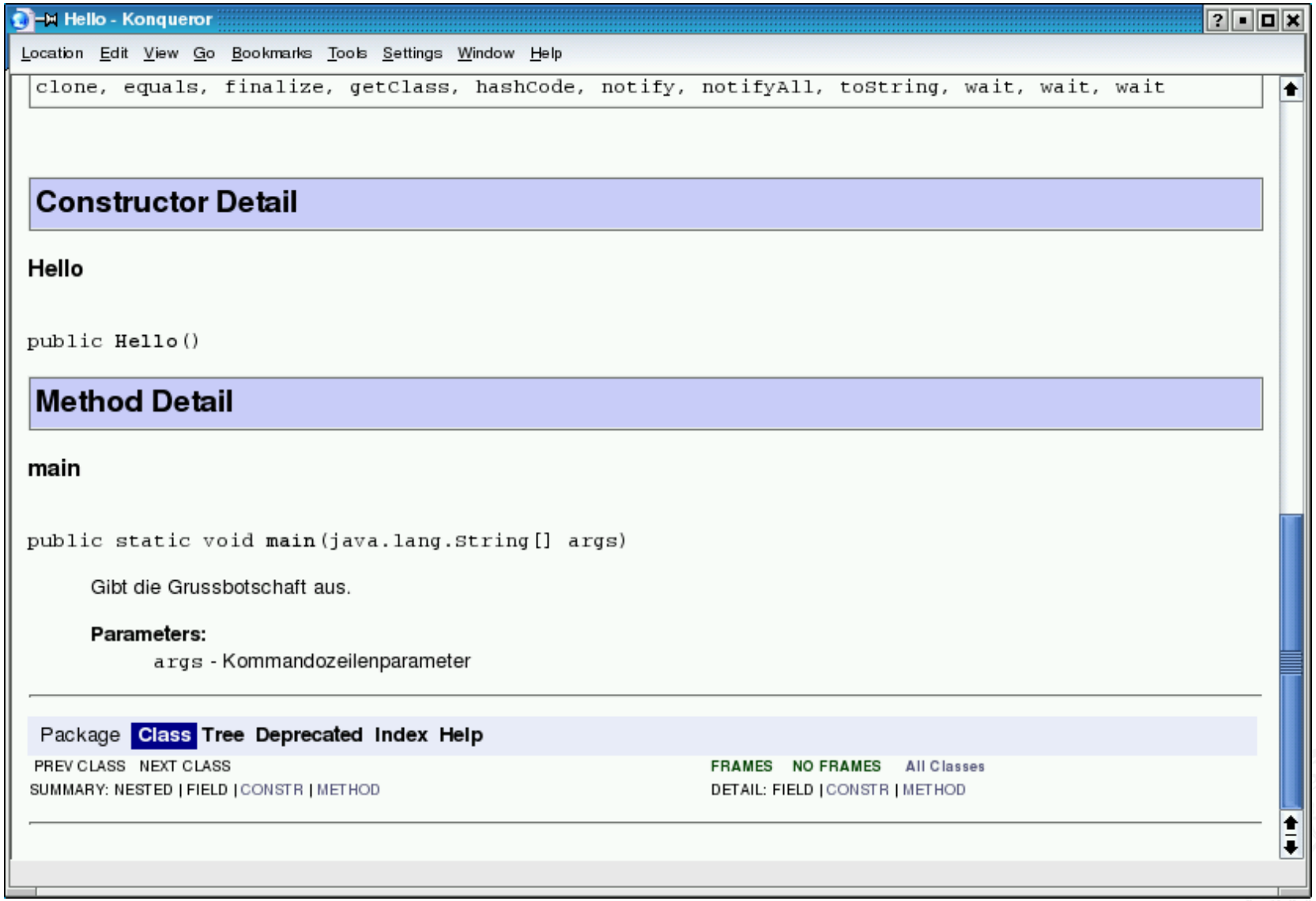
Der Befehl `javadoc -version -author` *Datei*`.java` erzeugt eine Datei *Datei*`.html`, die eine schön formatierte Dokumentation enthält.

Beispiel

```
/**
 * Enthaelt eine Methode zur Ausgabe einer Grussbotschaft.
 * @author Martin Hofmann
 * @version 0.1
 */
public class Hello
{    /** Gibt die Grussbotschaft aus.
     * @param args Kommandozeilenparameter
     */
    public static void main(String[] args)
    {
        /* Hier findet die Ausgabe statt */
        System.out.println("Hello, World!");
    }
}
```


Hello - Konqueror
Location Edit View Go Bookmarks Tools Settings Window Help
SuSE
Location:
file:/home/mhofmann/work/teaching/InfoII/Hello.html
Package Class Tree Deprecated Index Help
PREV CLASS NEXT CLASS
FRAMES NO FRAMES All Classes
SUMMARY: NESTED | FIELD | CONSTR | METHOD
DETAIL: FIELD | CONSTR | METHOD
Class Hello
java.lang.Object
|
+--Hello
public class Hello
extends java.lang.Object
Enthaelt eine Methode zur Ausgabe einer Grussbotschaft.
Version:
0.1
Author:
Martin Hofmann


Hello - Konqueror
Location Edit View Go Bookmarks Tools Settings Window Help
0.1
Author:
Martin Hofmann
Constructor Summary
Hello()
Method Summary
static void
main(java.lang.String[] args)
Gibt die Grussbotschaft aus.
Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait
Constructor Detail
Hello
public Hello()


Hello - Konqueror
Location Edit View Go Bookmarks Tools Settings Window Help
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait
Constructor Detail
Hello
public Hello()
Method Detail
main
public static void main(java.lang.String[] args)
Gibt die Grussbotschaft aus.
Parameters:
args - Kommandozeilenparameter
Package Class Tree Deprecated Index Help
PREV CLASS NEXT CLASS
FRAMES NO FRAMES All Classes
SUMMARY: NESTED | FIELD | CONSTR | METHOD
DETAIL: FIELD | CONSTR | METHOD