Java Standard

de.wikibooks.org

24. Juli 2016

Inhaltsverzeichnis

1 Einleitung

1.1 Nötiges Vorwissen

Als Vorwissen für dieses Buch wird vorausgesetzt, dass der Nutzer mit dem Computer und der Kommandozeile (*Shell*) umgehen kann. Desweiteren werden Kenntnisse der objektorientierten Programmierung[1] vorausgesetzt.

Nicht notwendig, aber nützlich, ist die Kenntnis einer anderen mit Java verwandten Programmiersprache, wie etwa C++.

1.2 Geschichte und Namensgebung

Die Programmiersprache **Java** wurde 1991 von James Gosling[2] und anderen im Hause Sun Microsystems[3] entwickelt. Der ursprüngliche Name lautete jedoch *Oak* für eine große Eiche außerhalb des goslingschen Büros.

Eine ausführliche Beschreibung[4] mit Spezifikationen der Programmiersprache ist in der deutschsprachigen Wikipedia[5] zu finden.

1.2.1 Versionen von Java

Es gibt mehrere Versionen von Java, wobei grob gesagt werden kann, dass die vorletzte Version die am häufigsten eingesetzte Version ist. Dieses Buch ist im Grundstil dafür geschrieben, dass mit dieser Version gearbeitet wird. Zur Zeit ist die Version 1.4 die am meisten eingesetzte Version.

Auf die neueren Features von Java 5.0 (Tiger) wird ebenfalls an den geeigneten Stellen eingegangen.

Version 1.0 (1996)

Bild:JDK-1.1-icon.png Version 1.1 (1997)

Wesentliche Änderung:

1 https://de.wikipedia.org/wiki/Objektorientierte_Programmierung
2 https://de.wikipedia.org/wiki/Gosling
3 https://de.wikipedia.org/wiki/Sun%20Microsystems
4 https://de.wikipedia.org/wiki/Programmiersprache_Java
5 https://de.wikipedia.org/wiki/Wikipedia

- Neues Event Handling

Bild:J2SE-1.2-icon.png Version 1.2 (1998) / Java 2

Wesentliche Änderung:

- Aufnahme von Swing in die J2SE

Bild:J2SE-1.3-icon.png Version 1.3 (2000)

Bild:J2SE-1.4-icon.png Version 1.4 (2002)

Wesentliche Änderungen:

- Unterstützung von Zusicherungen (Assertions)
- Aufnahme einer Logging API
- Aufnahme einer API für reguläre Ausdrücke

Bild:J2SE-5.0-icon.png Version 5.0 (2004)

Wesentliche Änderungen:

- Generische Datentypen
- Enumerationen
- Neue zusätzliche Syntax für for-Schleifen über Datenstrukturen
- Annotationen für Metadaten

Version 6.0 (2006)

Sun Microsystems hat erstmalig sehr früh einen Einblick in die neue Version (Codename Mustang[6]) gewährt. Einige Themen stehen zur Diskussion, darunter unter anderem aspekt orientierte Entwicklung [AOP], ein Thema, welches dem Entwickler sicher ebenso viel Freude machen dürfte wie die generischen Datentypen in Version 5 oder die Logging API in Version 1.4.

1.3 Warum Java?

Hier werden Ihnen die Vor- und Nachteile von Java aufgezeigt, damit Sie sich besser entscheiden können, ob es sich für Sie lohnt, diese Programmiersprache zu erlernen.

6 `http://mustang.dev.java.net`

1.3.1 Vorteile

Java ist mittlerweile für die verschiedensten Computersysteme verfügbar und hat eine weite Verbreitung gefunden. Ebenso bringt Java eine umfangreiche Klassenbibliothek mit, die für (fast) alle täglichen Programmieraufgaben eine Unterstützung enthält. Durch das Konzept der virtuellen Maschine ist ein einmal compilierter Programmcode auf jeder Plattform, für die eine Java VM vorhanden ist, lauffähig. Ein erneutes Übersetzen auf der jeweiligen Zielplattform (wie bei C/C++) ist nicht mehr notwendig. Ein weiterer Vorteil sind die sogenannten "Applets". Dies sind Java-Programme, die innerhalb eines Web-Browsers gestartet werden können und somit sehr einfach Applikationen über das Internet verfügbar machen.

Ein handfester Vorteil - nicht nur für große Projekte - ist die mittlerweile freie Verfügbarkeit von integrierten Entwicklungsumgebungen, allen voran die Eclipse-Plattform. Für viele Programmierer ist gerade die Werkzeugunterstützung ein entscheidendes Kriterium bei der Auswahl einer Programmiersprache. Hier ist Java eine der am aktivsten unterstützen Plattformen.

Java hat sich mittlerweile als Industriestandard etabliert.

1.3.2 Nachteile

Java-Programme benötigen zur Ausführung eine Laufzeit-Umgebung. Auf vielen Computern ist diese nicht vorinstalliert und muss erst separat eingerichtet werden. Gleiches gilt jedoch auch für andere beliebte Programmiersprachen, z.B. .NET oder Perl. Ebenso sind die in gängigen Browsern eingebauten Java-Applet-Laufzeit-Umgebungen häufig veraltet. Will man Java-Programme für diese Browser schreiben, so muss man sich auf eine alte Sprachversion beschränken oder auf dem Browser eine aktuelle Laufzeitumgebung nachinstallieren.

Des Weiteren ist Java nicht geeignet, um systemnahe Programme wie etwa Hardwaretreiber zu entwickeln. Das liegt im Wesentlichen daran, dass Java-Programme auf theoretisch beliebigen Rechnerarchitekturen und Betriebssystemen unverändert lauffähig sein sollen. Diese Abstraktion lässt einen direkten Zugriff auf spezifische hardwarenahe Funktionen nicht mehr zu. Andere Sprachen wie etwa C oder C++ sind für derartige Aufgaben besser geeignet.

1.3.3 Geschwindigkeit von Java-Programmen

Die vorherrschende Meinung zur Geschwindigkeit von Java-Programmen ist, dass Java-Programme langsamer als vergleichbare C- oder C++-Programme seien. Das kann man jedoch nicht pauschal so sagen, denn die Geschwindigkeit von Java-Programmen hängt von verschiedenen Faktoren ab.

Eine große Rolle spielt zunächst die *Virtuelle Maschine (VM)*, auf der das Java-Programm abläuft. In den Anfangszeiten wurde Java-Code interpretiert. Dies führte zu Leistungseinbußen, die bei modernen Java-VMs praktisch nicht mehr gegeben sind. Die Java-VM von SUN beispielsweise "kompiliert" Java-Programme sozusagen zur Laufzeit. Im Ergebnis erhält man ein Programm, das kaum noch langsamer ist als ein vergleichbares, das in C oder C++ geschrieben wurde.

Einen grundsätzlichen Geschwindigkeitsnachteil haben Java-Programme beim "Anfahren", denn jedes Java-Programm muss zunächst für sich eine eigene VM starten. Außerdem dauert das Laden der Klassen sehr lange. Neuere Implementierungen der VM (2004) beheben letzteren Nachteil dadurch, dass Klassen von mehreren Programmen gleichzeitig benutzt werden können, so dass, nachdem die erste Java-Applikation gestartet wurde, für die nachfolgenden das Laden entfällt.

Einen weiteren Geschwindigkeitsnachteil haben Java-Programme dadurch, dass bei jedem Feldzugriff die Bereichsgrenzen überprüft werden. Moderne VMs können aber die Überprüfung weitestgehend "wegoptimieren", indem bei Schleifendurchläufen - anstatt bei jedem Feldzugriff - die Bereichsüberprüfung vor dem Schleifenrumpf platziert wird. Auf diese Art lassen sich etwa 80 Prozent der ansonsten anfallenden Bereichsüberprüfungen eliminieren.

Noch ein Nachteil entsteht Java-Code dadurch, dass viele Methoden (genauer: nicht-finale Instanzmethoden) zunächst *virtuell* sind. Auch hier haben jedoch Compiler die Möglichkeit, Optimierungen durchzuführen; und weil die Laufzeitumgebung von Java auf die Ausführung von virtuellen Methoden optimiert ist, sind Aufrufe von virtuellen Methoden in Java erheblich schneller als in den meisten C++-Compilern.

Nicht zuletzt spielt auch die Bibliothek eine Rolle. Die von SUN favorisierte *Swing*-Bibliothek, die in erster Linie für die Grafikausgabe zuständig ist, steht nicht im Ruf, besonders schnell zu sein.

Es gibt aber auch Bedingungen, unter denen Java-Programme Geschwindgkeits*vorteile* gegenüber C- oder C++-Programmen haben. Beispielsweise ist die Objekterzeugungsgeschwindigkeit bei Java sehr hoch. Wenn es also darum geht, viele Objekte in kurzer Zeit zu erzeugen, können Java-Programme in der Praxis hier Vorteile ausspielen. Java hat auch weniger Aliasing-Probleme als C oder C++. Aliasing bedeutet, dass sich Speicherbereiche von formal unterschiedlichen Objekten überlappen. Da es in Java weniger Aliasing-Probleme gibt, können Java-Programme bei numerischen Aufgaben gewinnbringend eingesetzt werden. (Siehe Leistungsvergleich zwischen Java und C++[7].)

Ob Ihr eigenes Java-Programm in der Praxis schneller oder langsamer ist als eines, das in C++ geschrieben wurde, hängt aber noch von vielen anderen Faktoren ab. Wie schon angedeutet, können Java-Programme auf schnellen und auf langsamen VMs laufen. Und wenn Sie die Geschwindigkeit mit C++ vergleichen, dann spielt selbstverständlich auch die Implementierung des zugrundeliegenden C++-Compilers eine Rolle.

Groß ist der praktische Geschwindigkeitsunterschied heutzutage in den meisten Fällen jedenfalls nicht.

Jedes Java-Programm lässt sich aber unendlich langsam machen, wenn man nur unendlich schlecht programmiert. Der Hauptfaktor dabei sind also Sie.

Um mit Java zu arbeiten, benötigt man - wie in jeder anderen Programmiersprache auch - ein paar Werkzeuge, wie zum Beispiel einen Compiler und einen Editor. Diese Werkzeuge werden hier jetzt erläutert, damit Sie wissen, was Sie benötigen, wenn Sie mit Java anfangen wollen.

7 `http://www.kano.net/javabench/`

1.4 Allgemeines

1.4.1 SDK mit JRE

Zur Programmierung von Java benötigt man zum Anfang eigentlich wenige Werkzeuge. Minimal benötigt man:

- Ein Java-Software-Development-Kit (*Java SDK*, früher und jetzt wieder auch JDK genannt) für das verwendete Betriebssystem.
  Das SDK muss für das Betriebssystem bestimmt sein. Im Gegensatz zu Java-Programmen selbst ist das SDK nicht plattformunabhängig. Sun stellt verschiedene SDKs für gängige Betriebssysteme wie Windows und Linux und die Sun-spezifische Unix-Variante Solaris zur Verfügung, die von Suns Webseite[8] heruntergeladen werden können. Benötigt wird zu Anfang nur die mit JSE 6[9] bezeichnete aktuelle Version (JSE steht hierbei für Java Standard Edition)! Je nach Arbeitsrichtung können später noch spezielle APIs und/oder eine andere Java Edition hinzukommen.
  Wird man bei Sun für das eigene Betriebssystem nicht fündig, so ist der Betriebssystem-Hersteller der nächste Ansprechpartner. So bieten z.B. Apple und IBM Java SDKs für die eigenen Betriebssysteme auf ihren Webseiten an. Bitte achten Sie beim Herunterladen darauf, dass Sie wirklich das SDK herunterladen (ca. 50 MB) und nicht das JRE (Java Runtime Environment) - dies ist im SDK enthalten.
- Einen beliebigen Texteditor.
  Dies kann z.B. unter Windows *notepad* sein (nicht sonderlich bequem) oder aber ein typischer Programmiereditor wie *vi*[10] oder *emacs* aus der Unix-Welt.

Zusätzlich lohnt sich:

- Die Java Dokumentation - sie beinhaltet u. a. die Java API Dokumentation im sogenannten Javadoc-Format. Aber es enthält auch eine vollständige Sprachbeschreibung, die Beschreibung aller Werkzeuge, wie Compiler oder virtuelle Maschine, Tutorien und vieles mehr.
  Es lohnt sich, die Dokumentation lokal zu installieren. Man kann sie sich z.B. auch von der Sun Webseite herunterladen.
- Ein Webbrowser zum Lesen der Java Dokumentation.
  Einen Browser besitzen Sie zweifellos, da Sie diesen Text gerade lesen.

Wenn die Entwicklung in Java allerdings komfortabel sein soll, dann gibt es diverse so genannte integrierte Entwicklungsumgebungen (IDEs), die Ihnen die täglichen Routineaufgaben erleichtern bzw. abnehmen. Professionelle IDEs wie das sehr populäre Eclipse[11] sind groß, mächtig und es braucht einige Zeit sie komplett zu beherrschen. Daher wird auf IDEs hier nicht weiter eingegangen. Der Umgang mit einer speziellen IDE wird ggf. Bestandteil eines eigenen Wikibook werden.

8 `http://java.sun.com/`
9 `http://java.sun.com/javase/downloads/widget/jdk6.jsp`
10 `https://de.wikibooks.org/wiki/Learning_the_vi_editor%20`
11 `http://www.eclipse.org`

Ein Hinweis auf eine speziell für das Erlernen von Java und objektorientierter Konzepte gedachte IDE sei hier dennoch erlaubt: BlueJ[12] wird von diversen Universitäten gepflegt[13], ist bewusst einfach gehalten und wird gerne im Lehrbetrieb eingesetzt.

In diesem Buch wird jedoch vom einfachsten Fall ausgegangen, dass Sie einen Editor besitzen und das Java-Software-Development-Kit.

Neben dem Begriff *SDK* findet man auch den Begriff *JRE*. Das JRE (*Java Runtime Environment*) ist die Laufzeitumgebung, die dazu dient, Java-Programme auszuführen. Das SDK von Sun enthält bereits eine Version des JRE, so dass man dieses nicht separat herunterladen und installieren muss. Alleine mit dem JRE lassen sich keine Programme entwickeln (es fehlt z.B. der Compiler). Zur Entwicklung braucht man immer das SDK. Zur Ausführung von Java-Programmen reicht das JRE. Die JRE ist ebenso plattformspezifisch wie das SDK.

1.5 Installation des SDK

1.5.1 Installieren des SDKs unter Windows

Für Windows installieren Sie einfach die selbst-extrahierende Datei. Nach dem Neustart könnten Sie schon anfangen, möchten Sie jedoch z.b. die Programme "javac.exe", "java.exe", "javaw.exe", "jar.exe" etc. von jedem Ordner aus ausführen können ohne immer den vollständigen Pfad angeben zu müssen, müssen Sie die Umgebungsvariable "PATH" verändern.

Wir nehmen mal an, Sie haben JDK an dem Ort "C:\jdk1.6.0_<version>" installiert. Nun müssen Sie die PATH-Variable um den Eintrag "C:\jdk_1.6.0_<version>\bin" erweitern. Die PATH-Variable kann man **unter Windows 2000/XP** unter *Start -> Einstellungen -> Systemsteuerung* dann auf *System* und *Erweitert* verändern. In diesem Dialog wählen Sie die Schaltfläche *Umgebungsvariablen* und suchen sich unter *Systemvariablen* die PATH-Variable aus und fügen den Pfad des SDK an. Der neue Pfad wird von den Alten durch ein Semikolon getrennt. Nach dem Bestätigen können Sie testen, ob die PATH-Variable verändert wurde, in dem Sie in der Eingabeaufforderung PATH eingeben. In der Auflistung sollten Sie nun den Pfad des SDK wiederfinden.

Unter **Vista** kommen Sie dort hin mit: *Windows* + Pause -> *Erweiterte Systemeinstellungen* -> Reiter *Erweitert* -> Button *Umgebungsvariablen...* -> Scrollfeld *Systemvariablen*.

Nach den Änderungen sollten Sie Ihren PC neu starten.

Registry Einträge

Bei der Standardinstallation werden in der Windows Registry verschiedene Schlüssel eingetragen. Diese unterscheiden sich in ihrer Position je nach Windows Version:

Version	**Basisschlüssel**

12 http://www.bluej.org
13 http://www.bluej.org/about/what.html

Version	Basisschlüssel
Windows 95	HKLM\Software\JavaSoft
Windows 98	HKLM\Software\JavaSoft

Tipp: Auch wenn die Standardinstallation sich in der Registry einträgt, um somit u.a. die aktuellste Java Version zu finden, sind diese nicht zwingend. Jedoch muss ansonsten die JAVA_HOME Umgebungsvariable korrekt gesetzt sein.

Unterhalb dieser Schlüssel werden im *Prefs/* Teil die Benutzereinstellungen des Pakets *java.util.prefs* gesichert.

1.5.2 Installieren des SDK unter Linux

Die Schwierigkeit unter Linux ist die Organisation der Dateien, die von Distribution zu Distribution variieren können. Deshalb wird hier nur beispielhaft erklärt, wie für welche Distribution das SDK installiert wird.

Seit einiger Zeit bietet Sun selber ein SDK für Linux an. Man kann auch dieses von Suns Webseite herunterladen. Alternativ hat bereits früher die Blackdown-Gruppe[14] Suns Unix-SDK mit einer Lizenz von Sun nach Linux portiert. Enventuell sind für die saubere Installation des SDK von Sun bzw. Blackdown noch weitere Schritte notwendig um das SDK gut ins Betriebssystem einfügen zu können.

Zudem sind verschiedene (freie) Java-Clones für Linux erhältlich, die aber leider häufig nicht auf dem Stand von Suns SDK sind, so dass sie sich nur unter eingeschränkten Bedingungen zur Java-Entwicklung eignen.

SuSE Linux

Sie benötigen, um Java mit *SuSE Linux* programmieren zu können, die Professional-Ausgabe dieses Betriebssystems. Dies ist notwendig, da nur dort die benötigten Entwicklungstools mit auf den CDs bzw. DVDs enthalten sind. Alternativ können Sie sich die fehlenden Bestandteile aus dem Internet zusammen suchen. Ein guter Anlaufpunkt ist dafür der FTP-Server von [ftp://www.suse.com/ SuSE].

Nun die Schritte im Einzelnen:

1. Starten Sie Yast. Das geschieht über *Kontrollzentrum -> YAST2 Module -> Software -> Software installieren.*
2. Klicken Sie nun auf die Schaltfläche *Systemverwaltungsmodus* und geben Sie Ihr Root-Passwort ein.
3. Wählen Sie jetzt *Paketgruppen* aus und Klicken sie auf *Entwicklung -> Programmiersprachen -> Java.*
4. Jetzt müssen die die Pakete *Java 2* und *Java 2 JRE* auswählen.
5. Klicken Sie jetzt auf *Akzeptieren* und die benötigte Software wird auf Ihrem Linux-Rechner installiert.

14 `http://blackdown.org/`

Die CLASSPATH-Umgebungsvariable müssen Sie für dieses Beispiel nicht extra setzen ;).

Suse liefert in der aktuellen Distribution (z.Zt. Suse 9.3) Java 1.4 mit, was zum Erlernen von Java ausreichen sollte. Falls dies bei Ihnen nicht der Fall sein sollte (z.B. weil Sie spezielle Features von Java 5.0 benötigen oder ausprobieren wollen) müssen Sie folgende Schritte für Ihr System nachvollziehen:

1. Installation der JPackage-Pakete von der Suse CD/DVD (suchen nach "JPackage")

1. Download von Suns SDK (`jdk-1_5_0_04-linux-<architektur>.bin`, wobei "architektur" für die jeweilige Architektur steht also z.B. "i586" oder "Amd64".
2. Verschieben oder kopieren des heruntergeladenen SDK mit `cp jdk-1_5_0_04-linux-<architektur>.bin /usr/src/packages/SOURCES`
3. Wechseln des Verzeichnisses `cd /usr/src/packages`
4. Bauen der einzelen RPMs mit `rpmbuild -bb SPECS/java-1.5.0-sun.spec`
5. Die fertigen Pakete liegen jetzt in `/usr/src/packages/RPMS/<architektur>`" - diese können mit `rpm -Uhv java*` installiert werden.

Debian

Das Sun JDK wurde nicht unter GPL gestellt. Daher ist sowohl Suns JDK als auch die Blackdown-Variante nicht in den Free bzw. Non-Free-Zweigen der Debian-Distributionen[15] enthalten. Eine Linux-Binary kann jedoch von www.oracle.com[16] heruntergeladen werden. Dort sind auch alle Installationshinweise zu finden.

Alternativ bietet Debian ein SDK, welches komplett aus freien Komponenten besteht. Während es nicht unbedingt alle Funktionalität des Sun-SDKs bietet, reicht es für viele Anwendungen aus. Die Installation erfolgt mittels

```
aptitude install free-java-sdk
```

Nach dem Setzen der JAVA_HOME-Umgebungsvariable steht ein SDK mit gewöhnlicher Bedienung zur Verfügung.

```
export JAVA_HOME=/usr/lib/fjsdk
```

Red Hat

Bevor man das JDK auf Red Hat Linux installieren kann, muss man sicherstellen das GCJ und seine Komponenten nicht installiert sind oder wenn sie installiert sind, das man sie entfernt.

15 http://debian.org
16 http://www.oracle.com/technetwork/java/javase/index-137561.html

Gentoo Linux

Die einfachste Möglichkeit Java unter Gentoo zu installieren ist, die blackdown-version zu emergen.

```
emerge blackdown-jdk
```

Bei diesem ebuild sind alle nötigen Programme enthalten. Jetzt muss man nur noch blackdown als Standard-VM setzen, was am einfachsten mittels java-config geschieht.

```
java-config -L
```

Dieser Befehl listet alle verfügbaren VMs auf. Nun wählt man die Blackdown-VM aus.

```
java-config -S blackdown-jdk-[eure version]
```

Nun sollte das Kompilieren mit javac [datei.java] klappen. Aufgerufen werden die Programme mit java [datei], ohne die .class-Endung. Wer die Sun-j2sdk emergen will sollte trotzdem erst Blackdown installieren, da bei dieser z.B. java-config schon enthalten ist, bei der Sun-Version nicht. Man kann ja danach die VM auf Sun-j2sdk setzen.

Ubuntu

Die Installation des Java SDK(JDK) unter Ubuntu[17] ist sehr einfach: Das JDK von Sun befindet sich in den offiziellen Quellen (multiverse) ab Version 7.04 (Feisty Fawn). Über den Paketmanager installiert man das Paket *sun-java6-jdk* oder führt folgenden Befehl im Terminal aus:

```
$ sudo apt-get install sun-java6-jdk
```

Überprüfen kann man die Installation, indem man im Terminal den Befehl

```
$ javac -version
```

ausführt. Folgende Meldung sollte erscheinen:

```
javac 1.6.0_22
```

1.5.3 Mac OS X

Java ist integraler Bestandteil des Betriebssystems Mac OS X und muss dort nicht gesondert installiert werden. Es ist herstellerseitig unter /System/Library/Frameworks/JavaVM.framework/Versions installiert. In diesem Verzeichnis

17 https://de.wikipedia.org/wiki/Ubuntu

befinden sich jeweils Ordner mit den entsprechenden Java-Versionen. Standardmäßig sind folgende Versionen installiert bzw. erhältlich:

Mac OS X	**Vorinstalliert**	**erhältlich**
10.1	1.3.1	1.3.1
10.2	1.3.1	1.4.2
10.3	1.3.1	1.4.2
10.4	1.4.2	1.5.0
10.5	1.5.0	1.6.0

Die erhältlichen Java-Versionen können automatisch über Software-Update bezogen und installiert werden. Sind mehrere Java-Versionen installiert, so kann mit Hilfe eines kleinen Programmes eingestellt werden, welche JRE verwendet wird.

1.5.4 BSD

Unter BSDs gibt es zwei Probleme beim Installieren von Java: zum einen darf man die Installationsdateien aus lizenztechnischen Gründen nicht automatisiert runterladen, zum anderen stellt Sun keine nativen Binärdateien von Java für BSD bereit. Man muss entsprechende Dateien also manuell herunterladen. Falls es sich dabei nicht um inoffizielle BSD-Binarys handelt, muss man zuerst eine JDK-Umgebung mit Linux-Emulator installieren die als Bootstraper fungiert, aus der man danach native Binarys kompilieren kann. Da das aber unnötig lange dauert, wird hier nur der direkte Weg über inoffizielle Pakete beschrieben.

FreeBSD

Die einfachste Möglichkeit an ein JDK zu kommen ist der Port java/diablo-jdk15. Dazu lädt man einfach den entsprechenen Tarball von `http://www.freebsdfoundation.org/downloads/java.shtml` runter, legt ihn in /usr/ports/distfiles ab und gibt dann

```
portinstall diablo-jdk15
```

oder

```
cd /usr/ports/java/diablo-jdk15
make install clean
```

ein. Alternativ lädt man sich das Package von der Seite herunter und installiert es mittels

```
pkg_add diablo-jdk-freebsd<version>.<arch>.1.5.0.07.00.tbz
```

wobei Werte in spitzen Klammern vor der Eingabe ersetzt werden müssen.

Pkgsrc

Unter Pkgsrc muss man sich zuvor das JDK (jdk-1_5_0-p3-bin-duh1-bsd-i586.tar.bz2) von `http://www.duh.org/NetBSD/java2/` runterladen, in /usr/pkgsrc/distfiles/ ablegen und anschließend den Port lang/scsl-jdk15 mittels

```
cd /usr/pkgsrc/lang/scsl-jdk15
make install clean
```

installieren.

1.6 Eine IDE einrichten

1.6.1 Eclipse

Eclipse[18] ist eine kostenlose IDE, die ursprünglich von IBM ins Leben gerufen wurde und nun als Open-Source-Projekt, unterstützt von einem Konsortium namhafter Firmen wie z.B. Intel, HP, SAP oder SuSE, voran getrieben wird. Das von Haus aus als erweiterbare Plattform für Plug-ins konzipierte Framework erlangt erst durch diese seine Funktionen. Die mittlerweile beliebteste in Java geschriebene IDE ist für die Betriebssysteme Linux, Max OS X sowie MS-Windows als *Download* von der Hersteller-Website verfügbar und bringt in der Grundausstattung einen sehr komfortablen Java-Editor mit. Mithilfe von entsprechenden Plug-ins kann man unter Eclipse auch noch mit anderen Programmiersprachen (u. a. C/C++, Cobol und PHP) arbeiten. Die Anzahl der Plug-ins steigt sehr rasant an und reicht von kostenlosen Erweiterungen bis hin zu teueren kommerziellen Produkten.

- Zur Installation der Entwicklungsumgebung Eclipse[19] benötigen Sie nur das JRE oder das Java Software Development Kit (Java SDK). Letzteres enthält neben der JRE je nach Version eine mehr oder weniger umfangreiche Sammlung von Werkzeugen zum Entwickeln und Testen von Anwendungen.
- Unter Windows beschränkt sich die Installation auf das Entpacken der `*.zip` Datei in das gewünschte Verzeichnis. Beim ersten Start durch einen Doppelklick auf `eclipse.exe` wird die Installation vervollständigt und das Framework ist einsatzbereit.
- Unter Linux ist die Installation ebenfalls einfach: Entweder Sie wählen die bei Ihrer Distribution mitgelieferten Pakete und installieren diese, oder Sie laden die entsprechende Datei für Ihre Rechnerarchitektur herunter, entpacken diese, wechseln in das Verzeichnis, in das Sie Eclipse entpackt haben, und starten dann Eclipse mit `./eclipse`.
- Unter Mac OS X wird die Installation analog durchgeführt. Das Archiv wird entpackt. Dadurch wird ein Verzeichnis erstellt, das die ausführbare Datei enthält.

18 `http://www.eclipse.org`
19 `http://www.Eclipse.org`

1.6.2 JBuilder

JBuilder[20] ist eine IDE, die von Borland entwickelt wird. Sie ist nicht als Open Source verfügbar. Für die Foundation Version von JBuilder gibt es eine kostenlose Nutzungslizenz. Für die Professional- bzw. für die Enterprise-Version gibt es eine kostenpflichtige Nutzungslizenz.

- Der JBuilder hat ein entsprechendes SDK bereits mit dabei. Sie sollten sich jedoch trotzdem noch ein aktuelles SDK besorgen, da hier der JBuilder streikt.
- Aufgrund von Änderungen des Aufbaus der `class`-Dateien sollten Sie auf die JBuilder Version achten:
  - JBuilder bis Version 3: native implementiert, daher sehr schnell und auch für langsamere PCs geeignet.
  - JBuilder bis Version 5: JDK 1.3 und früher
  - JBuilder Version 6 bis Version X: JDK 1.4 und früher
  - JBuilder Version 2005, 2006: JDK 5 und früher, Betriebssysteme jedoch nur noch Win XP, Win 2000 oder höher
  - Für JDK 6 ist bereits eine neue Version des JBuilders in Entwicklung
- Unter Windows reicht ein Doppelklick auf die Installationsdatei, um den JBuilder in Windows üblicher Manier zu installieren. Es gibt auch die Möglichkeit einer Silent Installation.

1.6.3 NetBeans

NetBeans[21] ist eine IDE, die ursprünglich von der Firma NetBeans als Open Source entwickelt wurde. NetBeans wurde später von Sun übernommen und blieb als Open Source erhalten. Für NetBeans gibt es eine kostenlose Nutzungslizenz. Die NetBeans-IDE ist für die Betriebssysteme Linux, Mac OS X, MS-Windows sowie Solaris als *Download* von der Hersteller-Website verfügbar.

Diese wird auch beim Download des SDK mit angeboten, wir empfehlen Ihnen hier jedoch, die beiden Pakete NetBeans und SDK getrennt zu installieren, um diese bei Bedarf einfach deinstallieren zu können.

1.6.4 Java-Editor

Nur für Windwos

Der Java-Editor[22] ist eine sehr einfache und intuitiv bedienbare IDE für **Java** und **UML**. Sie wendet sich an alle User, die nicht erst stundenlang ein Tool installieren und konfigurieren möchten, und hat trotzdem alle Attribute einer ausgewachsenen Softwareentwicklungsumgebung, wie Syntax-Highlighting, Code-Vervollständigung (bei installierter Dokumentation der Pakete) und einen visuellen GUI-Builder. Das Tool erinnert von seiner Benutzung her an Borlands JBuilder oder Delphi (mit dem es auch geschrieben wurde) und lässt sich

20 `http://www.Borland.de`
21 `http://www.netbeans.org`
22 `http://www.javaeditor.org/index.php?title=Java-Editor/de`

ähnlich einfach benutzen. Klassen können modelliert und dann wie in BlueJ interaktiv getestet werden. Da das Tool ursprünglich für Schüler der Sekundarstufe II entwickelt wurde, gibt es eine recht ausführliche Dokumentation[23] in deutscher Sprache.

Hier kommen wir direkt zu unserem ersten Programm. Wir werden ein Programm schreiben, das auf der Eingabeaufforderung (oder Konsole, wenn Sie Linux verwenden sowie Terminal unter Mac) eine Textmeldung ausgibt und sich danach einfach wieder beendet.

Die Eingabeaufforderung - auch als DOS-Fenster bekannt - erreichen Sie unter Windows indem Sie Start → Ausführen anwählen und dort `cmd` eingeben (Windows NT, 2000, XP, 8, 8.1) bzw. `command` unter Windows 95, 98, ME bzw. Win32 (Windows 3.11).
In Windows 8 und 8.1 drücken sie die Start-Taste um das Metro-Menü aufzurufen und geben einfach `cmd` ein und drücken Return bzw. wählen Sie das Programm `cmd` aus den Suchergebnissen aus.

Unter Linux können Sie über das Menü die Konsole auswählen oder mit ALT + Funktionstasten zu einer freien Konsole wechseln.

Unter Mac OSX 10.4 (Tiger) gehen Sie in den Ordner Programme und danach auf den Ordner Dienstprogramme, dort finden Sie das Terminal.app. In Mac OSX (10.5-10.10) finden sie das Programm "Terminal.app" im Finder->Programme->Dienstprogramme.

1.7 Hello Java World!

Jetzt starten Sie bitte Ihren Editor oder Ihre IDE. Dort geben Sie das folgende Programm ein. Achten Sie dabei genau auf die Groß- und Kleinschreibung! Alternativ können Sie das Programm auch einfach aus dieser Webseite kopieren und in den Editor einfügen, so sind Sie ein paar Sekunden schneller und müssen sich nicht um die Groß-/Kleinschreibung kümmern.

Machen Sie sich jetzt noch keine Gedanken darüber, was die einzelnen Befehle zu bedeuten haben, die Erklärung folgt ein wenig weiter unten.

```
public class HelloWorld {
      public static void main(String[] args) {
          System.out.println("Hello Java World");
          System.exit(0);
      }
}
```

Speichern Sie das Programm jetzt in einem Verzeichnis Ihrer Wahl unter dem Namen `HelloWorld.java`. Beachten Sie bitte genau auch hier die Groß- und Kleinschreibung des Dateinamens und ändern Sie den Namen nicht. Es muss unbedingt der Name sein, den Sie in der Programmzeile `public class HelloWorld` verwendet haben, ergänzt um die Dateinamensendung `.java`.

Gehen Sie jetzt in der Eingabeaufforderung in dieses Verzeichnis und übersetzen Sie das Programm mit folgendem Befehl:

```
javac HelloWorld.java
```

23 http://lernarchiv.bildung.hessen.de/sek_ii/informatik/11.2/index.html

Ein möglicher Fehler ist ”error: cannot read: HelloWorld.java”. Schreiben Sie dann einfach den Pfad davor (C:\Pfad\HelloWorld.java). Ein kurzer Pfad ist empfehlenswert. Wenn die Übersetzung fehlerfrei durchgelaufen ist, können Sie das Programm mit der folgenden Eingabe ausführen:

```
java HelloWorld
```

Fallen Ihnen zwei kleine, aber wichtige Unterschiede bei den beiden Befehlen auf? Richtig:

1. Im ersten Fall (zum Compilieren) wird das Programm `javac` (mit einem ’c’ am Ende) verwendet. Im zweiten Fall (zum Ausführen) aber das Programm `java` (ohne ’c’ am Ende).
2. Beim Compilieren mit `javac` mussten Sie die Dateinamensendung `.java` mitangeben. Bei der Programmausführung mit `java` dürfen Sie diese Endung **nicht** mitangeben.

Falls eine Fehlermeldung wie

```
Exception in thread "main" java.lang.NoClassDefFoundError: HelloWorld
```

auftritt, müssen Sie nachschauen, ob der aktuelle Pfad auch im CLASSPATH enthalten ist. Wie das gemacht wird, ist weiter oben beschrieben. Ebenso sollten Sie genau prüfen, ob Sie der Java-Datei wie oben beschrieben den richtigen Namen gegeben haben, und ob Sie versehentlich die Endung des Dateinamens beim Aufruf von `java` angegeben haben.

Wenn das Programm fehlerfrei durchgelaufen ist, dann müsste auf der Konsole der folgende Text erscheinen:

```
Hello Java World
```

1.8 Hello Java!

An dieser Stelle haben Sie bereits Ihr erstes Java-Programm geschrieben und wollen nun wissen, was denn nun diese einzelnen Befehle und kryptischen Zeichen bedeuten. Wir gehen hier davon aus, dass Sie sich bereits mit der Objektorientierung beschäftigt haben und somit wissen was Klassen, Objekte (Instanzen), Methoden (Operationen) und Eigenschaften (Attribute) sind. Falls Sie dies noch nicht wissen sollten, folgen Sie doch einfach dem oberen Link, wo Sie sicher fündig werden.

Hier ist nochmals das Listing unseres Hello-World-Programms:

```
public class HelloWorld {
    public static void main(String[] args) {
        System.out.println("Hello Java World");
        System.exit(0);
    }
}
```

Wir haben die Zeilen durchnummeriert, um so die Bestandteile des Programms besser erklären zu können.

In der ersten Zeile definieren wir die Klasse "HelloWorld". Dabei ist zu beachten, dass die Klasse wie der Dateiname heißen muss, also in unserem Beispiel muss die Datei "HelloWorld.java" heißen. Jeder kann auf unsere Klasse zugreifen, deshalb ist sie auch "public". Man kann die Rechte von Klassen auch einschränken, aber dazu kommen wir später.

In Zeile zwei wird die Main-Methode definiert. Diese finden Sie bei jeder Java-Anwendung (Applikationen), nicht jedoch bei Applets, Servlets oder Midlets. Diese ist sozusagen das Herz jeder Java-Anwendung. Von dieser Methode ausgehend können Sie sich die komplette Funktionsweise eines Programms erschließen.

In der Zeile 3 geben wir unsere Meldung, bei uns "Hello Java World", aus, was nicht allzu viel zu sagen hat, aber der Zweck unseres Programms ist. Nebenbei bemerkt ist es sozusagen die "höhere Weihe" eines Programmierers in der Programmiersprache seiner Wahl ein "Hello World" auszugeben. Sie finden in fast jedem Tutorial zu irgendeiner Programmiersprache ein solches "Hello-World"-Programm.

Die vierte Zeile heißt, dass wir die Anwendung beenden wollen und zwar mit einem Rückgabewert von 0. Dies beendet auch gleichzeitig die Java Virtual Machine (JVM), sodass wir wieder auf der Eingabeaufforderung landen. Auch hierbei gilt, dass nur Java-Anwendungen mit diesem Befehl beendet werden. In der Eingabeaufforderung kann man diesen Wert auch auswerten, so dass man z.B. ein Java-Programm schreiben kann, das durch verschiedene Rückgabewerte den Ablauf eines Batch-Programmes oder Shellskripts steuern kann.

1.9 Pakete

Java-Klassen und auch Schnittstellen (Interface) können und werden in so genannte Pakete (Packages) gruppiert. Ein Paket ist dabei einfach eine Sammlung von Klassen. Pakete werden in den geläufigen Betriebssystemen in Form von Verzeichnissen abgebildet. Klassen und Schnittstellen, welche in einem Paket liegen, beginnen dabei mit der *package* Anweisung, gefolgt von den Paketnamen und dem Semikolon. Zum Beispiel:

```
package org.wikibooks.de.java;
```

Üblicherweise gruppiert man thematisch verwandte Klassen in einem Paket.

Oft ist der Domainname des Autors Teil des Paketnamens, so dass ein Programmierer schnell die Herkunft des Pakets ermitteln und auf der zugehörigen Webseite weitere Informationen zu dem Paket finden kann. Hierbei wird zudem eine Eindeutigkeit der Pakete und somit der Java Klassen / Schnittstellen erreicht. Beispiel: org.apache.log4j → www.apache.org

1.10 Klassenbibliothek

Java enthält eine sehr umfangreiche Klassenbibliothek. Es ist dringend zu empfehlen, bei der Installation des SDK auch die zugehörige Klassenbibliotheksdokumentation (*API Documentation*) herunterzuladen und lokal zu installieren, bzw. in die IDE zu integrieren.

Die beim SDK Standard Edition mitgelieferte Klassenbibliothek ist in Pakete (*packages*) eingeteilt. Es lohnt sich, sich im Laufe der Zeit zumindest einen Überblick über die vorhan-

denen Pakete und deren grundsätzliche Bedeutung zu verschaffen. Einige Pakete werden bei jeder Art von Java-Programmierung so häufig gebraucht, dass deren Inhalt früher oder später in Fleisch und Blut übergehen sollte.

Dabei ist es ist nicht unbedingt notwendig, jeden einzelnen Methodenparameter auswendig zu lernen (Ausnahme: Einige gerade im US-amerikanischen Raum beliebte Programmierer-Zertifizierungen fragen solches "Wissen" in ihren Zertifizierungstests ab). Methodenparameter kann man immer schnell in der API-Dokumentation nachschlagen, oder sie werden von modernen IDEs sogar direkt angezeigt. Eine grundsätzliche Vorstellung davon, was wo und wofür in der Klassenbibliothek zu finden ist, sollte man allerdings schon haben, um zügig arbeiten zu können und halbwegs effiziente Programme zu schreiben.

Zu Anfang lohnt es sich, einen Blick auf die Dokumentation der Klassen in den folgenden Paketen zu werfen:

Paket	**Inhalte**
java.lang	Basis-Klassen wie `Object`. Das Rückgrat der Sprache (*lang* = *Language* = Sprache)
java.io	Einfache Ein- und Ausgabeklassen für Text- und Binärdaten, Dateizugriffe und Kodierung und Enkodierung von Daten.
java.util	Sehr nützliche Hilfsklassen. Insbesondere finden sich hier die sog. *Collection classes*. Dies ist eine Sammlung von Klassen, die gängige Datenstrukturen zur Verknüpfung von Objekten bereitstellen. Dank dieser Klassen ist es für einen Java-Programmierer in den allermeisten Fällen unnötig z.B. selber Listen, Mengen oder eine Hashtable zu implementieren.
Je nachdem, welche Art von Programmen man entwickeln möchte, sollte ein Blick in folgende Pakete folgen:	
java.net	Basisklassen für Netzwerkkommunikation.
java.awt javax.swing	Klassen zur Programmierung grafischer Benutzeroberflächen

Java ist eine einfach aufgebaute, leicht zu erlernende und robuste rein objektorientierte Programmiersprache. Seit der Verbreitung der javafähigen Webbrowser Mitte der 1990er Jahre ist sie neben C++ zur einer der wichtigsten imperativen Programmiersprachen aufgestiegen und hat inzwischen nicht nur auf dem Desktop (**J2SE**), sondern auch auf mobilen Endgeräten (**J2ME**) und Unternehmensanwendungen (**J2EE**) einen festen Platz gefunden.

w:Java Virtual Machine[24] Bei Programmiersprachen wie C oder C++ wird beim Kompilieren Maschinencode erzeugt. Dieser kann dann direkt auf dem Computer ausgeführt werden.

Im Gegensatz dazu verfolgte SUN bei der Einführung der Programmiersprache das Konzept "Write once, run everywhere!". Das heißt, man wollte eine Programmiersprache entwickeln, mit der man einmal ein Programm schreibt, welches dann auf fast allen Betriebssystemen läuft. Dazu führte man das Konzept der Java Virtual Machine (JVM) ein. Die Java Virtual Machine ist eine Abstraktionsschicht zwischen dem Javaprogramm und dem eigentlichen Betriebssystem. Bei der Kompilierung mit **javac** wird kein nativer Maschinencode erzeugt,

24 `https://de.wikipedia.org/wiki/Java%20Virtual%20Machine`

sondern ein Bytecode, der dann von der JVM intepretiert werden kann. Deshalb können kompilierte Javaprogramme auch nicht direkt aufgerufen werden, sondern müssen immer über die virtuelle Maschine in Form von "java test" oder "java -jar test.jar" aufgerufen werden. Man kann sich die virtuelle Maschine als Blackbox um das Java Programm vorstellen, die die Interaktion mit dem Betriebssystem übernimmt.

Java Quelltext -- javac → Java Bytecode -- java → Betriebssystem

In Java werden standardmäßig verschiedene Anwendungsarten unterschieden.

1.11 Applets

Sind mehr oder weniger komplexe Anwendungen, die speziell für das Laufen in einem Webbrowser entwickelt werden. Sie benötigen zur Ausführung eine HTML-Datei mit einem Applet-Tag in dem sie ähnlich wie ein Bild in einer HTML-Seite eingebettet werden. Außerdem haben nichtsignierte Applets nur sehr beschränkte Zugriffsrechte auf den Host-Rechner. Damit unterliegen Sie besonderen Bedingungen.

UNKNOWN TEMPLATE lst:Java Standard: Applets

aufbau_von_applets

Durch die Entwicklung bei JavaScript und HTML5 sind Applets aus der Mode gekommen und werden in diesem Buch nicht länger behandelt.

1.12 Applikationen

Applikationen sind die "normalen" Anwendungen auf Ihrem Computer und haben volle Zugriffsmöglichkeiten auf alle Bestandteile des Rechners. Sie werden mit Hilfe des **javac** Compilers übersetzt und mit Hilfe des Java-Interpreters **java** bzw. **javaw** ausgeführt. Es gibt hierbei noch einige Besonderheiten in Zusammenhang mit Java Webstart.

1.13 Servlets

Servlets sind die Java-Variante der sog. CGI-Skripten und gehören in die Server--Welt. Hierbei wird Java-Code auf dem Server ausgeführt, der dort z.B. Dateien engegennimmt oder in Datenbanken schreibt.

Diese Anwendungsart gehört zur Java Enterprise[25]-Version (J2EE).

25 `https://de.wikibooks.org/wiki/Java%20Enterprise`

1.14 Midlets

Midlets sind die kleinen Anwendungen, welche typischerweise auf Handys etc. laufen. Diese werden mit Hilfe der Java Micro[26]-Version (J2ME) entwickelt.

26 https://de.wikibooks.org/wiki/Java%20Micro

2 Grundlagen

2.1 Worum geht es?

Variablen sind Speicherorte für Daten wie Zahlen, Texte oder Adressen. Java ist eine statisch typisierte Programmiersprache. Das bedeutet, dass zu dem Zeitpunkt, wo das Java-Programm erstellt wird bekannt sein muss, welchen Typ eine Variable hat. Typen können so genannte primitive Datentypen sein oder beliebig komplexe Klassen.

2.2 Was sind Primitive Datentypen?

Primitive Datentypen sind eine Gruppe von Typen, mit denen man Variablen erstellen kann, die Zahlen, einzelne Zeichen oder logische Werte aufnehmen. Für jeden primitiven Datentyp gibt es zudem eine eigene, so genannte *Wrapperklasse*, die diesen Datentyp aufnimmt. Nachdem wir die einzelnen primitiven Datentypen besprochen haben, besprechen wir die zugehörigen Wrapperklassen und zeigen Vor- und Nachteile auf.

Die primitiven Datentypen, ihr typischer Wertebereich und die zugehörige Wrapperklasse haben wir in der folgenden Tabelle für Sie aufgeführt.

Typname	**Größe**[1]	**Wrapper-Klasse**	**Wertebereich**	**Beschreibung**
boolean	undefiniert[2]	java.lang.Boolean	true / false	Boolescher Wahrheitswert, Boolescher Typ[3]
char	16 bit	java.lang.Character	0 ... 65.535 (z. B. 'A')	Unicode-Zeichen (UTF-16)
byte	8 bit	java.lang.Byte	-128 ... 127	Zweierkomplement-Wert
short	16 bit	java.lang.Short	-32.768 ... 32.767	Zweierkomplement-Wert
int	32 bit	java.lang.Integer	-2.147.483.648 ... 2.147.483.647	Zweierkomplement-Wert

2 `https://docs.oracle.com/javase/tutorial/java/nutsandbolts/datatypes.html`

3 Wir gebrauchen zur zeit in diesem Buch die Begriffe "Boolescher Typ" und "Wahrheitswert" synonym, wünschen uns aber eine Vereinheitlichung in Richtung "Boolescher Typ" oder "Boolesche Variable, wenn Variablen dieses Typs gemeint sind.

Typname	Größe[1]	Wrapper-Klasse	Wertebereich	Beschreibung
long	64 bit	java.lang.Long	-2^{63} bis 2^{63}-1, ab Java 8 auch 0 bis 2^{64} -1[4]	Zweierkomplement-Wert
float	32 bit	java.lang.Float	+/-1,4E-45 ... +/-3,4E+38	32-bit IEEE 754, es wird empfohlen, diesen Wert nicht für Programme zu verwenden, die sehr genau rechnen müssen.
double	64 bit	java.lang.Double	+/-4,9E-324 ... +/-1,7E+308	64-bit IEEE 754, doppelte Genauigkeit

Wir zeigen ihnen nun den Umgang mit den einzelnen Typen jeweils an einem Beispiel.

2.3 Zeichen

Der Datentyp `char` kann jedes beliebige Unicode-Zeichen aufnehmen, insbesondere europäische und asiatische Zeichen sind damit abgedeckt. Hierfür benötigt er 2 Bytes an Speicherplatz pro Zeichen. Zeichen werden in Hochkommata eingeschlossen.

```
char ersterBuchstabe = 'A';
System.out.println("Der erste Buchstabe des Alphabets ist ein");
System.out.println(ersterBuchstabe);
```

Es spielt für Java keine Rolle, ob Sie eine `char`-Variablen ein Zeichen oder eine Zahl zuweisen. Ferner können Sie mit Zeichen auch rechnen:

```
public class ZeichenTest {
      public static void main(String[] args) {
          char buchstabe1 = 'A';
          char buchstabe2 = 65;
          buchstabe1 += 1;
          System.out.println(buchstabe1);
          System.out.println(buchstabe2);
      }
}
```

Dieses Programm erzeugt die Ausgabe `"B A"`. Addiert man zu einem Zeichen eine 1, dann ist damit das nächste Zeichen gemeint. Ebenso können Sie Zeichen auch numerisch eingeben, wie `buchstabe2 = 65` zeigt. 65 ist der Zahlenwert für das Zeichen 'A'.

4 https://docs.oracle.com/javase/tutorial/java/nutsandbolts/datatypes.html

2.4 Ganze Zahlen

2.4.1 byte

Ein `byte` ist der kleinste numerische Datentyp in Java, er ist 8 bit lang. Dieser Datentyp tritt zumeist im Zusammenhang mit Feldern auf, auf die wir in einem späteren Kapitel zu sprechen kommen.

```
byte einByte = 10;
byte anderesByte = -10;
System.out.println(einByte + " " + anderesByte);
```

Obenstehendes Beispiel zeigt, wie man Variablen vom Typ `byte` deklariert und initialisiert.

2.4.2 short

Der Datentyp `short` ist 2 Bytes groß und vorzeichenbehaftet. Oft werden Konstanten mit diesem Datentyp angelegt, tatsächlich aber wird `short` nur selten wirklich gebraucht.

2.4.3 int

Der Datentyp `int` ist wohl der am häufigsten eingesetzte primitive Typ. Er belegt 4 Bytes, was in der Regel für viele Anwendungsbereiche ausreicht.

```
int a = 1;
int b = 1;
int c = a + b;
System.out.println("a mit dem Wert " + a + " plus B mit dem Wert " + b + "
 ergibt: C " + c);
```

2.4.4 long

der Datentyp `long` ist die erweiterte Form des `int`-Typs. Im Gegensatz zum `int` hat dieser Datentyp 8 Bytes. Wenn man bei der Darstellung von Zahlenliteralen Wert darauf legen möchte, dass sie zum Datentyp `long` gehören, dann fügt man den Suffix ”L” oder ”l” (kleines L) an. Das muss man aber nicht, der Compiler wandelt Literale entsprechen um.

Ein Beispiel:

```
long a = 10L;
long b = 20;
long c = a + b;
System.out.println(a + " + " + b + " = " + c);
```

Die Ausgabe ist `10 + 20 = 30`.

2.5 Gleitkommatypen

Mit den Datentypen `float` und `double` können Sie eine Teilmenge der rationalen Zahlen darstellen. Sprachlich sind Gleitkomma- und Fließkommazahlen gebräuchlich. Beide Typen unterscheiden sich durch ihre Genauigkeit und den Speicherverbrauch. Gemeinsam ist, dass solche Zahlen nur *ungefähr richtig* gespeichert werden.

2.5.1 float

Eine Variable vom Typ `float` speichert eine Gleitkommazahl mit einer Größe von 4 Byte. Die Genauigkeit liegt bei 7 signifikanten Stellen, man spricht hier von *einfacher Genauigkeit* (engl: *single precision*). Zahlenliteralen stellt man ein "f" oder "F" nach, sonst nimmt der Java-Compiler an, es handele sich um `double`-Literale.

Ein Beispiel:

```
float a = 1.00f;
float b = 3.00f;
float c = a/b;
float d = c * b; // sollte a ergeben
System.out.println(a + "/" + b + " = " + c);
System.out.println(c + "*" + b + " = " + d + " ( " + a + ")");
```

Ausgabe:

```
1.0/3.0 = 0.33333334
0.33333334*3.0 = 1.0 ( 1.0)
```

Dieses Beispiel zeigt gleichzeitig ein Problem mit Gleitkommazahlen, da hier (1/3) in der ersten Zeile falsch dargestellt wird, die Gesamtrechnung ist aber offenbar richtig.

Verwenden Sie den Datentyp `float` nur, wenn es ihnen nicht auf Präzision ankommt.

2.5.2 double

Der Typ `double` bietet Gleitkommavariablen mit einer Größe von 8 Byte. Die Genauigkeit liegt bei 15 signifikanten Stellen, dies bezeichnet man als *doppelte Genauigkeit* (engl. *double precision*).

```
double a = 1.00;
double b = 3.00;
double c = a/b;
double d = c * b; // sollte a ergeben
System.out.println(a + "/" + b + " = " + c);
System.out.println(c + "*" + b + " = " + d + " ( " + a + ")");
```

Ausgabe:

```
1.0/3.0 = 0.3333333333333333


0.3333333333333333*3.0 = 1.0 ( 1.0)
```

2.6 Boolescher Typ

Eine boolesche Variable `boolean` kann einen von zwei Zuständen annehmen: `true` oder `false`. Dies repräsentiert in Ausdrücken zumeist *eine Bedingung ist erfüllt* – oder eben nicht.

Boolsche Variablen werden zumeist im Zusammenhang mit Verzweigungen gebraucht, auf die wir im Kapitel über Kontrollstrukturen zu sprechen kommen.

```
boolean istWahr = true;

if (istWahr) {
        // Mach irgendwas
}
```

2.7 Casting in Java

Casting nennt man die *Überführung eines Datentypen in einen Anderen.* Es gibt implizites und explizites Casting. Ist die Wertemenge eines numerische Datentyps Teilmenge eines anderen Datentyps, dann lässt sich der erste Datentyp in den Zweiten überführen. Hierfür braucht man nichts weiter zu tun, man spricht vom impliziten Casting. Ist aber der Wertebereich eines Datentyps eine echte Obermenge eines anderen, dann kann es zu Datenverlusten führen, den ersten Datentyp in den Zweiten zu überführen. Beispielsweise kann man `byte` nach `int` überführen, umgekehrt aber nur explizit, das heißt, man schreibt explizit hin, dass man diese Umwandlung wünscht und akzeptiert, dass es zu Datenverlust kommen kann.

```
int  ii = 42;
byte bb = 100;
// implizites Casting ("automatisches Casting") weil int den größeren
 Wertebereich hat
ii = bb;
// explizites Casting weil der Wertebereich von int eine echte Obermenge von
 byte ist
bb = (byte) ii;
```

1. WEITERLEITUNG Java Standard: Arrays[5]

2.8 Bezeichner

Bezeichner (engl.: Identifier) bezeichnen in Java Klassen, Interfaces, Variablen (bzw. Objekte), Methoden und Konstanten. Bezeichner sind also, einfach gesprochen, die Namen der zuvor genannten Elementen.

Mit Hilfe von Bezeichnern kann man die diesen Namen zugeordneten Speicherbereiche ansprechen und nutzen.

5 https://de.wikibooks.org/wiki/Java%20Standard%3A%20Arrays

2.8.1 Gültige Namen

Legale Identifier können in Java aus folgenden Zeichen bestehen:

- Unicode-Zeichen
- Ziffern
- Währungssymbole
- Verbindungszeichen (Bsp.: Unterstrich)

Gültige Identifier:

- starten mit einem Zeichen, Währungssymbol oder Verbindungszeichen
- starten **nicht** mit einer Ziffer
- können nach dem ersten Zeichen alle Kombinationen von Zeichen, Ziffern, Währungssymbolen und Verbindungszeichen enthalten
- theoretisch ist die Anzahl der Zeichen pro Identifier unbegrenzt
- sind keine Java-Schlüsselwörter wie **new**
- Identifier sind case-sensitive, sprich Java unterscheidet zwischen Groß- und Kleinschreibung.

2.8.2 Konventionen

Ähnlich wie in der "realen Welt" sollte man sich bei der Programmierung ebenfalls um einen guten Ton bemühen.

```
esisteinunterschiedobichalleskleinundzusammenschreibe ODER

_leserlich_formatiere_sodass_mein_Code_besser_nachvollzogen_werden_kann
```

Grundlegende Regel hierbei ist die Verständlichkeit und leichte Nachvollziehbarkeit des verfassten Codes (und die Kommentare nicht vergessen). Aus diesem Grund wurden die Java-Code-Konventionen geschaffen:

Klassen und Interfaces

- Erster Buchstabe immer groß
- "UpperCamelCase" anwenden. Wenn mehrere Worte in einem Identifier verbunden werden, sollten die Anfangsbuchstaben groß geschrieben werden.
- Für Klassen Nomen verwenden (Hund, Katze...)
- Für Interfaces Adjektive verwenden (Essbar...)

Methoden

- erster Buchstabe immer klein.
- "lowerCamelCase" anwenden.
- Methodennamen sollten Kombinationen aus Verben und Nomen darstellen. So kann am besten beschrieben werden, wozu die Methode dient. Gute Beispiele findet man natürlich in der Java API.
- Bsp.: getTasche(), fuelleTasche(...)

Variablen

- wie bei Methoden "lowerCamelCase" und erster Buchstabe immer klein.

Konstanten

- Konstanten werden erzeugt, indem man Variablen static und final deklariert.
- Sie werden nur in Großbuchstaben geschrieben und einzelne Worte werden durch Unterstrich voneinander getrennt.
- Bsp.: MAX_MENGE

2.9 Variable

2.9.1 Deklaration

Variablen in Java sind immer lokale Variablen. Bevor man eine Variable benutzen kann, muss sie erst deklariert werden. Dies kann an beliebiger Stelle innerhalb einer Methode, in der statischen Initalisierung einer Klasse oder im Initialisierungsteil einer for-Schleife geschehen. Nur in diesem Bereich ist die Variable sichtbar. Der Name der Variablen muss innerhalb dieses Sichtbarkeitsbereichs eindeutig sein.

Da Java eine streng typisierte Sprache ist, muss bei Deklaration einer Variablen immer ein Datentyp mit angegeben werden. Dies kann ein primitiver Datentyp sein (boolean, char, byte, short, int, long, float, double) oder ein Referenztyp (eine Klasse oder Interface).

Optional kann eine Variable bei Deklaration durch einen Ausdruck zur Laufzeit mit einem Wert initialisiert werden.

Mit dem ebenfalls optionalen Schlüsselwort final wird verhindert, dass der Wert einer Variablen nach dessen Deklaration geändert werden kann; lediglich eine erstmalige Zuweisung im Konstruktor einer Klasse für Instanzvariablen ist möglich.

Die Syntax einer Variablendeklaration:

```
[ final ] [private|protected|public] Datentyp VARIABLENNAME [ = Ausdruck ] ;
```

zum Beispiel:

```
public class Beispiel {
  static {
    int i = 0;
    // etwas Sinnvolles mit i anstellen
  }
  public void machWas() {
    long grosseGanzeZahl;
    String name;
    final float PI = 3.1415;
    boolean [] wahrheitswerte = new boolean[10];
    // etwas Sinnvolles mit grosseGanzeZahl, name und PI anstellen
```

```
    // Die Zuweisung PI = 1; verursacht einen Compilerfehler
    // Ein Zugriff auf i verursacht einen Compilerfehler
    for (int i=0; i < wahrheitswerte.length; i++) {
      // hier existiert i (ein anderes als obiges im static Block)
    }
  }
}
```

2.9.2 Sichtbarkeits- und Zugriffsmodifizierer

Die drei Schlüsselwörter `private`, `protected` und `public` dienen der Modifikation der Sichtbarkeit und des Zugriffs.

- `private` erlaubt nur der eigenen Klasse den Zugriff.
- `protected` erlaubt der eigenen Klasse, der Paketklasse und der Elternklasse den Zugriff.
- `public` erlaubt jeder Klasse, auch fremden Klassen, den Zugriff.

Ist eine Methode als `private static final` gekennzeichnet, ist der Zugriff am schnellsten.

Um Fehlern vorzubeugen, sollte der Zugriff so stark wie möglich eingeschränkt werden.

2.9.3 Initialisierung

Vor der Verwendung der Variablen muss ihnen ein Wert zugewiesen werden, der dem Datentyp entspricht. So kann man einem Integerwert(int) nicht den String ”Hallo” zuweisen (zumindest nicht ohne vorherige Umwandlung)

```
int i;
i = 12; //i muss vorher deklariert werden
```

oder:

```
int i = 12;
```

2.10 Welche Operatoren gibt es in Java?

Java kennt eine eine Vielzahl von arithmetischen, logischen, und relationalen Operatoren, sowie einen, der außerhalb von Java keine Rolle spielt. Operatoren werden nach der Anzahl der möglichen Operanden unterteilt (unärer-, binärer- und ternärer Operator) und selbstverständlich nach der Funktion, die sie berechnen. Dieses Kapitel beschreibt die verfügbaren Operatoren in Tabellenform. Bis auf wenige Ausnahmen sollten alle Operatoren und das, was sie leisten, aus der Schule bekannt sein.

2.11 Arithmetische Operatoren

Operator	**Beschreibung**	**Kurzbeispiel**

Operator	**Beschreibung**	**Kurzbeispiel**
+	Addition	`int antwort = 40 + 2;`
-	Subtraktion	`int antwort = 48 - 6;`
*	Multiplikation	`int antwort = 2 * 21;`
/	Division,	`int antwort = 84 / 2;`
%	Teilerrest, Modulo-Operation, errechnet den Rest einer Division	`int antwort = 99 % 57;`
+	positives Vorzeichen, in der Regel überflüssig	`int j = +3;`
-	negatives Vorzeichen	`int minusJ = -j;`

Für zwei besonders in Schleifen häufig anzutreffende Berechnungen gibt es eine abkürzende Schreibweise.

Operator	**Beschreibung**	**Kurzbeispiel**
++	Postinkrement, Addiert 1 zu einer numerischen Variable	`x++;`
++	Preinkrement, Addiert 1 zu einer numerischen Variable	`++x;`

Operator	Beschreibung	Kurzbeispiel
--	Postdekrement, Subtrahiert 1 von einer numerischen Variable	`x--;`
--	Predekrement, Subtrahiert 1 von einer numerischen Variable	`--x;`

Post- und Pre-Operatoren verhalten sich bezüglich ihrer Berechnung absolut gleich, der Unterschied ist der Zeitpunkt, wann die Operation ausgeführt wird. Zum Tragen kommt das bei Zuweisungen:

```
i = 1;
a = ++i; // i = 2 und a = 2 (erst hochzählen, dann zuweisen)

i = 1;
b = i++; // i = 2 und b = 1 (erst zuweisen, dann hochzählen)
```

2.12 Operatoren für Vergleiche

Das Ergebnis dieser Operationen ist aus der Menge `true`, `false`:

Operator	Beschreibung	Kurzbeispiel
==	gleich	`3 == 3`
!=	ungleich	`4 != 3`
>	größer als	`4 > 3`
<	kleiner als	`-4 < -3`
>=	größer als oder gleich	`3 >= 3`

Operator	Beschreibung	Kurzbeispiel
<=	kleiner als oder gleich	`-4 <= 4`

2.13 Boolesche Operatoren

Operator	Beschreibung	Kurzbeispiel
!	Negation, invertiert den Ausdruck	`boolean lügnerSpricht = !wahrheit;`
&&	Und, `true`, genau dann wenn beide Argumente `true` sind	`boolean krümelmonster = istBlau && magKekse;`
||	Oder, `true`, wenn *mindestens* ein Operand `true` ist	`boolean machePause = hungrig || durstig;`
^	Exor, `true` wenn genau ein Operand `true` ist	`boolean zustandPhilosoph = denkt ^ isst;`

2.14 Operatoren zur Manipulation von Bits

Operator	Beschreibung	Kurzbeispiel
~	(unäre) invertiert alle Bits seines Operanden	`0b10111011 = ~0b01000100`
&	bitweises "und", wenn beide Operanden 1 sind, wird ebenfalls eine 1 produziert, ansonsten eine 0	`0b10111011 = 0b10111111 & 0b11111011`
|	bitweises "oder", produziert eine 1, sobald einer seiner Operanden eine 1 ist	0b00111011</syntaxhighlight>

Operator	Beschreibung	Kurzbeispiel
^	bitweises ”exklusives oder”, wenn beide Operanden den gleichen Wert haben, wird eine 0 produziert, ansonsten eine 1	`0b10111011 = 0b10001100 ^ 0b00110111`

Operator	Beschreibung	Kurzbeispiel
>>	Rechtsverschiebung, alle Bits des Operanden werden um eine Stelle nach rechts verschoben	(fehlt)
>>>	Rechtsverschiebung mit Auffüllung von Nullen	(fehlt)
<<	Linksverschiebung, entspricht bei positiven ganzen Zahlen einer Multiplikation mit 2, sofern keine ”1” rausgeschoben wird.	(fehlt)

2.15 Zuweisungsoperatoren

Zu vielen Operatoren aus den vorstehenden Tabellen gehören Schreibweisen, mit denen gleichzeitig zugewiesen werden kann. Damit spart man sich oft etwas Schreibarbeit. Also, statt etwa

```
x = x * 7;
```

zu schreiben kann man etwas verkürzt schreiben:

```
x *= 7;
```

.

Operator	Beschreibung	Kurzbeispiel
=	einfache Zuweisung	`int var = 7;`
+=	Addiert einen Wert zu der angegebenen Variablen	`plusZwei += 2;`
-=	Subtrahiert einen Wert von der angegebenen Variablen	`minusZwei -= 2;`
/=	Dividiert die Variable durch den angegebenen Wert und weist ihn zu	`viertel /= 4;`

Operator	Beschreibung	Kurzbeispiel
*=	Multipliziert die Variable mit dem angegebenen Wert und weist ihn zu	`vierfach *= 4;`
%=	Ermittelt den Modulo einer Variablen und weißt ihn der Variablen zu	`restModulo11 %= 11;`
&=	"und"-Zuweisung	`maskiert &= bitmaske;`
|=	"oder"-Zuweisung	
^=	"exklusives oder"-Zuweisung	
^=	bitweise "exklusive oder"-Zuweisung	
>>=	Rechtsverschiebungzuweisung	
>>>=	Rechtsverschiebungzuweisung mit Auffüllung von Nullen	
<<=	Linksverschiebungzuweisung	`achtfach <<= 3;`

2.16 Bedingungsoperator

Den einzigen ternären Operator `?:` stellen wir im Kapitel Kontrollstrukturen[6] vor.

2.17 Konkatenation

Zwei `String`s lassen sich mit "+" aneinanderschreiben, so wie Sie es schon aus früheren

```
System.out.println("Hallo" + " Welt" + "!");
```

-Beispielen kennen.

2.18 Vergleichsoperator instanceof

- `instanceof` überprüft ob ein Objekt eine Instanz einer bestimmten Klasse ist. Ein Beispiel hierzu stellen wir ihnen später vor.

6 Kapitel 2.29 auf Seite 44

2.19 Rangfolge von Operatoren

Die Rangfolge der Operatoren (engl. "*operator precedence*" oder auch "*precedence rules*") bestimmt in der Regel[7], in welcher Reihenfolge sie ausgewertet werden. Es geht darum, Klammern zu sparen. Weiß man, dass `&&` einen höheren Rang als `||` hat, dann wird der Ausdruck `(A && B) || C` zu `A && B || C`. Selbstverständlich *darf* man trotzdem Klammern setzen.

Ganz allgemein gilt, dass Ausdrücke von links nach rechts ausgewertet werden. Das gilt nicht für Zuweisungsoperatoren.

In der folgenden Tabelle[8] werden die Operatoren und ihre Ränge aufgeführt. Je weiter oben ein Operator in der Tabelle auftaucht, desto eher wird er ausgewertet. Operatoren mit dem gleichen Rang (in der gleichen Zeile) werden von links nach rechts ausgewertet.

Rangfolge	**Typ**	**Operatoren**
1	Postfix-Operatoren, Postinkrement, Postdekrement	`x++`, `x--`
2	Einstellige (unäre) Operatoren, Vorzeichen	`++x`, `--x`, `+x`, `-x`, `~b`, `!b`
3	Multiplikation, Teilerrest	`a*b`, `a/b`, `a % b`
4	Addition, Subtraktion	`a + b`, `a - b`
5	Bitverschiebung	`d << k`, `d >> k`, `d >>> k`
6	Vergleiche	`a < b`, `a > b`, `a <= b`, `a >= b`, `s instanceof S`
7	Gleich, Ungleich	`a == b`, `a != b`
8	UND (Bits)	`b & c`
9	Exor (Bits)	`b ^ c`
10	ODER (Bits)	`b | c`
11	Logisch UND	`B && C`
12	Logisch ODER	`B || C`
13	Bedingungsoperator	`a ? b : c`
14	Zuweisungen	`a = b`, `a += 3`, `a -= 3`, `a *= 3`, `a /= 3`, `a %= 3`, `b &= c`, `b ^= c`, `b |= c`, `d <<=k`, `d >>= k`, `d >>>= k`

2.20 Fallen

Es gibt zum Glück wenige Fallstricke im Gebrauch von Operatoren. Postfix-Operatoren werden immer zuerst nach dem *aktuellen* Wert, den sie haben ausgewertet, erst dann erfolgt die Operation darauf.

7 siehe nächsten Abschnitt Fallen ^{Kapitel2.20 auf Seite 34}

8 Zum Teil entnommen von `https://docs.oracle.com/javase/tutorial/java/nutsandbolts/operators.html`

Nicht in allen Fällen kann man sich bei Beachtung der Rangfolge Klammern sparen. Versuchen Sie doch einmal, den Ausdruck

```
int x = ++y++;
```

auszuwerten (wobei `y` vorher deklariert wurde). Trotz der klaren Vorrangregeln lässt sich dieser Ausdruck nicht kompilieren. Gut für alle, die einen solchen Quelltext lesen müssen...

2.21 Rechnengenauigkeit

Beim Rechnen mit zwei Datentypen unterschiedlicher Genauigkeit (z.B. `int` und `long`) muss beachtet werden, dass als Ergebnis der "genauere" Datentyp berechnet wird.

```
long = int + long;
```

Wird nur ein "ungenauerer" Wert benötigt, so muss das dem Compiler mitgeteilt werden (*cast*).

```
int = (int) (int + long);
```

Die Genauigkeit ist hier durch den Zahlenbereich gegeben. Ein Datentyp **int** kann eine 32-Bit-Ganzzahl darstellen, während der Typ **long** 64-Bit-Ganzzahlen aufnehmen kann (= 8 Byte). Deshalb "passt" ein int-Wert problemlos in einen long-Wert. Umgekehrt kann es da schon passieren, dass der long-Wert größer als der größtmögliche int-Wert ausfällt!

2.22 Verzweigung (`if`)

Eine einfache Verzweigung wird durch das "if-then-else"-Konstrukt gelöst, wobei lediglich `if` und `else` in Java Schlüsselwörter[9] sind.

```
if (<boolescher Ausdruck>) {
  // Anweisung(en) 1
} else {
  // Anweisung(en) 2
}
```

Eine `if`-Verzweigung führt stets die nachfolgende Anweisung aus, wenn der *<boolesche Ausdruck>* wahr ist. Sollen mehrere Anweisungen ausgeführt werden so sind diese in einen Block zusammenzufassen (mit Hilfe der geschweiften Klammern). Die `else` Bedingung ist optional - auch hier gilt dass mehrere Anweisungen in einen Block zusammenzufassen sind.

```
boolean beenden = false;
//[...]
if (beenden)
```

9 https://de.wikibooks.org/wiki/Java%20Standard%3A%20Schl%C3%BCsselw%C3%B6rter

```
  System.out.println ("Oh ich soll mich beenden");
  System.exit(0);
```

Das vorstehende Beispiel zeigt einen typischen Fehler in Zusammenhang mit dem if-Konstrukt. Der Befehl `System.exit(0);` wird immer ausgeführt, da kein Block gebildet wurde.

Es ist auch eine Frage des guten Programmierstils solche if-Abfragen immer in geschweifte Klammern zu fassen, dadurch lassen sich die einzelnen Ausdrücke besser dem jeweiligen if zuordnen.

```
boolean beenden = false;
//[...]
if (beenden) {
  System.out.println ("Oh ich soll mich beenden");
}
  System.exit(0);
```

Beim unteren Quelltext wird im Gegensatz zum oberen klar, das `System.exit(0)` immer ausgeführt wird, da es nicht in den Klammern steht.

2.23 Ternary Operator (`?:`)

Wenn nur zwischen zwei Ausgabewerten unterschieden werden soll, so gibt es für die If-Then-Else-Anweisung eine Kurzform; und zwar den "Ternary-Operator".

```
(<boolescher Ausdruck>) ? AusgabewertTrue : AusgabewertFalse;
```

Beispiel

Anstelle der Anweisung

```
 //[...]
 if (gender.equals("männlich

)) {
    System.out.println (SSehr geehrter Herr

);
 } else {
    System.out.println (SSehr geehrte Frau

);
 }
```

kann man kürzer schreiben

```
```

```
 System.out.println( (gender.equals("männlich
) ) ? SSehr geehrter Herr
 : SSehr
 geehrte Frau");
```

oder vielleicht etwas eleganter

```
 System.out.println( SSehr geehrte
 +(gender.equals("männlich
) ? "r Herr
 :
 Frau" ));
```

2.24 Mehrfachverzweigung (switch)

Das if[10]-Konstrukt führt in Zusammenhang mit mehreren Möglichkeiten meist zu einer unschönen Kaskadierung. Hier kann `switch` helfen. Um dieses einzusetzen müssen Sie einen Datentyp `int` abprüfen bzw. einen kleineren (also byte, short, char), da dieser durch die JVM automatisch gecastet wird.

```
 switch (myIntWert) {
 case 0 :
     System.out.println ("Mein Wert ist 0.
);
     break;
 case 1 :
     System.out.println ("Mein Wert ist 1.
);
 case 2 :
     System.out.println ("Mein Wert ist 1 oder 2.
);
     break;
 default :
     System.out.println (Ïch habe einen anderen Wert.
```

10 Kapitel 2.29 auf Seite 44

```
);
 }
```

Eine `switch` Anweisung wird stets dem Schlüsselwort `case` und meist mit `default` und `break` verwendet. Der Wert hinter dem `case` muss eine Konstante sein und dient dem Vergleich mit dem übergebenen Wert (*hier myIntWert*). Wenn diese gleich sind werden alle Anweisungen bis zum nächsten `break` oder dem Ende des Blocks ausgeführt. Wurden keine Übereinstimmung gefunden so werden die Anweisung nach `default` ausgeführt. Seit Java 7 ist es ebenfalls möglich, den Datentyp `String` in Switch-Anweisungen zu verwenden. Zuvor war es seit Java 5.0 lediglich möglich, anstelle von `String` den Datentyp `enums` zu benutzen.

2.25 For-Schleife

Die for-Schleife oder Zählschleife zählt von einem vorgegebenen Startwert zu einem ebenfalls vorgegebenen Endwert und führt für jeden Schritt von Start- bis Endwert alle Anweisungen im Schleifenkörper aus. Es muss also zusätzlich festgelegt werden, in welchen Intervallen bzw. Schritten von Start bis Ende gezählt wird.

For-Schleifen bestehen aus folgenden Teilen bzw. Anweisungen und Bedingungen:

- Schleifenkopf
  - Initialisierung der Zählervariable auf den Startwert
  - Startbedingung mit Limitierung auf den Endwert - muss immer einen boolschen Wert ergeben (true | false)
  - Anweisung zum Zählen
- Schleifenkörper
  - Anweisungen, die pro Schleifendurchlauf ausgeführt werden sollen

2.25.1 Die Syntax

Die grundlegende Syntax sieht folgendermaßen aus:

```
for(Initialisierung Zahlervariable; Startbedingung; Zählen) {
        ...
        Schleifenkörper mit Anweisungen
        ...
}

// Bsp.: for-Schleife, die von 0 bis 9 in 1er-Schritten
// durchlaufen wird
for(int i=0; i < 10; i++) {
        ...
        Anweisungen;
        ...
}
```

In obigem Beispiel sieht man eine for-Schleife, die von 0 bis 9 zählt. Warum nur bis 9? Weil die Startbedingung i kleiner als 10 und nicht kleiner gleich 10 lautet.

- Zunächst muss eine Zählervariable auf einen Startwert initialisiert werden `int i=0`. Wird diese Variable im Schleifenkopf deklariert, ist sie auch innerhalb ihres Namensraumes, also dem Schleifenkopf selbst sowie dem Schleifenkörper bekannt. Außerhalb der Schleife kann die Zählervariable folglich nicht angesprochen werden.
- Dann wird die Startbedingung für die for-Schleife festgelegt: `i < 10`. Sie läuft also **"solange i kleiner als 10 ist"**. Trifft diese Bedingung nicht zu, werden die Anweisungen im Schleifenkörper nicht ausgeführt. Die Startbedingung muss also immer einen boolschen Wert (true | false) ergeben.
- Zuletzt muss noch definiert werden in welchen Intervallen gezählt wird: `i++`. Die Schleife zählt also in Schritten von 1 nach oben bzw. addiert nach jedem Schleifendurchlauf 1 auf die Zählervariable i.

2.25.2 Ablauf einer for-Schleife

Nehmen wir folgende Schleife an:

```
for( int i=0; i < 3; i++ ){
        System.out.println( i );
}
```

Sie wird folgendermaßen durchlaufen:

1. Start: Initialisierung der Zählervariable auf den Wert 0
2. Prüfung der Startbedingung: i = 0 ist kleiner als 3 (i < 3 = true), also dürfen die Anweisungen des Schleifenkörpers ausgeführt werden
3. Erster Durchlauf: Die Zählervariable i wird am Bildschirm ausgegeben: 0
4. Hochzählen: Die Zählervariable wird um 1 erhöht: 0 + 1 = 1
5. Prüfung der Startbedingung: i = 1 ist kleiner als 3 (i < 3 = true), also dürfen die Anweisungen des Schleifenkörpers ausgeführt werden
6. Zweiter Durchlauf: Die Zählervariable i wird am Bildschirm ausgegeben: 1
7. Hochzählen: Die Zählervariable wird um 1 erhöht: 1 + 1 = 2
8. Prüfung der Startbedingung: i = 2 ist kleiner als 3 (i < 3 = true), also dürfen die Anweisungen des Schleifenkörpers ausgeführt werden
9. Dritter Durchlauf: Die Zählervariable i wird am Bildschirm ausgegeben: 2
10. Hochzählen: Die Zählervariable wird um 1 erhöht: 2 + 1 = 3
11. Prüfung der Startbedingung: i = 3 **genauso groß** wie 3 (i < 3 = false), also dürfen die Anweisungen des Schleifenkörpers **nicht mehr** ausgeführt werden

2.25.3 Verwendungszweck

For-Schleifen sind nützlich um eine bekannte Anzahl an immer wiederkehrenden Anweisungen auszuführen. Klassisch hierfür dürfte das Auslesen der Werte eines Arrays mittels einer Schleife sein. Beispielsweise dürfte ohne Schleifen die Ausgabe von in einem Array enthaltenen Werten sehr mühsam für den Programmierer sein. Dies würde bei zehn Werten wie folgt aussehen:

```
// erzeugt ein Array mit zehn Integer-Werten
```

```
int[] einArray = { 1,2,3,4,5,6,7,8,9,10 };

System.out.println( einArray[0] );
System.out.println( einArray[1] );
System.out.println( einArray[2] );
System.out.println( einArray[3] );
System.out.println( einArray[4] );
System.out.println( einArray[5] );
System.out.println( einArray[6] );
System.out.println( einArray[7] );
System.out.println( einArray[8] );
System.out.println( einArray[9] );
```

Diese Aufgabe lässt sich mittels einer Schleife erheblich erleichtern:

```
// erzeugt ein Array mit zehn Integer-Werten
int[] einArray = new int[10];
// nun wird das Befüllen des Arrays ebenfalls durch eine Schleife realisiert
for(int i=0; i < einArray.length; i++){
        einArray[i] = i + 1;
}

// For-Schleife, die die im Array enthaltenen Werte ausgibt
for( int i=0; i < einArray.length; i++ ){
        System.out.println( einArray[i] );
}
```

Mittels `.length` erhält man die Länge des Arrays als int-Wert. Somit lässt sich immer die Länge des Arrays bestimmen, auch wenn man diese zum Zeitpunkt der Programmierung noch nicht kennt.

Ein Array kann also mittels einer for-Schleife durchlaufen werden, da man einen Startwert = 0 sowie einen Endwert = `.length` hat. Da man alle Werte aus dem Array auslesen möchte (und nicht nur bestimmte), wird die Zählervariable immer um eins erhöht.

Im Schleifenkörper kann nun jeder Arrayplatz mit der Zählervariable angesprochen werden: `einArray[i]`.

Innerhalb des Schleifenkörpers kann jede Art von Anweisung durchgeführt werden. Muss man beispielsweise eine größere Anzahl an Parametern nacheinander an ein und dieselbe Methode übergeben, kann man dies mittels einer Schleife tun:

```
// wiederholter Methodenaufruf in einer Schleife
for(int i=0; i < einArray.length; i++){
        eineMethode( einArray[i] );

        einObjekt.eineMethode( einArray[i] );
}
```

Es ist natürlich ebenfalls möglich mehrmals Objekte des gleichen Typs zu erstellen.

```
// ein Array zur Aufnahme von Objekten erzeugen
Integer[] einArray = new Integer[100];

// Befüllen des Arrays mit 100 Objekten vom Typ Integer
for( int i=0; i < einArray.length; i++ ){
        einArray[i] = new Integer( i );
}
```

2.25.4 Abkürzung

Bei Arrays gibt es eine Kurzschreibweise, um über ein vorher belegtes Array zu iterieren:

```
int[] einArray  = {1, 2, 3};
for( int i : einArray ) {
      System.out.println(i);
}
```

2.25.5 Variationen

Bei der Implementierung des Schleifenkopfes ist man nicht strikt an die in obigem Beispiel vorgestellte Form gebunden. Vieles ist möglich, solange man sich an die Syntax hält. Die folgenden Beispiele sollen nur kleine Anregungen sein.

```
// Bsp.: Initialisierung der Zählvariable
for( byte i = -120; i < 100; i++ ){
        System.out.println( i );
}

int i = 50;
for(  ; i < 100; i++ ){
        System.out.println( i );
}

// Bsp.: Implementierung der Startbedingung
// Bsp.: abwärts zählen
for( int i = 150; i >= 100; i-- ){
        System.out.println( i );
}

// Bsp.: größeres Intervall
for( int i = 1; i <= 100; i+=10 ){
        System.out.println( i );
}

for( int i = 1; i <= 100; i*=2 ){
        System.out.println( i );
}

// Endlosschleife
for( ; ; ){
        System.out.println( 1 );
}
```

2.26 Schleife mit Vorabprüfung (`while`)

Die `while`-Schleife führt den Anweisungsblock aus, solange die Prüfung `true` ergibt. Die Prüfung der Bedingung erfolgt dabei vor dem Betreten der Schleife. Beispiele

```
while (true) {} // Endlosschleife
```

```
int i = 0;
while (i < 10) { i++; } // Ausführungsanzahl 10
```

```
int i = 0;
while (i < 0) {
  // wird nicht ausgeführt.
  System.out.println ("Come to me - PLEEEAAASE!
);
}
```

2.27 Schleife mit Nachprüfung (do)

Die do-Schleife führt alle beinhalteten Anweisungen solange aus, wie die Prüfung `true` ergibt. Die Prüfung der Bedingung erfolgt dabei nach dem ersten Schleifendurchlauf. Zu einer do Schleife wird stets das Schlüsselwort[11] `while` zur Bedingungsangabe benötigt. Beispiele

```
do {} while (true); // Endlosschleife
```

```
int i = 0;
do { i++; } while (i < 10); // Ausführungsanzahl 10
```

```
int i = 0;
do {
  System.out.println("Thanx to come to me.
);
} while (i < 0);
```

11 https://de.wikibooks.org/wiki/Java%20Standard%3A%20Schl%C3%BCsselw%C3%B6rter

2.28 Schleifen verlassen und überspringen (`break` und `continue`)

Innerhalb einer do, while oder for-Schleife kann mit `break` die gesamte Schleife verlassen werden. Soll nicht die gesamte Schleife, sondern nur der aktuelle Schleifendurchlauf verlassen und mit dem nächsten Durchlauf begonnen werden, kann die Anweisung `continue` verwendet werden. Im folgenden Beispiel werden in einem Iterator über eine Personenkartei alle Personen gesucht, die zu einer Firma gehören. Dabei werden gesperrte Karteikarten übersprungen. Wird eine Person gefunden, die zu dieser Firma gehört und als Freiberufler tätig ist, wird die weitere Suche abgebrochen, da angenommen wird, dass ein Freiberufler der einzige Angehörige seiner Firma ist.

```
  String desiredCompany = "Wikimedia

;
  Person[] persons;
  StringBuffer nameListOfWikimedia;

  for (int i=0;i<MAX_PERSONEN;i++) {
    if (persons[i].isLocked()) {
      continue; // Gesperrte Person einfach überspringen
    }
    String company = persons[i].getCompany();
    if (company.equals(desiredCompany)) {
      nameListOfWikimedia.append(persons[i].getName());
      if (persons[i].isFreelancer()) {
        break; // Annahme: Ein Freelancer hat keine weiteren Mitarbeiter
      }
    }
  }
```

Diese Aussprünge ähneln den goto-Befehlen anderer Programmiersprachen, sind aber nur innerhalb der eigenen Schleifen erlaubt. Bei Schachtelung mehrerer Schleifen können diese auch mit Bezeichnern (Labels) versehen werden, so dass sich die break- oder continue-Anweisung genau auf eine Schleife beziehen kann:

```
  catalog: while(catalogList.hasNext()) {
    product: while (productList.hasNext()) {
      price: while (priceList.hasNext()) {
        [...]
        if (priceTooHigh) {
          continue product;
        }
      }
    }
  }
```

2.29 Übungen

1. Schreibe ein einfaches Programm, welches für jeden Monat die Anzahl der Tage ausgibt. Nutze hierzu die einfache Verzweigung.
2. Schreibe ein einfaches Programm, welches für jeden Monat die Anzahl der Tage ausgibt. Nutze hierzu die Mehrfachverzweigung.
3. Schreibe eine einfache Applikation, welche mit Hilfe der Zählschleife die Übergabeparameter ausgibt. Sofern du das JDK 1.5 oder höher verwendest nutze hierfür die alte und neue Variante.
4. Wie oft wird die folgende `do`-Schleife ausgeführt und warum?

```
int i = 10;
do {
  i -= 3;
} while (i > 5);
```

Die Sprache Java gehört zu den objektorientierten Programmiersprachen. Die Grundidee der objektorientierten Programmierung ist die softwaretechnische Abbildung in einer Art und Weise, wie wir Menschen auch Dinge der realen Welt erfahren. Die Absicht dahinter ist, große Softwareprojekte einfacher verwalten zu können, und sowohl die Qualität von Software zu erhöhen als auch Fehler zu minimieren. Ein weiteres Ziel der Objektorientierung ist ein hoher Grad der Wiederverwendbarkeit von Softwaremodulen.

Allgemeines zum Thema Objektorientierung findet sich im dazugehörigen Wikipedia-Eintrag[12]. Dieses Kapitel beschäftigt sich speziell mit der Umsetzung in Java.

Ab der frühen Kindheit wird uns beigebracht, hinter den Dingen ein Schema zu erkennen. Wir lernen, dass Bello ein Hund ist. Zu diesen Bildern ordnen wir Eigenschaften und Fähigkeiten zu ("Farbe ist schwarz" oder "kann bellen"). Diese Verallgemeinerung ist eine der Stärken der Menschen. Es wird wohl daran liegen, dass die meisten Menschen nicht mehr als sieben Dinge gleichzeitig berücksichtigen können. Wen wundert es, wenn wir versuchen dieses Vorgehen in unsere Programmiersprachen zu übertragen. Somit können wir Dinge aus der Wirklichkeit auf die Maschine übertragen, weil sie abbildbar sind.

Der Hund wird in Java als Klasse repräsentiert. Bello wird als Objekt/Exemplar der Klasse verstanden.

Wesentliche Ziele von objektorientierter Programmierung sind Übersichtlichkeit, einfache Modifizierbarkeit, Flexibilität und einfache Wartung des Programmcodes. Erreicht wird dies durch den Ansatz, Objekte als relativ eigenständig zu verstehen. So ist es beispielsweise für das Gassigehen unerheblich, welchen Hund wir nun konkret an die Leine nehmen: Das kann Bello sein, es kann aber auch der Hund der Nachbarin sein, die im Urlaub ist. Das Vorgehen ist beide Male das Gleiche, unabhängig davon welche Farbe der Hund hat oder welcher Rasse er angehört.

12 `https://de.wikipedia.org/wiki/Objektorientierte%20Programmierung`

Um Gassigehen zu können, muss es sich um einen Hund handeln. Nur dieser Punkt interessiert. Ob der Hund womöglich auch noch prämierter Gewinner von Schönheitswettbewerben ist oder auf einem Skateboard fahren kann, ist unwichtig.

Das hört sich sehr banal an, hat aber weitreichende Konsequenzen. Es ist ein fundamentales Konzept von objektorientierter Programmierung nur diejenigen Eigenschaften eines Objektes zu betrachten, die einen auch wirklich interessieren.

Durch einen Übersetzungsfehler des Wortes „instance" aus dem Englischen wird immer wieder von einer Instanz gesprochen, wobei hier der Begriff „Instanz" falsch ist. Besser passende Übersetzungen für „instance" sind „Objekt", „Ausprägung", „Exemplar", oder auch „realisiertes Beispiel".

2.30 Aufbau einer Java-Klasse

```
public class Testklasse {
    private static final int klassenkonstante = 42;
    private static int anzahl;

    public static void gib_etwas_aus() {
        System.out.println(klassenkonstante + " Anzahl der übergebenen
 Argumente:" + anzahl);
    }

    public static void main(String[] args) {
        for(String i : args)
          System.out.println(i);
        anzahl = args.length;
        gib_etwas_aus();
    }
}
```

Jede mit "public" gekennzeichnete Klasse muss in einer eigenen Datei gespeichert werden, die Klassenname.java heißt. Schlüsselwörter wie "public" und "private" sind Zugriffsmodifizierer, "final" bezeichnet in diesem Kontext eine Konstante und wird ebenso wie "static" weiter unten erklärt. Die "Testklasse" enthält zwei Methoden: "gib_etwas_aus()" und "main(String[] args)". "klassenkonstante" und "anzahl" nennt man Klassenvariablen, man erkennt Klassenvariablen am Schlüsselwort "static".

```
class AndereKlasse {
    public int eineVariable;

    public AndereKlasse(int eingabe) {
        eineVariable = eingabe;
    }
}

public class Testklasse  {
    public static void main(String[] args) {
        AndereKlasse a = new AndereKlasse(42);
        System.out.println(a.eineVariable);
    }
}
```

Dieses Beispiel enthält zwei verschiedene Klassen. Die Variable "eineVariable" in der Klasse "AndereKlasse" ist eine Instanz-Variable, hingegen ist "a" aus der Methode "main()" eine

lokale Variable. In der Variablen ”a” ist das Objekt AndereKlasse gespeichert. Klasse und Objekt sind also zwei verschiedene Dinge.

2.31 Zugriffsmodifizierer

In der Objektorientierung kennt man so genannte Zugriffsmodifizierer (engl. *access modifier*), die die Rechte anderer Objekte einschränken (Kapselung) oder die ein bestimmtes Verhalten von einem Unterobjekt verlangen (Abstraktion/Vererbung).

Manchmal hört man auch die Bezeichnung Sichtbarkeitsmodifizierer (engl. *visibility modifier*). Diese Bezeichnung ist aber eigentlich falsch, weil die Zugriffsmodifizierer den Zugriff auf einen Member verbieten, der Member als solches bleibt jedoch sichtbar, z.B. über Reflection. Dennoch sollte man diese Bezeichnung verstehen, da sie unter Java-Programmierern weit verbreitet ist.

Java kennt folgende Zugriffsmodifizierer:

	Die Klasse selbst, innere Klassen	**Klassen im selben Package**	**Unterklassen**	**Sonstige Klassen**
private	Ja	Nein	Nein	Nein
(default)	Ja	Ja	Nein	Nein
protected	Ja	Ja	Ja	Nein
public	Ja	Ja	Ja	Ja

2.31.1 `private`

`private` ist der restriktivste Zugriffsmodifizierer. Er verbietet jeglichen Zugriff von außerhalb der Klasse auf den entsprechend modifizierten Member. Auf eine private Variable kann nur die Klasse selbst zugreifen, ebenso auf einen privaten Konstruktor, eine private Methode oder einen privaten geschachtelten Datentyp.

Klassenvariablen werden üblicherweise als `private` deklariert. Das Verändern der Variablen wird über passende Methoden, meist *Getter* und *Setter* ermöglicht. Dieses Prinzip nennt man Geheimnisprinzip.

```
/** Mutable 2D-Punkt-Klasse. */
public class Punkt2D {
    private int x, y; // private Variablen
    public Punkt2D(final int x, final int y) {
        setX(x);
        setY(y);
    }
    public int getX() { return x; }
    public int getY() { return y; }
    public void setX(final int x) { this.x = x; }
    public void setY(final int y) { this.y = y; }
}
```

Allgemein werden sämtliche Member als `private` deklariert, die ein Implementierungsdetail darstellen, auf das sich keine andere Klasse verlassen bzw. das von keiner anderen Klasse verwendet werden darf.

2.31.2 (default)

Als ”default” oder ”package private” bezeichnet man die Zugreifbarkeit für den Fall, dass kein Zugriffsmodifizierer angegeben wurde. Auf einen package private Member können nur Klassen zugreifen, die sich im selben Paket wie die Klasse des Members befinden.

Der ”default” Zugiffsmodifizierer wird dann eingesetzt, wenn eine Klasse auf die Daten einer anderen Klasse (innerhalb des selben Packages) Zugriff haben soll, aber diese Funktionalitäten nach außen nicht verfügbar gemacht werden sollen.

Dieser Zugriffmodifizierer wird insbesondere bei der Entwicklung von API eingesetzt.

2.31.3 `protected`

Mit dem Zugriffsmodifizierer `protected` ist der Zugriff nicht nur Klassen aus dem selben Package (wie ”default”), sondern auch Subklassen der Klasse erlaubt. Dies gilt auch, wenn die betreffenden Subklassen aus einem anderen Package sind als die Klasse des betreffenden Members.

Der Zugriffsmodifizierer `protected` wird verwendet, wenn es nur für Subklassen Sinn hat, den betreffenden Member zu verwenden.

Diese Zugriffsmodifizierer findet in der API-Programmierung Einsatz. Auch Muster wie das Schablonenmuster (engl. *template pattern*) nutzen diesen Mechanismus.

2.31.4 `public`

Der Zugriffsmodifizierer `public` gestattet sämtlichen Klassen Zugriff auf den betreffenden Member. Er ist der freizügigste und zugleich der am häufigsten eingesetzte Zugriffsmodifizierer.

Die Zugreifbarkeit `public` findet man hauptsächlich bei wichtigen Methoden, die von anderen Klassen verwendet werden sollen, bei Konstruktoren und bei Datentypen.

2.31.5 Welcher Zugriffsmodifizierer?

Welchen Zugriffsmodifizierer soll man nun verwenden? Im einfachsten Fall verwendet man `private` für Variablen sowie `public` für Methoden, Konstruktoren und Datentypen. Eine Regel, die sich in der Praxis sehr bewährt hat, lautet ”so streng wie möglich, so freizügig wie nötig”.

2.32 Polymorphie-Modifizierer

Polymorphie-Modifizierer sind dazu da, ein bestimmtes Verhalten einer Unterklasse zu erzwingen, bzw. dieses zu erleichtern. Das heißt, dass wenn eine Klasse von einer abstrakten Klasse abgeleitet werden soll, die Unterklasse diese Attribute bzw. Methoden implementieren muss, wenn es eine konkrete Klasse werden soll. Doch nun zu den einzelnen Modifizierern selbst:

2.32.1 abstract

Es können Methoden und Attribute als abstract bezeichnet (deklariert) werden, was bedeutet, dass entweder die Unterklasse diese implementieren muss oder aber die abgeleitete Klasse ebenfalls als abstrakt deklariert werden muss.

Von einer abstrakten Klasse können keine Instanzen gebildet werden, so dass diese immer erst implementiert werden muss, um das gewünschte Ergebnis zu erreichen. Ein Beispiel:

```
public abstract class Berechne {
    public abstract int berechne(int a, int b);
}
```

Dies ist eine abstrakte Klasse mit einer Methode. Was genau berechnet werden soll, steht hier aber nicht - deswegen heißt diese auch ”abstrakt”. Jetzt erweitern wir diese Klasse um eine Addition zu erhalten:

```
public class Addiere extends Berechne {
    public int berechne(int a, int b) {
        int c = a + b;
         return c;
     }
}
```

Wie man an diesem konkreten Beispiel sieht, wird erst in der ”konkreten” Klasse der Algorithmus implementiert.

2.32.2 final

Es können Klassen, Methoden, Attribute und Parameter als final bezeichnet (deklariert) werden. Einfach ausgedrückt bedeutet final in Java ”du kannst mich **jetzt** nicht überschreiben”.

Für finale Klassen bedeutet dies, dass man von ihr nicht erben kann (man kann keine Unterklasse erzeugen). Sie kann also nicht als Vorlage für eine neue Klasse dienen. Grundlegende Klassen, wie zum Beispiel die String-Klasse sind final. Wenn sie es nicht wäre, dann könnte man von ihr erben und ihre Methoden überschreiben und damit das Verhalten der erweiterten Klasse verändern.

Finale Methoden können in Subklassen nicht überschrieben werden.

Finale Attribute und auch Klassen-Variablen können nur ein einziges Mal zugewiesen werden. Sobald die Zuweisung erfolgt ist, kann eine finale Variable ihren Wert nicht mehr

ändern. Bei Member-Variablen muss die Zuweisung bei der Instanzierung, bei Klassen-Variablen beim Laden der Klasse erfolgen.

Finale Parameter können ausschliesslich den beim Methodenaufruf übergebenen Wert besitzen. In der Methode selbst lassen sie sich nicht überschreiben.

Ein Beispiel:

```
public int getSumme (final int summand1, final int summand2) {
   return summand1 + summand2;
}
```

Der Compiler hat die Möglichkeit, finale Member-Variablen, denen Konstanten zugewiesen werden, direkt im kompilierten Code zu ersetzen.

Ein Beispiel:

```
public class AntwortAufAlleFragenDesUniversums {
    private final long antwort = 23;
    public long gibAntwort() {
        return antwort;
    }
}
```

Der Compiler macht daraus:

```
public class AntwortAufAlleFragenDesUniversums {
    public long gibAntwort() {
        return 23;
    }
}
```

Wie Sie sehen, hat der Compiler hier einfach die finale Variable durch den Wert ersetzt. Diese Optimierung ist aber nur möglich, da der Wert konstant ist und direkt zugewiesen wird.

In folgendem Beispiel kann der Compiler nicht optimieren:

```
public class WievielMillisSindVerstrichen {
    private final long antwortInstanz = System.currentTimeMillis();
    private final static long antwortKlasse = System.currentTimeMillis();

    public long gibAntwortInstanz() {
        return antwortInstanz;
    }

    public long gibAntwortKlasse() {
        return antwortKlasse;
    }
}
```

Der Wert der Variable antwortKlasse wird beim Laden der Klasse zugewiesen. Die Klasse wird dann geladen, wenn sie das erste Mal zur Laufzeit benötigt wird. Der Wert der Variable antwortInstanz wird beim Instanzieren der Klasse, genauer gesagt bei der Initialisierung des Objekts auf Ebene der Klasse WievielMillisSindVerstrichen, also mit new WievielMillisSindVerstrichen() zugewiesen.

Noch eine Warnung beim Verwenden von Konstanten in Java. In Java deklariert man Konstanten wie folgt:

```
public class Konstante {
    public final static int ICH_BIN_EINE_KONSTANTE = 42;
}

public class Beispiel {
    public static void main(String[] args) {
        System.out.println(Konstante.ICH_BIN_EINE_KONSTANTE);
    }
}
```

```
javac Konstante.java Beispiel.java
```

Der Java Compiler ersetzt dann überall die Variable durch dessen Konstante. Dadurch gelangt die Konstante (42) und nicht die Variable (ICH_BIN_EINE_KONSTANTE) in das Kompilat (Konstante.class und Beispiel.class).

```
java Beispiel
```

gibt wie erwartet die Zahl 42 aus.

Wenn ich jetzt den Wert der Konstante von 42 auf 99 ändere und nur die Klasse Konstante compiliere, wird es gefährlich:

```
public class Konstante {
    public final static int ICH_BIN_EINE_KONSTANTE = 99;
}
```

```
javac Konstante.java
java Beispiel
```

gibt immer noch die Zahl 42 aus, obwohl ich doch den Wert der Variable ICH_BIN_EINE_KONSTANTE geändert habe. Warum? Als das Kompilat der Klasse Beispiel (Beispiel.class) erzeugt wurde, war der Wert von ICH_BIN_EINE_KONSTANTE noch 42. Der Compiler hat zu dem Zeitpunkt den Wert 42 in die Klasse Beispiel eingesetzt. Weil die Klasse Beispiel nicht nochmals kompiliert wurde steht im Kompilat immer noch den Wert 42 und der wird dann auch ausgegeben.

```
javac Beispiel.java
java Beispiel
```

hilft und gibt dann auch 99 aus.

Konsequenzen: Bei einer Änderung von public/protected final static sollte immer der gesamte Code neu kompiliert werden. Wenn das nicht möglich ist (z.B. in einer Klasse, die von verschiedenen anderen Projekten benutzt wird), sollte man auf die Verwendung von Konstanten verzichten und stattdessen einfach eine Methode verwenden:

```
public class Konstante {
    private final static int ICH_BIN_EINE_KONSTANTE = 99; // ungefährlich, da
nur innerhalb der Klasse der Zugriff möglich ist

    public static getDieKonstante() {
        return ICH_BIN_EINE_KONSTANTE;
```

```
    }
}
```

Ein letztes Beispiel soll aufzeigen, dass mit final lediglich die Zuweisung (das Überschreiben), nicht aber der Gebrauch geschützt wird.

```
import java.util.List;
import java.util.LinkedList;

public class NamensListe {
    private final List interneNamensliste = new LinkedList();

    public void neuerName(String name) {
        interneNamensliste.add(name);
    }

    public int anzahlNamen() {
        return interneNamensliste.size();
    }

    public List gibListe() {
        return interneNamensliste;
    }
}
```

Wie man hier schön sieht, kann man auf finalen Member-Variablen, hier die internenNamensliste, Methoden ausführen (wie das Hinzufügen mit interneNamensliste.add(name)). Da die Member-Variable interneNamensliste final ist, kann man ihr keinen anderen Wert zuweisen. Eine zusätzliche Methode

```
public void entferneListe() {
    interneNamensliste = null;
}
```

ist nicht zulässig und würde beim Kompilieren mit einer Fehlermeldung enden. Richtig riskant ist die Methode gibListe(). Indem die interneNamensliste der Außenwelt (außerhalb der Klasse) verfügbar gemacht wird, kann man von der Außenwelt auch alle Methoden der Liste aufrufen. Damit ist die Liste außerhalb des Einflussbereiches der Klasse. Da hilft es auch nicht, dass die Member-Variable interneNamensliste final ist.

2.32.3 static

Es können Methoden und Klassenvariablen als static bezeichnet (deklariert) werden.

Statische Methoden und Variablen benötigen keinerlei Instanzen einer Klasse, um aufgerufen zu werden. Ein Beispiel für einen statischen Member ist z.B. die Konstante PI in der Klasse java.lang.Math.

Auch die Methoden dieser Klasse können einfach aufgerufen werden, ohne vorher eine Instanz dieser Klasse anzulegen, z.B. java.lang.Math.max(3,4).

Sofern eine statische Klassenvariable erst zur Laufzeit dynamisch einen Wert erhalten soll, können Sie dies mit einem statischen Block erreichen.

Beispiel:

```
public class LoadTimer {
```

```
  static {
    ladeZeit = System.currentTimeInMillis ();
  }
  private static long ladeZeit;
}
```

Es ist dennoch möglich, statische Methoden oder Attribute über ein Objekt aufzurufen, davon wird aber dringend abgeraten. Denn dies führt zu unangenehmen Ergebnissen.

Beispiel:

```
public class SuperClass {
 public static void printMessage(){
  System.out.println("Superclass: printMessage");
 }
}

public class SubClass extends SuperClass {
 public static void printMessage(){
  System.out.println("Subclass: printMessage");
 }
}

public class StrangeEffect {
 public static void main(String[] args){
  SubClass object = new SubClass();
  object.printMessage();

  SuperClass castedObject = (SuperClass)object;
  castedObject.printMessage();
 }
}
```

Die Ausgabe ist:

```
Subclass: printMessage
Superclass: printMessage
```

Erstaunlich ist die zweite Zeile: Obwohl unser `object` vom Typ SubClass ist, wird die Methode von SuperClass aufgerufen. Offensichtlich funktioniert hier das Überschatten nicht. Das liegt daran, dass statische Methodenaufrufe nicht vom Laufzeittyp abhängen! Konkret bedeutet dies, dass die Entscheidung welche statische Methode nun aufgerufen werden soll, unabhängig davon getroffen wird, welcher Klasse das Exemplar angehört.

Um diesen irreführenden Effekt zu vermeiden, sollte man statische Methoden immer auf Klassen aufrufen und nicht auf Objekten.

2.32.4 strictfp

`strictfp` kennzeichnet Klassen und Methoden, deren enthaltene Fließkommaoperationen streng auf eine Genauigkeit von 32 bzw. 64 Bit beschränkt sind. Dadurch wird sichergestellt, dass die JVM darauf verzichtet, Fließkommaoperationen intern mit einer höheren Genauigkeit zu berechnen (beispielsweise 40 bzw. 80 Bit) und nach Abschluss der Berechnung wieder auf 32 bzw. 64 Bit zu kürzen. Dies ist wichtig, da es sonst bei verschiedenen JVMs

oder verschiedener Hardware zu unterschiedlichen Ergebnissen kommen kann und somit das Programm nicht mehr plattformunabhängig ist.

Ausdrücke, die zur Kompilierzeit konstant sind, sind immer FP-strict.

2.32.5 native

`native` kann nur vor Methoden stehen und bedeutet, dass die Implementierung der betreffenden Methode nicht in Java, sondern einer anderen Programmiersprache geschrieben wurde, und von der virtuellen Maschine über eine Laufzeitbibliothek gelinkt werden muss. Die Syntax der Methode entspricht dann einer abstrakten Methode. Ein Beispiel:

```
public native void macheEsNichtInJava ();
```

Um eine solche Methode zu verwenden muss sie nach dem Compilieren über einen "Java-Pre-Compiler" (javah) gejagt werden, d.h. dieser generiert aus einer "native" - Methode ein entsprechenden C-Header und Rumpf, der dann mit Leben gefüllt werden kann. Diese Rümpfe werden als dll unter Windows bzw. lib unter Linux/Unix compiliert. Diese compilierten Libs müssen dann aber auch zur Laufzeit des Programms zugreifbar sein, andernfalls erhält der Nutzer eine Exception.

2.33 Klassen, Objekte, Instanzen

Eine **Klasse** beschreibt die (allgemeine) Definition. Alles mit Ausnahme der primitiven Datentypen (int, boolean, long etc.) in Java ist abgeleitet von "java.lang.Object". Das heißt, dass jede Java-Klasse, die Sie schreiben, bestimmte Methoden bereits von "Object" geerbt hat.

Ein **Objekt** – auch **Instanz** genannt – ist ein bestimmtes Exemplar einer Klasse und wird zur Laufzeit des Programms erzeugt.

2.34 Konstruktoren

Um eine Instanz einer Klasse zu erschaffen, wird der Konstruktor benutzt. Der Konstruktor ist namentlich wie die Klasse zu benennen. Die Syntax entspricht hierbei einer Methode jedoch gibt der Konstruktor keinen Wert, das heißt auch kein void, zurück.

```
package org.wikibooks.de.javakurs.oo;
public class MeineKlasse
{
  // Nun der Konstruktor
  public MeineKlasse ()
  {
  ...
  }
}
```

2.35 Destruktoren

Destruktoren gibt es in Java nicht. Es besteht eine gewisse Wahrscheinlichkeit dafür, dass die `finalize` Methode einer Instanz vor dessen Zerstörung durch die Garbage Collection aufgerufen wird. Dies ist jedoch **nicht** sichergestellt.

2.36 Methoden in "java.lang.Object"

Hier sind nun einmal die wichtigsten Methoden, die man in allen Java-Objekten finden und aufrufen kann, aufgeführt.

2.36.1 toString()

Diese Methode ist während der Entwicklung eines Programmes von großem Nutzen. Jede Klasse, welche toString() überschreibt, kann hier den relevanten Inhalt der Klasse in Textform ausgeben. Mit System.out.println(object); kann man nun den Inhalt eines Objektes ansehen.

2.36.2 equals(Object)

Java unterscheidet zwei Arten von Gleichheit:

Identisch: Zwei Objekte sind identisch, wenn sie beide das selbe Objekt (exakt am selben Ort im Speicher) referenzieren, was mittels "==" getestet wird. In der Objektorientierung heißt das auch, dass diese eine gemeinsame Instanz haben. Stellen Sie sich das einfach so vor, als wenn Hausnummern in einer Stadt verglichen werden: Die Hausnummern sind zwar gleich, aber es können verschiedene Leute im Haus wohnen.

Beispiel:

```
Object obj1 = new Object();
Object obj2 = obj1;
if (obj1==obj2)
{
    // beide Objekte obj1 und obj2 sind identisch, da sie beide am selben Ort
im Speicher liegen.
}
```

Gleich resp. equal: Zwei Objekte sind gleich, wenn sie denselben semantischen Inhalt repräsentieren und das wird mittels equals(..) getestet. Um beim Hausbeispiel zu bleiben: Hier vergleichen wir die Leute, die in einem Haus wohnen, mit der Liste von Leuten, die in diesem Haus wohnen sollen.

Beispiel:

```
String name1 = new String("Peter Muster");
String name2 = new String("Peter Muster");
if (name1.equals(name2))
{
     // beide Objekte repräsentieren denselben Inhalt, sie sind aber nicht am
```

```
selben Ort gespeichert
     // name1 == name2 ergibt deshalb false, name1 und name2 sind gleich, aber
nicht identisch.
}
```

Ein Beispiel aus der realen Welt: Zwillinge sind gleich, aber nicht identisch. Weshalb: wenn man dem einen Zwilling z.B. die Haare färbe, dann behält der andere Zwilling seine Haarfarbe. Dasselbe gilt für Objekte:

Wenn zwei Objekte gleich, nicht aber identisch sind, dann kann ich im ersten Objekt etwas ändern, ohne dass sich dadurch das zweite Objekt ändert.

Wenn zwei Objekte identisch sind und ich ändere im ersten Objekt etwas, dann ist diese Änderung auch beim zweiten Objekt sofort vorhanden (beide Objekte sind eben identisch).

Wenn Objekte identisch sind, dann sind sie immer auch gleich (zumindest sollte es so sein, sofern man equals korrekt implementiert hat).

Benutzung der Methode equals: Hier muss man sicherstellen, dass das Objekt, auf welchem die Methode equals aufgerufen werden soll, nicht null ist. Typischweise sieht man dann sowas:

```
if (name1!=null && name1.equals(name2)) {...}
```

Eigene Implementierung Wenn Sie eine eigene Klasse anlegen sollten Sie stets diese Methode implementieren.

2.36.3 hashCode()

Die Methode hashCode() liefert einen int als Rückgabewert und wird überall dort verwendet wo direkt oder indirekt mit Hashtables oder Hash-Sets gearbeitet wird. Zur Definition einer Hashtable siehe weiter unten. Wichtig: wenn zwei Objekte gleich (equals) sind, dann sollten sie unbedingt denselben hashCode zurückliefern. Umgekehrt ist es aber erlaubt, dass, wenn zwei Objekte nicht gleich sind, ihre beiden hashCodes gleiche Werte zurückliefern. Deshalb sollte man, wenn man equals() überschreibt auch gerade hashCode() überschreiben.

Und was geschieht, wenn man equals überschreibt, aber den hashCode nicht? Solange man die Objekte nicht in einer Hashtable/HashMap oder einem HashSet unterbringt hat man keine Probleme. Das ändert sich aber, sobald man die Objekte in Hashtables/HashMap oder HashSets unterbringen will. Um zu verstehen, was dabei ”schiefgeht”, muss man verstehen, wie Hashtables funktionieren.

Zur Definition einer Hashtable: Es ist eine Datenstruktur, welche erlaubt, sehr schnell Objekte anhand ihres Schlüssels abzulegen und anhand dieses Schlüssels wieder aufzufinden. Einfach ausgedrückt ist eine Hashtable ein Array von Schlüssel-Objekt-Paaren einer bestimmten Größe. Der Hash-Wert des Schlüssels modulo Größe der Hashtable dient dabei als Index, wo der Schlüssel selbst und das dazugehörige Objekt abgelegt werden. Falls nun mehrere Schlüssel an denselben Platz in der Hashtable platziert werden, so wird (in Java, nicht zwingend aber in anderen Programmiersprachen) einfach eine Liste benutzt, wo alle Schlüssel-Werte-Paare der entsprechenden Hashtable-Position gespeichert werden. Um Objekte aus der Hashtable herauszuholen, wird fast dasselbe gemacht wie beim Einfügen. Es

wird die Position in der Hashtable berechnet (hash-Code des Schlüssels modulo Hashtable-Größe) und dann alle dort gefunden Objekte mittels der equals-Methode verglichen. Sobald die equals-Methode erfolgreich war, so hat man das richtige Objekte gefunden. Das Ganze ist dann besonders effizient, wenn die Hashtable klein ist und alle Schlüssel genau einen eigenen Index rsp. Hashtable-Position haben. Als Faustregel gilt: Ist eine Hashtable zu 50% gefüllt, sinkt die Effizienz bei weiterem Einfügen schnell. Zudem sollten die hashCode-Werte möglichst gut verteilt sein. Am schlimmsten wäre es, wenn alle Objekte derselben Klasse als hashCode denselben Wert zurückliefern. Die Größe einer Hashtable kann man beim Erzeugen definieren. Zusätzlich kann man ihr einen Füllgrad angeben, ab wann sie sich vergrößern soll. Die Vergrößerung einer Hashtable ist eine sehr teure Operation, die man indirekt durch Hinzufügen eines neuen Schlüssel-Werte-Paares auslöst. Dabei werden alle Schlüssel-Werte-Paare in einer neuen, größeren Hashtable abgelegt. Das geschieht zwar automatisch, ist aber zeitaufwändig.

Was geschieht nun, wenn man in einer Klasse K equals überschreibt, nicht aber hashCode? Dann wird der hashCode der Oberklasse benutzt (was bei der Objekt-Klasse mehr oder weniger der Speicheradresse gleichkommt). Wenn nun als Schlüssel Instanzen der Klasse K benutzt werden, dann werden alle Instanzen mit großer Wahrscheinlichkeit gleichmäßig in der Hashtable abgelegt (das wäre ja noch gut). Beim Auffinden eines Objektes anhand eines Schlüssels wird der hashCode des Schlüssels berechnet, um die Position in der Hashtable herauszufinden. Wenn nun zwei Schlüssel gleich sind, aber unterschiedliche hashCodes haben, dann wird jedes Schlüssel-Wert-Paar an eine andere Position abgelegt. Die Folge: man findet Objekte mit großer Wahrscheinlichkeit nicht mehr, da man am ”falschen” Ort sucht.

Beispiel, was falsch läuft:

```
import java.util.*;

public class K extends Object {
  private String content;

  public K(String content) {
     this.content = content;
  }

  public boolean equals(Object obj) {
     if (this==obj) {
        return true;
     }
     if (obj==null) {
        return false;
     }
     if (!(obj instanceof K )) {
        return false; // different class
     }
     K other = (K) obj;
     if (this.content==other.content) {
        return true;
     }
     if (this.content==null && other.content!=null) {
        return false;
     }
     // this.content can't be null
     return this.content.equals(other.content);
  }

  public static void main(String[] args) {
     K k1 = new K("k"); // let's say has hashCode 13
```

```
     K k2 = new K("k"); // let's say has hashCode 19

     Map map = new HashMap();
     map.put(k1, "this is k1");

     String str = (String) map.get(k2);
     // str will be null because of different hashCode's
     System.out.println(str);
     // next line will print "true" because k1 and k2 are the same, so they
 should also return the same hashCode
     System.out.println(k1.equals(k2));
   }
 }
```

Was fehlt, ist der hashCode():

```
public class K extends Object {
...

 public int hashCode() {
    if (this.content==null) {
       return 0;
    }
    return this.content.hashCode();
 }
}
```

2.36.4 wait(), notify() und notifyAll()

Die sogenannte **Vererbung** ermöglicht es Informationen (Variablen) und Verhalten (Methoden / Operationen) weiterzugeben. Dies ist eine wesentliche Möglichkeit um Redundanz zu vermeiden. Die Erben fügen dann weitere Informationen und/oder Verhalten hinzu. Zwei Klassen stehen dabei zueinander als Superklasse (Erblasser) und Subklasse (Erbe) in Beziehung.

Vererbung ist ein zentrales Thema in der objektorientierten Programmierung (OOP) - siehe auch w:Objektorientierte Programmierung[13].

2.37 Erzeugen und Zerstören von Subklassen

Das Erzeugen und Zerstören von Subklassen erfolgt analog zu einer normalen Klasse. Intern werden jedoch auch die Standardkonstruktoren der jeweiligen Superklasse beim Erzeugen und analog die finalize Methode beim Zerstören aufgerufen. Dies ist wichtig, da wir die Informationen und das Verhalten der Superklasse verwenden. Evtl. Grundstellen der Informationen oder Aufräumarbeiten beim Zerstören unserer Instanz sind somit auch bei den ererbten Informationen notwendig.

13 https://de.wikipedia.org/wiki/Objektorientierte%20Programmierung

2.38 Überschreiben von Methoden

In der Subklasse können Methoden der Basisklasse "überschrieben" werden. Damit wird der Inhalt der ursprünglichen Methode verändert.

```
package org.wikibooks.de.javakurs.oo;
public class MeineKlasse
{
  // Nun der Konstruktor
  public MeineKlasse ()
  {
  //...
  }
  //ursprüngliche Methode
  public methode()
  {
     System.out.println("Wir sind in der Basisklasse

);
  }
}

public class MeineSubklasse extends MeineKlasse
{
  public methode()
  {
     System.out.println("Wir sind in der Subklasse

);
  }
}
```

Wird die `methode()`-Methode für ein Objekt des Typs `MeineSubklasse` aufgerufen, wird der Satz "Wir sind in der Subklasse" ausgegeben.

Überschriebene Methoden werden für die Anwendung der Laufzeit-Polymorphie in Java benötigt. Dabei werden Methoden erst zur Laufzeit (im Gegensatz zur Compile-Zeit) den Objekten zugeordnet.

Anmerkung: Im Deutschen wird das Verb "überschreiben" benutzt. Im Englischen heißt es jedoch "override", nicht "overwrite".

Mit dem JDK 1.6 / Java 6 sollte die zudem eine Annotation (Anmerkung) angebracht werden:

```
...
public class MeineSubklasse extends MeineKlasse
{
  @Override
  public methode()
  {
     System.out.println("Wir sind in der Subklasse
```

```
);
   }
 }
 ...
```

2.39 Javaspezifische Implementierung

In Java sind alle Objekte von der Klasse Object abgeleitet. Object ist somit die Basisklasse von allen anderen Klassen. Alle Klassen sind Subklassen von Object.

Man leitet eine eigene Klasse von der Basisklasse durch extends ab:

```
public class Beispiel extends Object
{
  //...
}
```

2.40 static

Als `static`-deklarierte Elemente gehören keinem Objekt an, sondern der Klasse (oder Schnittstelle), in der sie definiert sind. Das berühmteste Beispiel ist hier

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

Die main-Methode muss als `static` deklariert sein, da vor ihrem Aufrufen ja kein Objekt instanziiert sein kann.

Als `static` können sowohl Variablen als auch Methoden deklariert werden.

Da `static`-Methoden von allen Objekten unabhängig sind, haben sie Einschränkungen:

- Es können aus ihnen heraus nur andere `static`-Methoden aufgerufen werden.
- Sie können auch nur auf `static`-Variablen zugreifen.
- `static`-Methoden und -Variablen können nicht mit `this` oder `super` angesprochen werden.

2.41 abstract

Als `abstract`-deklarierte Methoden werden in der Basisklasse nur deklariert. Die Definition findet dann in den Subklassen statt.

```
abstract class BeispielKlasse
{
   abstract void schreibIrgendwas();
}

class BeispielSubklasse extends BeispielKlasse
{
   void schreibIrgendwas()
   {
      System.out.println(Ïrgendwas

);
   }
}
```

In der Basisklasse wird nur angegeben, dass die Methode existiert, aber nicht wie sie implementiert ist. Damit wird also zugesichert, dass jedes Objekt des Typs `BeispielKlasse` - und damit auch jedes Objekt einer davon abgeleiteten Klasse - die Methode `schreibIrgendwas()` besitzen muss.

In `BeispielSubklasse` muss die abstrakte Methode daher implementiert werden, ansonsten gibt es einen Compilerfehler. Wird eine abstrakte Methode in einer Unterklasse nicht implementiert, so muss diese Klasse selbst auch wiederum als `abstract` gekennzeichnet werden.

Generell muss jede Klasse, die mindestens eine abstrakte Methode enthält, auch selbst als abstrakt deklariert werden. Dies hat den Effekt, dass keine Objekte direkt von dieser Klasse erstellt werden können. Im obigen Beispiel wäre es also nicht zulässig, ein Objekt der Klasse `BeispielKlasse` mit

```
new BeispielKlasse();
```

zu erzeugen.

2.42 final

Durch die Benutzung von `final` kann das Ableiten einer Klasse oder Überschreiben einer Methode verhindert werden.

```
final class BeispielKlasse
{
   void schreibIrgendwas()
   {
     //...
   }
}

class BeispielSubklasse extends BeispielKlasse
{
  //...
}
```

Das funktioniert nicht. Durch das `final`-Schlüsselwort kann von der Klasse `Beispielklasse` nicht abgeleitet werden.

```
class BeispielKlasse
{
   final void schreibIrgendwas()
   {
     //...
   }
}

class BeispielSubklasse extends BeispielKlasse
{
   void schreibIrgendwas(){
     //...
   }
}
```

Hier wird zwar die Subklasse erstellt, jedoch kann die Methode `schreibIrgendwas()` nicht überschrieben werden.

2.43 super

Um den Konstruktor der Basisklasse aufzurufen, wird die Methode `super()` verwendet. Sie wird im Konstruktor der Subklasse verwendet. `super()` wird benutzt, um `private`-Elemente der Basisklasse anzusprechen.

```
class Beispiel extends Object
{
   protected variable;

   Beispiel()
   {
      super();
      variable = 10;
   }
}
```

`super()` ruft hier den Konstruktor von Beispiel auf, und könnte somit `private`-Elemente manipulieren.

Als zweite Anwendungsmöglichkeit kann `super` im Zusammenhang mit einem Element der Basisklasse benutzt werden.

```
super.methodeABC();
super.variableXYZ;
```

Durch diese Aufrufe werden aus der Subklasse heraus die Methoden/ Variablen der Basisklasse aufgerufen.

2.44 Einfluss von Modifizierern auf die Vererbung

2.44.1 public und protected

Sowohl public- wie auch protected-Elemente werden an die abgeleitete Klasse vererbt. An ihrem public bzw. protected-Status ändert sich nichts.

```
public methode();
protected variable;
```

2.44.2 private

Private-Elemente werden nicht vererbt.

```
private variable;
```

2.44.3 packagevisible

Dies ist der voreingestellte Modifizierer, der angewandet wird, wenn kein anderer Modifizierer angegeben wird. Durch ihn können nur Objekte aus dem gleichen Paket auf die Elemente zugreifen.

```
class Beispielsklasse
{
  //...
}
```

Er wird ohne Einschränkungen an die Subklasse vererbt.

2.45 Was ist ein Interface?

Interfaces als Java-Typ beschreiben eine öffentliche Schnittstelle der implementierenden Klassen. Interfaces sind hierbei eine Möglichkeit die Probleme der Mehrfachvererbung (**in Java sowieso nicht möglich**) elegant zu umgehen.

Schlüsselwort für die Deklaration eines Interface ist `interface`, welches anstelle der `class` Deklaration tritt. Ein Interface selbst kann nicht instanziert werden.

```
package de.wikibooks.org.oo;
public interface MyInterface {}
```

2.46 Deklaratorische Interfaces

Ein deklaratorisches Interface deklariert keine weiteren Methoden. Es stellt somit lediglich sicher, dass alle implementierenden Klassen vom Typ des Interface sind.

Ein Beispiel für ein Deklaratorisches Interface ist unser `MyInterface`.

```
package de.wikibooks.org.oo;
public class IchBinVomTypMyInterface implements MyInterface {

  public static void main (final String [] args) {
    final IchBinVomTypMyInterface instance = new IchBinVomTypMyInterface ();
```

```
    System.out.println ('''instance instanceof MyInterface''');
  }

}
```

2.47 "Normale" Interfaces

Normalerweise deklarieren Interfaces Methoden, die durch die implementierenden Klassen definiert werden. Die deklarierten Methoden sind dabei immer von der Sichtbarkeit `public`, auch wenn dies nicht explizit erwähnt wird.

```
package de.wikibooks.org.oo;
public interface NormalInterface {
  void go ();
}
```

2.48 Interface als Konstantensammlung

Interfaces werden auch gern zur Sammlung von Konstanten verwendet. Ein schönes Beispiel hierfür ist in `javax.swing.WindowConstants` zu finden. Jedoch wird davon abgeraten, Interfaces als Konstantensammlungen zu benutzen. Denn eigentlich definieren Interface neue Typen. Sun Microsystems hat dieses Problem nun adressiert und bietet mittlerweile einen eigenen Ausdruck für den Import von Konstanten aus anderen Klassen an: import static. Siehe dazu die Schlüsselwortreferenz für import.

2.49 Interfaces benutzen

Um ein Interface zu benutzen, muss dieses mit `implements` hinter dem Klassennamen spezifiziert werden. Anschließend müssen alle deklarierten Methoden des Interfaces implementiert (*oder die Klasse als* `abstract` *deklariert*) werden.

Es können auch durchaus mehrere Interfaces implementiert werden. Diese werden dann durch Kommata getrennt: `... implements Interface1, Interface2, Interface3 {`

```
package de.wikibooks.org.oo;
public class Walker implements NormalInterface {
  public void go () {}
}
```

Das Paket java.lang enthält die elementaren Grundtypen von Java. Es wird per Default immer bereitgestellt, so dass es nicht notwendig ist einen `import` dieses Pakets vorzunehmen.

2.50 Object - die Mutter aller Klassen

Die Klasse Object ist die **Wurzel im Vererbungsgraphen** von Java. Jede Klasse, die nicht explizit von einer anderen Klasse erbt, erbt automatisch von Object. Somit ist Object die

einzige Javaklasse ohne Vorfahren (”parent”). In Object sind nur eine handvoll Methoden versammelt, die aber wichtig für das gesamte Java-Laufzeitsystem sind. Diese Methoden werden von den abgeleiteten Klassen überschrieben und jeweils angepasst. Die Methoden kann man verschiedenen Zwecken zuordnen:

Verwendungszweck	**Methodennamen**
Erzeugung- und Initialisierung	**Object()** - Defaultkonstruktor
Objektbeseitigung	**finalize()** - wird vom Garbage Collector aufgerufen
inhaltlicher Vergleich zweier Objekte	**equals()**
vollständige Kopie	**clone()** - erzeugt eine ”tiefe” (physikalische) Kopie eines Objektes (vergleichbar mit Copyconstructor in C++)
Serialisierung in String	**toString()** - schreibt das aktuelle Objekt mit allen Infos in einen String
Laufzeitklasse	**getClass()**- gibt die Laufzeitklasse zurück
eindeutiger Identifier	**hashCode()** - gibt die eindeutige ID der Klasse zurück
Threadverwaltung	**notify()**, **notifyAll()**, **wait()**

Die Klasse Math ist das Matheobjekt mit allen Operationen für einfache numerische Berechnungen. Neben Konstanten PI und E werden auch viele mathematische Operationen wie Wurzelziehen, Exponentialzahlen, Sinus und Cosinus zur Verfügung gestellt. Alle Konstanten und Methoden in der Math-Klasse sind **static**, damit man kein eigenes Math-Objekt für jede Berechnung anlegen muss. Der Ergebnistyp fast aller Operationen ist **double**.

Konstanten: PI, E

```
double wert = Math.PI;
```

Wurzelziehen, Logarithmus und Exponentialfunktion: sqrt, log, pow

```
double x = Math.sqrt( 2 );
double y = Math.pow( 2,10 );  // 2^10 = 1024
```

Trigonometrische Funktionen: sin, cos, acos, asin, atan, atan2

```
double wert  = Math.sin( 0.0 );
double wert2 = Math.sin( Math.PI );
```

Wertetabelle für die ersten 10 Sinuswerte im Intervall [0..2*PI]

```
double schrittweite = 2.0*Math.PI/10.0;
for( double x=0.0; x<=2*Math.PI; x=x+schrittweite )
{
  System.out.println( "f( "+x+" ) = "+Math.sin( x ));
}
```

Minimum und Maximum: min, max

```
int x = Math.min( 2,4 );  //Ergebnis ist 2
int y = Math.max( 2,4 );  //Ergebnis ist 4
```

Absolutwert, Runden und Abschneiden: abs, ceil, floor, rint, round

```
double x = Math.abs( -4.1 );   //Ergebnis ist 4.1
int    y = Math.round( 4.2 );  //Ergebnis ist 4
```

Umrechnung Grad (0..360) in Radian (0..2*PI)

```
x = Math.toDegrees( Math.PI );
y = Math.toRadians( 90 );
```

Pseudozufallswert ausgeben aus dem Bereich von größer oder gleich 0.0 bis kleiner 1.0

```
double x = Math.random();
```

2.51 Was ist der Sinn von Wrapperklassen?

Wrapperklassen (”Hüllklassen”) dienen als Verbindungsglied zwischen den **Ordinaltypen** von Java (byte, short, int, long, float, double, char) und den Klassen in Java und dabei insbesondere der Klasse **String**. Ein Grundproblem bei der Benutzung von Oberflächen wie AWT und Swing ist, dass bei Eingaben (z.B. in ein Textfeld) immer nur Texte verwaltet werden. Diese Texte müssen in ”richtige” Zahlen verwandelt werden, um mit ihnen zu rechnen, und dann wieder in Texte zurückgewandelt werden, um sie in den GUI-Komponenten wieder anzuzeigen. Für dieses Verfahren, was auch als **Boxing/Unboxing** bekannt ist, werden die Wrapper-Klassen gebraucht. Bis auf die Klasse `Character` stammen alle Wrapper-Klassen von `java.lang.Number` ab. Es gibt nun zu jedem ordinalen Datentyp eine Wrapperklasse:

Ordinaltyp	**Wrapperklasse**
byte	Byte
short	Short
int	Integer
long	Long
float	Float
double	Double
boolean	Boolean
char	Character

2.52 Umwandlung von Ordinaltyp nach String

Beispiele für die **Wandlung von Ordinaltyp -> String**:

```
int i=10;
Integer ii = new Integer( i );
String s = ii.toString();
```

oder kürzer:

```
String s1 = Integer.toString( 10 );
String s2 = Float.toString( 3.14f );
```

2.53 Umwandlung von String nach Ordinaltyp

Beispiele für die **Wandlung von String -> Ordinaltyp**:

```
String s = "10";
int   i = Integer.parseInt( s );
float f = Float.parseFloat( s );
```

2.54 Umwandlung in andere Zahlensysteme

Mit Wrapperklassen ist auch die Umwandlung vom Dezimalsystem in andere Zahlensysteme (Hex, Oct, Dual) möglich. Auch für diese Operationen braucht kein neues Integer-Objekt angelegt werden weil die Operationen **static** sind.

Beispiele für die Umwandlung (das Ergebnis ist aber ein **String**!):

```
System.out.println( "42 = "+Integer.toBinaryString( 42 )+"b (binär)");
System.out.println( "42 = "+Integer.toHexString(    42 )+"h (hexadezimal)");
System.out.println( "42 = "+Integer.toOctalString(  42 )+"o (octal)");
```

2.55 Autoboxing/Autounboxing

In Java 5.0 wird vom Compiler mittlerweile ein automatisches Boxing/Unboxing vorgenommen. Dadurch ist es unnötig geworden, Ordinaltypen aus den Wrapperklassen mittels den toByte/toShort/toInt/toLong/toFloat/toDouble Methoden zu gewinnen. Andersherum kann einem Objekt einer Wrapperklasse direkt ein Ordinaltyp zugewiesen werden. Dieses Feature verkürzt den Code und macht ihn lesbarer. Allerdings muss nun der Programmierer dafür Sorge tragen, dass nicht ständig zwischen einem Wrapper-Objekt und einem Ordinaltyp hin- und hergewechselt werden muss, da dies enorm viele Resourcen beansprucht.

Beispiel:

```
Integer i = 5;
byte b = Byte.valueOf("7");
```

2.56 Welche Vor- und Nachteile hat man durch Wrapperklassen?

WrapperKlassen haben den Nachteil, dass der Overhead beim instanziieren geringfügig höher ist, sie sind einige Bytes größer als der dazugehörige primitive Datentyp.

Erstellt man größere Mengen, wirkt sich dies negativ auf die Geschwindigkeit und den benötigten Speicher eines Programmes aus.

Wrapper sind zum Beispiel erforderlich, um threadsichere primitive Datentypen zur Verfügung zu stellen.

> ⚠ **Achtung**
>
> Dieses Kapitel überschneidet sich mit Java Standard: Zeichenketten[14]. Aus praktischen Gründen gehören die Inhalte auf jeden Fall zu den Grundlagen. Es ist zu überlegen, ob das Kapitel "String" als Teil von `java.lang` erhalten bleibt und sich schwerpunktmäßig mit den theoretischen Grundlagen befassen soll oder ob es gestrichen werden kann.

2.57 Die Klasse String und StringBuffer

Zur Verarbeitung von **Texten** oder neudeutsch **Zeichenketten** gibt es in Java die Klasse String. Strings ermöglichen das effiziente Erstellen, Vergleichen und Modifizieren von Texten. Wie man am großen "S" schon sieht, ist String eine **Klasse** und kein Ordinaltyp wie char (Ordinaltypen beginnen immer mit einem Kleinbuchstaben). Strings werden sehr häufig benutzt, wenn man Werte aus GUI-Oberflächen auslesen bzw. verändern oder Textverarbeitung betreiben möchte. Darum gibt es für Strings einige Vereinfachungen, um den Umgang mit ihnen zu erleichtern:

- zur Benutzung muss man kein String-Objekt erzeugen (man kann es natürlich trotzdem tun)
- man kann Strings einfach mit einer Zuweisung einer Zeichenkette = **".."** initialisieren
- Strings können einfach mit Hilfe des +-Zeichens aneinandergehängt werden (Konkatenation)
- Die Klasse String muss nicht explizit importiert werden
- Vorsicht: Um zwei Strings auf Gleichheit bzgl. des Inhalts zu überprüfen, kann man **nicht** == benutzen. Mit == wird überprüft, ob die beiden Objekte identisch sind, nicht ob deren Inhalt der gleiche ist. Für den Test auf Inhaltsgleichheit benutzt man deshalb die Methode equals() die von der Klasse String bereitgestellt wird.

ein paar Beispiele:

```
 String s1 = "T-Rex";               // verkürzte Initialisierung
 String s2 = "Dino";
 String s3 = new String( "Dino" );  // s2 ist nun eine Kopie des String "Dino"
 if( !s1.equals(s2) )               // wenn beide Strings ungleich sind
   s1 = s2 +" "+ s1;                // dann mache daraus einen String "Dino
T-Rex"
 if( s2 != s3)                      // Vorsicht, dies ergibt wahr!
   s1 = s2 +" "+ s3;                // hieraus wird nun "Dino Dino"
```

2.58 Unicode-Unterstützung

Die Zeichenketten (Strings) in Java verwenden zur Laufzeit eine UTF-16-Codierung, um die einzelnen Zeichen zu speichern. Das bedeutet, dass normalerweise ein Zeichen 16 Bit belegt. Beim Schreiben und Lesen von Dateien kann eine Konvertierung in verschiedene Zeichensatzcodierungen erfolgen.

2.59 Immutable-Objekte

Zeichenketten (Strings) sind unveränderbare Objekte (immutable). Das bedeutet, daß ein Zeichenkettenobjekt, das einmal erzeugt worden ist, nicht verändert werden kann. Methoden, die Zeichenketten verändern, wie z.B. substring(), erzeugen in Wirklichkeit ein neues Zeichenkettenobjekt. Das gilt auch für ”+” und ”+=”.

Man sollte beachten, daß substring() zwar ein neues Zeichenkettenobjekt erzeugt, aber weiterhin dieselbe interne Struktur für die Zeichen der Zeichenkette referenziert. Wenn man also eine sehr lange Zeichenkette hat, die nur temporär verwendet wird, und dann mit substring() davon eine kurze Zeichenkette gewinnt, die man lange aufbewahrt, bleiben intern die Zeichen der langen Zeichenkette weiterhin gespeichert. Normalerweise ist das erwünscht.

3 Autoren

Edits	User
2	Aggi[1]
4	Albmont[2]
24	Aler[3]
5	Anschmid13[4]
87	Axelf[5]
19	Axum[6]
157	Bastie[7]
1	Benedikt.Seidl[8]
3	Berni[9]
1	Bk1 168[10]
1	Buchfreund~dewikibooks[11]
11	Burghard.w.v.britzke[12]
4	CTC~dewikibooks[13]
1	CarsracBot[14]
1	Cembon[15]
1	Chaquotay~dewikibooks[16]
1	Chrhep[17]
5	ChristianHujer~dewikibooks[18]
1	Collinj[19]
3	Constantin Graf[20]
17	Daniel B[21]

1 https://de.wikibooks.org/wiki/Benutzer:Aggi
2 https://de.wikibooks.org/wiki/Benutzer:Albmont
3 https://de.wikibooks.org/wiki/Benutzer:Aler
4 https://de.wikibooks.org/wiki/Benutzer:Anschmid13
5 https://de.wikibooks.org/wiki/Benutzer:Axelf
6 https://de.wikibooks.org/wiki/Benutzer:Axum
7 https://de.wikibooks.org/wiki/Benutzer:Bastie
8 https://de.wikibooks.org/w/index.php%3ftitle=Benutzer:Benedikt.Seidl&action=edit&redlink=1
9 https://de.wikibooks.org/wiki/Benutzer:Berni
10 https://de.wikibooks.org/wiki/Benutzer:Bk1_168
11 https://de.wikibooks.org/w/index.php%3ftitle=Benutzer:Buchfreund~dewikibooks&action=edit&redlink=1
12 https://de.wikibooks.org/wiki/Benutzer:Burghard.w.v.britzke
13 https://de.wikibooks.org/wiki/Benutzer:CTC~dewikibooks
14 https://de.wikibooks.org/wiki/Benutzer:CarsracBot
15 https://de.wikibooks.org/w/index.php%3ftitle=Benutzer:Cembon&action=edit&redlink=1
16 https://de.wikibooks.org/w/index.php%3ftitle=Benutzer:Chaquotay~dewikibooks&action=edit&redlink=1
17 https://de.wikibooks.org/w/index.php%3ftitle=Benutzer:Chrhep&action=edit&redlink=1
18 https://de.wikibooks.org/w/index.php%3ftitle=Benutzer:ChristianHujer~dewikibooks&action=edit&redlink=1
19 https://de.wikibooks.org/wiki/Benutzer:Collinj
20 https://de.wikibooks.org/wiki/Benutzer:Constantin_Graf
21 https://de.wikibooks.org/wiki/Benutzer:Daniel_B

1 David23x[22]
4 Dexbot[23]
3 DimaS[24]
2 Dirk Huenniger[25]
24 Djamal~dewikibooks[26]
3 Dominik Schacht[27]
5 DrPepper~dewikibooks[28]
1 Drop~dewikibooks[29]
2 E^(nix)[30]
2 Erkan Yilmaz[31]
1 Frettckken[32]
2 Friparvus[33]
1 Ftiercel[34]
1 Galaktos[35]
1 Gettler23[36]
1 Gohnarch[37]
20 Gronau~dewikibooks[38]
1 Hagezussa[39]
1 Hardy42[40]
2 Heartdisease[41]
2 Heinrich~dewikibooks[42]
1 Herr Schroeder[43]
6 Heuler06[44]
1 Holpergeist[45]
28 Intruder[46]

22 https://de.wikibooks.org/wiki/Benutzer:David23x
23 https://de.wikibooks.org/wiki/Benutzer:Dexbot
24 https://de.wikibooks.org/wiki/Benutzer:DimaS
25 https://de.wikibooks.org/wiki/Benutzer:Dirk_Huenniger
26 https://de.wikibooks.org/w/index.php%3ftitle=Benutzer:Djamal~dewikibooks&action=edit&redlink=1
27 https://de.wikibooks.org/w/index.php%3ftitle=Benutzer:Dominik_Schacht&action=edit&redlink=1
28 https://de.wikibooks.org/w/index.php%3ftitle=Benutzer:DrPepper~dewikibooks&action=edit&redlink=1
29 https://de.wikibooks.org/w/index.php%3ftitle=Benutzer:Drop~dewikibooks&action=edit&redlink=1
30 https://de.wikibooks.org/wiki/Benutzer:E%255E(nix)
31 https://de.wikibooks.org/wiki/Benutzer:Erkan_Yilmaz
32 https://de.wikibooks.org/w/index.php%3ftitle=Benutzer:Frettckken&action=edit&redlink=1
33 https://de.wikibooks.org/w/index.php%3ftitle=Benutzer:Friparvus&action=edit&redlink=1
34 https://de.wikibooks.org/w/index.php%3ftitle=Benutzer:Ftiercel&action=edit&redlink=1
35 https://de.wikibooks.org/wiki/Benutzer:Galaktos
36 https://de.wikibooks.org/w/index.php%3ftitle=Benutzer:Gettler23&action=edit&redlink=1
37 https://de.wikibooks.org/wiki/Benutzer:Gohnarch
38 https://de.wikibooks.org/wiki/Benutzer:Gronau~dewikibooks
39 https://de.wikibooks.org/wiki/Benutzer:Hagezussa
40 https://de.wikibooks.org/wiki/Benutzer:Hardy42
41 https://de.wikibooks.org/w/index.php%3ftitle=Benutzer:Heartdisease&action=edit&redlink=1
42 https://de.wikibooks.org/wiki/Benutzer:Heinrich~dewikibooks
43 https://de.wikibooks.org/wiki/Benutzer:Herr_Schroeder
44 https://de.wikibooks.org/wiki/Benutzer:Heuler06
45 https://de.wikibooks.org/w/index.php%3ftitle=Benutzer:Holpergeist&action=edit&redlink=1
46 https://de.wikibooks.org/wiki/Benutzer:Intruder

5 JackPotte[47]
2 Jan~dewikibooks[48]
3 Javasdenn[49]
1 Jjjeykey[50]
82 Juetho[51]
2 Klaus Eifert[52]
1 Kopoltra[53]
1 Kristjan[54]
2 Lamettrie[55]
5 Leo141[56]
87 Lesath~dewikibooks[57]
1 Lucario100[58]
1 Maddog1985[59]
1 Marc Weigand[60]
2 Martin Fuchs[61]
38 MartinThoma[62]
3 Martinh1[63]
1 Matthias Heil[64]
6 MichaelFrey[65]
1 Michaziegler[66]
2 Mik~dewikibooks[67]
1 Miterion[68]
6 Mjchael[69]
1 Mkarg~dewikibooks[70]
2 Morten Haan[71]

47 https://de.wikibooks.org/wiki/Benutzer:JackPotte
48 https://de.wikibooks.org/wiki/Benutzer:Jan~dewikibooks
49 https://de.wikibooks.org/wiki/Benutzer:Javasdenn
50 https://de.wikibooks.org/w/index.php%3ftitle=Benutzer:Jjjeykey&action=edit&redlink=1
51 https://de.wikibooks.org/wiki/Benutzer:Juetho
52 https://de.wikibooks.org/wiki/Benutzer:Klaus_Eifert
53 https://de.wikibooks.org/w/index.php%3ftitle=Benutzer:Kopoltra&action=edit&redlink=1
54 https://de.wikibooks.org/wiki/Benutzer:Kristjan
55 https://de.wikibooks.org/w/index.php%3ftitle=Benutzer:Lamettrie&action=edit&redlink=1
56 https://de.wikibooks.org/wiki/Benutzer:Leo141
57 https://de.wikibooks.org/wiki/Benutzer:Lesath~dewikibooks
58 https://de.wikibooks.org/w/index.php%3ftitle=Benutzer:Lucario100&action=edit&redlink=1
59 https://de.wikibooks.org/w/index.php%3ftitle=Benutzer:Maddog1985&action=edit&redlink=1
60 https://de.wikibooks.org/w/index.php%3ftitle=Benutzer:Marc_Weigand&action=edit&redlink=1
61 https://de.wikibooks.org/w/index.php%3ftitle=Benutzer:Martin_Fuchs&action=edit&redlink=1
62 https://de.wikibooks.org/wiki/Benutzer:MartinThoma
63 https://de.wikibooks.org/w/index.php%3ftitle=Benutzer:Martinh1&action=edit&redlink=1
64 https://de.wikibooks.org/w/index.php%3ftitle=Benutzer:Matthias_Heil&action=edit&redlink=1
65 https://de.wikibooks.org/wiki/Benutzer:MichaelFrey
66 https://de.wikibooks.org/w/index.php%3ftitle=Benutzer:Michaziegler&action=edit&redlink=1
67 https://de.wikibooks.org/wiki/Benutzer:Mik~dewikibooks
68 https://de.wikibooks.org/w/index.php%3ftitle=Benutzer:Miterion&action=edit&redlink=1
69 https://de.wikibooks.org/wiki/Benutzer:Mjchael
70 https://de.wikibooks.org/w/index.php%3ftitle=Benutzer:Mkarg~dewikibooks&action=edit&redlink=1
71 https://de.wikibooks.org/wiki/Benutzer:Morten_Haan

1 MrOerni[72]
1 Netzwerkerin~dewikibooks[73]
1 Nowotoj[74]
27 Outruder[75]
12 Pc-world[76]
1 Phiarc[77]
1 Plaicy[78]
3 Progman[79]
125 Qwertz84[80]
1 Schmidt2[81]
1 Schoebu[82]
2 Shelmton[83]
3 Simdan2[84]
2 Spanky1408[85]
3 Stefan Majewsky[86]
21 Stephan Kulla[87]
1 Supaari[88]
1 Tandar[89]
7 ThePacker[90]
2 Thomy der Genuss[91]
1 Truhgoy[92]
2 Ttbya[93]
3 Ugh~dewikibooks[94]
11 Underdog[95]
12 Wanderinformatiker[96]

72 https://de.wikibooks.org/w/index.php%3ftitle=Benutzer:MrOerni&action=edit&redlink=1
73 https://de.wikibooks.org/w/index.php%3ftitle=Benutzer:Netzwerkerin~dewikibooks&action=edit&redlink=1
74 https://de.wikibooks.org/wiki/Benutzer:Nowotoj
75 https://de.wikibooks.org/wiki/Benutzer:Outruder
76 https://de.wikibooks.org/wiki/Benutzer:Pc-world
77 https://de.wikibooks.org/w/index.php%3ftitle=Benutzer:Phiarc&action=edit&redlink=1
78 https://de.wikibooks.org/wiki/Benutzer:Plaicy
79 https://de.wikibooks.org/wiki/Benutzer:Progman
80 https://de.wikibooks.org/wiki/Benutzer:Qwertz84
81 https://de.wikibooks.org/wiki/Benutzer:Schmidt2
82 https://de.wikibooks.org/w/index.php%3ftitle=Benutzer:Schoebu&action=edit&redlink=1
83 https://de.wikibooks.org/wiki/Benutzer:Shelmton
84 https://de.wikibooks.org/w/index.php%3ftitle=Benutzer:Simdan2&action=edit&redlink=1
85 https://de.wikibooks.org/w/index.php%3ftitle=Benutzer:Spanky1408&action=edit&redlink=1
86 https://de.wikibooks.org/wiki/Benutzer:Stefan_Majewsky
87 https://de.wikibooks.org/wiki/Benutzer:Stephan_Kulla
88 https://de.wikibooks.org/wiki/Benutzer:Supaari
89 https://de.wikibooks.org/wiki/Benutzer:Tandar
90 https://de.wikibooks.org/wiki/Benutzer:ThePacker
91 https://de.wikibooks.org/w/index.php%3ftitle=Benutzer:Thomy_der_Genuss&action=edit&redlink=1
92 https://de.wikibooks.org/w/index.php%3ftitle=Benutzer:Truhgoy&action=edit&redlink=1
93 https://de.wikibooks.org/wiki/Benutzer:Ttbya
94 https://de.wikibooks.org/w/index.php%3ftitle=Benutzer:Ugh~dewikibooks&action=edit&redlink=1
95 https://de.wikibooks.org/wiki/Benutzer:Underdog
96 https://de.wikibooks.org/wiki/Benutzer:Wanderinformatiker

7	WickiLissy[97]
171	WitweBolte[98]
9	Yuuki Mayuki[99]
1	Zero˜dewikibooks[100]
9	Zipper202[101]

97 https://de.wikibooks.org/w/index.php%3ftitle=Benutzer:WickiLissy&action=edit&redlink=1
98 https://de.wikibooks.org/wiki/Benutzer:WitweBolte
99 https://de.wikibooks.org/wiki/Benutzer:Yuuki_Mayuki
100 https://de.wikibooks.org/wiki/Benutzer:Zero˜dewikibooks
101 https://de.wikibooks.org/w/index.php%3ftitle=Benutzer:Zipper202&action=edit&redlink=1

Abbildungsverzeichnis

- GFDL: Gnu Free Documentation License. http://www.gnu.org/licenses/fdl.html
- cc-by-sa-3.0: Creative Commons Attribution ShareAlike 3.0 License. http://creativecommons.org/licenses/by-sa/3.0/
- cc-by-sa-2.5: Creative Commons Attribution ShareAlike 2.5 License. http://creativecommons.org/licenses/by-sa/2.5/
- cc-by-sa-2.0: Creative Commons Attribution ShareAlike 2.0 License. http://creativecommons.org/licenses/by-sa/2.0/
- cc-by-sa-1.0: Creative Commons Attribution ShareAlike 1.0 License. http://creativecommons.org/licenses/by-sa/1.0/
- cc-by-2.0: Creative Commons Attribution 2.0 License. http://creativecommons.org/licenses/by/2.0/
- cc-by-2.0: Creative Commons Attribution 2.0 License. http://creativecommons.org/licenses/by/2.0/deed.en
- cc-by-2.5: Creative Commons Attribution 2.5 License. http://creativecommons.org/licenses/by/2.5/deed.en
- cc-by-3.0: Creative Commons Attribution 3.0 License. http://creativecommons.org/licenses/by/3.0/deed.en
- GPL: GNU General Public License. http://www.gnu.org/licenses/gpl-2.0.txt
- LGPL: GNU Lesser General Public License. http://www.gnu.org/licenses/lgpl.html
- PD: This image is in the public domain.
- ATTR: The copyright holder of this file allows anyone to use it for any purpose, provided that the copyright holder is properly attributed. Redistribution, derivative work, commercial use, and all other use is permitted.
- EURO: This is the common (reverse) face of a euro coin. The copyright on the design of the common face of the euro coins belongs to the European Commission. Authorised is reproduction in a format without relief (drawings, paintings, films) provided they are not detrimental to the image of the euro.
- LFK: Lizenz Freie Kunst. http://artlibre.org/licence/lal/de
- CFR: Copyright free use.

- EPL: Eclipse Public License. `http://www.eclipse.org/org/documents/epl-v10.php`

Copies of the GPL, the LGPL as well as a GFDL are included in chapter Licenses[102]. Please note that images in the public domain do not require attribution. You may click on the image numbers in the following table to open the webpage of the images in your webbrower.

102 Kapitel 4 auf Seite 81

4 Licenses

4.1 GNU GENERAL PUBLIC LICENSE

Version 3, 29 June 2007

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed. Preamble

The GNU General Public License is a free, copyleft license for software and other kinds of works.

The licenses for most software and other practical works are designed to take away your freedom to share and change the works. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change all versions of a program–to make sure it remains free software for all its users. We, the Free Software Foundation, use the GNU General Public License for most of our software; it applies also to any other work released this way by its authors. You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for them if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs, and that you know you can do these things.

To protect your rights, we need to prevent others from denying you these rights or asking you to surrender the rights. Therefore, you have certain responsibilities if you distribute copies of the software, or if you modify it: responsibilities to respect the freedom of others.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must pass on to the recipients the same freedoms that you received. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

Developers that use the GNU GPL protect your rights with two steps: (1) assert copyright on the software, and (2) offer you this License giving you legal permission to copy, distribute and/or modify it.

For the developers' and authors' protection, the GPL clearly explains that there is no warranty for this free software. For both users' and authors' sake, the GPL requires that modified versions be marked as changed, so that their problems will not be attributed erroneously to authors of previous versions.

Some devices are designed to deny users access to install or run modified versions of the software inside them, although the manufacturer can do so. This is fundamentally incompatible with the aim of protecting users' freedom to change the software. The systematic pattern of such abuse occurs in the area of products for individuals to use, which is precisely where it is most unacceptable. Therefore, we have designed this version of the GPL to prohibit the practice for those products. If such problems arise substantially in other domains, we stand ready to extend this provision to those domains in future versions of the GPL, as needed to protect the freedom of users.

Finally, every program is threatened constantly by software patents. States should not allow patents to restrict development and use of software on general-purpose computers, but in those that do, we wish to avoid the special danger that patents applied to a free program could make it effectively proprietary. To prevent this, the GPL assures that patents cannot be used to render the program non-free.

The precise terms and conditions for copying, distribution and modification follow. TERMS AND CONDITIONS 0. Definitions.

"This License" refers to version 3 of the GNU General Public License.

"Copyright" also means copyright-like laws that apply to other kinds of works, such as semiconductor masks.

"The Program" refers to any copyrightable work licensed under this License. Each licensee is addressed as "you". "Licensees" and "recipients" may be individuals or organizations.

To "modify" a work means to copy from or adapt all or part of the work in a fashion requiring copyright permission, other than the making of an exact copy. The resulting work is called a "modified version" of the earlier work or a work "based on" the earlier work.

A "covered work" means either the unmodified Program or a work based on the Program.

To "propagate" a work means to do anything with it that, without permission, would make you directly or secondarily liable for infringement under applicable copyright law, except executing it on a computer or modifying a private copy. Propagation includes copying, distribution (with or without modification), making available to the public, and in some countries other activities as well.

To "convey" a work means any kind of propagation that enables other parties to make or receive copies. Mere interaction with a user through a computer network, with no transfer of a copy, is not conveying.

An interactive user interface displays "Appropriate Legal Notices" to the extent that it includes a convenient and prominently visible feature that (1) displays an appropriate copyright notice, and (2) tells the user that there is no warranty for the work (except to the extent that warranties are provided), that licensees may convey the work under this License, and how to view a copy of this License. If the interface presents a list of user commands or options, such as a menu, a prominent item in the list meets this criterion. 1. Source Code.

The "source code" for a work means the preferred form of the work for making modifications to it. "Object code" means any non-source form of a work.

A "Standard Interface" means an interface that either is an official standard defined by a recognized standards body, or, in the case of interfaces specified for a particular programming language, one that is widely used among developers working in that language.

The "System Libraries" of an executable work include anything, other than the work as a whole, that (a) is included in the normal form of packaging a Major Component, but which is not part of that Major Component, and (b) serves only to enable use of the work with that Major Component, or to implement a Standard Interface for which an implementation is available to the public in source code form. A "Major Component", in this context, means a major essential component (kernel, window system, and so on) of the specific operating system (if any) on which the executable work runs, or a compiler used to produce the work, or an object code interpreter used to run it.

The "Corresponding Source" for a work in object code form means all the source code needed to generate, install, and (for an executable work) run the object code and to modify the work, including scripts to control those activities. However, it does not include the work's System Libraries, or general-purpose tools or generally available free programs which are used unmodified in performing those activities but which are not part of the work. For example, Corresponding Source includes interface definition files associated with source files for the work, and the source code for shared libraries and dynamically linked subprograms that the work is specifically designed to require, such as by intimate data communication or control flow between those subprograms and other parts of the work.

The Corresponding Source need not include anything that users can regenerate automatically from other parts of the Corresponding Source.

The Corresponding Source for a work in source code form is that same work. 2. Basic Permissions.

All rights granted under this License are granted for the term of copyright on the Program, and are irrevocable provided the stated conditions are met. This License explicitly affirms your unlimited permission to run the unmodified Program. The output from running a covered work is covered by this License only if the output, given its content, constitutes a covered work. This License acknowledges your rights of fair use or other equivalent, as provided by copyright law.

You may make, run and propagate covered works that you do not convey, without conditions so long as your license otherwise remains in force. You may convey covered works to others for the sole purpose of having them make modifications exclusively for you, or provide you with facilities for running those works, provided that you comply with the terms of this License in conveying all material for which you do not control copyright. Those thus making or running the covered works for you must do so exclusively on your behalf, under your direction and control, on terms that prohibit them from making any copies of your copyrighted material outside their relationship with you.

Conveying under any other circumstances is permitted solely under the conditions stated below. Sublicensing is not allowed; section 10 makes it unnecessary. 3. Protecting Users' Legal Rights From Anti-Circumvention Law.

No covered work shall be deemed part of an effective technological measure under any applicable law fulfilling obligations under article 11 of the WIPO copyright treaty adopted on 20 December 1996, or similar laws prohibiting or restricting circumvention of such measures.

When you convey a covered work, you waive any legal power to forbid circumvention of technological measures to the extent such circumvention is effected by exercising rights under this License with respect to the covered work, and you disclaim any intention to limit operation or modification of the work as a means of enforcing, against the work's users, your or third parties' legal rights to forbid circumvention of technological measures. 4. Conveying Verbatim Copies.

You may convey verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice; keep intact all notices stating that this License and any non-permissive terms added in accord with section 7 apply to the code; keep intact all notices of the absence of any warranty; and give all recipients a copy of this License along with the Program.

You may charge any price or no price for each copy that you convey, and you may offer support or warranty protection for a fee. 5. Conveying Modified Source Versions.

You may convey a work based on the Program, or the modifications to produce it from the Program, in the form of source code under the terms of section 4, provided that you also meet all of these conditions:

* a) The work must carry prominent notices stating that you modified it, and giving a relevant date. * b) The work must carry prominent notices stating that it is released under this License and any conditions added under section 7. This requirement modifies the requirement in section 4 to "keep intact all notices". * c) You must license the entire work, as a whole, under this License to anyone who comes into possession of a copy. This License will therefore apply, along with any applicable section 7 additional terms, to the whole of the work, and all its parts, regardless of how they are packaged. This License gives no permission to license the work in any other way, but it does not invalidate such permission if you have separately received it. * d) If the work has interactive user interfaces, each must display Appropriate Legal Notices; however, if the Program has interactive interfaces that do not display Appropriate Legal Notices, your work need not make them do so.

A compilation of a covered work with other separate and independent works, which are not by their nature extensions of the covered work, and which are not combined with it such as to form a larger program, in or on a volume of a storage or distribution medium, is called an "aggregate" if the compilation and its resulting copyright are not used to limit the access or legal rights of the compilation's users beyond what the individual works permit. Inclusion of a covered work in an aggregate does not cause this License to apply to the other parts of the aggregate. 6. Conveying Non-Source Forms.

You may convey a covered work in object code form under the terms of sections 4 and 5, provided that you also convey the machine-readable Corresponding Source under the terms of this License, in one of these ways:

* a) Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by the Corresponding Source fixed on a durable physical medium customarily used for software interchange. * b) Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by a written offer, valid for at least three years and valid for as long as you offer spare parts or customer support for that product model, to give anyone who possesses the object code either (1) a copy of the Corresponding Source for all the software in the product that is covered by this License, on a durable physical medium customarily used for software interchange, for a price no more than your reasonable cost of physically performing this conveying of source, or (2) access to copy the Corresponding Source from a network server at no charge. * c) Convey individual copies of the object code with a copy of the written offer to provide the Corresponding Source. This alternative is allowed only occasionally and noncommercially, and only if you received the object code with such an offer, in accord with subsection 6b. * d) Convey the object code by offering access from a designated place (gratis or for a charge), and offer equivalent access to the Corresponding Source in the same way through the same place at no further charge. You need not require recipients to copy the Corresponding Source along with the object code. If the place to copy the object code is a network server, the Corresponding Source may be on a different server (operated by you or a third party) that supports equivalent copying facilities, provided you maintain clear directions next to the object code saying where to find the Corresponding Source. Regardless of what server hosts the Corresponding Source, you remain obligated to ensure that it is available for as long as needed to satisfy these requirements. * e) Convey the object code using peer-to-peer transmission, provided you inform other peers where the object code and Corresponding Source of the work are being offered to the general public at no charge under subsection 6d.

A separable portion of the object code, whose source code is excluded from the Corresponding Source as a System Library, need not be included in conveying the object code work.

A "User Product" is either (1) a "consumer product", which means any tangible personal property which is normally used for personal, family, or household purposes, or (2) anything designed or sold for incorporation into a dwelling. In determining whether a product is a consumer product, doubtful cases shall be resolved in favor of coverage. For a particular product received by a particular user, "normally used" refers to a typical or common use of that class of product, regardless of the status of the particular user or of the way in which the particular user actually uses, or expects or is expected to use, the product. A product is a consumer product regardless of whether the product has substantial commercial, industrial or non-consumer uses, unless such uses represent the only significant mode of use of the product.

"Installation Information" for a User Product means any methods, procedures, authorization keys, or other information required to install and execute modified versions of a covered work in that User Product from a modified version of its Corresponding Source. The information must suffice to ensure that the continued functioning of the modified object code is in no case prevented or interfered with solely because modification has been made.

If you convey an object code work under this section in, or with, or specifically for use in, a User Product, and the conveying occurs as part of a transaction in which the right of possession and use of the User Product is transferred to the recipient in perpetuity or for a fixed term (regardless of how the transaction is characterized), the Corresponding Source conveyed under this section must be accompanied by the Installation Information. But this requirement does not apply if neither you nor any third party retains the ability to install modified object code on the User Product (for example, the work has been installed in ROM).

The requirement to provide Installation Information does not include a requirement to continue to provide support service, warranty, or updates for a work that has been modified or installed by the recipient, or for the User Product in which it has been modified or installed. Access to a network may be denied when the modification itself materially and adversely affects the operation of the network or violates the rules and protocols for communication across the network.

Corresponding Source conveyed, and Installation Information provided, in accord with this section must be in a format that is publicly documented (and with an implementation available to the public in source code form), and must require no special password or key for unpacking, reading or copying. 7. Additional Terms.

"Additional permissions" are terms that supplement the terms of this License by making exceptions from one or more of its conditions. Additional permissions that are applicable to the entire Program shall be treated as though they were included in this License, to the extent that they are valid under applicable law. If additional permissions apply only to part of the Program, that part may be used separately under those permissions, but the entire Program remains governed by this License without regard to the additional permissions.

When you convey a copy of a covered work, you may at your option remove any additional permissions from that copy, or from any part of it. (Additional permissions may be written to require their own removal in certain cases when you modify the work.) You may place additional permissions on material, added by you to a covered work, for which you have or can give appropriate copyright permission.

Notwithstanding any other provision of this License, for material you add to a covered work, you may (if authorized by the copyright holders of that material) supplement the terms of this License with terms:

* a) Disclaiming warranty or limiting liability differently from the terms of sections 15 and 16 of this License; or * b) Requiring preservation of specified reasonable legal notices or author attributions in that material or in the Appropriate Legal Notices displayed by works containing it; or * c) Prohibiting misrepresentation of the origin of that material, or requiring that modified versions of such material be marked in reasonable ways as different from the original version; or * d) Limiting the use for publicity purposes of names of licensors or authors of the material; or * e) Declining to grant rights under trademark law for use of some trade names, trademarks, or service marks; or * f) Requiring indemnification of licensors and authors of that material by anyone who conveys the material (or modified versions of it) with contractual assumptions of liability to the recipient, for any liability that these contractual assumptions directly impose on those licensors and authors.

All other non-permissive additional terms are considered "further restrictions" within the meaning of section 10. If the Program as you received it, or any part of it, contains a notice stating that it is governed by this License along with a term that is a further restriction, you may remove that term. If a license document contains a further restriction but permits relicensing or conveying under this License, you may add to a covered work material governed by the terms of that license document, provided that the further restriction does not survive such relicensing or conveying.

If you add terms to a covered work in accord with this section, you must place, in the relevant source files, a statement of the additional terms that apply to those files, or a notice indicating where to find the applicable terms.

Additional terms, permissive or non-permissive, may be stated in the form of a separately written license, or stated as exceptions; the above requirements apply either way. 8. Termination.

You may not propagate or modify a covered work except as expressly provided under this License. Any attempt otherwise to propagate or modify it is void, and will automatically terminate your rights under this License (including any patent licenses granted under the third paragraph of section 11).

However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, you do not qualify to receive new licenses for the same material under section 10. 9. Acceptance Not Required for Having Copies.

You are not required to accept this License in order to receive or run a copy of the Program. Ancillary propagation of a covered work occurring solely as a consequence of using peer-to-peer transmission to receive a copy likewise does not require acceptance. However, nothing other than this License grants you permission to propagate or modify any covered work. These actions infringe copyright if you do not accept this License. Therefore, by modifying or propagating a covered work, you indicate your acceptance of this License to do so. 10. Automatic Licensing of Downstream Recipients.

Each time you convey a covered work, the recipient automatically receives a license from the original licensors, to run, modify and propagate that work, subject to this License. You are not responsible for enforcing compliance by third parties with this License.

An "entity transaction" is a transaction transferring control of an organization, or substantially all assets of one, or subdividing an organization, or merging organizations. If propagation of a covered work results from an entity transaction, each party to that transaction who receives a copy of the work also receives whatever licenses to the work the party's predecessor in interest had or could give under the previous paragraph, plus a right to possession of the Corresponding Source of the work from the predecessor in interest, if the predecessor has it or can get it with reasonable efforts.

You may not impose any further restrictions on the exercise of the rights granted or affirmed under this License. For example, you may not impose a license fee, royalty, or other charge for exercise of rights granted under this License, and you may not initiate litigation (including a cross-claim or counterclaim in a lawsuit) alleging that any patent claim is infringed by making, using, selling, offering for sale, or importing the Program or any portion of it. 11. Patents.

A "contributor" is a copyright holder who authorizes use under this License of the Program or a work on which the Program is based. The work thus licensed is called the contributor's "contributor version".

A contributor's "essential patent claims" are all patent claims owned or controlled by the contributor, whether already acquired or hereafter acquired, that would be infringed by some manner, permitted by this License, of making, using, or selling its contributor version, but do not include claims that would be infringed only as a consequence of further modification of the contributor version. For purposes of this definition, "control" includes the right to grant patent sublicenses in a manner consistent with the requirements of this License.

Each contributor grants you a non-exclusive, worldwide, royalty-free patent license under the contributor's essential patent claims, to make, use, sell, offer for sale, import and otherwise run, modify and propagate the contents of its contributor version.

In the following three paragraphs, a "patent license" is any express agreement or commitment, however denominated, not to enforce a patent (such as an express permission to practice a patent or covenant not to sue for patent infringement). To "grant" such a patent license to a party means to make such an agreement or commitment not to enforce a patent against the party.

If you convey a covered work, knowingly relying on a patent license, and the Corresponding Source of the work is not available for anyone to copy, free of charge and under the terms of this License, through a publicly available network server or other readily accessible means, then you must either (1) cause the Corresponding Source to be so available, or (2) arrange to deprive yourself of the benefit of the patent license for this particular work, or (3) arrange, in a manner consistent with the requirements of this License, to extend the patent license to downstream recipients. "Knowingly relying" means you have actual knowledge that, but for the patent license, your conveying the covered work in a country, or your recipient's use of the covered work in a country, would infringe one or more identifiable patents in that country that you have reason to believe are valid.

If, pursuant to or in connection with a single transaction or arrangement, you convey, or propagate by procuring conveyance of, a covered work, and grant a patent license to some of the parties receiving the covered work authorizing them to use, propagate, modify or convey a specific copy of the covered work, then the patent license you grant is automatically extended to all recipients of the covered work and works based on it.

A patent license is "discriminatory" if it does not include within the scope of its coverage, prohibits the exercise of, or is conditioned on the non-exercise of one or more of the rights that are specifically granted under this License. You may not convey a covered work if you are a party to an arrangement with a third party that is in the business of distributing software, under which you make payment to the third party based on the extent of your activity of conveying the work, and under which the third party grants, to any of the parties who would receive the covered work from you, a discriminatory patent license (a) in connection with copies of the covered work conveyed by you (or copies made from those copies), or (b) primarily for and in connection with specific products or compilations that contain the covered work, unless you entered into that arrangement, or that patent license was granted, prior to 28 March 2007.

Nothing in this License shall be construed as excluding or limiting any implied license or other defenses to infringement that may otherwise be available to you under applicable patent law. 12. No Surrender of Others' Freedom.

If conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot convey a covered work so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not convey it at all. For example, if you agree to terms that obligate you to collect a royalty for further conveying from those to whom you convey the Program, the only way you could satisfy both those terms and this License would be to refrain entirely from

conveying the Program. 13. Use with the GNU Affero General Public License.

Notwithstanding any other provision of this License, you have permission to link or combine any covered work with a work licensed under version 3 of the GNU Affero General Public License into a single combined work, and to convey the resulting work. The terms of this License will continue to apply to the part which is the covered work, but the special requirements of the GNU Affero General Public License, section 13, concerning interaction through a network will apply to the combination as such. 14. Revised Versions of this License.

The Free Software Foundation may publish revised and/or new versions of the GNU General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies that a certain numbered version of the GNU General Public License "or any later version" applies to it, you have the option of following the terms and conditions either of that numbered version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of the GNU General Public License, you may choose any version ever published by the Free Software Foundation.

If the Program specifies that a proxy can decide which future versions of the GNU General Public License can be used, that proxy's public statement of acceptance of a version permanently authorizes you to choose that version for the Program.

Later license versions may give you additional or different permissions. However, no additional obligations are imposed on any author or copyright holder as a result of your choosing to follow a later version. 15. Disclaimer of Warranty.

THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION. 16. Limitation of Liability.

IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MODIFIES AND/OR CONVEYS THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. 17. Interpretation of Sections 15 and 16.

If the disclaimer of warranty and limitation of liability provided above cannot be given local legal effect according to their terms, reviewing courts shall apply local law that most closely approximates an absolute waiver of all civil liability in connection with the Program, unless a warranty or assumption of liability accompanies a copy of the Program in return for a fee.

END OF TERMS AND CONDITIONS How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively state the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

<one line to give the program's name and a brief idea of what it does.> Copyright (C) <year> <name of author>

This program is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program. If not, see <http://www.gnu.org/licenses/>.

Also add information on how to contact you by electronic and paper mail.

If the program does terminal interaction, make it output a short notice like this when it starts in an interactive mode:

<program> Copyright (C) <year> <name of author> This program comes with ABSOLUTELY NO WARRANTY; for details type 'show w'. This is free software, and you are welcome to redistribute it under certain conditions; type 'show c' for details.

The hypothetical commands 'show w' and 'show c' should show the appropriate parts of the General Public License. Of course, your program's commands might be different; for a GUI interface, you would use an "about box".

You should also get your employer (if you work as a programmer) or school, if any, to sign a "copyright disclaimer" for the program, if necessary. For more information on this, and how to apply and follow the GNU GPL, see <http://www.gnu.org/licenses/>.

The GNU General Public License does not permit incorporating your program into proprietary programs. If your program is a subroutine library, you may consider it more useful to permit linking proprietary applications with the library. If this is what you want to do, use the GNU Lesser General Public License instead of this License. But first, please read <http://www.gnu.org/philosophy/why-not-lgpl.html>.

4.2 GNU Free Documentation License

Version 1.3, 3 November 2008

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed. 0. PREAMBLE

The purpose of this License is to make a manual, textbook, or other functional and useful document "free"in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference. 1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you". You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A "Modified Versionöf the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A SSecondary Sectionïs a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The Ïnvariant Sectionsäre certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The "Cover Textsäre certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A "Transparent"copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not "Transparentïs called Öpaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The "Title Page"means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page"means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

The "publisher"means any person or entity that distributes copies of the Document to the public.

A section Ëntitled XYZ"means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as Äcknowledgements", "Dedications", Ëndorsements", or "History".) To "Preserve the Titleöf such a section when you modify the Document means that it remains a section Ëntitled XYZäccording to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License. 2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies. 3. COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document. 4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

* A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission. * B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement. * C. State on the Title page the name of the publisher of the Modified Version, as the publisher. * D. Preserve all the copyright notices of the Document. * E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices. * F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below. * G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice. * H. Include an unaltered copy of this License. * I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "Historyïn the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence. * J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History§ection. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission. * K. For any section Entitled Äcknowledgementsör "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein. * L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles. * M. Delete any section Entitled Ëndorsements". Such a section may not be included in the Modified Version. * N. Do not retitle any existing section to be Entitled Ëndorsementsör to conflict in title with any Invariant Section. * O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled Ëndorsements", provided it contains nothing but endorsements of your Modified Version by various parties—for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version. 5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled "Historyïn the various original documents, forming one section Entitled "History"; likewise combine any sections Entitled Äcknowledgements", and any sections Entitled "Dedications". You must delete all sections Entitled Ëndorsements". 6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document. 7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an ägggregateïf the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate. 8. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

If a section in the Document is Entitled Äcknowledgements", "Dedications", or "History", the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title. 9. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, or distribute it is void, and will automatically terminate your rights under this License.

However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, receipt of a copy of some or all of the same material does not give you any rights to use it. 10. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See http://www.gnu.org/copyleft/.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License ör any later versionäpplies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation. If the Document specifies that a proxy can decide which future versions of this License can be used, that proxy's public statement of acceptance of a version permanently authorizes you to choose that version for the Document. 11. RELICENSING

"Massive Multiauthor Collaboration Site"(or "MMC Site") means any World Wide Web server that publishes copyrightable works and also provides prominent facilities for anybody to edit those works. A public wiki that anybody can edit is an example of such a server. A "Massive Multiauthor Collaboration"(or "MMC") contained in the site means any set of copyrightable works thus published on the MMC site.

"CC-BY-SA"means the Creative Commons Attribution-Share Alike 3.0 license published by Creative Commons Corporation, a not-for-profit corporation with a principal place of business in San Francisco, California, as well as future copyleft versions of that license published by that same organization.

Ïncorporate"means to publish or republish a Document, in whole or in part, as part of another Document.

An MMC is ëligible for relicensingïf it is licensed under this License, and if all works that were first published under this License somewhere other than this MMC, and subsequently incorporated in whole or in part into the MMC, (1) had no cover texts or invariant sections, and (2) were thus incorporated prior to November 1, 2008.

The operator of an MMC Site may republish an MMC contained in the site under CC-BY-SA on the same site at any time before August 1, 2009, provided the MMC is eligible for relicensing. ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

Copyright (C) YEAR YOUR NAME. Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.3 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled "GNU Free Documentation License".

If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the "with ... Texts."line with this:

with the Invariant Sections being LIST THEIR TITLES, with the Front-Cover Texts being LIST, and with the Back-Cover Texts being LIST.

If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.

4.3 GNU Lesser General Public License

GNU LESSER GENERAL PUBLIC LICENSE

Version 3, 29 June 2007

Copyright © 2007 Free Software Foundation, Inc. <http://fsf.org/>

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

This version of the GNU Lesser General Public License incorporates the terms and conditions of version 3 of the GNU General Public License, supplemented by the additional permissions listed below. 0. Additional Definitions.

As used herein, “this License” refers to version 3 of the GNU Lesser General Public License, and the “GNU GPL” refers to version 3 of the GNU General Public License.

“The Library” refers to a covered work governed by this License, other than an Application or a Combined Work as defined below.

An “Application” is any work that makes use of an interface provided by the Library, but which is not otherwise based on the Library. Defining a subclass of a class defined by the Library is deemed a mode of using an interface provided by the Library.

A “Combined Work” is a work produced by combining or linking an Application with the Library. The particular version of the Library with which the Combined Work was made is also called the “Linked Version”.

The “Minimal Corresponding Source” for a Combined Work means the Corresponding Source for the Combined Work, excluding any source code for portions of the Combined Work that, considered in isolation, are based on the Application, and not on the Linked Version.

The “Corresponding Application Code” for a Combined Work means the object code and/or source code for the Application, including any data and utility programs needed for reproducing the Combined Work from the Application, but excluding the System Libraries of the Combined Work. 1. Exception to Section 3 of the GNU GPL.

You may convey a covered work under sections 3 and 4 of this License without being bound by section 3 of the GNU GPL. 2. Conveying Modified Versions.

If you modify a copy of the Library, and, in your modifications, a facility refers to a function or data to be supplied by an Application that uses the facility (other than as an argument passed when the facility is invoked), then you may convey a copy of the modified version:

* a) under this License, provided that you make a good faith effort to ensure that, in the event an Application does not supply the function or data, the facility still operates, and performs whatever part of its purpose remains meaningful, or * b) under the GNU GPL, with none of the additional permissions of this License applicable to that copy.

3. Object Code Incorporating Material from Library Header Files.

The object code form of an Application may incorporate material from a header file that is part of the Library. You may convey such object code under terms of your choice, provided that, if the incorporated material is not limited to numerical parameters, data structure layouts and accessors, or small macros, inline functions and templates (ten or fewer lines in length), you do both of the following:

* a) Give prominent notice with each copy of the object code that the Library is used in it and that the Library and its use are covered by this License. * b) Accompany the object code with a copy of the GNU GPL and this license document.

4. Combined Works.

You may convey a Combined Work under terms of your choice that, taken together, effectively do not restrict modification of the portions of the Library contained in the Combined Work and reverse engineering for debugging such modifications, if you also do each of the following:

* a) Give prominent notice with each copy of the Combined Work that the Library is used in it and that the Library and its use are covered by this License. * b) Accompany the Combined Work with a copy of the GNU GPL and this license document. * c) For a Combined Work that displays copyright notices during execution, include the copyright notice for the Library among these notices, as well as a reference directing the user to the copies of the GNU GPL and this license document. * d) Do one of the following: o 0) Convey the Minimal Corresponding Source under the terms of this License, and the Corresponding Application Code in a form suitable for, and under terms that permit, the user to recombine or relink the Application with a modified version of the Linked Version to produce a modified Combined Work, in the manner specified by section 6 of the GNU GPL for conveying Corresponding Source. o 1) Use a suitable shared library mechanism for linking with the Library. A suitable mechanism is one that (a) uses at run time a copy of the Library already present on the user’s computer system, and (b) will operate properly with a modified version of the Library that is interface-compatible with the Linked Version. * e) Provide Installation Information, but only if you would otherwise be required to provide such information under section 6 of the GNU GPL, and only to the extent that such information is necessary to install and execute a modified version of the Combined Work produced by recombining or relinking the Application with a modified version of the Linked Version. (If you use option 4d0, the Installation Information must accompany the Minimal Corresponding Source and Corresponding Application Code. If you use option 4d1, you must provide the Installation Information in the manner specified by section 6 of the GNU GPL for conveying Corresponding Source.)

5. Combined Libraries.

You may place library facilities that are a work based on the Library side by side in a single library together with other library facilities that are not Applications and are not covered by this License, and convey such a combined library under terms of your choice, if you do both of the following:

* a) Accompany the combined library with a copy of the same work based on the Library, uncombined with any other library facilities, conveyed under the terms of this License. * b) Give prominent notice with the combined library that part of it is a work based on the Library, and explaining where to find the accompanying uncombined form of the same work.

6. Revised Versions of the GNU Lesser General Public License.

The Free Software Foundation may publish revised and/or new versions of the GNU Lesser General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Library as you received it specifies that a certain numbered version of the GNU Lesser General Public License “or any later version” applies to it, you have the option of following the terms and conditions either of that published version or of any later version published by the Free Software Foundation. If the Library as you received it does not specify a version number of the GNU Lesser General Public License, you may choose any version of the GNU Lesser General Public License ever published by the Free Software Foundation.

If the Library as you received it specifies that a proxy can decide whether future versions of the GNU Lesser General Public License shall apply, that proxy’s public statement of acceptance of any version is permanent authorization for you to choose that version for the Library.