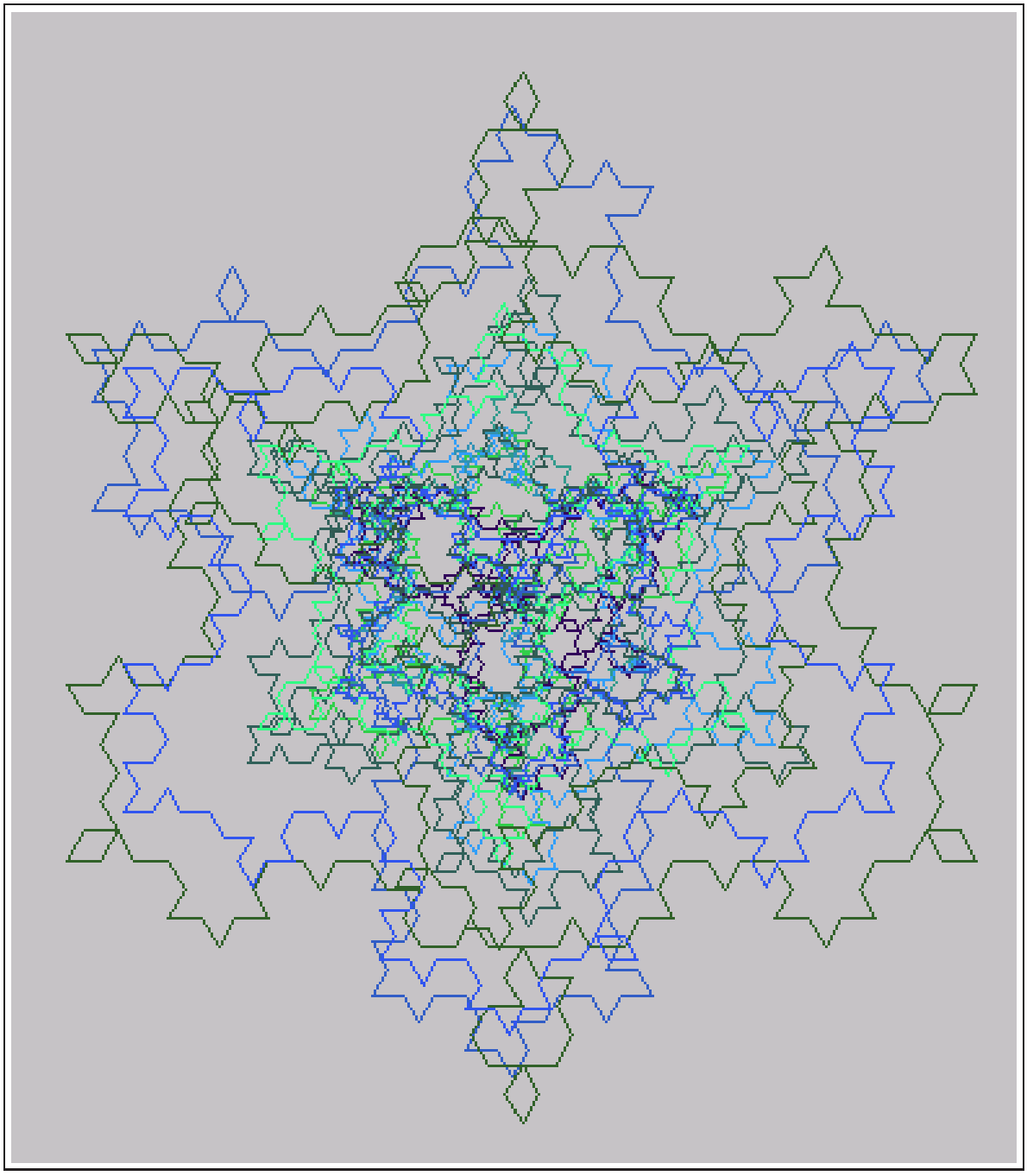
Faszination Programmierung

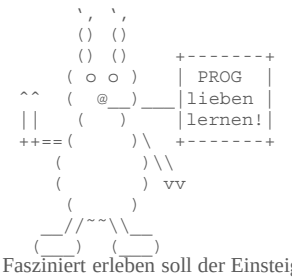
Freude mit Graphik und vielfältigen Konstrukten

Hinrich E. G. Bonin[1]

[1]Prof. Dr. rer. publ. Dipl.-Ing. Dipl.-Wirtsch.-Ing. Hinrich E. G. **Bonin** lehrte bis Ende März 2010 „Informatik in der Öffentlichen Verwaltung“ an der Leuphana Universität Lüneburg, Institut für Wirtschaftsinformatik (IWI), Email: Hinrich@hegb.de, Adresse: An der Eulenburg 6, D-21391 Reppenstedt, Germany.



Zusammenfassung



Wie in vielen Hochschulen — so auch in der Leuphana Universität Lüneburg — basiert die Einführung in die *Grundlagen der Programmierung* (PROG) für Informatiker und Wirtschaftsinformatiker primär auf der Programmiersprache *Java*^{TM} von *Sun Microsystems*, Inc. USA. Gemeinsames Ziel ist es, programmieren als einen systematischen Konstruktionsvorgang zu vermitteln.

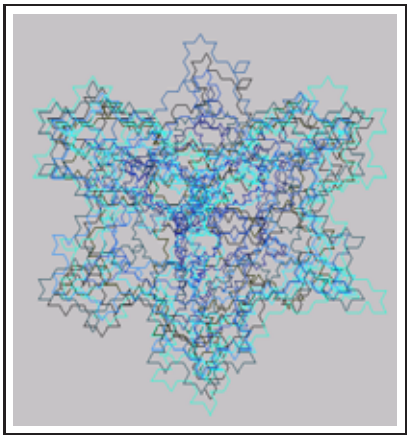
Fasziniert erleben soll der Einsteiger die konkrete Entwicklung von Algorithmen und Datenstrukturen als ein diszipliniertes Vorgehen :-). Zum Einordnen und Verstehen des *Objekt-Orientierte Paradigmas* (Denkmodells) wird die imperativ-orientierte Programmierung in PostScript^{TM} skizziert.

Das Fach *Programmierung* verlangt vom Einsteiger auf seinem Weg zum Versteher viel Geduld und Ausdauer :-). Der harte Weg läßt sich erfolgreich durchlaufen, wenn programmieren viel Freude macht und Faszinaion vermittelt. Deshalb sind die Beispiele in Faszination Programmierung aus dem Bereich Graphik gewählt worden.

Klar ist, das programmieren erlernen Sie zunächst am besten durch „abkupfern“ und spielen mit gelungenen Programmen. Klar ist aber auch, gute eigene Programme setzen ein tiefes Verständnis der tragfähigen Modelle und Konzepte der jeweiligen Programmiersprache voraus. Faszination Programmierung vermittelt daher das *imperative* und das *objekt-orientierte* Denkmodell (OO-Paradigma).

Faszination Programmierung[1] versucht ein solches Verständnis Schritt für Schritt aufzubauen. Bei den Beispielen, Übungen und Musterlösungen geht es primär um ein Begreifen und Umgehen mit der Komplexität, die einem Programm innewohnt. Plakativ formuliert möchte Faszination Programmierung Ihnen helfen Java^{TM} als ein Akronym für *Just a valid application* zu verstehen ;-).

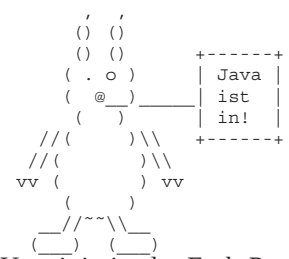
[1] Hinweis: Dieses Dokument ist wird fortgeschrieben. Die aktuelle Version befindet sich unter http://www.hegb.de. Anmerkungen und Kommentare schicken Sie bitte an: Hinrich@hegb.de



Vorwort

Kunst ist nicht nur die statische Errungenschaft
der Meister der Vergangenheit.
Kunst ist die kreative Dynamik-Qualität
der Künstler von heute.[2]
Robert M. Pirsig ↪ [Glover03] p. vii

Sie wollen und/oder müssen sich mit der Programmierung befassen? Das ist gut so! Schreiben von Programmen kann viel Freude machen. Welche Programmiersprache Sie dazu am besten nutzen sollten hängt weitgehend vom jeweiligen Zeitgeist ab :-). Um 1980 war es *List Processing* (LISP) (↪ [Bonin91b]). Derzeit ist es Java™. Ob nun Java™ noch im Jahre 2020 relevant ist oder sich in einer Rolle wie heute LISP befindet, ist unerheblich. Es geht um die Frage wie kann effektiv ein Begreifen der Java™ innewohnenden Modelle und Konzepte ermöglicht werden.



Ziel ist es, ein Spektrum von Möglichkeiten der Notation von Quellcode aufzuzeigen. Deshalb wird neben Java auch die Programmierung in *PostScript* beispielhaft skizziert.

Unstrittig ist das Fach *Programmierung* ein Kern der großen Disziplin Informatik. Es verlangt vom Einsteiger auf seinem Weg zum Versteher viel Geduld und Ausdauer :-). Erwerben muss er fundiertes Wissen über Schwerpunktthemen wie zum Beispiel:

[2] „*Art is not just the static achievements of the masters of the past. Art is the creative Dynamic Quality of the artist of the present.*“

- Syntax und Semantik elementarer Konstrukte (Sequenz, Alternative, Iteration, Rekursion),
- Repräsentation von Daten (Klassenkonstrukt, Variablen, Datentyp),
- Kommunikation von Objekten (Methoden, Signaturen),
- Erzeugung von Objekten (Vererbung, Interface),
- Handhabung von Klassenmengen (Pakete, Archive, Zugriffsrechte),
- Gebräuchliche Konstruktionen (Pattern) und
- allgemeine Konstruktionsempfehlungen (Bezeichner, Dokumentationsregeln)

Dieser harte Weg läßt sich nur dann erfolgreich durchlaufen, wenn programmieren letztlich Freude macht und Faszinaion vermittelt. Deshalb sind die Beispiele in Faszination Programmierung primär aus dem Bereich Graphik gewählt worden.

Faszination Programmierung wendet sich an alle, die mittels schöner Bilder die Lust nicht verlieren wollen, damit sie die harte Arbeit des Begreifens von Programmen meistern. Dabei spielt Ihr Alter keine Rolle, denn nach den neueren Erkenntnissen der Hirnforschung verfügt das Hirn über eine hohe Plastizität und die Fähigkeit der Neurogenese, bei der neue Nervenzellen in bestehende Verschaltungen eingefügt werden. Dank dieser Hirneigenschaften *kann Hans also durchaus noch lernen, was Hänschen nicht gelernt hat* :-) — auch wenn es mit den Jahren deutlich schwerer fällt.

Der Anfänger muss viele Konstrukte erlernen und bewährte Konstruktionen nachbauen. Ebensowenig wie zum Beispiel Autofahren allein aus Büchern erlernbar ist, wird niemand zum „Progamming-Wizard“ (deutsch: Hexenmeister) durch das Nachlesen von erläuterten Beispielen. Das intensive Durchdenken der Beispiele, im Dialog mit einer passenden Entwicklungsumgebung, vermittelt jedoch im Sinne der gezogenen Analogie, unstrittig die Kenntnis der Straßenverkehrsordnung und ermöglicht ein erfolgreiches Teilnehmen am Verkehrsgeschehen — auch im Großstadtverkehr. Für diesen Lernprozeß wünsche ich Ihnen, der „Arbeiterin“ oder dem „Arbeiter“, viel Freude.

Faszination Programmierung ist konzipiert als ein Buch zum Selbststudium und für Lehrveranstaltungen. Mit den umfassenden Quellenangaben und vielen Vor- und Rückwärtsverweisen, dient es auch als Nachschlagewerk und Informationslieferant für Spezialfragen. Jedoch ist es kein umfassendes Java-Kompendium.

Faszination Programmierung wurde im Jahre 2005 begonnen und zwar nachdem der JAVA^{TM}–COACH ($\hookrightarrow$ [Bonin04b]) — ein mehr als 500 Seiten starkes Buch — weitgehend fertiggestellt und ein erstes Manuskript über Kunst und Programmierung enstanden war. Während der Fortschreibung eines Buches, lernt man erfreulicherweise stets dazu. Das hat jedoch auch den Nachteil, daß man laufend neue Unzulänglichkeiten erkennt. Schließlich ist es, trotz solcher Schwächen, der Öffentlichkeit zu übergeben. Ich bitte Sie daher im voraus um Verständnis für Unzulänglichkeiten. Willkommen sind Ihre konstruktiven Vorschläge, um die Unzulänglichkeiten Schritt für Schritt weiter zu verringern.

`:-)`	Your basic smiley. This smiley is used to inflect a sarcastic or joking statement since we can't hear voice inflection over e-mail.
`;-)`	Winky smiley. User just made a flirtatious and/or sarcastic remark. More of a ¨don't hit me for what I just said¨ smiley.
`:-(`	Frowning smiley. User did not like that last statement or is upset or depressed about something.
`:-I`	Indifferent smiley. Better than a `:-(` but not quite as good as a `:-)`.
`:->`	User just made a really biting sarcastic remark. Worse than a `;-)`.
`>:->`	User just made a really devilish remark.
`>;->`	Winky and devil combined. A very lewd remark was just made.

Legende:
Quelle ↪ `http://members.aol.com/bearpage/smileys.htm` (online 21-Nov-2003)

Tabelle 1: Internet Smileys

Notation

Englische Fachbegriffe

In Faszination Programmierung wird erst gar nicht der Versuch unternommen, die weltweit übliche Informatik-Fachsprache Englisch zu übersetzen. Es ist daher teilweise „mischsprachig“: Deutsch und Englisch. Aus Lesbarkeitsgründen sind nur die männlichen Formulierungen genannt; die Leserinnen seien implizit berücksichtigt. So steht das Wort „Programmierer“ hier für Programmiererin und Programmierer.

Für die Notation der Modelle und Algorithmen werden auch im Text PostScript^{TM} und Java^{TM} genutzt. Beispielsweise wird im Fall von Java zur Kennzeichnung einer Methode — also eines „aktivierbaren“ Objektteils — eine leere Liste an den Bezeichner angehängt, zum Beispiel `createSceneGraph()`.

Typewriter

Ein Programm (Quellcode) ist in der Schriftart `Typewriter` dargestellt. Ausgewiesene Zeilennummern in einer solchen Programmdarstellung sind kein Bestandteil des Quellcodes. Sie dienen zur Vereinfachung der Erläuterung.

Primär aus Sicherheitsgründen und aus didaktischen Gründen werden die zu importiernden Java-Klassen explizit genannt und nicht nur ihr Paket. So umfasst der Kopf einer Klasse beispielsweise:

```
import com.sun.j3d.utils.geometry.Primitive;
import com.sun.j3d.utils.geometry.Box;
import com.sun.j3d.utils.geometry.Sphere;
```

statt einfach:

```
import com.sun.j3d.utils.geometry.*;
```

Hinweis: Die Programme sind über einen längeren Zeitraum entstanden. Zu Beginn gab es noch die Institution *Fachhochschule Nordost Niedersachsen*

(FHNON) `:-)`. Daher sind einige Java-Programme im *Package* `de.fhnon.as.`*xxx* ausgewiesen.

Das *Package* für den ersten Zeitraum nach der Fusion zur Universität Lüneburg wurde `de.unilueneburg.as.`*xxx* benannt. Auf den üblichen Bindestrich zwischen `uni` und `lueneburg` musste verzichtet werden, da andernfalls Fehler auftreten. Nach Benennung zur *Leuphana Universität Lüneburg* entstand das *Package* `de.leuphana.ics.`*xxx* und später noch `de.leuphana.iwi.`*xxx*.

Alle Angaben in diesem Buch erfolgen nach bestem Wissen und Gewissen. Sorgfalt bei der Umsetzung ist indes dennoch geboten. Der Verlag, der Autor und die Herausgeber übernehmen keinerlei Haftung für Personen-, Sach- oder Vermögensschäden, die aus der Anwendung der vorgestellten Materialien und Methoden entstehen könnten.

PS: Ab und zu werden zur Aufmunterung und zum Schmunzeln im Text *Internet Smileys* benutzt. Ihre Bedeutung erläutert Tabelle 1 S. 6.

Danksagung

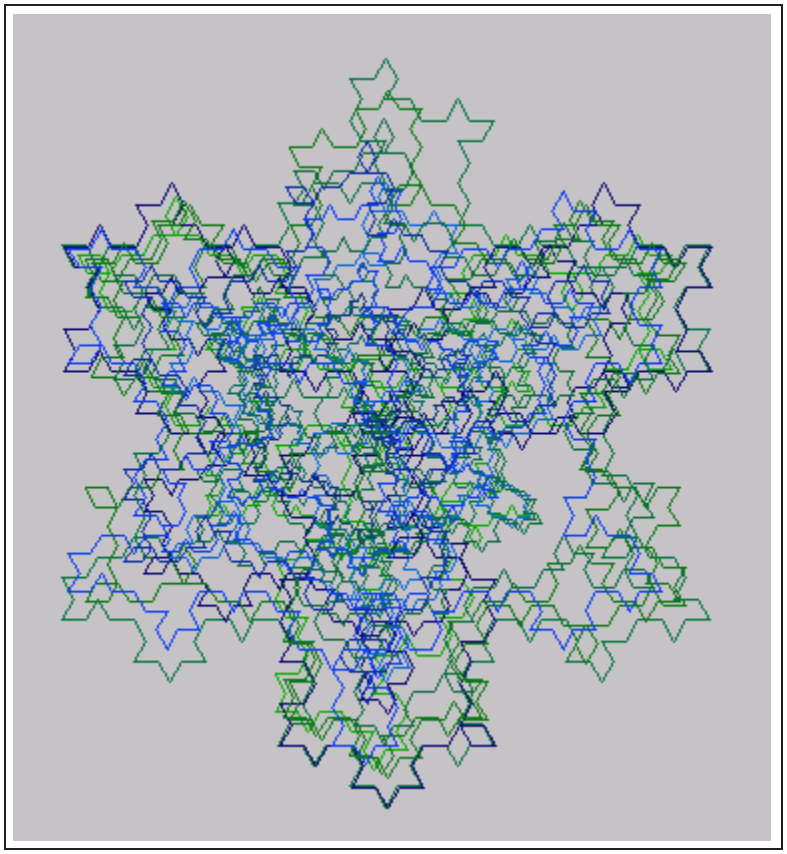
Für das Interesse und die Durchsicht einer der ersten Fassungen danke ich meinem Kollegen Herrn Dipl.-Ing. Christian Wagner. Für ein gründliches Korrekturlesen danke ich Herrn Dipl.-Kfm. Nobert Tschritter.

Ohne die kritischen Diskussionen mit Studierenden im Rahmen der Lehrveranstaltungen wäre Faszination Programmierung nicht in dieser Form entstanden. Ihnen möchte ich an dieser Stelle ganz besonders danken.

Lüneburg, 19. März 2004 – 17. Februar 2010

```
<Erfasser>
     <Verfasser>
          Hinrich E. G. Bonin
     </Verfasser>
</Erfasser>
```

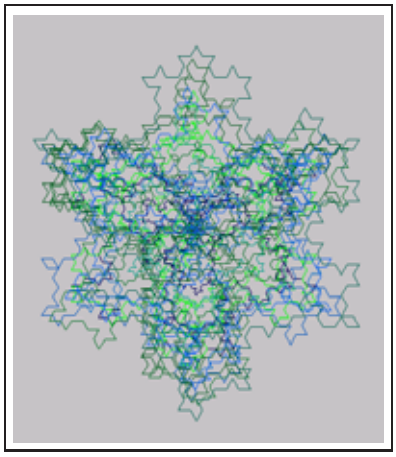
8



Inhaltsverzeichnis

Kapitel 1

Ist programmieren eine Kunst?



Kunst ist Mathematik![1]
Alexej von Jawlensky, 1864–1941, russ.-dt. Maler
zitiert in ↪ [Mäckler00] S. 27

[1] In Walter Dexel; Der Bauhausstil — ein Mythos, Starnberg (Josef Keller Verlag), 1976, S. 39

techne

„Der handwerksbedingte Ursprung des Kunstbegriffs manifestiert sich in dem griechischen Wort *techne*, in dem Aristoteles die Fähigkeit ausgedrückt sieht, ein Produkt herzustellen, mit dem Wissen um das innewohnende Prinzip. Gelingt die Arbeit, wird von *können* als *Kunst* gesprochen; auch die Heil- und Kriegskunst zählen in der Antike zu den *artes*.“ :-) (↪ [Mäckler00] S. 9)

Trainingsplan

Das Kapitel „Ist programmieren eine Kunst?“ gibt einen Überblick über:

- die Frage: Was ist programmieren?, und
  ↪ Seite 14 . . .
- das Zusammenwirken aus Kreativität und strikt diszipliniertem Vorgehen.
  ↪ Seite 16 . . .

1.1 Was ist programmieren?

Wenn programmieren den gesamten Vorgang zur Schaffung eines Progamms umfasst, dann stellt sich die Frage: Was ist ein Programm? — oder anders formuliert, was ist eine allgemein anerkannte Definition für den Begriff *Programm*.

1.1.1 Definition: Programm

Leider ist der Begriff *Programm* sehr unscharf (*fuzzy*) und wird zusätzlich in vielfältigen Zusammenhängen verwendet auch unabhängig von Rechnern. Zum Beispiel spricht man vom Programm der politischen Partei XY oder vom Programm der Bundesregierung :-(. Üblicherweise und ganz allgemein thematisiert der Begriff *Programm* einen geplanten Ablauf, der von Menschen oder Rechnern abgearbeitet werden soll.[2] Wir unterstellen, dass ein solcher Ablauf irgendwie aufgeschrieben sein muss und definieren dann:

[2] Zum Beispiel ↪ `http://de.wikipedia.org/wiki/Programm` (online 06-Oct-2005)

Ein Programm[3] ist ein im Voraus festgesetzter Ablauf, der so notiert ist, dass er von Menschen und/oder Rechnern vollzogen werden kann.

Algorithmus

So gesehen entspricht der Begriff *Programm* dem Begriff *Algorithmus*. Die Formulierung eines Algorithmus unterliegt jedoch keinen so engen Sprachvorschriften, wie die eines Programms. Dieses muss exakt die *Semiotik*, das heißt,

- die *Syntax*[4] (≈ Formvorschriften für die Sprachkonstrukte),
- die *Semantik*[5] (≈ Bedeutung der Sprachkonstrukte) und
- die *Pragmatik*[6] (≈ korrekte Anwendungskontext der einzelnen Sprachkonstrukte),

der gewählten Programmiersprache einhalten. Ein Algorithmus kann auf beliebige Art formuliert sein, wenn er nur im Voraus einen bestimmten Ablauf festlegt. Anders formuliert: Ein und derselbe Algorithmus kann in verschiedenen Programmiersprachen formuliert werden.

Im Zusammenhang mit Rechnern versteht man unter einem Programm eine Einheit zur Erfüllung von vordefinierten Aufgaben. Ein Programm kann in sehr unterschiedlichen Größenordnungen vorkommen. Aus der Größenperspektive wird ein Programm als Baustein, Modul, Procedur, Funktion etc. bezeichnet. Wirken mehrere Einheiten (Programme) zusammen spricht man in der Regel von Software.

1.1.2 Prozess der Realisierung von Qualität

Bei der Schaffung eines Programms (P) – oder allgemeiner von Software – geht es um die Realisierung der fallspezifisch notwendigen Qualität (Q). Sie ergibt sich primär aus den folgenden Eigenschaften eines Programms: $P(Q)$

1. *Leistungsfähigkeit*,
   das heißt, das Programm erfüllt die gewünschten Anforderungen.

2. *Zuverlässigkeit*,
   das heißt, das Programm arbeitet auch bei ungewöhnlichen Bedienungsmaßnahmen und bei Ausfall gewisser Komponenten weiter und liefert aussagekräftige Fehlermeldungen (Robustheit),

[3] französisch: *programme* ≡ schriftliche Bekanntmachung, Tagesordnung; griechisch: *prógramma* ≡ Vorgeschriebenes, Vorschrift

[4] Griechisch *Syntaxis* ≡ Lehre vom Satzbau, Satzlehre

[5] Semantik ≡ Bedeutungslehre

[6] Pragmatik von griechisch *pragma* ≡ Tat; Sachkunde, besonders die Geschäftsordnung im Staatsdient

3. *Durchschaubarkeit & Wartbarkeit*,
   das heißt, das Programm kann auch von anderen Programmierern als dem Autor verstanden, verbessert und auf geänderte Verhältnisse eingestellt werden,

4. *Portabilität & Anpassbarkeit*,
   das heißt, das Programm kann ohne großen Aufwand an weitere Anforderungen angepasst werden,

5. *Ergonomie & Benutzerfreundlichkeit*,
   das heißt, das Programm ist leicht zu handhaben,

6. *Effizienz*,
   das heißt, das Programm benötigt möglichst wenig Ressourcen.

Aus der Perspektive dieser Eigenschaften betrachtet, läßt sich ein Programm, im Kontext von Rechnern, als die Umsetzung eines Algorithmus mit hinreichender Qualität betrachten. Wir halten fest, nicht jede Folge von Befehlen, die „läuft“ und irgendetwas macht, kann den Ehrennamen Programm bekommen, denn es fehlen ihr wesentliche Eigenschaften (Qualitätsansprüche).

1.2 Kreativität & Disziplin

Man braucht Ideen, also Kreativität, wenn es gilt Konstrukte einer gewählten Programmiersprache so zu notieren, das ein Programm entsteht, das die erforderlichen Qualitätsansprüche erfüllt. Kurz und holzschnittartig formuliert: *Der Programmierer gehört zur Gilde der Kreativen.*

Bei komplexen Zusammenhängen reicht aber Kreativität nicht aus. Die schönen Ideen kommen selten zielorientiert genau zur rechten Zeit. Vielmehr gilt es, durch ein systematisches Vorgehen, die Kreativität zu kanalisieren, also durch die Einhaltung einer bewährten Schrittfolge, das kreative Denken auf die jeweils angebrachten Themen (Punkte) zu konzentrieren. Kurz und holzschnittartig formuliert: *Der Programmierer gehört zur Gilde der Buchhalter*, weil dieser gewohnt ist, diszipliniert nach Regeln, komplexe Fälle zu bearbeiten.

Wenn man Programmieren anhand von Graphiken (Bildern) erlernen will, dann sollte das Ergebnis einer Programmausführung (hoffentlich) Kunst sein. Was Kunst ist, dürfte eine kaum klärbare philosophische Frage sein. Zwei Zitate mögen dazu jedoch etwas einstimmen, also Diskussionsstoff liefern.

> „Wer meint, dass man Bilder außerhalb der ästhetischen Erfahrung erzeugen kann, der bewegt sich auf demselben Grat wie die Alchimisten oder wie diejenigen, die nach dem Perpetuum Mobile suchen. Aber was heißt nun ästhetische Erfahrung? … Ästhetische Erfahrung ist die Selbstkonstituierung des Einzelnen in der Praxis des Lebens (egal in welcher Form dieser Praxis — als Jäger, Sammler, Bauer, Arbeiter, Wissenschaftler, Lehrer, Pfarrer, Politiker usw.) nach dem universellen

> Gesetz der Selbstoptimierung.“
> ↪ [Nadin03] S. 119.

> „der künstler als programmierer *schafft werke als klassen von werken*. er denkt grundsätzlich, wenn er schafft. er denkt an alle bilder, die ein inneres band verbindet. er denkt an bilder als klassen, die es berechenbar zu realisieren gilt.“ :-)
> ↪ [Nake03] S. 139. (Hinweis: Im Orginal klein geschrieben.)

Ein Kunstwerk behält seine einzigartige Schönheit, auch wenn man den Algorithmus genau versteht, mit dem es erzeugt wurde. Beispielsweise verliert eine Fuge von Bach keinesfalls ihre Faszination, nur weil man exakt weiß, wie sie konstruiert ist. Klar, die Musikwissenschaft kann zu Bachs Fuge einiges fundiert sagen, aber zur Erklärung ihrer individuellen Faszination bleibt sie überfragt ;-).

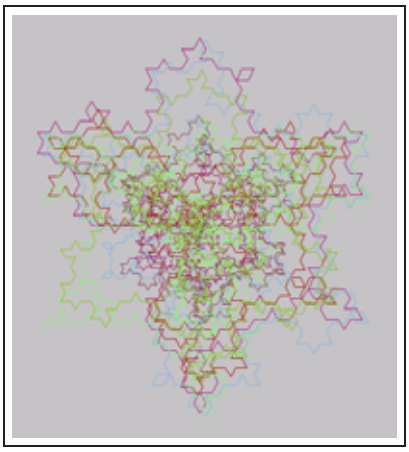
P = **Kunst?**

Ob nun ein Programm (*P*) selbst, also der notierte Code in der jeweiligen Programmiersprache (*Quellcode*), als ein Kunstwerk betrachtbar ist, mag mancher bestreiten. Wer jedoch wirklich vom Programmieren fasziniert ist, der erkennt einen gelungenen Quellcode durchaus als Kunstwerk an.

Kapitel 2

PostScript^{TM}-Paradigma — Kommandos überall

PostScript has become
an established part of worldwide
graphics, design, publishing, and printing.
(↪ [McGilton/Campione92] p. xiii)



Die Sprache PostScript^{TM}[1], seit ≈ 1985 entwickelt von Adobe Systems Incorporated, integriert drei Punkte:

- *PostScript* ist eine universelle Programmiersprache (*General-Purpose Programming Language*) mit leistungsfähigen Graphikbausteinen (*Built-in Graphics Primitives*).

[1] PostScript is a registered trademark of Adobe Systems Incorporated.
Post ≡ Vorsilbe mit der Bedeutung „nach, hinter“
Skript ≡ „Geschriebenes“, schriftliche Ausarbeitung, Nachschrift einer Hochschulvorlesung, Drehbuch

- *PostScript* ist eine Sprache zur dynamischen Seitenbeschreibung (*Page Description Language*).
- *PostScript* ist ein interaktives System zur Steuerung von Druckern und Bildschirmen (*System for Controlling Raster Output Devices*)
- *PostScript* ist ein Austauschformat für Seitenbeschreibungen, unabhängig von speziellen Anwendungen und Geräten (*Application- and Device-Independent Interchange Format*)

Trainingsplan

Das Kapitel „PostScript^{TM}-Paradigma — Kommandos überall“ gibt einen Überblick über:

- die Notation von Kommandos (Befehlen),
  ↪ Seite 14 . . .
- einen Programmiereinstieg anhand von einigen PostScript-Kostproben,
  ↪ Seite 22 . . .
- die Aufgaben von *Stacks* mit *Push*- und *Pop*-Operationen,
  ↪ Seite 28 . . .
- die elementaren Datentypen und
  ↪ Seite 30 . . .
- die Möglichkeiten der Gestaltung von (Druck-)Seiten anhand eines Beispiels mit *Clipping Region*.
  ↪ Seite 32 . . .

2.1 Notation von Kommandos

Syntax, Semantik und Pragmatik sind prägende Aspekte jeder Programmiersprache, daher gestalten sie auch die Art und Weise wie wir in PostScript formulieren und notieren. Wie üblich, so sind sie, auch bei PostScript, nicht völlig neu erfunden, sondern basieren auf verschiedenen Programmiersprachen. Die

PostScript-Syntax orientiert sich primär an der der Programmiersprache *Forth* [2] und damit an der Postfix-Notation, auch *Umgekehrten Polnischen Notation* genannt, das heißt, der Operator folgt seinen Operanden.

2.1.1 Postfix-Notation

Der Begriff Postfix-Notation (UPN ≡ *Umgekehrte Polnischen Notation*) klingt zunächst kompliziert und möglichweise unnatürlich. Damit wird jedoch nur die Reihenfolg von Substantiv (Objekt) und Verb (Operation) thematisert. Man formuliert in UPN beispielsweise einen Alltagsvorgang folgendermaßen:

1. Hände waschen
2. Brot schneiden
3. Brot belegen

und nicht:

1. Wasche Hände!
2. Schneide Brot!
3. Belege Brot!

Postfix-Notation

Man gehe zu einer Stelle auf einer Seite, die durch die Koordinaten mit $x = 150$ Einheiten und $y = 200$ Einheiten definiert ist. Diese Aufgabe wird in PostScript, also in seiner Postfix-Notation, folgendermaßen notiert:

```
150 200 moveto
```

Wollte man beispielsweise zur Zahl 20 die Zahl 40 addieren und das Ergebnis anschließend durch 2 teilen, dann stellt sich diese Aufgabe in PostScript folgendermaßen dar:

```
20 40 add 2 div
```

2.1.2 Präfix-Notation

Wären die beiden genannten Beispielaufgaben in der gegensätzlichen Notation, der sogenannten Präfix-Notation (Prefix-Notation), auch *Polnische Notation* [3]

[2] *Forth* wurde von Charles H. Moore ≈ 1969 entwickelt. *Forth* ist eine Progammiersprache (incl. Betriebssystem & Entwicklungsumgebung) deren Hauptdatenstruktur der *Stack* ist. *Forth*-Konstrukte werden in der *Umgekehrten Polnischen Notation* formuliert. Ein Interpreter für *Forth* kann sehr ressourcenschonend gebaut werden. *Forth* eignet sich daher besonders gut für die Programmierung von kleinen Rechnern (z. B. Microcontrollern).

[3] Die *Polnische Notation* auch Łukasiewicz-Notation genannt verdankt ihren Namen dem polnischen Mathematiker Jan Łukasiewicz, der sie 1920 vorstellte. Diese Notation, bei der der Operator vor den Operanden steht, bedarf keiner Klammerung, da die Präzedenz der Operationen klar ist. Der gleiche Effekt wird erreicht, wenn der Operator nicht vor den Operanden, sondern danach steht. Diese Postfix-Notation wurde später ebenfalls von Łukasiewicz entwickelt.

genannt, zu formulieren, wie sie beispielsweise LISP (*List Processing*) nutzt, dann wäre zu notieren:

```
(moveto 150 200)
```

und

```
(div (add 20 40) 2)
```

Das Datenmodell in PostScript verfügt über Bausteine einer modernen Programmiersprache wie zum Beispiel Zahlen, Zeichenketten (*Strings*) und Matrizen (*Arrays*). Zur Manipulation von Programm und Daten ist bedeutsam, das PostScript beides quasi gleich, als Daten behandelt (↪ Abschnitt 2.4 S. 30). An dieser Stelle halten wir nur fest, die Sprache PostScript verfügt über weitreichende Möglichkeiten und ist für den Soforteinstieg hinreichend einfach zu nutzen — letztlich aufgrund ihrer interpretativen Ausführung.

Es gibt viele Programme zur Ausführung (Interpretation) von PostScript-Quellcode, beispielsweise die Open-Source-Software `GSview` von *Ghostgum Software Pty. Ltd.* oder das lizenzpflichtige Softwarepaket `Corel` von *Corel Corporation* oder die Programme in vielen Laserdruckern, beispielsweise im Drucker `HP ColorLaserJet4600PS`.

2.2 PostScript-Kostproben

2.2.1 Einfache Linie

Als Einstieg soll der PostScript-Quellcode für die Darstellung einer einfachen Linie notiert werden. Dazu nutzt man seinen Lieblingseditor, zum Beispiel `Emacs` oder `Microsoft Word` oder ... . Mit diesem erzeugt man eine Datei mit dem Namen `examplePSa` und dem Suffix `ps` (Extension).

PostScript *LanguageLevel*

Jede PostScript-Quellcode-Datei beginnt mit einer Zeile, die definiert, das es sich um PostScript handelt (`%!PS`) und um welchen Sprachlevel es geht. Man unterscheidet in der Entwicklung von PostScript Phasen, die man als Sprachlevel (*LanguageLevel* [4]) bezeichnet. Im folgenden wird aus Einfachheitsgründen generell der Level `Adobe-3.0` angenommen. Die Datei `examplePSa.ps` enthält daher folgende erste Zeile:

```
%!PS-Adobe-3.0
```

DIN A4

DIN A4 Diese Linie soll auf einer DIN A4 Seite dargestellt werden. Eine DIN A4 Seite ist $21cm$ breit und $29,70cm$ hoch. In PostScript-Einheiten ausgedrückt, ist die DIN A4 Seite $595PSunit$ breit und

[4] Im Computerjargon auch *Magic Number* genannt.

842*PSunit* hoch. Der Mittelpunkt M der Seite liegt dann bei $M(298,421)$. Ein Zentimeter entspricht damit $28,35$*PSunit* und ein Inch 72*PSunit*.

Den Anfangspunkt (A) der Linie setzt man mit dem Operator `moveto`. Mit dem Operator `lineto` zieht man die Linie bis zum gewünschten Endpunkt (E). Die Beispielslinie soll in PostSript-Einheiten gehen von $A(150,200)$ bis $E(300,400)$. Dazu wird in der Datei `examplePSa.ps` notiert:

```
150 200 moveto
300 400 lineto
```

Aus holzschnittartiger, stark vereinfachter Sicht basiert das PostScript-Modell auf Graphik-Objekten deren Konturen spezifiziert werden. Mit dem PostScript--Konstrukt `closepath` beendet man die Spezifikation einer Kontur und geht zum Ausgangspunkt zurück. Mit dem PostScript-Konstrukt `stroke`[5] wird die Kontur als Linie dargestellt. Hat man beispielsweise ein Fläche spezifiziert dann wird mit `stroke` ihr Rand (↪ z. B. S. 25) dargestellt und mit dem PostScript-Konstrukt `fill`[6] ihre Fäche (↪ z. B. S. 26). Die Datei `examplePSa.ps` wird daher ergänzt um folgende Zeilen:

stroke

```
closepath
stroke
```

Die Darstellung eines spezifizierten Objektes vollzieht das PostScript-Konstrukt `showpage`. Im Sinne des Anspruchs, ein Programm nicht nur lauffähig zu machen, sondern eine hinreichende Qualität zu erreichen, kommentiert man den bisherigen PostScript-Quellcode mit den Konstrukten:

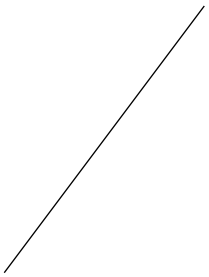
```
%%Creator:
%%Title:
%%CreationDate:
%%EndComments
...
%%EOF
```

Damit ist die erste simple Graphik, eine einfache Linie von $A(150,200)$ bis $E(300,400)$, vollständig programmiert und kann nun von einem PostScript-Interpretor abgearbeitet werden.

Die Abbildung 2.1 S. 24 zeigt das Ergebnis, nachdem die Datei examplePSa.ps in die *Encapsulated PostScript* Datei `examplePSa.eps` konvertiert wurde und dann in ein LaTeX-Dokument eingebunden wurde. Aus diesem LaTeX-Dokument wurde über das Programm `DVIPS` eine PostScript-Datei erzeugt, die auch den PostScript-Quellcode von `examplePSa.eps` enthält.

PS→EPS Die Konvertierung von PostScript zu *Encapsulated PostScript* **EPS**

[5] `stroke` ≡ mit einem Strich kennzeichnen
[6] tt fill ≡ (sich) füllen, ausfüllen



Legende:
PostScript-Quellcode ↪ S. 24

Abbildung 2.1: PostScript: Einfache Linie

(EPS) geschieht durch die Einfügung einer Positionierungsangabe der Graphik als Rechteck (*BoundingBox*).

Die Datei `examplePSa.eps` enthält daher die beiden Kopfzeilen:

```
%!PS-Adobe-3.0 EPSF-3.0
%%BoundingBox: 148 199 302 401
```

und sonst den Inhalt der Datei `examplePSa.ps`. Die `BoundingBox` wird durch ihre linke untere Ecke (*LU*) und ihre rechte obere Ecke (*RO*) in Form von *PSunit*-Angaben spezifiziert. In diesem Fall also mit den Punkten $LU(148,199)$ und $RO(302,401)$.

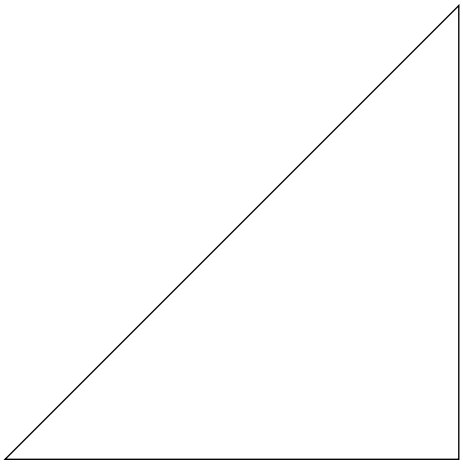
Listing 2.1: `examplePSa`

```
%!PS-Adobe-3.0
%%Creator: Hinrich E.G. Bonin
%%Title: Einfache Linie!
%%CreationDate: 08-Oct-2005
%%EndComments
150 200 moveto
300 400 lineto
closepath
stroke
showpage
%%EOF
```

2.2.2 Einfaches Dreieck

Listing 2.2: `examplePSb`

```
%!PS-Adobe-3.0
%%Creator: Hinrich E.G. Bonin
%%Title: Einfaches Dreieck!
%%CreationDate: 08-Oct-2005
%%EndComments
```



Legende:
PostScript-Quellcode ↪ S. 24

Abbildung 2.2: PostScript: Einfaches Dreieck

```
%%BeginProlog
/cm { 28.35 mul } def
%%EndProlog
4 cm 4 cm moveto
16 cm 16 cm  lineto
16 cm  4 cm lineto
closepath
stroke
showpage
%%EOF
```

2.2.3 Grau ausgefülltes Dreieck

Listing 2.3: `examplePSc`

```
%!PS-Adobe-3.0
%%Creator: Hinrich E.G. Bonin
%%Title: Grau ausgefuelltes Dreieck!
%%CreationDate: 08-Oct-2005
%%EndComments
%%BeginProlog
/cm { 28.35 mul } def
%%EndProlog
4 cm 4 cm moveto
0.8 setgray
16 cm 16 cm  lineto
16 cm  4 cm lineto
closepath
fill
showpage
```

Legende:
PostScript-Quellcode ↪ S. 25

Abbildung 2.3: PostScript: Graues Dreieck

```
%%EOF
```

2.2.4 Hello World!

Listing 2.4: `examplePSd`

```
%!PS-Adobe-3.0
%%Creator: Hinrich E.G. Bonin
%%Title: Hello World!
%%CreationDate: 08-Oct-2005
%%EndComments
%%BeginProlog
/cm { 28.35 mul } def
%%EndProlog
/Palatino-Roman findfont
83 scalefont
setfont
4 cm 4 cm moveto
0.4 setgray
45 rotate
(Hello World!) show
showpage
%%EOF
```

2.2.5 Zusammengesetzte Kostprobe

Listing 2.5: `examplePSe`

```
%!PS-Adobe-3.0
%%Creator: Hinrich E.G. Bonin
```

Legende:
PostScript-Quellcode ↪ S. 26

Abbildung 2.4: PostScript: Gedrehter Schriftzug

Legende:
PostScript-Quellcode ↪ S. 26

Abbildung 2.5: PostScript: `Hello World!` mit grauem Dreieck

```
%%Title: Hello World mit grauem Dreieck!
%%CreationDate: 08-Oct-2005
%%EndComments
%%BeginProlog
/cm { 28.35 mul } def
%%EndProlog
4 cm 4 cm moveto
0.8 setgray
16 cm 16 cm  lineto
16 cm  4 cm lineto
closepath
fill
/Palatino-Roman findfont
83 scalefont
setfont
4 cm 4 cm moveto
0.4 setgray
45 rotate
(Hello World!) show
showpage
%%EOF
```

2.3 Stack

Bisher wurde ein PostScript-Objekt entweder ausgefüllt dargestellt (`fill`) oder es wurde seine Kontur als Strich gezeichnet (`stroke`). Wollte man beispielsweise das graue Dreieck in der Abblidung 2.5 S. 27 umranden, dann könnte man es nochmal notieren und mit `stroke` spezifizieren. Ein solche Lösung hätte den Nachteil, dass der PostScript-Quellcode sehr lang und damit wenig durchschaubar würde (Mangel an Transparenz!). Bei der besseren Lösung (↪ Abbildung 2.6 S. 29; Quellcode S. 29) hält man einen spezifizierten Zustand der Graphik fest und setzt dann später wieder auf diesen auf. Zum Festhalten des Graphikzustandes dient das PostScript-Konstrukt `gsave`, zum Wiederaufsetzen das Konstrukt `grestore`. Beide Konstrukte arbeiten mit dem *Graphik State Stack*, wobei `gsave` eine *Push*-Operation und `grestore` eine *Pop*-Operation durchführt.

gsave

grestore

2.3.1 Last-In-First-Out (LIFO) —*Stack*: *Push* und *Pop*

Ein klassischer *Stack* (≡ Stapelspeicher) ist eine Datenstruktur mit festgelegen Operationen zu ihrer Manipulation. Das Hinzufügen von Daten erfolgt ausschließlich, indem diese Daten auf den Stapel „oben aufgelegt“ werden (*Push*-Operation). Das Ändern des Stapelspeichers ist nur möglich durch Entnahme der Daten, die auf dem Stapel „oben aufliegen“ (*Pop*-Operation). Hinweis: Zur Effizenzsteigerung ist bei manchem *Stack*, wie auch bei PostScript, der Lesezugriff auf tiefer liegende Daten möglich. Dadurch wird ein Umschichten eines Stapels, mit Hilfe eines anderen Stapels vermieden.

Programmiersprachen wie Java, C oder FORTRAN verstecken ihr Stack-Modell vor der Manipulation durch den Programmierer. PostScript jedoch er-

Legende:
PostScript-Quellcode ↪ S. 30

Abbildung 2.6: PostScript: `Hello World!` mit umrandetem Dreieck

möglicht, wie bei interpretierten Sprachen üblich, die direkte Kontrolle und den Zugriff auf seine Stacks. Bei PostScript sind vier Stacks zu unterscheiden:

- *Operand Stack* wird verwendet als Speicher der Operanden für fast alle PostScript Operatoren.
- *Dictionary Stack* wird verwendet als Speicher für Objekte, die mittels ihres Namens auffindbar sind (*Naming Contexts*, *Name-Lookup-Mechanisms*).
- *Graphic State Stack* wird verwendet als Speicher, für Zustände der Graphik, wie sie beispielsweise im folgenden Quellcode (S. 29) mit `gsave` und `grestore` genutzt werden.
- *Execution Stack* wird verwendet als Speicher für ausführbare Prozeduren (*Executable Procedures*), insbesondere zum Zeitpunkt ihrer Ausführung.

2.3.2 *Graphic State Stack*-Beispiel

In diesem PostScript-Beispiel wird die Stärke der Umrandungslinien, mit der sie dann vom `stroke`-Konstrukt dargestellt werden, explizit gesetzt (`setlinewidth`). Das Zusammentreffen dieser Linien wird mit dem Konstrukt

`setlinejoin` gestaltet. Dieser Wert des Arguments bestimmt den Grad der „Abrundung“.

Listing 2.6: `examplePSf`

```
%!PS-Adobe-3.0
%%Creator: Hinrich E.G. Bonin
%%Title: Hello World mit umrandetem Dreieck!
%%CreationDate: 08-Oct-2005
%%EndComments
%%BeginProlog
/cm { 28.35 mul } def
%%EndProlog
4 cm 4 cm moveto
16 cm 16 cm  lineto
16 cm  4 cm lineto
closepath
gsave
0.8 setgray
fill
grestore
0.9 setgray
4.5 cm setlinewidth
2 setlinejoin
stroke
/Palatino-Roman findfont
83 scalefont
setfont
4 cm 4 cm moveto
0.4 setgray
45 rotate
(Hello World!) show
showpage
%%EOF
```

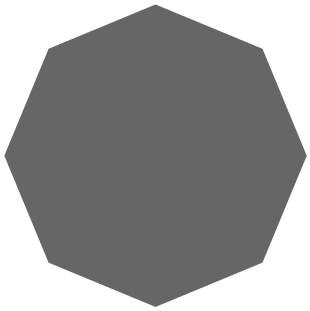
2.4 Datentypen

Ein PostScript-Stack kann Elemente von unterschiedlichen Datentypen speichern. Dabei unterscheidet man einfache Datentypen (*Simple Data Types*) und zusammengesetzte Datentypen (*Composite Data Types*).

Simple Data Types Beispiele:

```
true    % Boolean Value
7       % Integer Value
-13     % Integer Value
3.141   % Real Value
8#377   % Radix Value
        % (Octal representation of 255)
```

Composite Data Types Beispiele:



Legende:
PostScript-Quellcode ↪ S. 31

Abbildung 2.7: PostScript: *Octagon*-Figur

```
(Nachhaltigkeit!) % Character String
<0123456789ABCDEF>% Hexadecimal String
[ 1 2 3 4 5 ]     % Array Object
{ 72 mul }        % Executable Array Object
```

Die Werte in einer Matrix (*Array*) können auch das Ergebnis einer Auswertung beim Einleseprozess sein. Zum Beispiel wird aus dem folgenden notierten Eingabedaten:

```
[ 7 2 mul 2 2 mul ]
```

ein *Array* mit nur zwei Elementen:

```
[ 14 4 ]
```

Das Beispiel `octagonPS` (↪Abbildung 2.7 S. 31, Quellcode S. 31) nutzt diese Array-Eigenschaft zur Berechnung der Eckpunkte des Achtecks.

```
[  45 cos radius mul    45 sin radius mul ]
  % 1. Eckpunkt rechts oben im Gegenuhrzeiger
```

Mit den Konstrukt `get` wird auf das Array zugegriffen, wobei in PostScript das erste Element mit den Index 0 adressiert wird (*zero-based*). Das Konstrukt `aload` transferiert die Elemente eines Array zum *Operand Stack*. Im Beispiel also die berechneten x,-y-Werte eines Eckpunktes. Das Konstrukt `pop` entfernt des oberste Element des Stacks, hier vom Datentyp Array.

aload

Listing 2.7: `octagonPS`

```
%!PS-Adobe-3.0
%%Creator: Hinrich E.G. Bonin
%%Title: Octagon
```

```
4  %%CreationDate: 08-Oct-2005
   %%EndComments
6  %%BeginProlog
   /cm { 28.35 mul } def
8  /radius 4 cm def
   /cornerpoints [
10 [  45 cos radius mul    45 sin radius mul ]
   [  90 cos radius mul    90 sin radius mul ]
12 [ 135 cos radius mul   135 sin radius mul ]
   [ 180 cos radius mul   180 sin radius mul ]
14 [ 225 cos radius mul   225 sin radius mul ]
   [ 270 cos radius mul   270 sin radius mul ]
16 [ 315 cos radius mul   315 sin radius mul ]
   [ 360 cos radius mul   360 sin radius mul ]
18 ] def
   %%EndProlog
20 4.2 cm 4.2 cm translate
   cornerpoints 0 get aload pop moveto
22 cornerpoints 1 get aload pop lineto
   cornerpoints 2 get aload pop lineto
24 cornerpoints 3 get aload pop lineto
   cornerpoints 4 get aload pop lineto
26 cornerpoints 5 get aload pop lineto
   cornerpoints 6 get aload pop lineto
28 cornerpoints 7 get aload pop lineto
   closepath
30 0.4 setgray
   fill
32 showpage
   %%EOF
```

2.5 Clipping Region

Die Spezifikation eines Pfades (*Path*) lässt sich auch zum Schneiden eines komplexen Seitenausschnittes (*Clipping*[7] *Region*) benutzen. An die Stelle des PostScript-Konstruktes `stroke` oder `fill` tritt dann das Konstrukt `clip`, wie das folgende Beispiel (↪ PostScript-Quellcode S. 33) zeigt. Der Ausschnitt ist dort als Ellipse spezifiziert. Die Ellipse wird als Kreisbogen von 0 bis 360 Grad und einem unterschiedlichen Skalierungsfaktor in Richtung der Breite (x-Werte) und Höhe (y-Werte) definiert und zwar folgendermaßen:

```
1.2 1.8 scale
0 0 2.75 inch 0 360 arc
```

Der Skalierungsfaktor der x-Werte ist 1.2, der der y-Werte 1.8. Das PostSript-Konstrukt `arc` hat folgende Syntax:

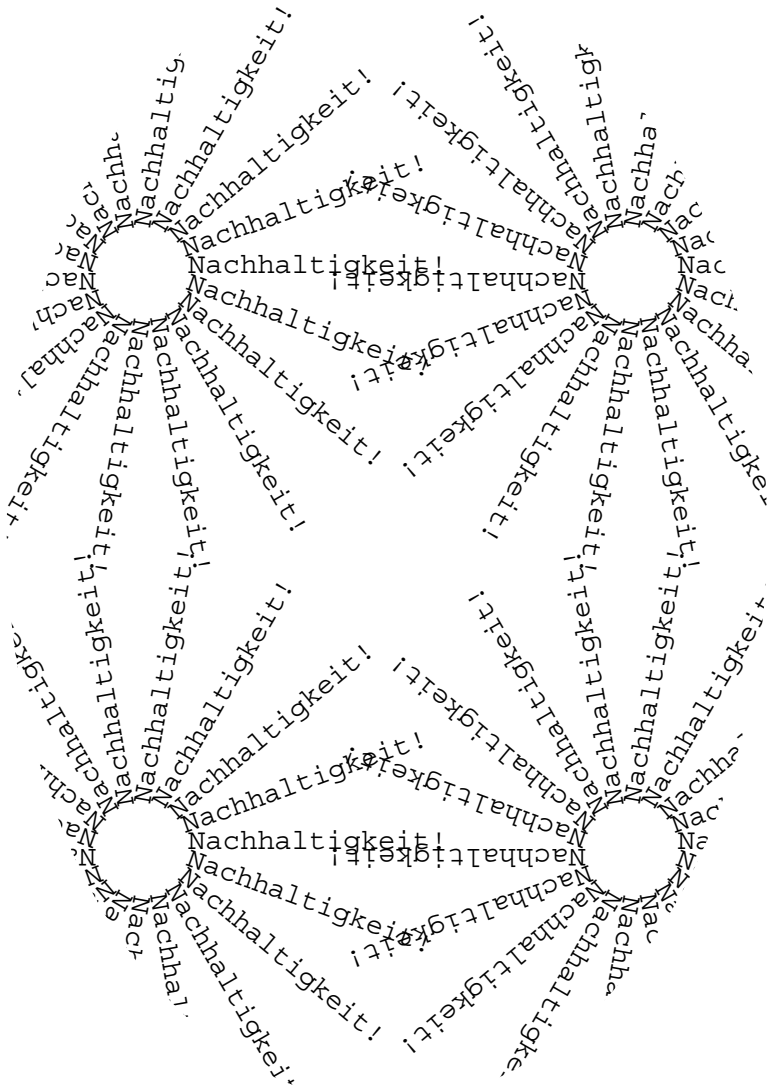
$$x_{Mittelpunkt}\ y_{Mittelpunkt}\ Radius\ Start_{Winkel}\ End_{Winkel}\ \texttt{arc}$$

[7] Clipping ≡ (Be)Schneiden, Stutzen, Ausschnitt

Die Werte für $Start_{Winkel}$ und End_{Winkel} werden in Grad von der x-Achse in Gegen-Uhrzeiger-Richtung ermittelt. Im Beispiel ist ein Kreisbogen im Mittelpunkt (0,0) mit einem Radius von 2.75*inch* von 0^0 bis 360^0 angeben, also ein voller Kreis, der durch die unterschiedliche Skalierung der Achsen zu einer Ellipse wird.

Listing 2.8: `clippingRegionPS`

```
%!PS-Adobe-3.0
%%Creator: Hinrich E.G. Bonin
%%Title: Using a path as clipping region
%%CreationDate: 06-Oct-2005
%%EndComments
%%BeginProlog
/inch { 72 mul } def
%%EndProlog
matrix currentmatrix
4.25 inch 5.5 inch translate
1.2 1.8 scale
0 0 2.75 inch 0 360 arc
closepath
clip
newpath
setmatrix
/Courier findfont
18 scalefont
setfont
/TextOnCircle {
    gsave
    translate
    18 {
        0.4 inch 0 moveto
        (Nachhaltigkeit!) show
        20 rotate
    } repeat
    grestore
} def
2.125 inch 7.75 inch TextOnCircle
6.375 inch 7.75 inch TextOnCircle
2.125 inch 2.75 inch TextOnCircle
6.375 inch 2.75 inch TextOnCircle
showpage
%%EOF
```

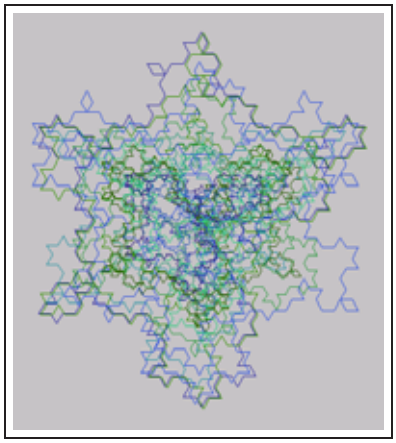


Legende:
Idee für dieses Beispiel entnommen aus [McGilton/Campione92] p. 34
PostScript-Quellcode ↪ S. 33

Abbildung 2.8: Path als Clipping Region

Kapitel 3

Java^{TM}-Paradigma — Objekte überall



Just a valid application

Unter den Fehlleistungen der Programmierung wird ständig und überall gelitten :-): Zu kompliziert, zu viele Mängel, nicht übertragbar, zu teuer, zu spät und so weiter. *The Java Factor*[1] soll es besser machen. Der Hoffnungsträger basiert auf einer beinahe konsequent objekt-orientierten Programmierung. Sie soll die gewünschte Qualität ermöglichen. Dabei wird Qualität durch die Leistungsfähigkeit, Zuverlässigkeit, Durchschaubarkeit & Wartbarkeit, Portabilität & Anpassbarkeit, Ergonomie & Benutzerfreundlichkeit und Effizienz beschrieben.

Faszination Programmierung schwärmt wohl vom Glanz der „Java-Philosophie“, ist aber nicht euphorisch eingestimmt. Es wäre schon viel erreicht, wenn Sie, liebe Programmiererin, lieber Programmierer, nach dem Arbeiten

[1] Titelüberschrift von *Communications of the ACM*, June 1998, Volume 41, Number 6.

mit diesem Buch Java^{TM} als ein Akronym für *Just a valid application* betrachten können.

Trainingsplan

Das Kapitel „Java^{TM}-Paradigma — Objekte überall“ gibt einen Überblick über:

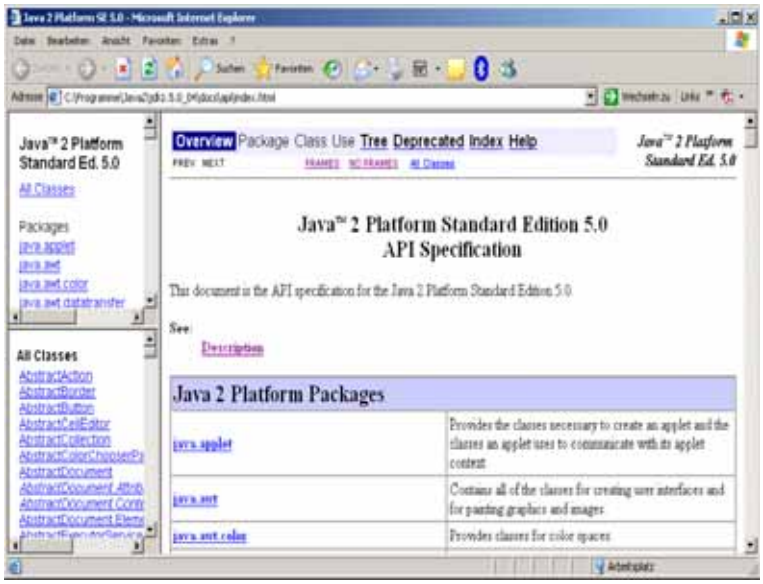
- die Java 2 Standard Edition (J2SE)
  ↪ Seite 36 ...
- die Java3D-Welt (*Virtual Universe* und
  ↪ Seite 41 ...
- einige Java3D-Kostproben.
  ↪ Seite 46 ...

3.1 J2SE (Java 2 Standard Edition)

Zum Arbeiten mit Java^{TM} benötigt man die Beschreibung der vorgegebenen Klassen und Schnittstellen (*Interfaces*) der jeweiligen Version. Arbeitet man beispielsweise mit der *Java^{TM} 2 Platform Standard Edition 5.0* dann findet man üblicherweise unter dem Pfad `docs` die benötigte API-Beschreibung (*Application Programming Interface*). Beispielsweise befindet sich diese Dialoghilfe auf meinem Rechner unter:
`file:///C:/Programme/Java2/jdk1.5.0_04/docs/api/index.html`
Die Abbildung 3.1 S. 37 zeigt die Einstiegsseite diese Java-Dialoghilfe.

Üblicherweise werden die ersten Java-Schritte mit ganz einfachen Beispielprogrammen vollzogen. Man erfährt dabei, was der Java-Compiler `javac` und die sogenannte *Java Virtual Machine* `java` machen. Solche elementaren Grundkenntnisse vermittelt mein Manuskript JAVA^{TM}–COACH (↪ [Bonin04b]) `:-)`. Da wir der Informatiktradition folgend schon das obligatorische Einstiegsprogramm „Hello World!“ in Postscript realisiert haben (↪ Abbidung 2.4 S. 27), kann hier unmittelbar eine etwas komplexere erste Kostproben präsentiert werden.

Texteingabe und 2D-Darstellung Mit Hilfe eines Java-Applets, das heißt, einer Java-Klasse eingebunden in eine Web-Seite, wird ein Text in einem Dialogfenster erfasst (↪ Abbildung 3.3 S. 42) und anschließend in einer farbigen Ellipse dargestellt (↪ Abbildung 3.4 S. 43).



Legende:
Dialoghilfe für Java^{TM}

Abbildung 3.1: Beispiel: Java API

Listing 3.1: `Text`

```
/**
 *  Applet example
 *
 *@author       Bonin
 *@version      1.2
 */
package de.unilueneburg.as.figure2D;

import java.applet.Applet;
import java.awt.Color;
import java.awt.Font;
import java.awt.font.FontRenderContext;
import java.awt.font.TextLayout;
import java.awt.geom.Ellipse2D;
import java.awt.geom.Rectangle2D;
import java.awt.Graphics;
import java.awt.Graphics2D;
import java.util.Random;
import javax.swing.JOptionPane;

public class Text extends Applet
{
   final int LARGE_SIZE = 36;

   private Color color1;
   private Color color2;
   private Font largeFont;
   private String input;


   public Text()
   {
      super();
      System.out.println("Text()");

   }


   public void init()
   {
      System.out.println("init()");

      float r1;

      float g1;

      float b1;
      float r2;
      float g2;
      float b2;
      Random generator = new Random();

      /*
       *  Applet Parameters
       *  <param name="..." value="..." />
```

```
56          */
            r1 = Float.parseFloat(getParameter("red"));
58          g1 = Float.parseFloat(getParameter("green"));
            b1 = Float.parseFloat(getParameter("blue"));
60          color1 = new Color(r1, g1, b1);

62          /*
             * Second color generated randomly.
64           * nextDouble() returns a random floating-point
             * number between 0 (inclusive) and 1 (exclusive).
66           */
            r2 = (float) generator.nextDouble();
68          g2 = (float) generator.nextDouble();
            b2 = (float) generator.nextDouble();
70          color2 = new Color(r2, g2, b2);

72          /*
             * Modal dialog (Swing toolkit)
74           * Waits until the user has entered a string
             * and clicks on the "OK" button.
76           */
            input = JOptionPane.showInputDialog(
78                "Your text?");
            if (input.length() == 0)
80          {
               input = "No input!";
82          }
            largeFont = new Font(
84                "Courier", Font.ITALIC, LARGE_SIZE);
        }
86

88      public void paint(Graphics g)
        {
90
           System.out.println("paint(g)");
92
           Graphics2D g2 = (Graphics2D) g;
94
           /*
96          * The font render context is an object, that knows how
            * to transform letter shapes (which are described as
98          * curves) into pixels.
            */
100        FontRenderContext context = g2.getFontRenderContext();

102        /*
            * The TextLayout object gets typographic measurements
104         * of the string TEXT.
            */
106        TextLayout layout =
               new TextLayout(input, largeFont, context);
108
           float xTextWidth = layout.getAdvance();
110        float yTextHeight =
               layout.getAscent() + layout.getDescent();
```

```
112         float xLeft = (getWidth() - xTextWidth) * 0.5F;
            float yTop = (getHeight() - yTextHeight) * 0.5F;
114         float yBase = yTop + layout.getAscent();

116         Ellipse2D.Float egg =
                new Ellipse2D.Float(
118             xLeft, yTop, xTextWidth, yTextHeight);
            Rectangle2D.Float box =
120             new Rectangle2D.Float(
                xLeft, yTop, xTextWidth, yTextHeight);
122
            g2.setColor(color1);
124         g2.fill(egg);
            g2.setColor(color2);
126         g2.draw(box);

128         g2.setFont(largeFont);
            g2.drawString(input, xLeft, yBase);
130     }

132
        public void stop()
134     {
            System.out.println("stop()");
136
        }
138

140     public void destroy()
        {
142         System.out.println("destroy()");
        }
144 }
```

HTML-Datei `Text.html` **mit Applet über** `<object>`**-Element**

Listing 3.2: `Text.html`

```
<?xml version="1.0" encoding="utf-8" ?>
2 <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
4 <!-- Bonin Version 1.0 -->
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en">
6 <head>
<meta http-equiv="Content-Type"
8   content="text/html; charset=utf-8" />
<title>Little Applet</title>
10 </head>
<body>
12 <h1>Your Text:</h1>
<object
14   codetype="application/java"
    classid="java:de.unilueneburg.as.figure2D.Text.class"
16   code="de.unilueneburg.as.figure2D.Text"
    width="600" height="200"
```

```
18    alt="Java: Just A Valid Application">
      <param name="red" value="1.0" />
20    <param name="green" value="0.0" />
      <param name="blue" value="0.0" />
22  </object>
    <p>Copyright bonin@uni-lueneburg.de</p>
24  </body>
    </html>
```

Protokoll `Text.log`

```
d:\bonin\prog\code>java -version
java version "1.5.0_04"
Java(TM) 2 Runtime Environment,
  Standard Edition (build 1.5.0_04-b05)
Java HotSpot(TM) Client VM
  (build 1.5.0_04-b05, mixed mode, sharing)

d:\bonin\prog\code>javac de/unilueneburg/as/figure2D/Text.java

d:\bonin\prog\code>appletviewer Text.html
Text()
init()
paint(g)
paint(g)
stop()
destroy()

d:\bonin\prog\code>
```

Das Ergebnis mit dem *Appletviewer* zeigt die ↪ Abbildung 3.2, S. 42. Das Ergebnis mit dem Browser *Firefox 1.0.7*[2] zeigt die ↪ Abbildung 3.4, S. 43. Das Fenster für die Eingabe zeigt die ↪ Abbildung 3.3, S. 42

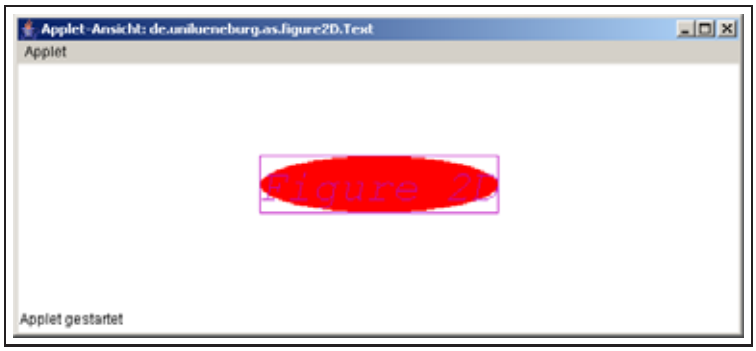
3.2 Virtual Universe — Java3D-Paket

Scene Graph

Eine Java3D-Welt (*Virtual Universe*) wird vom sogenannten *Scene Graph* erzeugt. Der *Scene Graph* setzt sich zusammen aus Objekten (Instanzen der Java3D-Klassen). Er definiert die Geometrie, den Klang, den Ort, die Orientierung und das Erscheinungsbild (*Appearance*) der sicht- und hörbaren Objekte. Als Graph setzt er sich aus Knoten und Kanten zusammen.[3] Ein Knoten (*node*) ist ein „Datenelement“ und eine Kante (*arc*) ist eine Beziehung (*Relationship*) zwischen Datenenelementen. Eine Kante bildet daher die Beziehung zwischen Instanzen von Java3D-Klassen ab. Die Abbildung 3.5 S. 44 zeigt ein erstes Beispiel für einen einfachen *Scene Graph*. Die verwendeten Symbole erläutert Abbildung 3.6

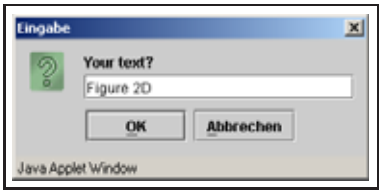
[2] Mozilla/5.0 (Windows; U; Windows NT 5.1; de-DE; rv:1.7.12) Gecko/20050919 Firefox/1.0.7.

[3] Mathematisch betrachtet ist der *Scene Graph* ein gerichteter azyklischer Graph (*directed acyclic graph* (DAG)).



Legende:
Modal dialog ↪ Abbildung 3.3, S. 42

Abbildung 3.2: `Text`-Applet mit `appletviewer`

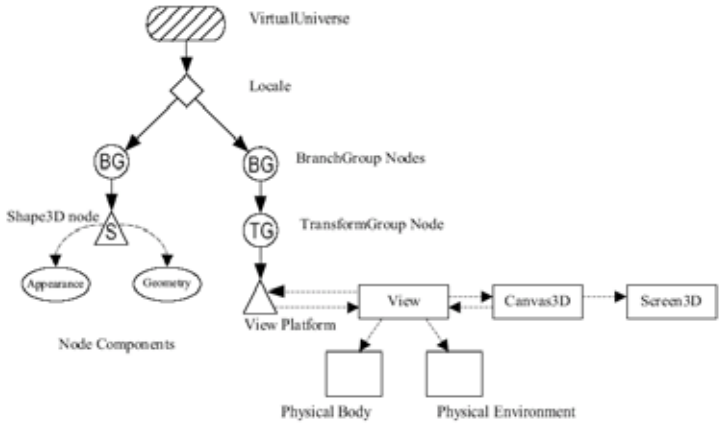


Legende:
Modal dialog: Texteingabe `Figure 2D`

Abbildung 3.3: Eingabefenster von `Text`-Applet

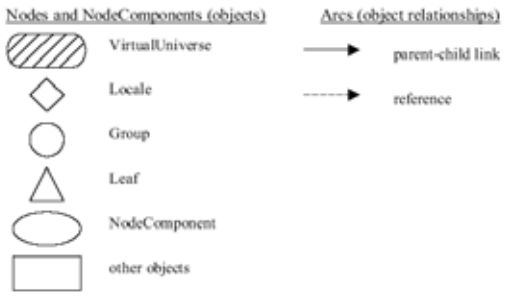
Legende:
Modal dialog ↪ Abbildung 3.3, S. 42

Abbildung 3.4: `Text`-Applet mit Browser *Firefox 1.0.7*



Legende:
Inhalt ≡ linke Teilgraph von `Locale` (*content branch graph*)
Sichtweise ≡ rechte Teilgraph von `Locale` (*viewing branch graph*)
Quelle ↪ *The Java 3D Tutorial, Chapter 1, p. 1–5*
Erläuterung der Symbole ↪ Abbildung 3.6 S. 44

Abbildung 3.5: *Scene Graph* — erstes Beispiel



Legende:
Quelle ↪ *The Java 3D Tutorial, Chapter 1, p. 1–4*
Einfaches Beispiel ↪ Abbildung 3.5 S. 44

Abbildung 3.6: *Scene Graph* — Symbole

Eine übliche Beziehung in einem Graphen ist die Eltern-Kind-Beziehung (*parent-child relationship*). Dabei kann ein Gruppenknoten eine Menge von Kindknoten haben, aber nur einen Elternknoten. In einem solchen Baum hat ein Blatt(knoten) (*leaf node*) keine Kindknoten.

Eine andere Beziehung ist die Referenz. Eine Referenz assoziiert ein *NodeComponent object*, also ein Bestandteil eines Objektes, mit einem *Scene Graph*-Knoten. Die *NodeComponent*-Objekte definieren Attribute der Geometrie und des Erscheinungsbildes, die benutzt werden, um das sichtbare Objekt zu berechnen (*render*n).

Ein *Scene Graph Tree* wird gebildet durch Bäume mit einer Wurzel aus *Locale*-Objekte. Die *NodeComponents* und ihre Referenzpfeile sind nicht Teil dieser Bäume. Somit gibt es stets nur einen Weg von der Wurzel (*root*) eines Baums zu einem Blatt (*leaf*). Diesen Weg bezeichnet man als den *leaf nodes scene graph path*. So ein Weg spezifiziert vollständig die Zustandsinformation eines Baumblattes. Die Zustandsinformation umfasst den Ort, die Orientierung und die Grösse des sichtbaren Objektes.

Locale

Ein *Locale*-Objekt stellt einen Referenzpunkt im *Virtual Universe* bereit. Es ist quasi ein Orientierungspunkt (*landmark*) um den Ort von sichtbaren Objekten im Universum festzulegen.

Ein *BranchGroup*-Objekt ist die Wurzel eines Teilgraphen (*subgraph* auch *branch graph* (BG)). Es gibt zwei Kategorien von Teilgraphen:

- *content branch graph* und
  Er spezifiziert die Geometrie, das Erscheinungsbild, Verhalten, den Ort, den Klang und das Licht.
- *viewing branch graph*
  Er spezifiziert den *viewing*-Ort und die -Richtung.

`SimpleUniverse`

Die Klasse `SimpleUniverse` aus dem Paket `com.sun.j3d.utils.universe` vereinfacht die Java3D-Programmierung wesentlich, da sie einen schon definierten *view branch graph* nutzt. Ihr Konstruktor erzeugt einen *Scene Graph*, der das *VirtualUniverse*-Objekt und *Locale*-Objekte sowie einen vollständigen *view branch graph* umfasst.

Mit dieser Vereinfachung programmieren wir unsere ersten Java3D-Kostproben nach folgender Vorgehensweise:

1. Schritt: Erzeugen einer dreidimensionalen „Leinwand“, also eines `Canvas3D`-Objektes
2. Schritt: Erzeugen eines `SimpleUniverse`-Objektes
   mit Referenz zu unserem `Canvas3D`-Objekt
3. Schritt: Anpassen (*customizing*) des `SimpleUniverse`-Objektes
4. Schritt: Konstruieren *content branch*
5. Schritt: Kompilieren *content branch graph*

6. Schritt: Einfügen *content branch graph* in *Locale* des *SimpleUniverse*

Das Java3D-API umfasst mehr als hundert Klassen, die im Paket `javax.media.j3d` zusammengefasst sind. Üblichwerweise werden diese Klassen als *Java3D core classes* bezeichnet. Dieses Kernklassenpaket enthält die notwendigen *low-level* Klassen. Es wird ergänzt durch das Paket `com.sun.j3d.utils`, das als *Java3D utility classes* bezeichnet wird. Die *utility classes* bilden eine leistungsfähige Ergänzung zum Kern. Sie umfassen folgende Kategorien:

- Laden von Inhalt (*Content Loaders*),
- Konstruktion des *Scene Graph*,
- Geometrie und
- Komfort (*convenience utilities*).

AWT

Zusätzlich zu diesen beiden Paketen werden Klassen der Pakete `java.awt` und `javax.vecmath` genutzt. Das erste enthält das *Abstract Windowing Toolkit* (AWT). Es dient zum Erzeugen eines Fensters, in dem das Ergebnis angezeigt werden kann. Das zweite enthält die Klassen der Vektormathematik, also Punkte, Vektoren, Matrizen und andere mathematischen Objekte.

Im folgenden bezeichnen wir mit „sichtbaren Objekt“ (*visual object*) ein „Objekt im *Scene Graph*“, zum Beispiel eine Kugel (*sphere*) oder einen Quader (*cube*). Der Begriff „Objekt“ verweist stets auf eine Instanz einer Klasse, während der Begriff „Inhalt“ benutzt wird, um sich auf ein sichtbares Objekt im *Scene Graph* als Ganzes zu beziehen.

3.3 Java3D-Kostproben

In der Softwareentwicklung wird traditionell mit einem Programm „Hello World!“ gestartet (↪ PostScript-Beispiel 2.2.4 S. 26). Dieses zeigt üblicherweise eine entsprechende Zeichenkette auf dem jeweiligen Standardausgabegerät. In der Java3D-Welt wäre eine solche Zeichenkette kein passender Einstieg, sondern eher eine „Beleidigung“ für die Java3D-Schöpfer `:-)`.

Wir steigen daher mit dem Programm „Hello Universe“ ein. Anschließend bringen wir dann auch die obligatorische „Hello World!“-Meldung allerdings in einer grafisch anspruchsvolleren Form (↪ Abschnitt 3.9 S. 56).

3.3.1 `HelloUniverse`

„Hello Universe“ zeigt einen Zylinder im Universum und zwar mit den Polygonlinien und ihren Eckpunkten (*Vertices*) (↪ Abbildung 3.7 S. 49). Sicherlich ist der Quellcode zunächst nicht einfach nachvollziehbar. In diesem Kapitel werden die genutzten Java3D-Konstrukte nach und nach eingehend erläutert.

Also keine Sorge — gleich wird es einfacher und es kommt die Zeit, in der Sie diesen Einstieg gut verstehen.

Die Superklasse für `HelloUniverse` (↪ Quellecode S. 52) ist ein Applet. Üblicherweise ist ein Applet ein (kleines) Java-Programm, das konzipiert wurde um nicht eigenständig zu laufen, sondern eingebettet in eine andere Anwendung. In der Regel ist diese andere Anwendung ein marktüblicher Web-Browser. Ein Applet erfordert die Implementation von Methoden wie `init()` und `destroy()`. In der ersten gestalten wir unser `SimpleUniverse`, in der zweiten wenden wir die Methode `cleanup()` auf unser `SimpleUniverse` an. Um das Applet auch als eigenständige Java-Applikation aufrufen zu können, bauen wir eine `main`-Methode ein und nutzen darin die Klasse `Mainframe`. Den *Scene Graph* konstruieren wir mit der Methode `createSceneGraph()`. Damit hat unsere Klasse `HelloUniverse` folgende Grundstruktur:

`createSceneGraph()`

Listing 3.3: Grundstrukur von `HelloUniverse`

```
public class HelloUniverse extends Applet
{
   private SimpleUniverse u = null;

   private HelloUniverse() {...};

   public BranchGroup createSceneGraph()
   {

      ...
      Cylinder cylinder = new Cylinder(...)
      ...

   }


   public void init()
   {
      u = new SimpleUniverse(...);
      ...
      u.addBranchGraph(this.createSceneGraph());

   }


   public void destroy()
   {
      u.cleanup();
   }


   public static void main(String[] args)
   {
      new MainFrame(new HelloUniverse(), 300, 400);
   }

}
```

Mit Hilfe eines Objektes der Klasse `Canvas3D` lösen wir die Aufgabe eine dreidimensionale Szene auf der graphischen Benutzungsoberfläche [4] darzustellen. Zur Konstruktion dieses Objektes sind Daten über die konkrete graphische Konfiguration erforderlich. Diese erhalten wir mit Hilfe der Methode `getPreferredConfiguration()` von der Klasse `SimpleUniverse`. Außerdem gilt es, die Position des Beobachters, also die Position der `ViewingPlatform`, zu spezifizieren. Ohne Spezifikation liegt diese Position im Koordinatennullpunkt, also bei $P(0,0,0)$. Mit der Methode `setNominalViewingTransform()` modifizieren wir diese Position, so dass sich der Beobachter etwas entfernt vom Nullpunkt des `SimpleUniverse` befindet.[5] Die `init`-Methode ist damit wie folgt konstruiert:

Listing 3.4: Detail `init()`

```
public void init()
{
    this.setLayout(new BorderLayout());
    GraphicsConfiguration config =
          SimpleUniverse.
              getPreferredConfiguration();
    Canvas3D c = new Canvas3D(config);
    this.add("Center", c);
    u = new SimpleUniverse(c);
    u.getViewingPlatform().
        setNominalViewingTransform();
    u.addBranchGraph(
        this.createSceneGraph());

}
```

Mit der Methode `createSceneGraph()` erzeugen wir ein Objekt `bg` der Klasse `BranchGroup`, modifizieren dieses durch das Hinzufügen eines Zylinders, dessen Ausrichtung und Aussehen vorher spezifiziert wurde und geben `bg` aus Performance-Gründen in compilierter Form [6] als Wert zurück.

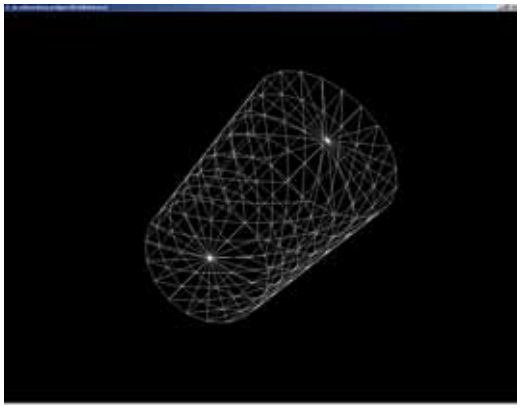
Listing 3.5: Detail `createSceneGraph()`

```
public BranchGroup createSceneGraph()
{
    BranchGroup bg = new BranchGroup();

    // Ausrichtung
    TransformGroup tg = new TransformGroup();
    ...
    tg.setTransform(...);

    // Aussehen
    Appearance app = new Appearance();
```

[4] Häufig auch als „Benutzeroberfläche“ bezeichnet — obwohl die Oberfläche des Benutzers Haut ist.

[5] Die *Nominal Viewing Distance* beträgt ungefähr $2,4m$. In dieser Entfernung füllen Objekte von einer Höhe bzw. Breite von $2m$ die „Bildplatte“ (*image plate*).

[6] Hinweis: Mit der Compilierung unterliegt der *Scene Graph* einigen Einschränkungen. Beispielsweise lassen sich die compilierten Elemente nur noch bedingt verändern.

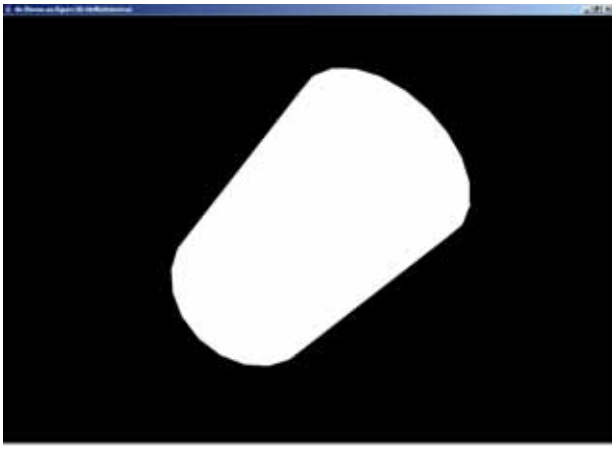


Legende:
Quellcode ↪ S. 52

Abbildung 3.7: Beispiel: `HelloUniverse`

```
12          ...
            Material m = new Material(...);
14          // Licht einschalten
            m.setLightingEnable(true);
16          app.setMaterial(m);

18          ...

20          // Zylinder mit Aussehen erzeugen
            Cylinder cylinder =
22                new Cylinder(
                  0.3f, // radius
24                0.9f, // height
                  Primitive.GENERATE_NORMALS,
26                20,   // xDivision
                  7,    // yDivision
28                app   // Aussehen
                  );
30
            // Zylinder transformieren
32          tg.addChild(cylinder);

34          // Zylinder in BranchGroup-Knoten üeinfgen
            bg.addChild(tg);
36          bg.compile();
            return bg;
38      }
```

Das Ergebnis einer solchen Methode `createSceneGraph()` zeigt Abbildung 3.8 S. 50. Dieses Ergebnis ist kaum beeindruckender als eine übliche „Hello World“ Textausgabe. Wir ergänzen daher die Methode `createSceneGraph()` durch eine Modifikation der Instanz `app`.



Legende:
Quellcode ↪ S. 52
— Ergebnis mit vereinfachter Methode `createSceneGraph()` (↪ S.48)

Abbildung 3.8: Beispiel: `HelloUniverse` — vereinfacht

Unsere Änderung des Aussehens bringt ein leichteres Verstehen für die Polygone, aus denen der Zylinder konstruiert ist und zwar insbesondere für deren Spitzen („Ecken“), den sogenannten *Vertices*. Wir stellen den Zylinder jetzt mit sichtbaren *Vertices* dar und zwar aus 20 Abschnitten für den Umfang des Grundkreises (`xDivision`-Wert) und 7 Abschnitten zur Einteilung der Zylindermantelfläche (`yDivision`-Wert).

Listing 3.6: Detail `PolygonAttributes`

```
...
2       PolygonAttributes polyAtt =
              new PolygonAttributes();
4       polyAtt.setPolygonMode(
              PolygonAttributes.POLYGON_LINE);
6       polyAtt.setCullFace(
              PolygonAttributes.CULL_NONE);
8
        app.setPolygonAttributes(polyAtt);
10  ...
```

Face-Culling

Der Wert `PolygonAttributes.POLYGON_LINE` sorgt dafür, das nur die Verbindungslinien zwischen den einzelnen *Vertices* gezeichnet werden. Das Ergebnis ist daher eine Darstellung als „Drahtgitter“. Mit dem Wert `PolygonAttributes.` Mit `CULL_NONE` verhindern wir ein *Face-Culling*. Die Basis der Java3D-Maschine, meist `OpenGL` oder `DirectX`, führt ein *Face-Culling* durch, das heißt, bestimmte *Faces*, also Polygone, werden entfernt. Beispielsweise wird beim *Backface-Culling*, die Rückseite eines Objektes nicht gezeichnet.

Zum Fertigstellen der Klasse `HelloUniverse` bedarf es dann nur noch der Ergänzung des selbst gewählten Paketnamens, hier `de.unilueneburg.as.figure3D` und der Importierung der genutzten Klassen, also folgender Ergänzungen:

Listing 3.7: Detail *Pakete*

```
package de.fhnon.as.figure3D;

import java.applet.Applet;
import java.awt.BorderLayout;
import java.awt.GraphicsConfiguration;

import com.sun.j3d.utils.applet.MainFrame;

import com.sun.j3d.utils.geometry.Primitive;
import com.sun.j3d.utils.geometry.Cylinder;
import com.sun.j3d.utils.universe.SimpleUniverse;

import javax.media.j3d.Appearance;
import javax.media.j3d.BranchGroup;
import javax.media.j3d.Canvas3D;
import javax.media.j3d.Material;
import javax.media.j3d.TransformGroup;
import javax.media.j3d.Transform3D;
import javax.media.j3d.PolygonAttributes;

import javax.vecmath.AxisAngle4f;
import javax.vecmath.Color3f;
```

Nicht jedes Konstrukt in der Klasse `HelloUniverse` ist damit hinreichend erläutert. Beispielsweise bedarf die Klasse `AxisAngle4f` einer Erklärung. Sie ermöglicht eine Drehung um ein definiertes Koordinatensystem. Die Details sind in der *Java3D-API* [7] wie folgt erläutert: *A four-element axis angle represented by single-precision floating point* x, y, z, angle *components. An axis angle is a rotation of angle (radians) about the vector (x,y,z).*

Beim Konstruktor `AxisAngle4f` wird der Winkel, um den gedreht werden soll, über den vierten Parameter angegeben, hier um 90 Grad und zwar mit der Angabe `(float) Math.toRadians(90)`. Die Drehachse selbst legen die ersten drei Parameter fest. Sie sind für die jeweils gewünschte Achse auf den Wert 1f zu setzen, hier also eine Drehung um die x- und die y-Achse.

Auch die Klasse `Material` mit ihren mannigfaltigen Optionen bedarf noch einer eingehenden Erläuterung. Hier seien nur ganz holzschnittartig die Parameter des genutzten Konstruktors skizziert:

[7] In meiner Installation unter
↪ `file:///D:/bonin/myJava/j3dapi/index.html`.
Quelle für die Java3D-API
↪ `http://java3d.virtualworlds.de` (online 20-Apr-2004)

- `ambientColor` spezifiziert die Farbe, die mit dem gleichmäßigen Umgebungslicht korrespondiert.
- `emissiveColor` spezifiziert die Farbe, in der das 3D-Objekt selbst leuchtet.
- `diffuseColor` spezifiziert die Farbe, die reflektiert wird, wenn das Objekt beleuchtet wird.
- `specularColor` spezifiziert die Farbe des Glanzpunktes des Objektes.
- `shininess` ist ein Faktor im Bereich von 1..128, der das Reflexionsverhalten des Materials bestimmt.

Aus Vereinfachungsgründen ist hier stets die Farbe weiß gewählt. Die Klasse `com.sun.j3d.utils.applet.MainFrame` ermöglicht den Aufruf eines Applets als Java-Applikation, wobei der Wert:

- des ersten Parameters das Applet ist, hier durch seinen Konstruktor `HelloUniverse()` erzeugt,
- des zweiten Parameters die Breite des Applets in Pixel, hier 300 und
- des dritten Parameters die Höhe in Pixel, hier 400, angibt.

Listing 3.8: `HelloUniverse`

```
/**
 *  "Java 3D Example" Hello Universe
 *
 *@author       Bonin
 *@version      1.1
 */

package de.unilueneburg.as.figure3D;

import java.applet.Applet;
import java.awt.BorderLayout;
import java.awt.GraphicsConfiguration;

import com.sun.j3d.utils.applet.MainFrame;

import com.sun.j3d.utils.geometry.Primitive;
import com.sun.j3d.utils.geometry.Cylinder;
import com.sun.j3d.utils.universe.SimpleUniverse;

import javax.media.j3d.Appearance;
import javax.media.j3d.BranchGroup;
import javax.media.j3d.Canvas3D;
import javax.media.j3d.Material;
import javax.media.j3d.TransformGroup;
import javax.media.j3d.Transform3D;
import javax.media.j3d.PolygonAttributes;
```

```
28 import javax.vecmath.AxisAngle4f;
   import javax.vecmath.Color3f;
30
   public class HelloUniverse extends Applet
32 {
      public SimpleUniverse u = null;
34

36    public HelloUniverse()
      {
38       super();
      }
40

42    public BranchGroup createSceneGraph()
      {
44       BranchGroup bg = new BranchGroup();

46       // Zum Transformieren der untergeordneten Knoten
         TransformGroup tg = new TransformGroup();
48
         // Spezifiziert die Transformation, hier Drehung
50       Transform3D t3d = new Transform3D();
         t3d.setRotation(
52             new AxisAngle4f(
               1f,
54             1f,
               0f,
56             (float) Math.toRadians(90)));
         tg.setTransform(t3d);
58
         Appearance app = new Appearance();
60
         Color3f white = new Color3f(1.0f, 1.0f, 1.0f);
62       Color3f ambientColor = white;
         Color3f emissiveColor = white;
64       Color3f diffuseColor = white;
         Color3f specularColor = white;
66       float shininess = 64.0f;

68       Material m = new Material(
               ambientColor,
70             emissiveColor,
               diffuseColor,
72             specularColor,
               shininess);
74
         m.setLightingEnable(true);
76       app.setMaterial(m);

78       PolygonAttributes polyAtt =
               new PolygonAttributes();
80       polyAtt.setPolygonMode(
               PolygonAttributes.POLYGON_LINE);
82       polyAtt.setCullFace(
```

```
            PolygonAttributes.CULL_NONE);
84
        app.setPolygonAttributes(polyAtt);
86
        Cylinder cylinder =
88          new Cylinder(
            0.3f,
90          0.9f,
            Primitive.GENERATE_NORMALS,
92          20,
            7,
94          app);

96      tg.addChild(cylinder);

98      bg.addChild(tg);
        bg.compile();
100     return bg;
    }
102

104 public void init()
    {
106     this.setLayout(new BorderLayout());
        GraphicsConfiguration config =
108         SimpleUniverse.getPreferredConfiguration();
        Canvas3D c = new Canvas3D(config);
110     this.add("Center", c);
        u = new SimpleUniverse(c);
112     u.getViewingPlatform().
            setNominalViewingTransform();
114     u.addBranchGraph(
            this.createSceneGraph());
116
    }
118

120 public void destroy()
    {
122     u.cleanup();
    }
124

126 public static void main(String[] args)
    {
128     new MainFrame(
            new HelloUniverse(), 300, 400);
130 }

132 }
```

Protokoll `HelloUniverse.log`

```
D:\bonin\artsprog\code>java -version
java version "1.4.2_03"
Java(TM) 2 Runtime Environment,
```

```
  Standard Edition (build 1.4.2_03-b02)
Java HotSpot(TM) Client VM
  (build 1.4.2_03-b02, mixed mode)

D:\bonin\artsprog\code>javac de/fhnon/as/figure3D/HelloUniverse.java

D:\bonin\artsprog\code>java de.fhnon.as.figure3D.HelloUniverse

D:\bonin\artsprog\code>appletviewer HelloUniverse.html

D:\bonin\artsprog\code>
```

3.3.2 HelloWorld

In der Java3D-Welt unterscheiden wir zwei Formen von Texten; einen Text einerseits konstruiert mit der Klasse `com.sun.j3d.utils.geometry.Text2D` und andererseits konstruiert mit der Klasse `javax.media.j3d.Text3D`. Ein `Text2D`-Objekt besteht aus rechteckigen Polygonen, wobei der Text durch Anwendung einer Textur realisiert ist. Im Abschnitt 3.15 S. 79 nutzen wir ein solches `Text2D`-Objekt. Es wird wie folgt konstruiert:

Text2D

```
Text2D textObject = new Text2D(
  java.lang.String text, // z.B. "Bonin"
  Color3f color, // z.B. new Color3f(0f, 0f, 0f)
  java.lang.String fontName, // z.B. "Serif"
  int fontSize, // z.B. 120
  int fontStyle); // z.B. Font.BOLD
```

Ein `Text3D`-Objekt ist ein geometrisches 3D-Objekt. Die textliche Geometrie ist eine Extrusion[8] von einem „normalen“ Font. Daher wird zunächst ein entsprechender räumlicher Font auf der Basis eines üblichen AWT-Fonts konstruiert. Dies erfolgt mit der Klasse `javax.media.j3d.Font3D` folgendermaßen:

```
Font3D font3D = new Font3D(
  java.awt.Font font,
    // z.B. new Font("Times", Font.PLAIN, 1)
  FontExtrusion extrudePath);
    // z.B. new FontExtrusion()
```

Der 3D-Text entsteht dann mit Hilfe der Klasse `javax.media.j3d.Font3D` wie folgt:

Text3D

```
Text3D textGeom = new Text3D(
  Font3D font3D,    // z.B. font3D siehe oben
  String string,    // z.B. new String("Hello World!")
  Piont3f position, // z.B. new Point3f(0.0f, 0.0f, 0.0f)
  int alignment,    // z.B. Text3D.ALIGN_CENTER
  int path);        // z.B. Text3D.PATH_RIGHT
```

[8] Der Begriff *Extrusion* wird häufig im Zusammenhang mit Pressvorgängen bei der Metallverarbeitung genutzt.

Legende:
Quellcode ↪ S. 56

Abbildung 3.9: Beispiel: `HelloWorld`

Listing 3.9: `HelloWorld`

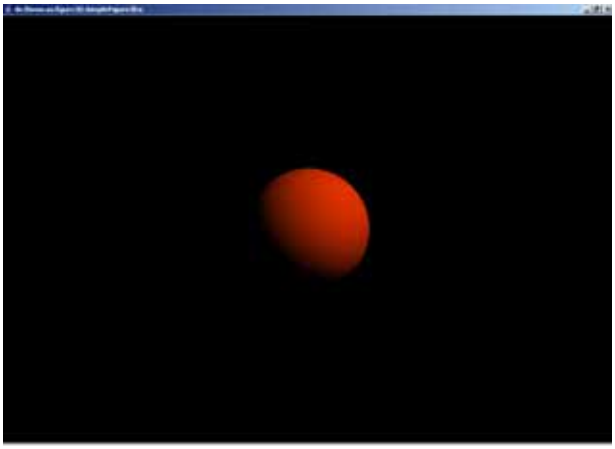
```
/**
 *  "Java 3D Example" Hello World
 *
 *@author       Bonin
 *@version      1.0
 */

package de.fhnon.as.figure3D;

import java.applet.Applet;
import java.awt.BorderLayout;
import java.awt.GraphicsConfiguration;
import java.awt.Font;

import com.sun.j3d.utils.applet.MainFrame;
import com.sun.j3d.utils.geometry.Primitive;
import com.sun.j3d.utils.geometry.Cylinder;
import com.sun.j3d.utils.universe.SimpleUniverse;

import javax.media.j3d.Appearance;
import javax.media.j3d.BranchGroup;
import javax.media.j3d.Canvas3D;
import javax.media.j3d.Font3D;
import javax.media.j3d.FontExtrusion;
import javax.media.j3d.Material;
import javax.media.j3d.Shape3D;
import javax.media.j3d.Text3D;
import javax.media.j3d.TransformGroup;
import javax.media.j3d.Transform3D;

```

```
import javax.vecmath.AxisAngle4f;
import javax.vecmath.Color3f;
import javax.vecmath.Point3f;

public class HelloWorld extends Applet
{
    private SimpleUniverse u = null;


    private HelloWorld()
    {
        super();
    }


    public BranchGroup createSceneGraph()
    {
        BranchGroup bg = new BranchGroup();
        // Zum Transformieren der untergeordneten Knoten
        TransformGroup tg = new TransformGroup();

        // Spezifiziert die Transformation
        Transform3D t3d = new Transform3D();
        // hier Drehung
        t3d.setRotation(
                new AxisAngle4f(
                1f,
                1f,
                0f,
                (float) Math.toRadians(45)));
        // ßMastab
        t3d.setScale(0.3);
        tg.setTransform(t3d);

        Font3D font3D = new Font3D(
                new Font("Times", Font.PLAIN, 1),
                new FontExtrusion());
        Text3D textGeom = new Text3D(
                font3D,
                new String("Hello World!"),
                new Point3f(0.0f, 0.0f, 0.0f),
                Text3D.ALIGN_CENTER,
                Text3D.PATH_RIGHT);

        Appearance app = new Appearance();

        Material m = new Material();
        m.setEmissiveColor(
                new Color3f(0.0f, 0.0f, 1.0f));
        m.setLightingEnable(true);
        app.setMaterial(m);

        Shape3D textObject = new Shape3D(textGeom, app);

        tg.addChild(textObject);

```

```
        bg.addChild(tg);
88      bg.compile();
        return bg;
90  }

92
    public void init()
94  {
        this.setLayout(new BorderLayout());
96      GraphicsConfiguration config =
              SimpleUniverse.getPreferredConfiguration();
98      Canvas3D c = new Canvas3D(config);
        this.add("Center", c);
100     u = new SimpleUniverse(c);
        u.getViewingPlatform().
102           setNominalViewingTransform();
        u.addBranchGraph(
104           this.createSceneGraph());

106 }

108
    public void destroy()
110 {
        u.cleanup();
112 }

114
    public static void main(String[] args)
116 {
        new MainFrame(
118           new HelloWorld(), 1000, 700);
    }
120
}
```

Protokoll `HelloWorld.log`

```
D:\bonin\artsprog\code>java -version
java version "1.4.2_03"
Java(TM) 2 Runtime Environment,
  Standard Edition (build 1.4.2_03-b02)
Java HotSpot(TM) Client VM
  (build 1.4.2_03-b02, mixed mode)

D:\bonin\artsprog\code>javac
  de/fhnon/as/figure3D/HelloWorld.java

D:\bonin\artsprog\code>java
  de.fhnon.as.figure3D.HelloWorld

D:\bonin\artsprog\code>
```



Legende:
Quellcode ↪ S. 59

Abbildung 3.10: Beispiel: `SimpleFigure3Da`

3.3.3 `SimpleFigure3D` — 3 Fälle

Die Beispielsidee ist der Java3D-Distribution entnommen und wird auch von Daniel Selman beschrieben (↪ [Selman02] S. 30–36). Der Quellcode der folgenden drei Varianten (`a`, `b` und `c`) ist von mir gestaltet.

Angestrahlte Kugel — 1. Fall

Wir strahlen mit einem Licht, erzeugt mit der Klasse `javax.media.j3d.DirectionalLight`, unser 3D-Objekt an. Das Licht wirkt nur auf das Objekt, wenn dieses in seinen spezifizierten Einflussgrenzen, den sogenannten *Influencing Bounds*, liegt. Diese Einflussgrenzen der jeweiligen Lichquelle spezifizieren wir hier ganz einfach als eine Kugel im Nullpunkt mit dem Radius 200*m*. Diese Grenzkugel selbst ist nicht sichtbar. Sie begrenzt nur den Bereich unserer Lichtquelle.

Influencing Bounds

Unser Licht kommt aus einer Richtung, die wir mit Hilfe eines Vektors festlegen. Die Lichtquelle selbst befindet sich in unendlicher Entfernung. Ihre Lichtstrahlen laufen daher parallel. Es findet keine entfernungsabhängige Änderung des Lichtes statt.

Listing 3.10: `SimpleFigure3Da`

```
/**
 * "Java 3D Example" Einfache Kugel
 *
 *@author      Bonin
 *@version     1.1
```

```
6   */

8   package de.fhnon.as.figure3D;

10  import java.applet.Applet;
    import java.awt.BorderLayout;
12  import java.awt.GraphicsConfiguration;

14  import com.sun.j3d.utils.geometry.Primitive;
    import com.sun.j3d.utils.geometry.Sphere;
16  import com.sun.j3d.utils.universe.SimpleUniverse;

18  import com.sun.j3d.utils.applet.MainFrame;

20  import javax.media.j3d.Appearance;
    import javax.media.j3d.BranchGroup;
22  import javax.media.j3d.BoundingSphere;
    import javax.media.j3d.Canvas3D;
24  import javax.media.j3d.DirectionalLight;
    import javax.media.j3d.Material;
26  import javax.media.j3d.TransformGroup;

28  import javax.vecmath.Color3f;
    import javax.vecmath.Point3d;
30  import javax.vecmath.Vector3f;

32  public class SimpleFigure3Da extends Applet
    {
34      private SimpleUniverse u = null;

36
        private SimpleFigure3Da()
38      {
            super();
40      }

42
        public BranchGroup createSceneGraph()
44      {
            BranchGroup bg = new BranchGroup();
46          Appearance app = new Appearance();
            Color3f ambientC =
48                  new Color3f(0.9f, 0.2f, 1.0f);
            Color3f emissiveC =
50                  new Color3f(0.0f, 0.0f, 0.0f);
            Color3f diffuseC =
52                  new Color3f(0.9f, 0.2f, 1.0f);
            Color3f specularC =
54                  new Color3f(0.0f, 0.0f, 0.0f);
            float shininess = 80.0f;
56
            app.setMaterial(
58                  new Material(
                    ambientC,
60                  emissiveC,
                    diffuseC,
```

```
62              specularC,
                shininess));
64          Sphere sphere =
                new Sphere(
66              0.2f, Primitive.GENERATE_NORMALS, 40, app);
            bg.addChild(sphere);
68          return bg;
        }
70

72      public void addLights(BranchGroup bg)
        {
74          DirectionalLight light =
                new DirectionalLight(
76              new Color3f(1.0f, 1.0f, 0.0f),
                new Vector3f(-1.0f, -1.0f, -1.0f));
78          light.setInfluencingBounds(
                this.getBoundingSphere());
80          bg.addChild(light);
        }
82

84      public TransformGroup createBehaviors(
            BranchGroup bg)
86      {
            TransformGroup objTrans =
88              new TransformGroup();
            bg.addChild(objTrans);
90          return objTrans;
        }
92

94      BoundingSphere getBoundingSphere()
        {
96          return new BoundingSphere(
                new Point3d(0.0, 0.0, 0.0), 200.0);
98      }

100

102     public void init()
        {
104         this.setLayout(new BorderLayout());
            GraphicsConfiguration config =
106             SimpleUniverse.getPreferredConfiguration();
            Canvas3D c = new Canvas3D(config);
108         this.add("Center", c);
            u = new SimpleUniverse(c);
110         u.getViewingPlatform().
                setNominalViewingTransform();
112
            BranchGroup bgRoot = new BranchGroup();
114         TransformGroup tg = this.createBehaviors(bgRoot);
            tg.addChild(this.createSceneGraph());
116         this.addLights(bgRoot);
            u.addBranchGraph(bgRoot);
```

```
118
        }
120
122     public void destroy()
        {
124        u.cleanup();
        }
126
128     public static void main(String[] args)
        {
130        new MainFrame(new SimpleFigure3Da(), 300, 400);
        }
132
}
```

Protokoll `SimpleFigure3Da.log`

```
D:\bonin\artsprog\code>java -version
java version "1.4.2"
Java(TM) 2 Runtime Environment,
  Standard Edition (build 1.4.2-b28)
Java HotSpot(TM) Client VM
  (build 1.4.2-b28, mixed mode)

D:\bonin\artsprog\code>javac
  de/fhnon/as/figure3D/SimpleFigure3Da.java

D:\bonin\artsprog\code>java
  de.fhnon.as.figure3D.SimpleFigure3Da

D:\bonin\artsprog\code>
```

Angestrahlte Kugel mit Hintergrund — 2. Fall

Feine Strukturen und Farbenvielfalt zeichnen Objekte der realen Welt aus. Beispielsweise hat ein Ziegelmauerwerk eine große Menge von unterschiedlichen Farben und Steinstrukturen (↪ Abbildung 3.11 S. 63). Dieses hochkomplexe Muster orginalgetreu mit entsprechenden Polygonen und Farbzuweisungen wiederzugeben, wäre zu aufwendig, das heißt, der Ressourcenverbrauch wäre nicht akzeptabel. Man löst dieses Darstellungsproblem mit Hilfe von sogenannten Texturen. Eine Textur ist ein normales Foto (Bild) von der Oberfläche des realen Objektes. Wir legen dann eine solche Textur auf das spezifizierte 3D-Objekt. Für eine Objektdarstellung benötigen wir dann nur Polygone um das 3D-Objekt zu spezifizieren, aber nicht um seine Oberfläche zu beschreiben. Dazu reicht seine Textur. Damit reduzieren wir den Ressourcenverbrauch erheblich. Geboten ist dieses Verfahren insbesondere bei Bewegtbildern.

Textur

Das Oberflächenbild laden wir mit der Klasse `com.sun.j3d.utils.image.TextureLoader`. In welcher Lage und Position das Bild auf das Objekt gelegt werden soll, spezifizieren wir mit Hilfe der Klasse `javax.media.j3d.TexCoordGeneration`. Mit dem ersten Parameterwert

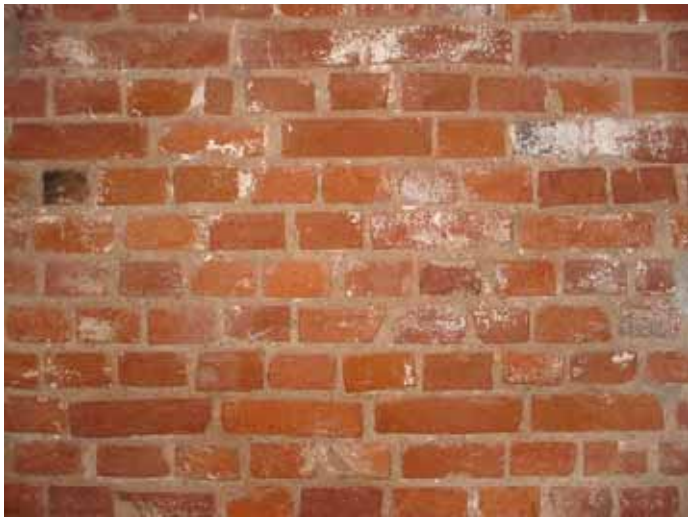
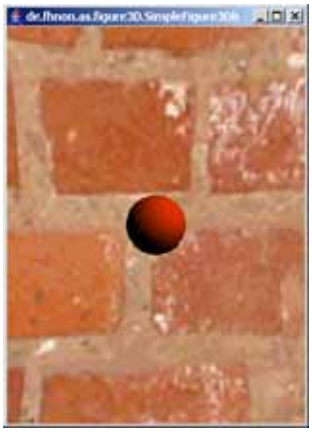


Abbildung 3.11: Hintergrundbild: `brickwork.jpg`

`SPHERE_MAP` sorgen wir für eine sphärische Projektion. Mit dem zweiten Parameterwert `TEXTURE_COORDINATE_2` spezifizieren wir eine zweidimensionale Textur. Mit der Klasse `javax.media.j3d.TextureAttributes` beeinflussen wir den Vorgang zum Beispiel im Hinblick auf die Qualität. Mit dem Parameterwert `TextureAttributes.NICEST` geben wir der Qualität Vorzug vor der Geschwindigkeit (`FASTEST`).

Listing 3.11: `SimpleFigure3Db`

```
/**
 *  "Java 3D Example" Kugel mit Hintergrund
 *
 *@author      Bonin
 *@version     1.1
 */

package de.fhnon.as.figure3D;

import java.applet.Applet;
import java.awt.BorderLayout;
import java.awt.GraphicsConfiguration;

import com.sun.j3d.utils.geometry.Primitive;
import com.sun.j3d.utils.geometry.Sphere;

import com.sun.j3d.utils.image.TextureLoader;
import com.sun.j3d.utils.universe.SimpleUniverse;

import com.sun.j3d.utils.applet.MainFrame;
```



Legende:
Quellcode ↪ S. 63
Textur ↪ Abbildung 3.11 S. 63

Abbildung 3.12: Beispiel: `SimpleFigure3Db` — Textur `brickwork.jpg`

```
22 import javax.media.j3d.Appearance;
   import javax.media.j3d.Background;
24
   import javax.media.j3d.BranchGroup;
26 import javax.media.j3d.BoundingSphere;
   import javax.media.j3d.Canvas3D;
28 import javax.media.j3d.DirectionalLight;
   import javax.media.j3d.Material;
30 import javax.media.j3d.TexCoordGeneration;

32 import javax.media.j3d.Texture;
   import javax.media.j3d.TextureAttributes;
34
   import javax.media.j3d.Transform3D;
36 import javax.media.j3d.TransformGroup;

38 import javax.vecmath.Color3f;
   import javax.vecmath.Color4f;
40 import javax.vecmath.Point3d;
   import javax.vecmath.Vector3f;
42
   public class SimpleFigure3Db extends Applet
44 {
      private SimpleUniverse u = null;
46

48    private SimpleFigure3Db()
      {
50       super();
```

```
    }
52

54  public BranchGroup createSceneGraph()
    {
56      BranchGroup bg = new BranchGroup();
        Appearance app = new Appearance();
58      Color3f ambientC =
            new Color3f(0.9f, 0.2f, 1.0f);
60      Color3f emissiveC =
            new Color3f(0.0f, 0.0f, 0.0f);
62      Color3f diffuseC =
            new Color3f(0.9f, 0.2f, 1.0f);
64      Color3f specularC =
            new Color3f(0.0f, 0.0f, 0.0f);
66      float shininess = 80.0f;

68      app.setMaterial(
            new Material(
70          ambientC,
            emissiveC,
72          diffuseC,
            specularC,
74          shininess));
        Sphere sphere =
76          new Sphere(
            0.2f, Primitive.GENERATE_NORMALS, 40, app);
78      bg.addChild(sphere);
        return bg;
80  }

82
    public BranchGroup createBackground()
84  {
        BranchGroup bg =
86          new BranchGroup();
        Background back = new Background();
88      back.setApplicationBounds(
            getBoundingSphere());
90      BranchGroup bgGeometry =
            new BranchGroup();
92      Appearance app = new Appearance();

94      Texture tex = new TextureLoader(
            "de/fhnon/as/figure3D/brickwork.jpg",
96          null).getTexture();
        app.setTexture(tex);
98
        app.setTexCoordGeneration(
100         new TexCoordGeneration(
            TexCoordGeneration.SPHERE_MAP,
102         TexCoordGeneration.TEXTURE_COORDINATE_2));
        app.setTextureAttributes(
104         new TextureAttributes(
            TextureAttributes.REPLACE,
106         new Transform3D(),
```

```
            new Color4f(),
108         TextureAttributes.NICEST));

110     Sphere sphere = new Sphere(1.0f,
            Primitive.GENERATE_TEXTURE_COORDS |
112         Primitive.GENERATE_NORMALS_INWARD, 40, app);

114     bgGeometry.addChild(sphere);
        back.setGeometry(bgGeometry);
116     bg.addChild(back);
        return bg;
118 }

120
    public void addLights(BranchGroup bg)
122 {
        DirectionalLight light =
124         new DirectionalLight(
            new Color3f(1.0f, 1.0f, 0.0f),
126         new Vector3f(-1.0f, -1.0f, -1.0f));
        light.setInfluencingBounds(
128         this.getBoundingSphere());
        bg.addChild(light);
130 }

132
    public TransformGroup createBehaviors(
134         BranchGroup bg)
    {
136     TransformGroup objTrans =
            new TransformGroup();
138     bg.addChild(objTrans);
        return objTrans;
140 }

142
    BoundingSphere getBoundingSphere()
144 {
        return new BoundingSphere(
146         new Point3d(0.0, 0.0, 0.0), 200.0);
    }
148

150 public void init()
    {
152     this.setLayout(new BorderLayout());
        GraphicsConfiguration config =
154         SimpleUniverse.getPreferredConfiguration();
        Canvas3D c = new Canvas3D(config);
156     this.add("Center", c);
        u = new SimpleUniverse(c);
158     u.getViewingPlatform().
            setNominalViewingTransform();
160     u.addBranchGraph(this.createBackground());

162     BranchGroup bgRoot = new BranchGroup();
```

```
        TransformGroup tg = this.createBehaviors(bgRoot);
164     tg.addChild(this.createSceneGraph());
        this.addLights(bgRoot);
166     u.addBranchGraph(bgRoot);

168  }

170
     public void destroy()
172  {
        u.cleanup();
174  }

176
     public static void main(String[] args)
178  {
        new MainFrame(new SimpleFigure3Db(), 300, 400);
180  }

182 }
```

Protokoll `SimpleFigure3Db.log`

```
D:\bonin\artsprog\code>java -version
java version "1.4.2"
Java(TM) 2 Runtime Environment,
  Standard Edition (build 1.4.2-b28)
Java HotSpot(TM) Client VM
  (build 1.4.2-b28, mixed mode)

D:\bonin\artsprog\code>javac de/fhnon/as/figure3D/SimpleFigure3Db.java

D:\bonin\artsprog\code>java de.fhnon.as.figure3D.SimpleFigure3Db

D:\bonin\artsprog\code>
```

Wir ändern die Bilddatei zur Schaffung der Textur um die Auswirkungen auf das Ergebnis zu verdeutlichen. Nun nutzen wir die Datei `code.jpg` (↪ Abbildung 3.13 S. 68). Das Ergebnis zeigt ↪ Abbildung 3.14 S. 69.

Animierte Kugel — 3. Fall

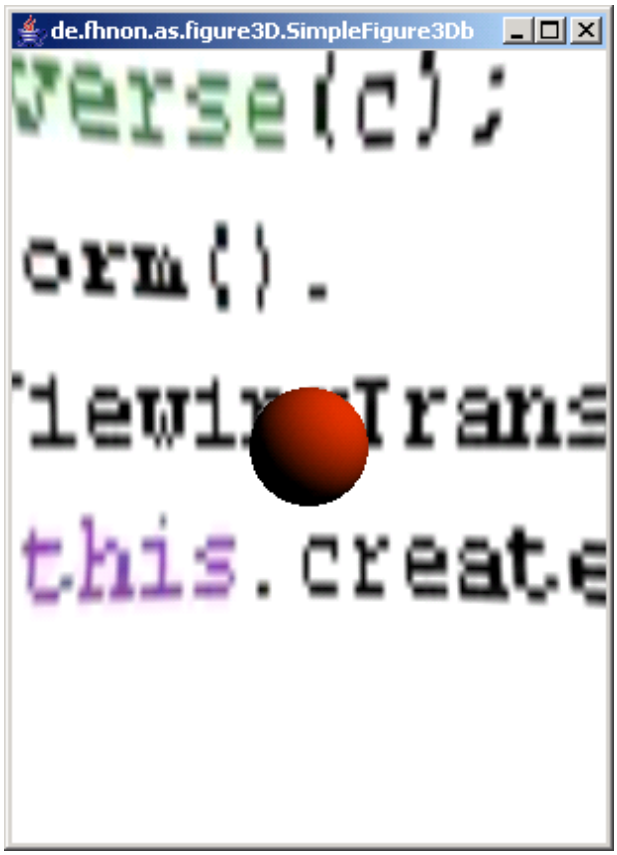
Listing 3.12: `SimpleFigure3Dc`

```
/**
2  *  "Java 3D Example"
   *  Animierte Kugel mit
4  * Zylinder und Kegel erweitert
   *
6  *@author      Bonin
   *@version     1.1
8  */

10 package de.fhnon.as.figure3D;
```

```
public void init()
{
   this.setLayout(new BorderLayout());
   GraphicsConfiguration config =
         SimpleUniverse.getPreferredConfiguration();
   Canvas3D c = new Canvas3D(config);
   this.add("Center", c);
   u = new SimpleUniverse(c);
   u.getViewingPlatform().
         setNominalViewingTransform();
   u.addBranchGraph(this.createBackground());

   BranchGroup bgRoot = new BranchGroup();
   TransformGroup tg = this.createBehaviors(bgRoot);
   tg.addChild(this.createSceneGraph());
   this.addLights(bgRoot);
   u.addBranchGraph(bgRoot);

}
```

Abbildung 3.13: Hintergrundbild: `code.jpg`

```
12 import java.applet.Applet;
   import java.awt.BorderLayout;
14 import java.awt.GraphicsConfiguration;

16 import com.sun.j3d.utils.geometry.Cone;
   import com.sun.j3d.utils.geometry.Cylinder;
18 import com.sun.j3d.utils.geometry.Primitive;
   import com.sun.j3d.utils.geometry.Sphere;
20
   import com.sun.j3d.utils.image.TextureLoader;
22 import com.sun.j3d.utils.universe.SimpleUniverse;

24 import com.sun.j3d.utils.applet.MainFrame;

26 import javax.media.j3d.Alpha;
   import javax.media.j3d.Appearance;
28 import javax.media.j3d.Background;

30 import javax.media.j3d.BranchGroup;
   import javax.media.j3d.BoundingSphere;
32 import javax.media.j3d.Canvas3D;
   import javax.media.j3d.DirectionalLight;
34 import javax.media.j3d.Material;
   import javax.media.j3d.PositionInterpolator;
36 import javax.media.j3d.TexCoordGeneration;
   import javax.media.j3d.Texture;
38 import javax.media.j3d.TextureAttributes;
   import javax.media.j3d.Transform3D;
40 import javax.media.j3d.TransformGroup;

42 import javax.vecmath.Color3f;
   import javax.vecmath.Color4f;
44 import javax.vecmath.Point3d;
```



Legende:
Quellcode ↪ S. 63
Textur ↪ Abbildung 3.13 S. 68

Abbildung 3.14: Beispiel: `SimpleFigure3Db` — Textur `code.jpg`

```
import javax.vecmath.Vector3f;
46
/**
48  *  Description of the Class
    *
50  *@author       bonin
    *@created      14. September 2004
52  */
public class SimpleFigure3Dc extends Applet {
54      private SimpleUniverse u = null;

56
        /**
58       *  Constructor for the SimpleFigure3Dc object
         */
60      private SimpleFigure3Dc() {
            super();
62      }

64
        /**
66       *  Description of the Method
         *
68       *@return     Description of the Return Value
         */
70      public BranchGroup createSceneGraph() {
            BranchGroup bg = new BranchGroup();
72          Appearance app = new Appearance();
            Color3f ambientC =
74                      new Color3f(0.9f, 0.2f, 1.0f);
            Color3f emissiveC =
76                      new Color3f(0.0f, 0.0f, 0.0f);
            Color3f diffuseC =
78                      new Color3f(0.9f, 0.2f, 1.0f);
            Color3f specularC =
80                      new Color3f(0.0f, 0.0f, 0.0f);
            float shininess = 80.0f;
82
            app.setMaterial(
84                      new Material(
                        ambientC,
86                      emissiveC,
                        diffuseC,
88                      specularC,
                        shininess));
90
            Sphere sphere =
92                      new Sphere(
                        0.1f,
94                      Primitive.GENERATE_NORMALS,
                        40,
96                      app);
            Cylinder cylinder =
98                      new Cylinder(
                        0.05f, 0.2f,
100                     Primitive.GENERATE_NORMALS,
```

```
                                40, 40,
102                             app);
            Cone cone =
104                             new Cone(
                                0.05f, 0.6f,
106                             Primitive.GENERATE_NORMALS,
                                40, 40,
108                             app);
            bg.addChild(sphere);
110         bg.addChild(cylinder);
            bg.addChild(cone);
112         return bg;
        }
114

116     /**
         *  Description of the Method
118      *
         *@return     Description of the Return Value
120      */
        public BranchGroup createBackground() {
122         BranchGroup bg =
                                new BranchGroup();
124         Background back = new Background();
            back.setApplicationBounds(
126                             getBoundingSphere());
            BranchGroup bgGeometry =
128                             new BranchGroup();
            Appearance app = new Appearance();
130
            Texture tex = new TextureLoader(
132         "de/fhnon/as/figure3D/brickwork.jpg",
                                null).getTexture();
134         app.setTexture(tex);

136         app.setTexCoordGeneration(
              new TexCoordGeneration(
138           TexCoordGeneration.SPHERE_MAP,
              TexCoordGeneration.TEXTURE_COORDINATE_2));
140         app.setTextureAttributes(
                                new TextureAttributes(
142                             TextureAttributes.REPLACE,
                                new Transform3D(),
144                             new Color4f(),
                                TextureAttributes.NICEST));
146
            Sphere sphere = new Sphere(1.0f,
148           Primitive.GENERATE_TEXTURE_COORDS |
              Primitive.GENERATE_NORMALS_INWARD, 40, app);
150
            bgGeometry.addChild(sphere);
152         back.setGeometry(bgGeometry);
            bg.addChild(back);
154         return bg;
        }
156
```

```
158         /**
             *  Adds a feature to the Lights
160          *  attribute of the SimpleFigure3Dc object
             *
162          *@param  bg  The feature to be added
             * to the Lights attribute
164          */
            public void addLights(BranchGroup bg) {
166             DirectionalLight light =
                  new DirectionalLight(
168               new Color3f(1.0f, 1.0f, 0.0f),
                  new Vector3f(-1.0f, -1.0f, -1.0f));
170             light.setInfluencingBounds(
                            this.getBoundingSphere());
172             bg.addChild(light);
            }
174

176         /**
             *  Description of the Method
178          *
             *@param  bg  Description of the Parameter
180          *@return     Description of the Return Value
             */
182         public TransformGroup createBehaviors(
                        BranchGroup bg) {
184             TransformGroup objTrans =
                            new TransformGroup();
186
                objTrans.setCapability(
188               TransformGroup.ALLOW_TRANSFORM_WRITE);
                Transform3D xAxis = new Transform3D();
190             xAxis.rotY(Math.PI / 2);
                xAxis.rotZ(Math.PI / 8);
192             Alpha xAlpha = new Alpha(-1,
                            Alpha.DECREASING_ENABLE |
194                         Alpha.INCREASING_ENABLE,
                            1000, 1000, 5000,
196                         1000, 1000, 10000,
                            2000, 4000);
198             PositionInterpolator posInt =
                  new PositionInterpolator(xAlpha,
200                         objTrans,
                            xAxis, -0.8f, 0.8f);
202             posInt.setSchedulingBounds(
                            getBoundingSphere());
204             objTrans.addChild(posInt);

206             bg.addChild(objTrans);
                return objTrans;
208         }

210
            /**
212          *  Gets the boundingSphere
```

```
         * attribute of the SimpleFigure3Dc object
214      *
         *@return     The boundingSphere value
216      */
        BoundingSphere getBoundingSphere() {
218             return new BoundingSphere(
                                new Point3d(0.0, 0.0, 0.0), 200.0);
220     }

222

        /**
224      * Description of the Method
         */
226     public void init() {
                this.setLayout(new BorderLayout());
228             GraphicsConfiguration config =
                  SimpleUniverse.getPreferredConfiguration();
230             Canvas3D c = new Canvas3D(config);
                this.add("Center", c);
232             u = new SimpleUniverse(c);
                u.getViewingPlatform().
234                             setNominalViewingTransform();
                u.addBranchGraph(this.createBackground());
236
                BranchGroup bgRoot = new BranchGroup();
238             TransformGroup tg = this.createBehaviors(bgRoot);
                tg.addChild(this.createSceneGraph());
240             this.addLights(bgRoot);
                u.addBranchGraph(bgRoot);
242
        }
244

246     /**
         * Description of the Method
248      */
        public void destroy() {
250             u.cleanup();
        }
252

254     /**
         * The main program for the SimpleFigure3Dc class
256      *
         *@param  args  The command line arguments
258      */
        public static void main(String[] args) {
260             new MainFrame(new SimpleFigure3Dc(), 1000, 700);
        }
262
}
```

Protokoll `SimpleFigure3Dc.log`

```
D:\bonin\artsprog\code>java -version
java version "1.4.2"
Java(TM) 2 Runtime Environment,
```

Legende:
Quellcode ↪ S. 67

Abbildung 3.15: Beispiel: `SimpleFigure3Dc`

```
  Standard Edition (build 1.4.2-b28)
Java HotSpot(TM) Client VM
  (build 1.4.2-b28, mixed mode)

D:\bonin\artsprog\code>javac
  de/fhnon/as/figure3D/SimpleFigure3Dc.java

D:\bonin\artsprog\code>java
  de.fhnon.as.figure3D.SimpleFigure3Dc

D:\bonin\artsprog\code>
```

3.3.4 Durchsichtiger Zylinder

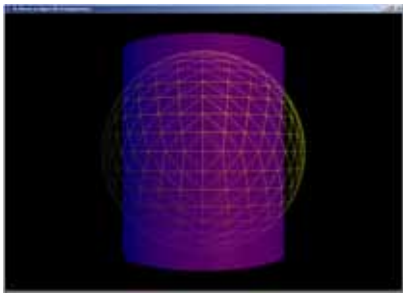
Transparency

Die Transparenz von Objekten ist für die 3D-Maschine eine rechenintensive Aufgabe. Um die Transparenzfrage zu entscheiden, bedarf es der Festlegung, welche Objekte durch ein Objekt hindurchscheinen sollen und wie deren Farbe sich als resultierende Farbe der Objekte errechnet. Ein Objekt, welches hinter einem durchsichtigen Objekt liegt, soll vom durchsichtigen Objekt in seinem Erscheinungsbild beeinflusst werden. Ein undurchsichtiges Objekt, welches vor einem durchsichtigen Objekt liegt, soll unbeeinflusst dargestellt werden. Es gilt daher, die Reihenfolge der Objekte in der räumlichen Tiefe zu bestimmen. Diese Reihenfolge nennt man die *Z-Order*. Sie ergibt sich aus der Position eines Objektes auf der z-Achse der 3D-Welt.

Wir spezifizieren die Durchsichtigkeit des Zylinders wie folgt:

Listing 3.13: Detail `TransparencyAttributes`

```
cylinderApp.setTransparencyAttributes(
```



Legende:
Quellcode ↪ S. 75

Abbildung 3.16: Beispiel: `Transparency`

```
2                new TransparencyAttributes(
                 TransparencyAttributes.NICEST,
4                0.6f));
```

Der erste Parameterwert des Konstruktors, hier `TransparencyAttributes.NICEST`, definiert die Qualität des Ergebnisses. Um die Berechnungszeit zu verkürzen, wäre der Wert `TransparencyAttributes.FASTEST` zu wählen. Der zweite Parameterwert legt den Grad der Durchsichtigkeit des Objektes fest. Dabei steht der Wert $1.0f$ für ein völlig durchsichtiges Objekt, was faktisch selbst nicht sichtbar ist. Der Wert $0.0f$ spezifiziert ein völlig undurchsichtiges Objekt.

Listing 3.14: `Transparency`

```
/**
2  *  "Java 3D Example" Durchsichtiges Objekt
   *
4  *@author      Bonin
   *@version     1.0
6  */

8  package de.fhnon.as.figure3D;

10 import java.applet.Applet;
   import java.awt.BorderLayout;
12 import java.awt.GraphicsConfiguration;

14 import com.sun.j3d.utils.geometry.Cylinder;
   import com.sun.j3d.utils.geometry.Primitive;
16 import com.sun.j3d.utils.geometry.Sphere;
   import com.sun.j3d.utils.universe.SimpleUniverse;
18
   import com.sun.j3d.utils.applet.MainFrame;
20
   import javax.media.j3d.AmbientLight;
22 import javax.media.j3d.Appearance;
   import javax.media.j3d.BranchGroup;
```

```
24 import javax.media.j3d.BoundingSphere;
   import javax.media.j3d.Canvas3D;
26 import javax.media.j3d.DirectionalLight;
   import javax.media.j3d.Material;
28 import javax.media.j3d.PolygonAttributes;
   import javax.media.j3d.Transform3D;
30 import javax.media.j3d.TransformGroup;
   import javax.media.j3d.TransparencyAttributes;
32
   import javax.vecmath.Color3f;
34 import javax.vecmath.Point3d;
   import javax.vecmath.Vector3f;
36
   public class Transparency extends Applet
38 {
      private SimpleUniverse u = null;
40

42    private Transparency()
      {
44       super();
      }
46

48    public BranchGroup createSceneGraph()
      {
50       BranchGroup bg = new BranchGroup();
         TransformGroup sphereTG = new TransformGroup();
52       Transform3D sphereT3D = new Transform3D();

54       sphereT3D.setTranslation(new Vector3f(1f, 0f, -1.5f));
         sphereTG.setTransform(sphereT3D);
56
         Appearance sphereApp = new Appearance();
58       Appearance cylinderApp = new Appearance();

60       sphereApp.setMaterial(new Material(
               new Color3f(0.1f, 0.1f, 0.1f),
62             new Color3f(0.0f, 0.0f, 0.0f),
               new Color3f(0.8f, 0.8f, 0.8f),
64             new Color3f(0.6f, 0.6f, 0.6f),
               100f));
66       sphereApp.setPolygonAttributes(new PolygonAttributes(
               PolygonAttributes.POLYGON_LINE,
68             PolygonAttributes.CULL_NONE,
               0));
70       Sphere sphere =
               new Sphere(
72             0.5f, Sphere.GENERATE_NORMALS, 40, sphereApp);
         bg.addChild(sphere);
74
         cylinderApp.setMaterial(new Material(
76             new Color3f(0.0f, 0.0f, 1.0f),
               new Color3f(0.0f, 0.0f, 0.0f),
78             new Color3f(1.0f, 0.0f, 0.0f),
               new Color3f(1.0f, 1.0f, 1.0f),
```

```
80              100f));
        cylinderApp.setTransparencyAttributes(
82              new TransparencyAttributes(
                TransparencyAttributes.NICEST,
84              0.6f));
        Cylinder cylinder =
86              new Cylinder(
                0.4f, 1f,
88              Cylinder.GENERATE_NORMALS,
                40, 1,
90              cylinderApp);
        bg.addChild(cylinder);
92
        bg.compile();
94      return bg;
    }
96

98  public void addLights(BranchGroup bg)
    {
100     AmbientLight aLight = new AmbientLight(
                new Color3f(1f, 1f, 1f));
102
        DirectionalLight dLight =
104             new DirectionalLight(
                new Color3f(1.0f, 1.0f, 0.0f),
106             new Vector3f(-0.5f, -0.5f, -1.0f));

108     aLight.setInfluencingBounds(
                this.getBoundingSphere());
110     dLight.setInfluencingBounds(
                this.getBoundingSphere());
112
        bg.addChild(aLight);
114     bg.addChild(dLight);
    }
116

118 public TransformGroup createBehaviors(
        BranchGroup bg)
120 {
        TransformGroup objTrans =
122             new TransformGroup();
        bg.addChild(objTrans);
124     return objTrans;
    }
126

128 BoundingSphere getBoundingSphere()
    {
130     return new BoundingSphere(
                new Point3d(0.0, 0.0, 0.0), 100000.0);
132 }

134
```

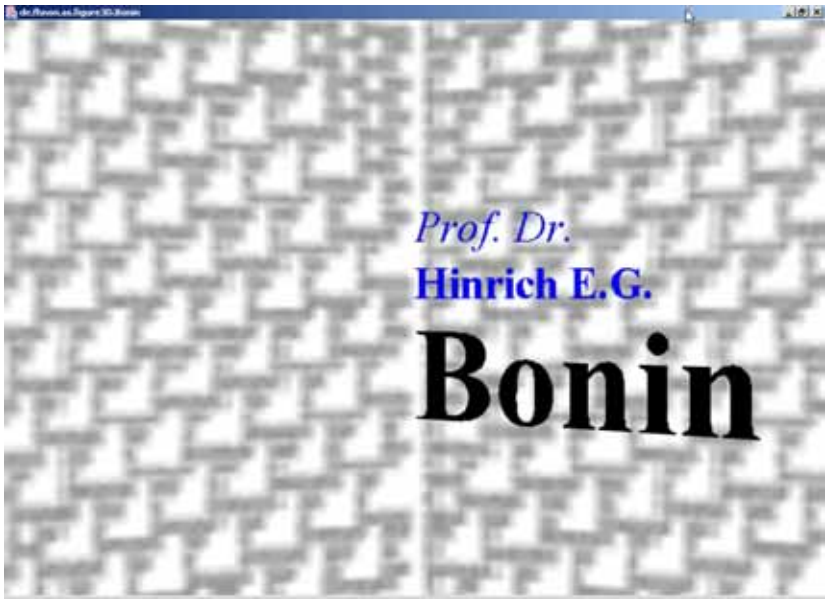
```
136     public void init()
        {
138        this.setLayout(new BorderLayout());
           GraphicsConfiguration config =
140              SimpleUniverse.getPreferredConfiguration();
           Canvas3D c = new Canvas3D(config);
142        this.add("Center", c);
           u = new SimpleUniverse(c);
144        u.getViewingPlatform().
                 setNominalViewingTransform();
146
           BranchGroup bgRoot = new BranchGroup();
148        TransformGroup tg = this.createBehaviors(bgRoot);
           tg.addChild(this.createSceneGraph());
150        this.addLights(bgRoot);
           u.addBranchGraph(bgRoot);
152
        }
154

156     public void destroy()
        {
158        u.cleanup();
        }
160

162     public static void main(String[] args)
        {
164        new MainFrame(new Transparency(), 1000, 700);
        }
166
}
```

Protokoll `Transparency.log`

```
D:\bonin\artsprog\code>java -version
java version "1.4.2_03"
Java(TM) 2 Runtime Environment,
  Standard Edition (build 1.4.2_03-b02)
Java HotSpot(TM) Client VM
  (build 1.4.2_03-b02, mixed mode)

D:\bonin\artsprog\code>javac
  de/fhnon/as/figure3D/Transparency.java

D:\bonin\artsprog\code>java
  de.fhnon.as.figure3D.Transparency

D:\bonin\artsprog\code>
```

3.3.5 Drehender Text

Die Beispielsidee ist der Java3D-Distribution entnommen. Der Quellcode ist modifiziert und ergänzt. Mit Drücken der Taste ↑ wird der Text größer. Die



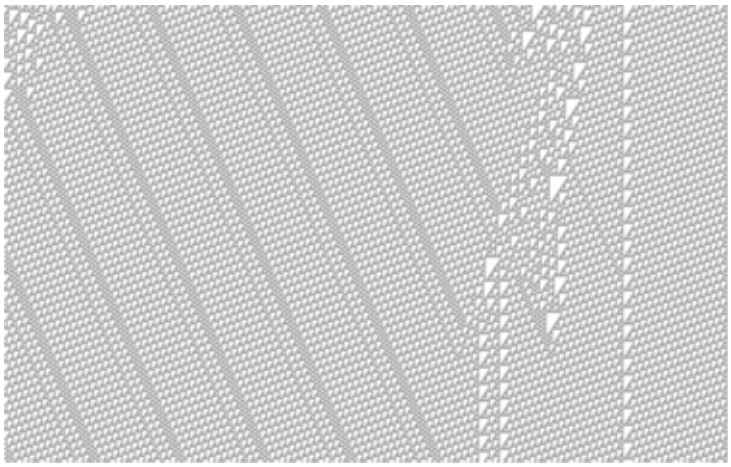
Legende:
Quellcode ↪ S. 79

Abbildung 3.17: Beispiel: Drehender Text

Taste [↓] verkleinert ihn. Mit der Taste [←] wird der Text nach links verschoben. Die Taste [→] verschiebt ihn nach rechts.

Listing 3.15: `Bonin`

```
/**
 *  Beispiel "Drehender Text"
 *  Idee aus Sun Demo-Sammlung entnommen; siehe:
 *  \j2sdk1.4.2\demo\java3d\Text2D\Text2DTest.java
 *  Copyright (c) 1996-2002 Sun Microsystems.
 *  Quellcode modified
 *
 *@author      Bonin
 *@version     1.0
 */
package de.fhnon.as.figure3D;
import java.applet.Applet;
import java.awt.GraphicsConfiguration;
import java.awt.BorderLayout;
import java.awt.Font;
import com.sun.j3d.utils.applet.MainFrame;
import com.sun.j3d.utils.geometry.Text2D;
import com.sun.j3d.utils.geometry.Primitive;
import com.sun.j3d.utils.geometry.Sphere;
```



Legende:
Quelle zur Erzeugung des Hintergrundbildes ↪ [Bonin04a].

Abbildung 3.18: Hintergrundbild: `AutomatonR110.eps`

```
20  import com.sun.j3d.utils.image.TextureLoader;
    import com.sun.j3d.utils.universe.SimpleUniverse;
22
    import javax.media.j3d.Alpha;
24  import javax.media.j3d.Appearance;
    import javax.media.j3d.Background;
26  import javax.media.j3d.BranchGroup;
    import javax.media.j3d.BoundingSphere;
28  import javax.media.j3d.Canvas3D;
    import javax.media.j3d.PolygonAttributes;
30  import javax.media.j3d.RotationInterpolator;
    import javax.media.j3d.Shape3D;
32  import javax.media.j3d.Texture;
    import javax.media.j3d.Transform3D;
34  import javax.media.j3d.TransformGroup;

36  import javax.vecmath.Point3d;
    import javax.vecmath.Color3f;
38  import javax.vecmath.Vector3f;

40  public class Bonin extends Applet
    {
42
       private SimpleUniverse u = null;
44

46     public BranchGroup createSceneGraph()
       {
48
          BranchGroup objRoot = new BranchGroup();
```

```
50
        TransformGroup objScale = new TransformGroup();
52      Transform3D t3d = new Transform3D();
        t3d.setScale(0.4);
54      objScale.setTransform(t3d);
        objRoot.addChild(objScale);
56
        TransformGroup objTrans = new TransformGroup();
58      objTrans.setCapability(
              TransformGroup.ALLOW_TRANSFORM_WRITE);
60
        BoundingSphere bounds =
62            new BoundingSphere(
              new Point3d(0.0, 0.0, 0.0), 100.0);
64
        TransformGroup textTranslationGroup;
66      Transform3D textTranslation;
        float yPos = -0.5f;
68      Shape3D textObject = new Text2D("Bonin",
              new Color3f(0f, 0f, 0f),
70            "Serif",
              120,
72            Font.BOLD);

74      Appearance app = textObject.getAppearance();

76      PolygonAttributes pa = app.getPolygonAttributes();
        if (pa == null)
78      {
           pa = new PolygonAttributes();
80      }
        pa.setCullFace(PolygonAttributes.CULL_NONE);
82      if (app.getPolygonAttributes() == null)
        {
84         app.setPolygonAttributes(pa);
        }
86      objTrans.addChild(textObject);

88      textTranslation = new Transform3D();
        textTranslation.setTranslation(
90            new Vector3f(0f, yPos, 0f));
        textTranslationGroup = new TransformGroup(
92            textTranslation);
        textTranslationGroup.addChild(objTrans);
94      objScale.addChild(textTranslationGroup);
        yPos += 0.5f;
96
        textObject = new Text2D("Hinrich E.G.",
98            new Color3f(0f, 0f, 1f),
              "Serif",
100           40,
              Font.BOLD);
102     textTranslation = new Transform3D();
        textTranslation.setTranslation(
104           new Vector3f(0f, yPos, 0f));
        textTranslationGroup = new TransformGroup(
```

```
106             textTranslation);
        textTranslationGroup.addChild(textObject);
108     objScale.addChild(textTranslationGroup);
        yPos += 0.2f;
110
        textObject = new Text2D("Prof. Dr.",
112             new Color3f(0f, 0f, 1f),
                "Serif",
114             40,
                Font.ITALIC);
116     textTranslation = new Transform3D();
        textTranslation.setTranslation(
118             new Vector3f(0f, yPos, 0f));
        textTranslationGroup =
120             new TransformGroup(textTranslation);
        textTranslationGroup.addChild(textObject);
122     objScale.addChild(textTranslationGroup);
        yPos += 0.5f;
124
        Transform3D yAxis = new Transform3D();
126     Alpha rotationAlpha =
                new Alpha(-1,
128             Alpha.INCREASING_ENABLE,
                0, 0,
130             4000, 0, 0,
                0, 0, 0);
132
        RotationInterpolator rotator =
134             new RotationInterpolator(
                rotationAlpha, objTrans, yAxis,
136             0.0f, (float) Math.PI * 2.0f);
        rotator.setSchedulingBounds(bounds);
138     objTrans.addChild(rotator);

140     return objRoot;
    }
142

144 public BranchGroup createBackground()
    {
146     BranchGroup backgroundGroup =
                new BranchGroup();
148     Background back = new Background();
        back.setApplicationBounds(
150             getBoundingSphere());
        BranchGroup bgGeometry =
152             new BranchGroup();
        Appearance app = new Appearance();
154     Texture tex = new TextureLoader(
                "./de/fhnon/as/figure3D/AutomatonR110.jpg",
156             this).getTexture();
        app.setTexture(tex);
158     Sphere sphere = new Sphere(1.0f,
                Primitive.GENERATE_TEXTURE_COORDS |
160             Primitive.GENERATE_NORMALS_INWARD, app);
        bgGeometry.addChild(sphere);
```

```
162         back.setGeometry(bgGeometry);
            backgroundGroup.addChild(back);
164         return backgroundGroup;
        }
166

168     BoundingSphere getBoundingSphere()
        {
170         return new BoundingSphere(
                new Point3d(0.0, 0.0, 0.0), 200.0);
172     }

174
        public Bonin() { }
176

178     public void init()
        {
180         setLayout(new BorderLayout());
            GraphicsConfiguration config =
182
            SimpleUniverse.getPreferredConfiguration();
184
            Canvas3D c = new Canvas3D(config);
186         add("Center", c);

188         BranchGroup scene = createSceneGraph();
            u = new SimpleUniverse(c);
190         MoverBehavior navigator =
                new MoverBehavior(
192             u.getViewingPlatform().
                getViewPlatformTransform());
194         scene.addChild(navigator);

196         scene.compile();

198         u.getViewingPlatform().
                setNominalViewingTransform();
200         u.addBranchGraph(createBackground());
            u.addBranchGraph(scene);
202     }

204
        public void destroy()
206     {
            u.cleanup();
208     }

210

212     public static void main(String[] args)
        {
214         new MainFrame(new Bonin(), 256, 256);
        }
216 }
```

Listing 3.16: `MoverBehavior`

```
/**
 *  Beispiel "Drehender Text" Idee aus Sun
 *  Demo-Sammlung entnommen, siehe
 *  \j2sdk1.4.2\demo\java3d\Text2D\MoverBehavior.java
 *  Copyright (c) 1996-2002 Sun Microsystems,
 *
 *@author      Bonin
 *@version     1.0
 */
package de.fhnon.as.figure3D;

import java.awt.event.KeyEvent;
import java.awt.AWTEvent;
import javax.media.j3d.*;
import java.util.Enumeration;
import javax.vecmath.*;

/*
 *  Mover behavior class -
 *  used to allow viewer to move using arrow keys
 */
class MoverBehavior extends Behavior
{
    WakeupOnAWTEvent w1 =
        new WakeupOnAWTEvent(KeyEvent.KEY_PRESSED);
    WakeupCriterion[] w2 = {w1};
    WakeupCondition w = new WakeupOr(w2);
    TransformGroup viewTransformGroup;
    /*
     *  holds current rotation radians
     */
    double rotation = 0.0;


    public void initialize()
    {
        /*
         *  Establish initial wakeup criteria
         */
        wakeupOn(w);
    }


    /**
     *  Override Behavior's stimulus method to
     *  handle the event.
     *
     *@param  criteria  Description of the
     *       Parameter
     */
    public void processStimulus(Enumeration criteria)
    {
        WakeupOnAWTEvent ev;
        WakeupCriterion genericEvt;
        AWTEvent[] events;
```

```
56
        while (criteria.hasMoreElements())
58      {
           genericEvt = (WakeupCriterion)
60               criteria.nextElement();
           if (genericEvt instanceof WakeupOnAWTEvent)
62         {
              ev = (WakeupOnAWTEvent) genericEvt;
64            events = ev.getAWTEvent();
              processManualEvent(events);
66         }
        }
68      /*
         * Set wakeup criteria for next time
70       */
        wakeupOn(w);
72   }

74

     /**
76    * Process a keyboard event to move or rotate
      * the viewer.
78    *
      *@param events Description of the Parameter
80    */
     void processManualEvent(AWTEvent[] events)
82   {

84      for (int i = 0; i < events.length; ++i)
        {
86         if (events[i] instanceof KeyEvent)
           {
88            KeyEvent event = (KeyEvent) events[i];
              if (event.getKeyCode()
90                  == KeyEvent.VK_EQUALS)
              {
92               continue;
              }
94            Transform3D t = new Transform3D();
              viewTransformGroup.getTransform(t);
96            Vector3f viewDir =
                    new Vector3f(0f, 0f, -1f);
98            Vector3f translation = new Vector3f();
              t.get(translation);
100           t.transform(viewDir);
              if (event.getKeyCode()
102                 == KeyEvent.VK_UP)
              {
104              translation.x += viewDir.x;
                 translation.y += viewDir.y;
106              translation.z += viewDir.z;
              } else if (event.getKeyCode()
108                 == KeyEvent.VK_DOWN)
              {
110              translation.x -= viewDir.x;
                 translation.y -= viewDir.y;
```

```
112                 translation.z -= viewDir.z;
                } else if (event.getKeyCode()
114                     == KeyEvent.VK_RIGHT)
                {
116                 rotation += -.1;
                } else if (event.getKeyCode()
118                     == KeyEvent.VK_LEFT)
                {
120                 rotation += .1;
                }
122             t.rotY(rotation);
                t.setTranslation(translation);
124             viewTransformGroup.setTransform(t);
            }
126         }
        }
128

130     /**
         *  Constructor
132      *
         *@param  trans  Description of the Parameter
134      */
        public MoverBehavior(TransformGroup trans)
136     {
            viewTransformGroup = trans;
138         Bounds bound =
                  new BoundingSphere(
140               new Point3d(0.0, 0.0, 0.0), 10000.0);
            this.setSchedulingBounds(bound);
142     }
}
```

Protokoll `Bonin.log`

```
D:\bonin\artsprog\code>java -version
java version "1.4.2"
Java(TM) 2 Runtime Environment,
  Standard Edition (build 1.4.2-b28)
Java HotSpot(TM) Client VM
  (build 1.4.2-b28, mixed mode)

D:\bonin\artsprog\code>javac de/fhnon/as/figure3D/*.java

D:\bonin\artsprog\code>java de.fhnon.as.figure3D.Bonin

D:\bonin\artsprog\code>
```

3.3.6 Konstruierte Geometrie

Wir konstruieren unser 3D-Objekt, eine Yo-Yo-Spindel, als Instanz der Klasse `javax.media.j3d.Shape3D`. Diese Beispielidee „Yo-Yo“ wurde dem

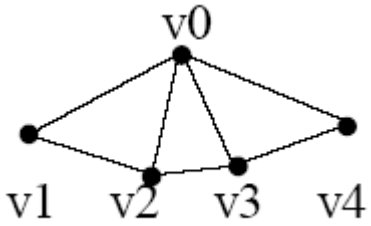


Abbildung 3.19: TriangleFanArray — *Vertices* $v_{0..4}$

Java3D-Tutorial[9]entnommen. Seine spezifizierte Geometrie übergeben wir als Parameter des Konstruktors. Diese Geometrie spezifizieren wir auf der Basis der Klasse `javax.media.j3d.TriangleFanArray`. Diese hat folgenden Konstruktor:

TriangleFanArray

```
public TriangleFanArray(int vertexCount,
                        int vertexFormat,
                        int[] stripVertexCounts)
```

Ein TriangleFanArray-Skizze mit $v_{0..4}$ zeigt Abbildung 3.19 S. 87. Wir konstruieren unser Objekt aus vier *triangle fan strips* mit einer „Auflösung“ (*N*) von 30 *Vertices* ($v_{1..30}$). Deren Nullpunkte (v_0) setzen wir direkt ausgehend vom Koordinatennullpunkt und einer Verschiebung auf der z-Achse, wie folgt:

```
coords[0 * (N + 1)] = new Point3f(0.0f, 0.0f, w);
coords[1 * (N + 1)] = new Point3f(0.0f, 0.0f, 0.0f);
coords[2 * (N + 1)] = new Point3f(0.0f, 0.0f, 0.0f);
coords[3 * (N + 1)] = new Point3f(0.0f, 0.0f, -w);
```

Die Koordinaten für $v_{1..30}$ ermitteln wir für jeden der vier *triangle fan strips* auf die gleiche Weise. Die jeweiligen x,y-Werte nach folgender Formel und den z-Wert setzen wir dann direkt auf *w* oder $-w$.

```
for (a = 0, n = 0; n < N;
     a = 2.0 * Math.PI / (N - 1) * ++n)
  {
     x = (float) (r * Math.cos(a));
     y = (float) (r * Math.sin(a));
...
}
```

[9]Tutorial v1.6 (Java 3D API v1.2) — Chapter 2 (Creating Geometry)
↪ `http://java.sun.com/products/java-media/3D/learning/tutorial/-index.html` (online 29-Apr-2004)

Legende:
Quellcode ↪ S. 88

Abbildung 3.20: Beispiel: Konstruierte Geometrie — YoYo

Die Koordinaten legen wir hintereinander im Array `coords` ab. Mit dem Wert `stripVertexCounts`, ein `int`-Array, geben wir die Anzahl der *Vertices* für jeden der *triangle fan strips* an. Seine Länge entspricht der Anzahl der *strips*.

Um die vier *triangle fan strips* einfach zu erkennen, ordnen wir ihnen jeweils eine Farbe zu.

Listing 3.17: `GeoArray`

```
/**
 *  "Java 3D Example" Geometry Array
 *
 *@author       Bonin
 *@version      1.0
 */

package de.fhnon.as.figure3D;

import java.applet.Applet;
import java.awt.BorderLayout;
import java.awt.GraphicsConfiguration;

import com.sun.j3d.utils.geometry.Primitive;
import com.sun.j3d.utils.geometry.Sphere;
import com.sun.j3d.utils.universe.SimpleUniverse;

import com.sun.j3d.utils.applet.MainFrame;

import javax.media.j3d.BranchGroup;
import javax.media.j3d.BoundingSphere;
import javax.media.j3d.Canvas3D;
```

```
import javax.media.j3d.DirectionalLight;
import javax.media.j3d.Geometry;
import javax.media.j3d.Transform3D;
import javax.media.j3d.TransformGroup;
import javax.media.j3d.TriangleFanArray;
import javax.media.j3d.Shape3D;

import javax.vecmath.AxisAngle4f;
import javax.vecmath.Color3f;
import javax.vecmath.Point3d;
import javax.vecmath.Point3f;
import javax.vecmath.Vector3f;

public class GeoArray extends Applet
{
   private SimpleUniverse u = null;


   private GeoArray()
   {
      super();
   }


   public BranchGroup createSceneGraph()
   {
      BranchGroup bg = new BranchGroup();
      TransformGroup tg = new TransformGroup();
      Transform3D t3d = new Transform3D();
      t3d.setRotation(
            new AxisAngle4f(
            1f,
            1f,
            0f,
            (float) Math.toRadians(60)));
      tg.setTransform(t3d);

      Shape3D shape3D =
            new Shape3D(this.yoyoGeometry());

      tg.addChild(shape3D);
      bg.addChild(tg);

      return bg;
   }


   public void addLights(BranchGroup bg)
   {
      DirectionalLight light =
            new DirectionalLight(
            new Color3f(1.0f, 1.0f, 0.0f),
            new Vector3f(-1.0f, -1.0f, -1.0f));
      light.setInfluencingBounds(
            this.getBoundingSphere());
      bg.addChild(light);
```

```
    }
80

82  public TransformGroup createBehaviors(
        BranchGroup bg)
84  {
        TransformGroup objTrans =
86          new TransformGroup();
        bg.addChild(objTrans);
88      return objTrans;
    }
90

92  BoundingSphere getBoundingSphere()
    {
94      return new BoundingSphere(
            new Point3d(0.0, 0.0, 0.0), 200.0);
96  }

98

    private Geometry yoyoGeometry()
100 {
        TriangleFanArray tfan;
102     final int N = 30;
        int vertexCount = 4 * (N + 1);
104     Point3f coords[] = new Point3f[vertexCount];
        Color3f colors[] = new Color3f[vertexCount];
106     Color3f blue = new Color3f(0.0f, 0.0f, 1.0f);
        Color3f yellow = new Color3f(1.0f, 1.0f, 0.0f);
108     Color3f cyan = new Color3f(0.0f, 1.0f, 1.0f);
        Color3f magenta = new Color3f(1.0f, 0.0f, 1.0f);
110
        int stripVertexCounts[] =
112         {N + 1, N + 1, N + 1, N + 1};

114     float r = 0.5f;
        float w = 0.4f;
116     int n;
        double a;
118     float x;
        float y;
120     /*
         * set the central points
122      * for the four triangle fan strips
         */
124     coords[0 * (N + 1)] =
            new Point3f(0.0f, 0.0f, w);
126     coords[1 * (N + 1)] =
            new Point3f(0.0f, 0.0f, 0.0f);
128     coords[2 * (N + 1)] =
            new Point3f(0.0f, 0.0f, 0.0f);
130     coords[3 * (N + 1)] =
            new Point3f(0.0f, 0.0f, -w);
132     colors[0 * (N + 1)] = blue;
        colors[1 * (N + 1)] = yellow;
134     colors[2 * (N + 1)] = cyan;
```

```
        colors[3 * (N + 1)] = magenta;

        for (a = 0, n = 0;
             n < N;
             a = 2.0 * Math.PI / (N - 1) * ++n)
        {
           x = (float) (r * Math.cos(a));
           y = (float) (r * Math.sin(a));
           coords[0 * (N + 1) + n + 1] =
                 new Point3f(x, y, w);
           coords[1 * (N + 1) + N - n] =
                 new Point3f(x, y, w);
           coords[2 * (N + 1) + n + 1] =
                 new Point3f(x, y, -w);
           coords[3 * (N + 1) + N - n] =
                 new Point3f(x, y, -w);
           colors[0 * (N + 1) + N - n] = blue;
           colors[1 * (N + 1) + n + 1] = yellow;
           colors[2 * (N + 1) + N - n] = cyan;
           colors[3 * (N + 1) + n + 1] = magenta;
        }
        tfan = new TriangleFanArray(vertexCount,
              TriangleFanArray.COORDINATES |
              TriangleFanArray.COLOR_3,
              stripVertexCounts);
        tfan.setCoordinates(0, coords);
        tfan.setColors(0, colors);
        return tfan;
    }


    public void init()
    {
        this.setLayout(new BorderLayout());
        GraphicsConfiguration config =
              SimpleUniverse.getPreferredConfiguration();
        Canvas3D c = new Canvas3D(config);
        this.add("Center", c);
        u = new SimpleUniverse(c);
        u.getViewingPlatform().
              setNominalViewingTransform();

        BranchGroup bgRoot = new BranchGroup();
        TransformGroup tg = this.createBehaviors(bgRoot);
        tg.addChild(this.createSceneGraph());
        this.addLights(bgRoot);
        u.addBranchGraph(bgRoot);

    }


    public void destroy()
    {
        u.cleanup();
    }

```

```
192     public static void main(String[] args)
        {
194        new MainFrame(new GeoArray(), 700, 700);
        }
196
}
```

Protokoll `GeoArray.log`

```
D:\bonin\artsprog\code>java -version
java version "1.4.2_03"
Java(TM) 2 Runtime Environment,
  Standard Edition (build 1.4.2_03-b02)
Java HotSpot(TM) Client VM
  (build 1.4.2_03-b02, mixed mode)

D:\bonin\artsprog\code>javac de/fhnon/as/figure3D/GeoArray.java

D:\bonin\artsprog\code>java de.fhnon.as.figure3D.GeoArray

D:\bonin\artsprog\code>
```

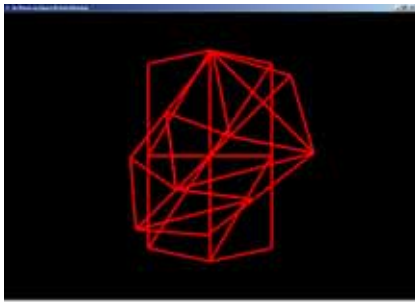
3.3.7 `DataSharing`

In diesem Beispiel greifen wir unser Startbeispiel, also die Klasse `HelloUniverse` (↪ Listing 3.8 S. 52) wieder auf. Allerdings sind jetzt zwei Zylinder darzustellen. Wir wollen jedoch nicht zweimal den Konstruktor `Cylinder(...)` aufrufen, sondern nur unser Zylinderobjekt zweimal darstellen und zwar einmal gedreht und einmal nicht gedreht.

Zu diesem *Data Sharing* nutzen wir die Klassen `SharedGroup` und `Link` des Paketes `javax.media.j3d`. Zwei `Link`-Knoten zeigen beide auf den `SharedGroup`-Knoten, der die eine `Cylinder`-Instanz aufnimmt. Der folgende Quellcode zeigt, dass die `SharedGroup`-Instanz, hier `sg`, die Verbindung von den beiden `Link`-Instanzen, hier `link1toSg` und `link2toSg`, zum gemeinsamen Objekt, hier `cylinder`, darstellt.

Listing 3.18: Detail `SharedGroup` & `Link`

```
    public BranchGroup createSceneGraph()
2   {
        BranchGroup bg = new BranchGroup();
4
        TransformGroup tg = new TransformGroup();
6
        SharedGroup sg = new SharedGroup();
8       Link link1toSg = new Link(sg);
        Link link2toSg = new Link(sg);
10
        Transform3D t3d = new Transform3D();
12      t3d.setRotation(...);
        tg.setTransform(t3d);
14
```



Legende:
Quellcode ↪ S. 93

Abbildung 3.21: Beispiel: `DataSharing`

```
        ...
16
        Cylinder cylinder = new Cylinder(...);
18
        ...
20
        sg.addChild(cylinder);
22      tg.addChild(link1toSg);
        bg.addChild(tg);
24      bg.addChild(link2toSg);
        bg.compile();
26      return bg;
    }
```

Gegenüber dem Startbeispiel wurden weitere Änderungen vorgenommen. So wurde der Zylinder nur noch mit vier Unterteilungen des Grundkreises, also mit $xDivision = 4$, spezifiziert. Das Ergebnis (↪ Abbildung 3.21 S. 93) zeigt daher den Zylinder als Quader.

Auch die Unterteilung des Zylindermantels wurde reduziert und zwar auf $yDivision = 2$. Zusätzlich wurde die Linienbreite vergrößert (auf den Wert `4.0f`) und mit dem Parameter `PolygonAttributes.CULL_BACK` wurde die Darstellung der hinteren Polygone unterdrückt.

Listing 3.19: `DataSharing`

```
/**
2  *  "Java 3D Example" Data Sharing
 *
4  *@author      Bonin
 *@version     1.0
6  */

8 package de.fhnon.as.figure3D;

10 import java.applet.Applet;
import java.awt.BorderLayout;
```

```
12 import java.awt.GraphicsConfiguration;

14 import com.sun.j3d.utils.applet.MainFrame;

16 import com.sun.j3d.utils.geometry.Primitive;
   import com.sun.j3d.utils.geometry.Cylinder;
18 import com.sun.j3d.utils.universe.SimpleUniverse;

20 import javax.media.j3d.Appearance;
   import javax.media.j3d.BranchGroup;
22 import javax.media.j3d.Canvas3D;
   import javax.media.j3d.LineAttributes;
24 import javax.media.j3d.Link;
   import javax.media.j3d.Material;
26 import javax.media.j3d.SharedGroup;
   import javax.media.j3d.TransformGroup;
28 import javax.media.j3d.Transform3D;
   import javax.media.j3d.PolygonAttributes;
30
   import javax.vecmath.AxisAngle4f;
32 import javax.vecmath.Color3f;

34 public class DataSharing extends Applet
   {
36    private SimpleUniverse u = null;

38
      public BranchGroup createSceneGraph()
40    {
         BranchGroup bg = new BranchGroup();
42
         TransformGroup tg = new TransformGroup();
44
         SharedGroup sg = new SharedGroup();
46       Link link1toSg = new Link(sg);
         Link link2toSg = new Link(sg);
48
         Transform3D t3d = new Transform3D();
50       t3d.setRotation(
               new AxisAngle4f(
52             1f,
               1f,
54             0f,
               (float) Math.toRadians(90)));
56       tg.setTransform(t3d);

58       Appearance app = new Appearance();
         Material m = new Material();
60       Color3f red = new Color3f(1.0f, 0.0f, 0.0f);
         m.setEmissiveColor(red);
62       m.setLightingEnable(true);
         app.setMaterial(m);
64
         PolygonAttributes polyAtt =
66             new PolygonAttributes();
         polyAtt.setPolygonMode(
```

```
68              PolygonAttributes.POLYGON_LINE);
        polyAtt.setCullFace(
70              PolygonAttributes.CULL_BACK);
        app.setPolygonAttributes(polyAtt);
72
        LineAttributes latt = new LineAttributes();
74      latt.setLineWidth(6.0f);
        latt.setLineAntialiasingEnable(true);
76      app.setLineAttributes(latt);

78      Cylinder cylinder =
              new Cylinder(
80            0.3f,
              0.9f,
82            Primitive.GENERATE_NORMALS,
              4,
84            2,
              app);
86
        sg.addChild(cylinder);
88      tg.addChild(link1toSg);
        bg.addChild(tg);
90      bg.addChild(link2toSg);
        bg.compile();
92      return bg;
    }
94

96  public void init()
    {
98      this.setLayout(new BorderLayout());
        GraphicsConfiguration config =
100           SimpleUniverse.getPreferredConfiguration();
        Canvas3D c = new Canvas3D(config);
102     this.add("Center", c);
        u = new SimpleUniverse(c);
104     u.getViewingPlatform().
              setNominalViewingTransform();
106     u.addBranchGraph(
              this.createSceneGraph());
108
    }
110

112 public void destroy()
    {
114     u.cleanup();
    }
116

118 public static void main(String[] args)
    {
120     new MainFrame(
              new DataSharing(), 300, 400);
122 }
```

```
124 }
```

Protokoll `DataSharing.log`

```
D:\bonin\artsprog\code>java -version
java version "1.4.2_03"
Java(TM) 2 Runtime Environment,
  Standard Edition (build 1.4.2_03-b02)
Java HotSpot(TM) Client VM
  (build 1.4.2_03-b02, mixed mode)

D:\bonin\artsprog\code>javac de/fhnon/as/figure3D/DataSharing.java

D:\bonin\artsprog\code>java de.fhnon.as.figure3D.DataSharing

D:\bonin\artsprog\code>
```

3.3.8 `ExampleWavefrontLoad`

Um 3D-Daten in die Java3D-Welt importieren zu können, gibt es das Paket `com.sun.j3d.loaders`. Mit dessen Hilfe läßt sich eine eigene *File Loader Class* entwickeln, die das jeweilige Datenformat importieren kann. Für eine große Menge marktüblicher Datenformate exisitieren schon entsprechende Dateilader.

Wavefront

In diesem Beispiel importieren wir ein Objekt, gespeichert im *Wavefront*-Format. Eine solche Objektdatei ist eine ASCII-codierte Datei mit der Namenserweiterung `*.obj`. Prinzipiell kann sie eine Geometrie enthalten, bestehend aus Polygonen und *Free-form Geometry* (Kurven und Oberflächen). Letztere wird vom hier genutzten Lader nicht unterstützt.

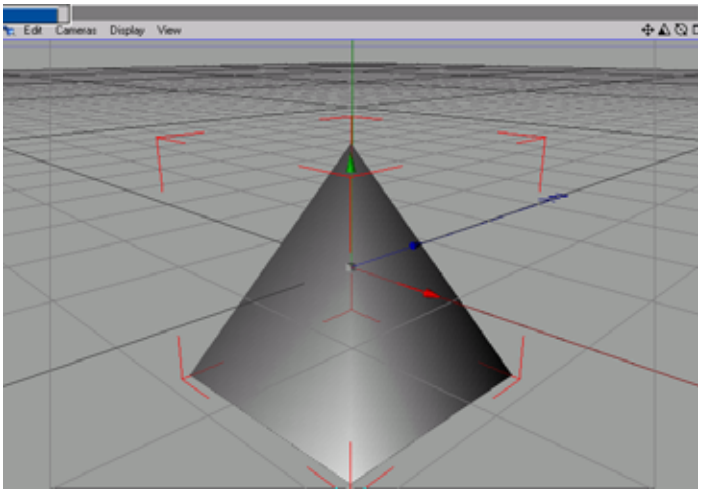
Das zu importierende Objekt wurde mit dem Werkzeug *Cinema4D Release 8* der Firma *Maxon* Computer GmbH[10] erzeugt und dort mit der Option „Export Wavefront“ gespeichert.

Der eigenliche Vorgang des Ladens geschieht durch eine Instanz der entsprechenden *File Loader Class*, hier der Klasse `com.sun.j3d.loaders.objectfile.ObjectFile`. Deren Methode `load(`filename`)` gibt eine Instanz der Klasse `com.sun.j3d.loaders.Scene` zurück, falls nicht Ausnahmefälle beim Lesen und Auswerten (Parsen) eintreten. Das folgende Quellcodefragment zeigt die Konstruktion für den Ladevorgang:

Listing 3.20: Detail `obj`-Datei laden

```
public BranchGroup createSceneGraph()
{
  BranchGroup bg = new BranchGroup();

  ObjectFile f = new ObjectFile();

  Scene s = null;

```

[10] Homepage ↪ `http://www.maxon.net` (online 8-May-2004)



Legende:
Quellcode ↪ S. 97

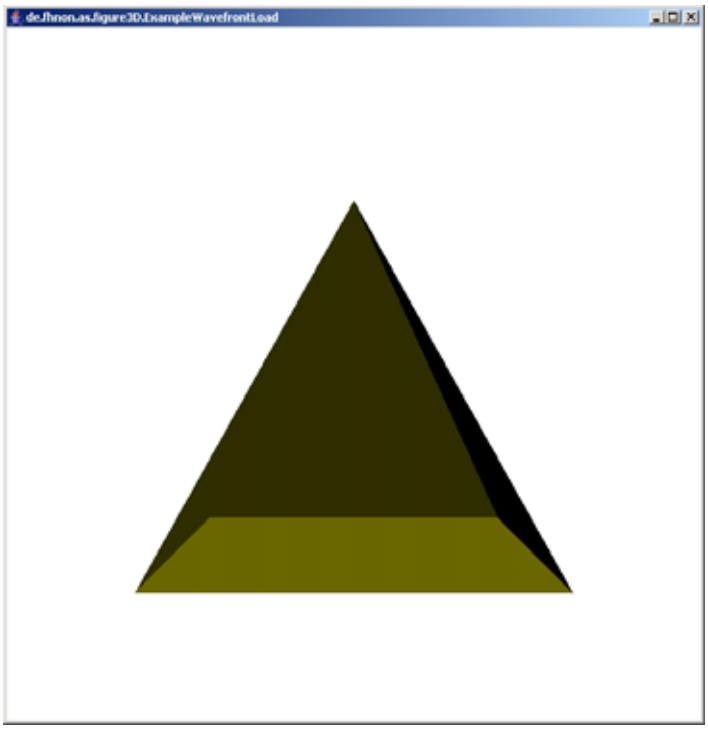
Abbildung 3.22: Beispiel: `ExampleWavefrontLoad` — Objekt in Cinema4D

```
    try
10  {
       s = f.load(filename);
12  } catch (FileNotFoundException e)
       {...;
14  } catch (ParsingErrorException e)
       {...;
16  } catch (IncorrectFormatException e)
       { ...;
18  }

20  bg.addChild(s.getSceneGroup());

22  return bg;
    }
```

Um das erfolgreiche Importieren zu verdeutlichen, wurde das Objekt in seiner Größe mit dem Konstrukt `t3d.setScale(0.005)` reduziert. Das Objekt im Werkzeug *Cinema4D Release* zeigt Abbildung 3.22 S. 97. Die ASCII-Datei `pyramide.obj` (↪ S. 101) zeigt es im exportierten Wavefront-Format. Das Importergebnis zeigt Abbildung 3.23 S. 98. Hinweis: Man beachte die Unterschiede in der Form und dem Erscheinungsbild.

Listing 3.21: `ExampleWavefrontLoad`

```
/**
2  * "Java 3D Example" Laden eines Objektes im Format
   * Wavefront. Erzeugt wurde das Objekt mit Cinema4D
```



Legende:
Quellcode ↪ S. 97

Abbildung 3.23: Beispiel: `ExampleWavefrontLoad` — Objekt in Java3D

```
4   *
    *@author      Bonin
6   *@version     1.0
    */
8
    package de.fhnon.as.figure3D;
10
    import java.io.*;
12  import java.applet.Applet;
    import java.awt.BorderLayout;
14  import java.awt.GraphicsConfiguration;

16  import com.sun.j3d.loaders.objectfile.ObjectFile;
    import com.sun.j3d.loaders.ParsingErrorException;
18  import com.sun.j3d.loaders.IncorrectFormatException;
    import com.sun.j3d.loaders.Scene;
20
    import com.sun.j3d.utils.universe.SimpleUniverse;
22
    import com.sun.j3d.utils.applet.MainFrame;
24
    import javax.media.j3d.Appearance;
26  import javax.media.j3d.Background;
    import javax.media.j3d.BranchGroup;
28  import javax.media.j3d.BoundingSphere;
    import javax.media.j3d.Canvas3D;
30  import javax.media.j3d.DirectionalLight;
    import javax.media.j3d.Material;
32  import javax.media.j3d.TransformGroup;
    import javax.media.j3d.Transform3D;
34
    import javax.vecmath.Color3f;
36  import javax.vecmath.Point3d;
    import javax.vecmath.Vector3f;
38
    public class ExampleWavefrontLoad extends Applet
40  {
        private SimpleUniverse u = null;
42
        private String filename =
44              "de/fhnon/as/figure3D/pyramide.obj";


46
        private ExampleWavefrontLoad()
48      {
            super();
50      }


52
        public BranchGroup createSceneGraph()
54      {
            BranchGroup bg = new BranchGroup();
56          TransformGroup tg = new TransformGroup();
            Transform3D t3d = new Transform3D();
58          t3d.setScale(0.005);
            tg.setTransform(t3d);
```

```
60
        ObjectFile f = new ObjectFile();
62
        Scene s = null;
64
        try
66      {
           s = f.load(filename);
68      } catch (FileNotFoundException e)
        {
70         System.err.println(e);
           System.exit(1);
72      } catch (ParsingErrorException e)
        {
74         System.err.println(e);
           System.exit(1);
76      } catch (IncorrectFormatException e)
        {
78         System.err.println(e);
           System.exit(1);
80      }

82      System.out.println(s.getNamedObjects());

84      tg.addChild(s.getSceneGroup());
        bg.addChild(tg);
86      return bg;
    }
88

90  public void addLights(BranchGroup bg)
    {
92      DirectionalLight light =
              new DirectionalLight(
94            new Color3f(1.0f, 1.0f, 0.0f),
              new Vector3f(-1.0f, -1.0f, -1.0f));
96      light.setInfluencingBounds(
              this.getBoundingSphere());
98      bg.addChild(light);
    }
100

102 public TransformGroup createBehaviors(
        BranchGroup bg)
104 {
    TransformGroup objTrans =
106       new TransformGroup();
    bg.addChild(objTrans);
108 return objTrans;
    }
110

112 BoundingSphere getBoundingSphere()
    {
114 return new BoundingSphere(
          new Point3d(0.0, 0.0, 0.0), 200.0);
```

```
116     }

118
        BranchGroup createBackground()
120     {
            BranchGroup bg = new BranchGroup();
122         Background background = new Background();
            background.setColor(1.0f, 1.0f, 1.0f);
124         background.setApplicationBounds(
                  getBoundingSphere());
126         bg.addChild(background);
            return bg;
128     }

130
        public void init()
132     {
            this.setLayout(new BorderLayout());
134         GraphicsConfiguration config =
                  SimpleUniverse.getPreferredConfiguration();
136         Canvas3D c = new Canvas3D(config);
            this.add("Center", c);
138         u = new SimpleUniverse(c);
            u.getViewingPlatform().
140               setNominalViewingTransform();

142         BranchGroup bgRoot = new BranchGroup();
            TransformGroup tg = this.createBehaviors(bgRoot);
144         tg.addChild(this.createSceneGraph());
            this.addLights(bgRoot);
146         u.addBranchGraph(this.createBackground());
            u.addBranchGraph(bgRoot);
148
        }
150

152     public void destroy()
        {
154         u.cleanup();
        }
156

158     public static void main(String[] args)
        {
160         new MainFrame(new ExampleWavefrontLoad(), 700, 700);
        }
162
    }
```

Listing 3.22: `Wavefront`-Daten `pyramide.obj`

```
# WaveFront *.obj file (generated by Cinema4D)
2
g Pyramid
4 v -100 -100 100
v 100 -100 100
6 v 100 -100 -100
```

```
v -100 -100 -100
v 0 100 0

vt 0 0 0
vt 1 0 0
vt 0 1 0
vt 1 0 0
vt 0 0 0
vt 1 1 0
vt 0 0 0
vt 1 0 0
vt 0 0 0
vt 1 0 0
vt 0.5 1 0

f 1/3 2/6 3/8 4/9
f 5/11 3/8 2/5
f 5/11 4/10 3/7
f 5/11 1/2 4/9
f 5/11 2/4 1/1
```

*.obj

In einer `*.obj`-Datei bedeutet:

- `v` float float float
  Eine Angabe einer Eckenposition im Raum (*vertex's geometric position*). Die erste Nennung in der Datei hat den Index 1, die folgendene werden der Reihe nach durchnummeriert.

- `vn` float float float
  Eine Angabe einer Normalen mit einer Indexierung analog zu `v`.

- `vt` float float
  Eine Angabe einer Textur-Koordinate mit einer Indexierung analog zu `v`.

- `f` int int int ... oder
  `f` int/int int/int int/int ... oder
  `f` int/int/int int/int/int int/int/int ...
  Eine Angabe einer Fläche, die durch ein Polygon spezifiziert wird (*Polygonal Face*). Die Zahlen sind Indizes im Feld der *Vertex*-Positionen, Textur-Koordinaten und der Normalen. Die Anzahl der *Vertices* für ein Polygon ist nicht begrenzt. Hinweis: Wird beim `ObjectFile`-Konstruktor die Angabe `ObjectFile.TRIANGULATE` genutzt, dann wird jede Fläche vom Java3D-`Triangulator` verarbeitet. Näheres zur Klasse `Triangulator` ↪ Listing 5.1 S. 126.

- `g` name
  Faces, die nach dieser benannten Gruppe vorkommen, werden als ein separates `Shape3D`-Objekt konstruiert. Dieses Objekt wird verknüpft mit der `SceneGroup` des Elternknotens. Mit der Methode `getNamedObjects()` erhält man diesen Gruppennamen.

Legende:
Objekt *Human Meg* aus Cinema4D-Bibliothek.
Quellcode ↪ S. 97

Abbildung 3.24: Beispiel: `ExampleWavefrontLoad` — Komplexes Objekt

Protokoll `ExampleWavefrontLoad.log`

```
D:\bonin\artsprog\code>java -version
java version "1.4.2_03"
Java(TM) 2 Runtime Environment,
  Standard Edition (build 1.4.2_03-b02)
Java HotSpot(TM) Client VM
  (build 1.4.2_03-b02, mixed mode)

D:\bonin\artsprog\code>javac
  de/fhnon/as/figure3D/ExampleWavefrontLoad.java

D:\bonin\artsprog\code>java
  de.fhnon.as.figure3D.ExampleWavefrontLoad
{pyramid=javax.media.j3d.Shape3D@4ac00c}

D:\bonin\artsprog\code>
```

Wenn man komplexe Objekte, zum Beispiel aus der *Cinema4D*-Bibliothek (hier das Objekt *Human Meg*), im Wavefront-Format importieren will, dann reicht der Speicherplatz, den die JVM (*Java Virtual Maschine*) beim Standardaufruf reserviert, nicht aus. Um die Applikation ausführen zu können, werden die *HotSpot Memory Options* genutzt. Die Parameter `-Xms` und `-Xmx` stellen die *Heap Allocation* auf die benötigten Werte ein. Im Beispiel werden 512 MB allokiert:

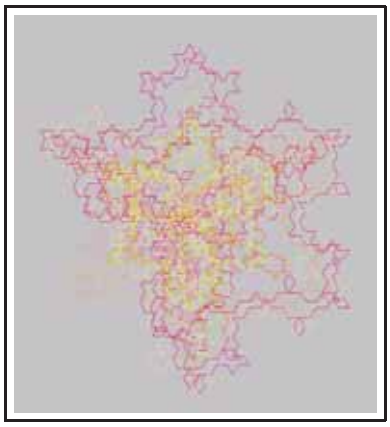
Memory Options

```
>java -Xms512m -Xmx512m
  de.uni-lueneburg.as.figure3D.ExampleWavefrontLoad
```

`-Xms`Size setzt den initialen Java^{TM}-Heap-Wert und `-Xmx`Size den maximalen Java^{TM}-Heap-Wert beim Start der Anwendung. Ein Beispiel mit diesem Aufruf zeigt Abbildung 3.24 S. 103.

Kapitel 4

Objekt-Verhalten — `Behavior`



Objekte können sich für den Beobachter im *Virtual Universe* im Laufe der Zeit ändern, beispielsweise können sie sich bewegen. Eine Verhaltensänderung des Objektes kann durch den Benutzer angestoßen werden oder aufgrund einer Kollision mit einem anderen Objekt erfolgen.

Zusätzlich zu Ort und Richtung der Bewegung kann sich die Geometry des Objektes, das heißt, seine Form und seine Farbe ändern. Auch der Beobachter kann seinen Standort und seinen Blickwinkel ändern.

Dieses Kapitel skizziert anhand von einigen Beispielen solche Änderungen des Objekt-Verhaltens.

Trainingsplan

Das Kapitel „Objekt-Verhalten“ gibt einen Überblick über:

- die Möglichkeiten der Anregung und die daraufhin eintretende Aktion,
  ↪ Seite 106 . . .
- die *Custom Behavior Class*,
  ↪ Seite 106 . . .
- beschreibt ein Bespiel zur Modifizierung der *View Platform* per Tastendruck,
  ↪ Seite 114 . . .
- und per Mausklick.
  ↪ Seite 120 . . .

4.1 Anregung und Aktion

Soll ein grafisches Objekt sich im *Virtual Universe* bewegen, dann sind Konstrukte zur Interaktion, Animation und Navigation anzuwenden. Die Tabelle 4.1 S. 107 skizziert diese Begriffe im Java3D-Kontext.

Ein einfaches Beispiel ist eine Box, die sich in Zeitschritten von einer halben Sekunde permanent dreht, wenn vorher eine Taste gedrückt worden ist. Solch ein spezielles Verhalten von Objekten (*Scene-Graph*-Teil) programmieren man auf der Basis der abstrakten Klasse `Behavior`.

4.2 *Custom Behavior Class*

Die abstrakten Klasse `Behavior` spezifiziert die beiden folgenden Methoden, die wir in unserer Subklasse überschreiben:

Start

- `void initialize()`
  Diese Methode wird einmal aufgerufen, wenn das Verhalten aktiviert wird. Sie übernimmt daher das Setzen des auslösenden Ereignisses, genannt *Trigger* und spezifiziert die Zustandsvariablen zu Beginn.

Anregung	**Aktion**			
Änderungsgrund (*Stimulus*)	**Objekt der Änderung**			
	`TransformGroup` (Ausrichtung/Ort)	`Geometry` (Form/-Farbe)	*Scene-Graph* (Hinzu/-Entfernen)	`View` (Ort/-Richtung)
Benutzer Taste, Maus	**Interaktion**	Anwendungsfall spezifisch		**Navigation**
Kollision	sichtbares Objekt			Ansichtsänderung
	ändert Ausrichtung / Ort	Erscheinung	verschwindet	
Zeit	**Animation**			
Betrachtungsort	*Billboard*	*Level of Detail*	Anwendungsfall spezifisch	

Legende:
Einordnung der Begriffe: Interaktion, Animation und Navigation
Ähnlich *The Java 3D Tutorial — Chapter 4*
↪http://java.sun.com/products/java-media/3D/collateral/j3d_tutorial_ch4.pdf (online 04-May-2004)

Tabelle 4.1: Objekt-Verhalten: Anregung und Aktion

Stimulus

- `void processStimulus(java.util.Enumeration criteria)`
  Diese Methode wird immer aufgerufen, wenn das Trigger-Ereignis für das spezifizierte Verhalten eintritt.

Trigger

Mit der Instanzmethode `wakeupOn(WakeupCondition criteria)` der Klasse `Behavior` überwachen wir die Bedingung für das Auslösen des jeweiligen Triggers (`WakeupCondition`). Da die Klasse `javax.media.j3d.WakeupOnAWTEvent` eine Subklasse von `javax.media.j3d.WakeupCondition` ist, können wir mit ihr ein `WakeupCondition`-Objekt konstruieren. Mit der Übergabe des Wertes von `KeyEvent.KEY_PRESSED` aus der Klasse `java.awt.event.KeyEvent` ist die Verknüpfung zur Tastatur realisiert. Mit der Klasse `javax.media.j3d.WakeupOnElapsedTime` konstruieren wir ein `WakeupCondition`-Objekt, das in 500 Millisekunden triggert. Der folgende Codeausschnitt skizziert diese Konstruktion:

Listing 4.1: Detail *Trigger*

```
MyBehavior(TransformGroup tg)
2       {
            this.tg = tg;
4       }
public void initialize()
6   {
    this.wakeupOn(
8           new WakeupOnAWTEvent(
            KeyEvent.KEY_PRESSED));
10  }
public void processStimulus(Enumeration criteria)
12  {
```

Legende:
Quellcode ↪ S. 108

Abbildung 4.1: Beispiel: KeyPressRotation — Hintergrundtextur

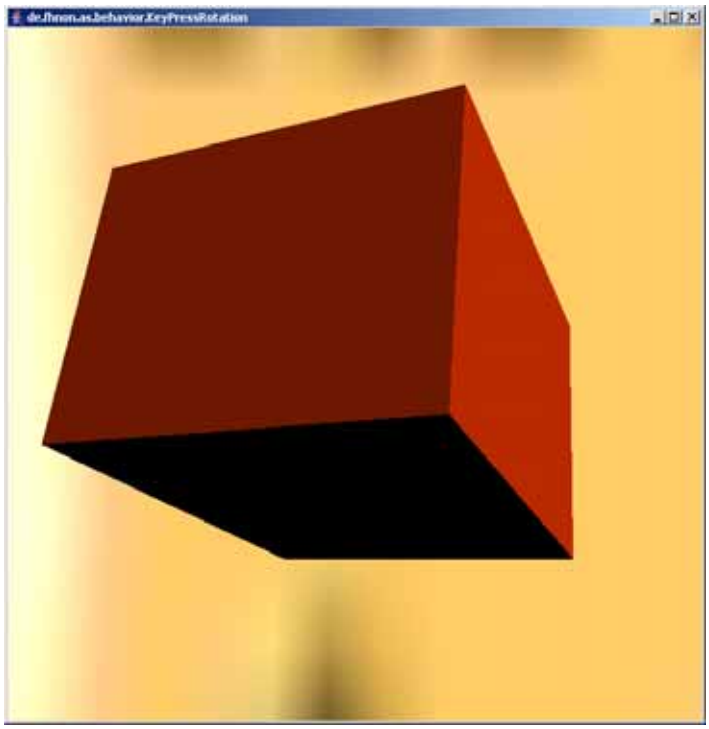
```
...
14      this.wakeupOn(
                new WakeupOnElapsedTime(500));
16  }
```

4.3 KeyPressRotation

Als Basis haben wir die Klasse SimpleFigure3Db (↪ Abschnitt 3.3.3 S. 62) genutzt und die Kugel durch ein Objekt der Klasse com.sun.j3d.-utils.geometry.Box ersetzt. Diese Konstruktion wurde um die innere Klasse MyBehavior ergänzt. Die Textur für den Hintergrund zeigt Abbildung 4.1 S. 108. Das Ergebnis, nach dem drücken einer beliebigen Taste und dem Warten von einigen Sekunden, zeigt Abbildung 4.2 S. 109.

Listing 4.2: KeyPressRoatation

```
/**
2  *  "Java 3D Example" Rotation auf Tastendruck
   *
4  *@author      Bonin
   *@version     1.0
6  */

8  package de.fhnon.as.behavior;

10 import java.applet.Applet;
   import java.awt.BorderLayout;
12 import java.awt.GraphicsConfiguration;
   import java.awt.event.KeyEvent;
14 import java.util.Enumeration;

16 import com.sun.j3d.utils.geometry.Primitive;
   import com.sun.j3d.utils.geometry.Box;
18 import com.sun.j3d.utils.geometry.Sphere;

20 import com.sun.j3d.utils.image.TextureLoader;
   import com.sun.j3d.utils.universe.SimpleUniverse;
22
   import com.sun.j3d.utils.applet.MainFrame;
24
   import javax.media.j3d.Appearance;
26 import javax.media.j3d.Background;
   import javax.media.j3d.Behavior;
28 import javax.media.j3d.BranchGroup;
```



Legende:
Quellcode ↪ S. 108

Abbildung 4.2: Beispiel: KeyPressRotation

```
import javax.media.j3d.BoundingSphere;
import javax.media.j3d.Canvas3D;
import javax.media.j3d.DirectionalLight;
import javax.media.j3d.Material;
import javax.media.j3d.TexCoordGeneration;
import javax.media.j3d.Texture;
import javax.media.j3d.TextureAttributes;
import javax.media.j3d.Transform3D;
import javax.media.j3d.TransformGroup;
import javax.media.j3d.WakeupOnAWTEvent;
import javax.media.j3d.WakeupOnElapsedTime;

import javax.vecmath.AxisAngle4f;
import javax.vecmath.Color3f;
import javax.vecmath.Color4f;
import javax.vecmath.Point3d;
import javax.vecmath.Vector3f;

public class KeyPressRotation extends Applet
{
   private SimpleUniverse u = null;


   public class MyBehavior extends Behavior
   {
      private int angle = 0;

      private TransformGroup tg;
      private Transform3D t3d = new Transform3D();


      MyBehavior(TransformGroup tg)
      {
         this.tg = tg;
      }


      public void initialize()
      {

         this.wakeupOn(
               new WakeupOnAWTEvent(
               KeyEvent.KEY_PRESSED));
      }


      public void processStimulus(
            Enumeration criteria)
      {

         angle += 5;

         t3d.setRotation(
               new AxisAngle4f(
               1f,
               1f,
```

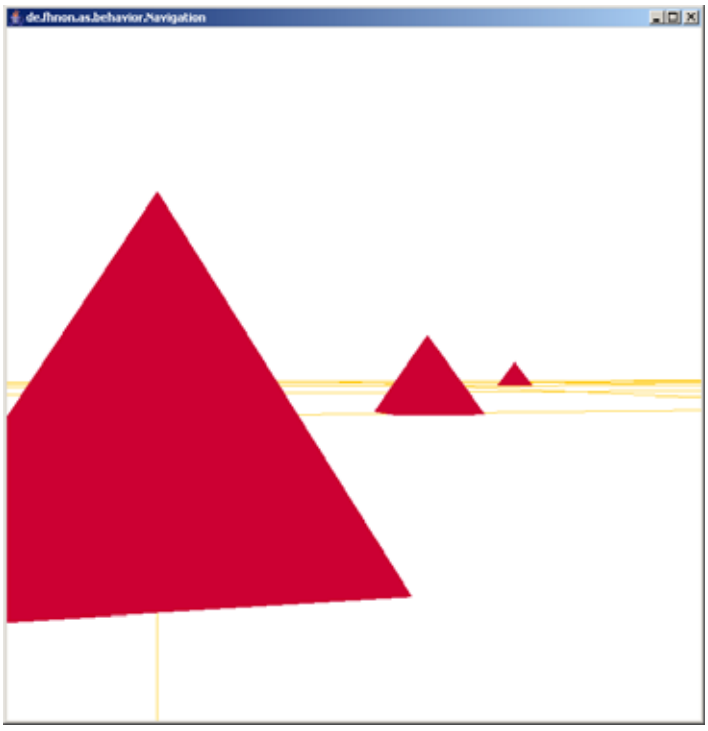
```
                0f,
86              (float) Math.toRadians(angle)));

88          tg.setTransform(t3d);
            this.wakeupOn(
90                  new WakeupOnElapsedTime(500));
        }
92
    }
94


96
    private KeyPressRotation()
98  {
        super();
100 }

102
    public BranchGroup createSceneGraph()
104 {
        BranchGroup bg = new BranchGroup();
106     TransformGroup tg = new TransformGroup();
        tg.setCapability(
108             TransformGroup.ALLOW_TRANSFORM_WRITE);

110     Appearance app = new Appearance();

112     app.setMaterial(
                new Material(
114             new Color3f(0.9f, 0.2f, 1.0f),
                new Color3f(0.0f, 0.0f, 0.0f),
116             new Color3f(0.9f, 0.2f, 1.0f),
                new Color3f(0.0f, 0.0f, 0.0f),
118             60.0f));

120     Box box =
                new Box(
122             0.4f, 0.5f, 0.6f,
                Primitive.GENERATE_NORMALS, app);
124
        tg.addChild(box);
126     bg.addChild(tg);

128     MyBehavior keyPressBehavior =
                new MyBehavior(tg);
130     keyPressBehavior.setSchedulingBounds(
                new BoundingSphere());
132     bg.addChild(keyPressBehavior);

134     bg.compile();
        return bg;
136 }

138
    public BranchGroup createBackground()
140 {
```

```
        BranchGroup bg =
142             new BranchGroup();
        Background back = new Background();
144     back.setApplicationBounds(
                getBoundingSphere());
146     BranchGroup bgGeometry =
                new BranchGroup();
148     Appearance app = new Appearance();

150     Texture tex = new TextureLoader(
                "de/fhnon/as/behavior/validxhtml.jpg",
152             null).getTexture();
        app.setTexture(tex);
154
        app.setTexCoordGeneration(
156             new TexCoordGeneration(
                TexCoordGeneration.SPHERE_MAP,
158             TexCoordGeneration.TEXTURE_COORDINATE_2));
        app.setTextureAttributes(
160             new TextureAttributes(
                TextureAttributes.REPLACE,
162             new Transform3D(),
                new Color4f(),
164             TextureAttributes.NICEST));

166     Sphere sphere = new Sphere(1.0f,
                Primitive.GENERATE_TEXTURE_COORDS |
168             Primitive.GENERATE_NORMALS_INWARD, 40, app);

170     bgGeometry.addChild(sphere);
        back.setGeometry(bgGeometry);
172     bg.addChild(back);
        return bg;
174 }

176
    public void addLights(BranchGroup bg)
178 {
        DirectionalLight light =
180             new DirectionalLight(
                new Color3f(1.0f, 1.0f, 0.0f),
182             new Vector3f(-1.0f, -1.0f, -1.0f));
        light.setInfluencingBounds(
184             this.getBoundingSphere());
        bg.addChild(light);
186 }

188
    public TransformGroup createBehaviors(
190         BranchGroup bg)
    {
192     TransformGroup objTrans =
                new TransformGroup();
194     bg.addChild(objTrans);
        return objTrans;
196 }
```

```
198
    BoundingSphere getBoundingSphere()
200 {
       return new BoundingSphere(
202          new Point3d(0.0, 0.0, 0.0), 200.0);
    }
204

206 public void init()
    {
208    this.setLayout(new BorderLayout());
       GraphicsConfiguration config =
210          SimpleUniverse.getPreferredConfiguration();
       Canvas3D c = new Canvas3D(config);
212    this.add("Center", c);
       u = new SimpleUniverse(c);
214    u.getViewingPlatform().
             setNominalViewingTransform();
216    u.addBranchGraph(this.createBackground());

218    BranchGroup bgRoot = new BranchGroup();
       TransformGroup tg = this.createBehaviors(bgRoot);
220    tg.addChild(this.createSceneGraph());
       this.addLights(bgRoot);
222    u.addBranchGraph(bgRoot);

224 }

226
    public void destroy()
228 {
       u.cleanup();
230 }

232
    public static void main(String[] args)
234 {
       new MainFrame(new KeyPressRotation(), 700, 700);
236 }

238 }
```

Protokoll `KeyPressRotation.log`

```
D:\bonin\artsprog\code>java -version
java version "1.4.2_03"
Java(TM) 2 Runtime Environment,
  Standard Edition (build 1.4.2_03-b02)
Java HotSpot(TM) Client VM
  (build 1.4.2_03-b02, mixed mode)

D:\bonin\artsprog\code>javac
  de/fhnon/as/behavior/KeyPressRotation.java

D:\bonin\artsprog\code>java
```



Legende:
Quellcode ↪ S. 115

Abbildung 4.3: Beispiel: `Navigation`

```
  de.fhnon.as.behavior.KeyPressRotation

D:\bonin\artsprog\code>
```

4.4 Navigation

Die Klasse `Navigation` (↪ S. 115) ist ein Beispiel zum Modifizieren der *View Platform* per Tastendruck. Die Idee für dieses Beispiel wurde entnommen aus: *The Java 3D Tutorial — Chapter 4*[1]. Der Quellcode wurde wesentlich anders gestaltet. Grundlage ist die Klasse `com.sun.j3d.utils.behaviors.keyboard.KeyNavigatorBehavior`. Ihr Konstruktor:

```
public KeyNavigatorBehavior(TransformGroup tg)
```

[1] ↪http://java.sun.com/products/java-media/3D/collateral/j3d_tutorial_ch4.pdf (online 04-May-2004)

Taste	Wirkung	mit Alt-Taste
←	Linksdrehung	Querverschiebung nach links
→	Rechtsdrehung	Querverschiebung nach rechts
↑	Verschiebung nach vorn	
↓	Verschiebung nachc hinten	
PgUp	Drehung nach oben	Verschiebung nach oben
PgDn	Drehung nach unten	Verschiebung nach unten

Tabelle 4.2: Klasse `KeyNavigatorBehavior` — Wirkungen der Tasten

konstruiert ein Verhalten, bezogen auf einen Tastendruck für den Graphen, der an den Parameter `tg` gebunden ist. Die Tabelle 4.2 S. 115 zeigt die Wirkung von Tasten, wenn das Fenster vorab aktiviert wurde (per linken Mausklick).

Listing 4.3: `Navigation`

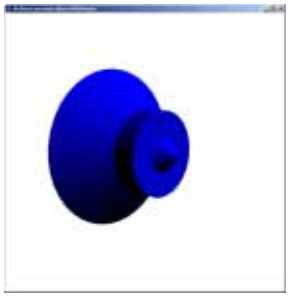
```
/**
 *  "Java 3D Example" Navigation mit Class
 *  com.sun.j3d.utils.behaviors.keyboard.KeyNavigatorBehavior;
 *
 *@author      Bonin
 *@version     1.0
 */

package de.fhnon.as.behavior;

import java.applet.Applet;
import java.awt.BorderLayout;
import java.awt.GraphicsConfiguration;

import com.sun.j3d.utils.behaviors.keyboard.KeyNavigatorBehavior;
import com.sun.j3d.utils.geometry.Primitive;
import com.sun.j3d.utils.geometry.Sphere;
import com.sun.j3d.utils.universe.SimpleUniverse;
import com.sun.j3d.utils.applet.MainFrame;

import javax.media.j3d.Appearance;
import javax.media.j3d.Background;

import javax.media.j3d.BranchGroup;
import javax.media.j3d.BoundingSphere;
import javax.media.j3d.Canvas3D;
import javax.media.j3d.DirectionalLight;
import javax.media.j3d.GeometryArray;
import javax.media.j3d.IndexedTriangleArray;
import javax.media.j3d.LineArray;
import javax.media.j3d.Link;
import javax.media.j3d.Material;
import javax.media.j3d.Shape3D;
```

```
34 import javax.media.j3d.SharedGroup;
   import javax.media.j3d.Transform3D;
36 import javax.media.j3d.TransformGroup;

38 import javax.vecmath.Color3f;
   import javax.vecmath.Point3d;
40 import javax.vecmath.Point3f;
   import javax.vecmath.Vector3f;
42
   public class Navigation extends Applet
44 {
      private SimpleUniverse u = null;
46

48    private Navigation()
      {
50       super();
      }
52

54    public BranchGroup createSceneGraph()
      {
56       BranchGroup bg = new BranchGroup();

58       TransformGroup tg = null;
         Vector3f vector = new Vector3f();
60       Transform3D t3d = new Transform3D();

62       TransformGroup tgLink = null;

64       bg.addChild(createLand());

66       SharedGroup share = new SharedGroup();
         share.addChild(createPyramid());
68
         float[][] position =
70             {{0.0f, 0.0f, -3.0f},
               {6.0f, 0.0f, 0.0f},
72             {6.0f, 0.0f, 6.0f},
               {3.0f, 0.0f, -10.0f},
74             {13.0f, 0.0f, -30.0f},
               {-13.0f, 0.0f, 30.0f},
76             {-13.0f, 0.0f, 23.0f},
               {13.0f, 0.0f, 3.0f}};
78
         for (int i = 0; i < position.length; i++)
80       {
            vector.set(position[i]);
82          t3d.setTranslation(vector);
            tgLink = new TransformGroup(t3d);
84          tgLink.addChild(new Link(share));
            bg.addChild(tgLink);
86       }
         tg = u.getViewingPlatform().
88           getViewPlatformTransform();
         vector.set(0.0f, 0.45f, 0.0f);
```

```
90          t3d.setTranslation(vector);
            tg.setTransform(t3d);
92          KeyNavigatorBehavior keyNavigating =
                new KeyNavigatorBehavior(tg);
94          keyNavigating.setSchedulingBounds(
                this.getBoundingSphere());
96          bg.addChild(keyNavigating);

98          bg.compile();

100         return bg;
        }
102

104     public BranchGroup createBackground()
        {
106         BranchGroup bg =
                new BranchGroup();
108         Background back = new Background();
            back.setApplicationBounds(
110             getBoundingSphere());
            BranchGroup bgGeometry =
112             new BranchGroup();
            Appearance app = new Appearance();
114
            Sphere sphere = new Sphere(1.0f,
116             Primitive.GENERATE_TEXTURE_COORDS |
                Primitive.GENERATE_NORMALS_INWARD,
118             40, app);

120         bgGeometry.addChild(sphere);
            back.setGeometry(bgGeometry);
122         bg.addChild(back);
            return bg;
124     }

126
        public void addLights(BranchGroup bg)
128     {
            DirectionalLight light =
130             new DirectionalLight(
                new Color3f(0.0f, 0.0f, 1.0f),
132             new Vector3f(-1.0f, -1.0f, -1.0f));
            light.setInfluencingBounds(
134             this.getBoundingSphere());
            bg.addChild(light);
136     }

138
        public TransformGroup createBehaviors(
140         BranchGroup bg)
        {
142         TransformGroup objTrans =
                new TransformGroup();
144         bg.addChild(objTrans);
            return objTrans;
```

```
146     }
148
        BoundingSphere getBoundingSphere()
150     {
            return new BoundingSphere(
152                 new Point3d(0.0, 0.0, 0.0), 10000.0);
        }
154
156     Shape3D createPyramid()
        {
158         IndexedTriangleArray pyramid =
                new IndexedTriangleArray(
160             5, GeometryArray.COORDINATES
                  | GeometryArray.COLOR_3
162             , 12);
164         pyramid.setCoordinate(
                0, new Point3f(0.0f, 0.9f, 0.0f));
166         pyramid.setCoordinate(
                1, new Point3f(-0.5f, 0.0f, -0.5f));
168         pyramid.setCoordinate(
                2, new Point3f(-0.5f, 0.0f, 0.5f));
170         pyramid.setCoordinate(
                3, new Point3f(0.5f, 0.0f, 0.5f));
172         pyramid.setCoordinate(
                4, new Point3f(0.5f, 0.0f, -0.5f));
174
            pyramid.setCoordinateIndex(0, 0);
176         pyramid.setCoordinateIndex(1, 1);
            pyramid.setCoordinateIndex(2, 2);
178
            pyramid.setCoordinateIndex(3, 0);
180         pyramid.setCoordinateIndex(4, 2);
            pyramid.setCoordinateIndex(5, 3);
182
            pyramid.setCoordinateIndex(6, 0);
184         pyramid.setCoordinateIndex(7, 3);
            pyramid.setCoordinateIndex(8, 4);
186
            pyramid.setCoordinateIndex(9, 0);
188         pyramid.setCoordinateIndex(10, 4);
            pyramid.setCoordinateIndex(11, 1);
190
            Color3f color = new Color3f(0.8f, 0.0f, 0.2f);
192         pyramid.setColor(0, color);
            pyramid.setColor(1, color);
194         pyramid.setColor(2, color);
            pyramid.setColor(3, color);
196         pyramid.setColor(4, color);

198         return new Shape3D(pyramid);
        }
200
```

```
202     Shape3D createLand()
        {
204        LineArray landGeom = new LineArray(
                44, GeometryArray.COORDINATES
206                | GeometryArray.COLOR_3);
           float l = -50.0f;
208        for (int c = 0; c < 44; c += 4)
           {
210
              landGeom.setCoordinate(
212                   c + 0, new Point3f(-50.0f, 0.0f, l));
              landGeom.setCoordinate(
214                   c + 1, new Point3f(50.0f, 0.0f, l));
              landGeom.setCoordinate(
216                   c + 2, new Point3f(l, 0.0f, -50.0f));
              landGeom.setCoordinate(
218                   c + 3, new Point3f(l, 0.0f, 50.0f));
              l += 10.0f;
220        }

222        Color3f c = new Color3f(1.0f, 0.8f, 0.1f);
           for (int i = 0; i < 44; i++)
224        {
              landGeom.setColor(i, c);
226        }

228        return new Shape3D(landGeom);
        }
230

232     public void init()
        {
234        this.setLayout(new BorderLayout());
           GraphicsConfiguration config =
236             SimpleUniverse.getPreferredConfiguration();
           Canvas3D c = new Canvas3D(config);
238        this.add("Center", c);
           u = new SimpleUniverse(c);
240        u.getViewingPlatform().
                setNominalViewingTransform();
242        u.addBranchGraph(this.createBackground());

244        BranchGroup bgRoot = new BranchGroup();
           TransformGroup tg = this.createBehaviors(bgRoot);
246        tg.addChild(this.createSceneGraph());
           this.addLights(bgRoot);
248        u.addBranchGraph(bgRoot);

250     }

252
        public void destroy()
254     {
           u.cleanup();
256     }
```



Legende:
Quellcode ↪ S. 121

Abbildung 4.4: Beispiel: `ObjectWithMouse`

```
258
    public static void main(String[] args)
260 {
       new MainFrame(new Navigation(), 700, 700);
262 }

264 }
```

Protokoll `Navigation.log`

```
D:\bonin\artsprog\code>java -version
java version "1.4.2_03"
Java(TM) 2 Runtime Environment,
  Standard Edition (build 1.4.2_03-b02)
Java HotSpot(TM) Client VM
  (build 1.4.2_03-b02, mixed mode)

D:\bonin\artsprog\code>javac
  de/fhnon/as/behavior/Navigation.java

D:\bonin\artsprog\code>java
  de.fhnon.as.behavior.Navigation

D:\bonin\artsprog\code>
```

4.5 `ObjectWithMouse`

Die Klasse `ObjectWithMouse` (↪ S. 121) ist ein Beispiel zum Modifizieren der *View Platform* per Mausklick. Grundlage bildet die Klasse `com.sun.j3d.utils.behaviors.mouse.MouseRotate`. Sie wird folgendermaßen genutzt:

```
MouseRotate myMouseRotate = new MouseRotate();
myMouseRotate.setTransformGroup(tg);
```

```
myMouseRotate.setSchedulingBounds(new BoundingSphere());
bg.addChild(myMouseRotate);
```

Wird die linke Maustaste gedrückt gehalten und die Maus im Fenster bewegt, dann dreht sich das Objekt, hier ein Kegel (Instanz von `com.sun.j3d.utils.geometry.Cone`) kombiniert mit einem Zylinder (Instanz von `com.sun.j3d.utils.geometry.Cylinder`), siehe ↪ Abbildung 4.4 S. 120.

Listing 4.4: `ObjectWithMouse`

```
/**
 *  "Java 3D Example" Rotation mit Class
 *  com.sun.j3d.utils.behaviors.mouse.MouseRotate
 *
 *@author      Bonin
 *@version     1.0
 */

package de.fhnon.as.rotate;

import java.applet.Applet;
import java.awt.BorderLayout;
import java.awt.GraphicsConfiguration;

import com.sun.j3d.utils.geometry.Cone;
import com.sun.j3d.utils.geometry.Cylinder;
import com.sun.j3d.utils.geometry.Primitive;
import com.sun.j3d.utils.geometry.Sphere;
import com.sun.j3d.utils.universe.SimpleUniverse;
import com.sun.j3d.utils.applet.MainFrame;
import com.sun.j3d.utils.behaviors.mouse.MouseRotate;
import javax.media.j3d.Appearance;
import javax.media.j3d.Background;

import javax.media.j3d.BranchGroup;
import javax.media.j3d.BoundingSphere;
import javax.media.j3d.Canvas3D;
import javax.media.j3d.DirectionalLight;
import javax.media.j3d.GeometryArray;
import javax.media.j3d.Material;
import javax.media.j3d.Shape3D;
import javax.media.j3d.Transform3D;
import javax.media.j3d.TransformGroup;

import javax.vecmath.Color3f;
import javax.vecmath.Point3d;
import javax.vecmath.Vector3f;

public class ObjectWithMouse extends Applet
{
   private SimpleUniverse u = null;


   private ObjectWithMouse()
   {
      super();
```

```
    }
48
50  public BranchGroup createSceneGraph()
    {
52      BranchGroup bg = new BranchGroup();

54      TransformGroup tg = new TransformGroup();
        tg.setCapability(TransformGroup.ALLOW_TRANSFORM_WRITE);
56      tg.setCapability(TransformGroup.ALLOW_TRANSFORM_READ);

58      bg.addChild(tg);

60      Appearance app = new Appearance();
        Color3f ambientC =
62          new Color3f(0.9f, 0.2f, 1.0f);
        Color3f emissiveC =
64          new Color3f(0.0f, 0.0f, 0.0f);
        Color3f diffuseC =
66          new Color3f(0.9f, 0.2f, 1.0f);
        Color3f specularC =
68          new Color3f(0.0f, 0.0f, 0.0f);
        float shininess = 80.0f;
70
        app.setMaterial(
72          new Material(
            ambientC,
74          emissiveC,
            diffuseC,
76          specularC,
            shininess));
78
        tg.addChild(new Cone(0.6f, 0.6f,
80          Primitive.GENERATE_NORMALS,
            40, 40,
82          app));

84      tg.addChild(new Cylinder(0.3f, 0.35f,
            Primitive.GENERATE_NORMALS,
86          40, 40,
            app));
88

90      MouseRotate myMouseRotate = new MouseRotate();
        myMouseRotate.setTransformGroup(tg);
92      myMouseRotate.setSchedulingBounds(
            new BoundingSphere());
94      bg.addChild(myMouseRotate);

96      bg.compile();

98      return bg;
    }
100

102 public BranchGroup createBackground()
```

```
    {
104     BranchGroup bg =
                new BranchGroup();
106     Background back = new Background();
        back.setApplicationBounds(
108             getBoundingSphere());
        BranchGroup bgGeometry =
110             new BranchGroup();
        Appearance app = new Appearance();
112
        Sphere sphere = new Sphere(1.0f,
114             Primitive.GENERATE_TEXTURE_COORDS |
                Primitive.GENERATE_NORMALS_INWARD,
116             40, app);

118     bgGeometry.addChild(sphere);
        back.setGeometry(bgGeometry);
120     bg.addChild(back);
        return bg;
122 }

124
    public void addLights(BranchGroup bg)
126 {
        DirectionalLight light =
128             new DirectionalLight(
                new Color3f(0.0f, 0.0f, 1.0f),
130             new Vector3f(-1.0f, -1.0f, -1.0f));
        light.setInfluencingBounds(
132             this.getBoundingSphere());
        bg.addChild(light);
134 }

136
    public TransformGroup createBehaviors(
138         BranchGroup bg)
    {
140     TransformGroup objTrans =
                new TransformGroup();
142     bg.addChild(objTrans);
        return objTrans;
144 }

146
    BoundingSphere getBoundingSphere()
148 {
        return new BoundingSphere(
150             new Point3d(0.0, 0.0, 0.0), 10000.0);
    }
152

154 public void init()
    {
156     this.setLayout(new BorderLayout());
        GraphicsConfiguration config =
158             SimpleUniverse.getPreferredConfiguration();
```

```
        Canvas3D c = new Canvas3D(config);
160     this.add("Center", c);
        u = new SimpleUniverse(c);
162     u.getViewingPlatform().
              setNominalViewingTransform();
164     u.addBranchGraph(this.createBackground());

166     BranchGroup bgRoot = new BranchGroup();
        TransformGroup tg = this.createBehaviors(bgRoot);
168     tg.addChild(this.createSceneGraph());
        this.addLights(bgRoot);
170     u.addBranchGraph(bgRoot);

172   }

174
      public void destroy()
176   {
        u.cleanup();
178   }

180
      public static void main(String[] args)
182   {
        new MainFrame(new ObjectWithMouse(), 700, 700);
184   }

186 }
```

Protokoll `ObjectWithMouse.log`

```
D:\bonin\artsprog\code>java -version
java version "1.4.2_03"
Java(TM) 2 Runtime Environment,
  Standard Edition (build 1.4.2_03-b02)
Java HotSpot(TM) Client VM
  (build 1.4.2_03-b02, mixed mode)

D:\bonin\artsprog\code>javac
  de/fhnon/as/rotate/ObjectWithMouse.java

D:\bonin\artsprog\code>java
  de.fhnon.as.rotate.ObjectWithMouse

D:\bonin\artsprog\code>
```

Kapitel 5

Beispielprojekt Jagdhund

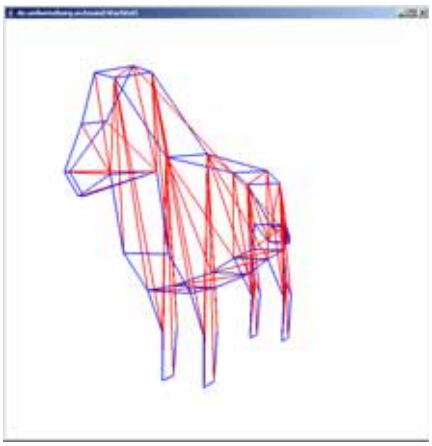
Um einen Jagdhund, beispielsweise der Rasse „Deutscher Wachtelhund“ (DW), darzustellen, ist dieser in Java3D vollständig aus Dreiecken (*triangles*) und/oder Vierecken (*quads*) zu konstruieren.

Die Erfassung der großen Menge solcher Konstrukte ist sehr aufwendig. Sie wird in Java3D etwas erleichtert, weil Polygone mit Hilfe der Klasse `com.sun.j3d.utils.geometry.GeometryInfo` automatisch in solche Konstrukte konvertiert werden.

Trainingsplan

Das Kapitel „Beispielprojekt Jagdhund“ zeigt:

- eine Jagdhundskizze konstruiert aus Polygonen mit 30 Punkten und
  ↪ Seite 126 . . .
- die entsprechende Java-Klasse mit ihren genutzen Daten.
  ↪ Seite 128 . . .



Legende:
Quellcode ↪ S. 128

Abbildung 5.1: Beispiel: `Wachtel1` — Dreiecke

5.1 1. Ansatz mit `GeometryInfo` — `Wachtel1`

Im 1. Ansatz des Projektes „Jagdhund“ konstruieren wir einen Wachtel aus zwei quasi parallel verlaufenden Polygonen mit 30 Punkten (von P_0 bis P_{28}), wobei der erste Punkt dieselben Koordinatenwerte hat wie der dreißigste Punkt. So entsteht ein „geschlossenes Polygon“. Die beiden Polygone unterscheiden sich nur durch die z-Werte ihrer Punkte. Die Dicke des Hundes ist am Rutenende und am Fang wesentlich kleiner als am Rücken. Das wird durch kleinere z-Werte erreicht, weil die x-Achse als Längsachse des Hundes dient. Der Nullpunkt des Koordinatensystems liegt ungefähr im Hundemagen. Zusätzlich spezifizieren wir Polygone, jeweils aus fünf Koordinatenangaben (Anfangspunkt gleich Endpunkt), die die beiden Hundeseiten miteinander verknüpfen. Das Ergebnis zeigt Abbildung 5.1 S. 126.

Die automatische Berechnung der Dreiecke aus den Polygonen verläuft nach folgendem Verfahren:

Listing 5.1: Detail *Triangles*-Ermittlung

```
float[] coordinateData = createCoordinateData();

// üfr jedes Polygon die Anzahl seiner Punkte
int[] stripCount = {30, 30, 5, 5, ...};

GeometryInfo gi = new GeometryInfo(
  GeometryInfo.POLYGON_ARRAY);
gi.setCoordinates(coordinateData);
gi.setStripCounts(stripCount);

```

```
Triangulator tr = new Triangulator();

tr.triangulate(gi);

NormalGenerator ng = new NormalGenerator();
ng.generateNormals(gi);

Stripifier st = new Stripifier();
st.stripify(gi);

Shape3D figure = new Shape3D();

figure.setAppearance(...);

figure.setGeometry(gi.getGeometryArray());
```

Mit dem Parameterwert `GeometryInfo.POLYGON_ARRAY` zeigen wir der `GeometryInfo`-Instanz `gi` an, dass unsere Koordinaten auch nichtplanare Polygone spezifizieren. Mit der Klasse `com.sun.j3d.utils.geometry.Triangulator` erzeugen wir daraus die Dreiecke. Die zugehörende *Normals* berechnet die Instanzmethode `generateNormals(gi)`, der Klasse `com.sun.j3d.utils.geometry.NormalGenerator`. Anschließend ändern wir die Primitiven (Dreiecke) in unserem `GeometryInfo`-Objekt `gi` in *Triangle Strips*. Den Wachtel, das eigenliche `Shape3D`-Objekt `figure`, erzeugen wir dann mit der Geometrie, die wir von `gi` durch Anwendung der Methode `getGeometryArray()` erhalten.

Mit der Klasse `javax.media.j3d.LineStripArray` stellen wir unsere Ausgangspolygone dar (↪ Abbildung 5.2 S. 128). Diese Klasse hat folgenden Konstruktor:

```
LineStripArray(
  int vertexCount,  // hier 160
  int vertexFormat, // hier LineArray.COORDINATES
  int[] stripVertexCounts) // hier stripCount
```

Damit erzeugen wir das leere `LineStripArray`-Objekt `lineArray`. Die Koordinaten verknüpfen wir mit der Methode:

```
setCoordinate(
  int index, // hier 0
  float[] coordinate) // hier coordinateData
```

Ihr erster Parameter `index` gibt die Einstiegsecke in der Geometry an. Der zweite Parameter `coordinate` ist die Referenz zu den einzelnen Polygonpunkten, strukturiert als 3 Werte für einen Punkt.

Um die erzeugten Dreiecke besser sehen zu können, haben wir ein langsames, permanentes Drehen des Hundes eingebaut und zwar folgendermaßen:

Listing 5.2: Detail *Rotation*

Legende:
Quellcode ↪ S. 128

Abbildung 5.2: Beispiel: `Wachtel1` — Ausgangspolygone

```
Alpha rotationAlpha = new Alpha(-1, 144000);

RotationInterpolator rotator =
            new RotationInterpolator(
            rotationAlpha, objSpin);

rotator.setSchedulingBounds(getBoundingSphere());

objSpin.addChild(rotator);
```

Mit dem Wert `-1` für den ersten Parameter beim Konstruktor der Klasse `javax.media.j3d.Alpha` wird eine permanente Rotation erreicht. Der zweite Wert, hier 144000, legt die Periodendauer fest, die *Alpha* benötigt um von null bis eins zu gehen.

5.2 Klasse `Wachtel1`

Listing 5.3: `Wachtel1`

```
/**
 *  Projekt Jagdhund 1. Ansatz
 *
 *@author       Bonin
 *@version      1.0
 */

package de.unilueneburg.as.hound;

import java.applet.Applet;
```

```
import java.awt.BorderLayout;
import java.awt.GraphicsConfiguration;

import com.sun.j3d.utils.geometry.GeometryInfo;
import com.sun.j3d.utils.applet.MainFrame;
import com.sun.j3d.utils.geometry.NormalGenerator;
import com.sun.j3d.utils.universe.SimpleUniverse;
import com.sun.j3d.utils.geometry.Stripifier;
import com.sun.j3d.utils.geometry.Triangulator;

import javax.media.j3d.Alpha;
import javax.media.j3d.AmbientLight;
import javax.media.j3d.Appearance;
import javax.media.j3d.Background;
import javax.media.j3d.BoundingSphere;
import javax.media.j3d.BranchGroup;
import javax.media.j3d.Canvas3D;
import javax.media.j3d.ColoringAttributes;
import javax.media.j3d.DirectionalLight;
import javax.media.j3d.LineArray;
import javax.media.j3d.LineAttributes;
import javax.media.j3d.LineStripArray;
import javax.media.j3d.Material;
import javax.media.j3d.RotationInterpolator;
import javax.media.j3d.Shape3D;
import javax.media.j3d.TransformGroup;
import javax.media.j3d.Transform3D;
import javax.media.j3d.PolygonAttributes;

import javax.vecmath.AxisAngle4f;
import javax.vecmath.Color3f;
import javax.vecmath.Vector3f;

public class Wachtel1 extends Applet
{

    private SimpleUniverse u = null;


    private Wachtel1()
    {
        super();
    }


    BranchGroup createSceneGraph()
    {

        BranchGroup bg = new BranchGroup();

        float[] coordinateData =
                createCoordinateData();

        int[] stripCount = {30, 30, 5, 5, 5, 5, 5, 5, 5,
                5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5};
```

```
68      GeometryInfo gi = new GeometryInfo(
            GeometryInfo.POLYGON_ARRAY);
70      gi.setCoordinates(coordinateData);
        gi.setStripCounts(stripCount);
72
        Triangulator tr = new Triangulator();
74
        tr.triangulate(gi);
76
        /*
78       * to guarantee connection information
         */
80      gi.recomputeIndices();

82      NormalGenerator ng = new NormalGenerator();
        ng.generateNormals(gi);
84      gi.recomputeIndices();

86      Stripifier st = new Stripifier();
        st.stripify(gi);
88      gi.recomputeIndices();

90      Shape3D figure = new Shape3D();

92      figure.setAppearance(
            createWireFrameAppearance());
94
        figure.setGeometry(gi.getGeometryArray());
96
        TransformGroup objSpin = new TransformGroup();
98      objSpin.setCapability(
            TransformGroup.ALLOW_TRANSFORM_WRITE);
100     bg.addChild(objSpin);

102     objSpin.addChild(figure);

104     LineStripArray lineArray = new LineStripArray(
            160,
106         LineArray.COORDINATES,
            stripCount);
108
        lineArray.setCoordinates(0, coordinateData);
110
        objSpin.addChild(
112         new Shape3D(
            lineArray, createMyColorAppearance()));
114
        Alpha rotationAlpha = new Alpha(-1, 144000);
116     RotationInterpolator rotator =
            new RotationInterpolator(
118         rotationAlpha, objSpin);

120     rotator.setSchedulingBounds(getBoundingSphere());
        objSpin.addChild(rotator);
122
```

```
        addLights(bg);
124
        bg.compile();
126
        return bg;
128     }

130
    Appearance createMyColorAppearance()
132 {
        Appearance app =
134         new Appearance();
        ColoringAttributes coloringAtt =
136         new ColoringAttributes();
        coloringAtt.setColor(0.0f, 0.0f, 1.0f);
138     app.setColoringAttributes(
            coloringAtt);
140     LineAttributes lineAttrib =
            new LineAttributes();
142     lineAttrib.setLineWidth(1.5f);
        app.setLineAttributes(
144         lineAttrib);
        return app;
146 }

148

150 Appearance createWireFrameAppearance()
    {
152
        Appearance app =
154         new Appearance();
        PolygonAttributes polyAtt =
156         new PolygonAttributes();
        polyAtt.setPolygonMode(
158         PolygonAttributes.POLYGON_LINE);
        app.setPolygonAttributes(polyAtt);
160     ColoringAttributes coloringAtt =
            new ColoringAttributes();
162     coloringAtt.setColor(1.0f, 0.0f, 0.0f);
        app.setColoringAttributes(coloringAtt);
164     LineAttributes lineAttrib =
            new LineAttributes();
166     lineAttrib.setLineWidth(1.5f);
        app.setLineAttributes(
168         lineAttrib);
        return app;
170 }

172
    BranchGroup createBackground()
174 {
        BranchGroup bg = new BranchGroup();
176     Background background = new Background();
        background.setColor(1.0f, 1.0f, 1.0f);
178     background.setApplicationBounds(
```

```
            getBoundingSphere());
180         bg.addChild(background);
            return bg;
182     }

184

186     void addLights(BranchGroup bg)
        {
188         DirectionalLight lightD = new DirectionalLight();
            lightD.setDirection(
190             new Vector3f(0.0f, -0.7f, -0.7f));
            lightD.setInfluencingBounds(
192             getBoundingSphere());
            bg.addChild(lightD);
194
            AmbientLight lightA = new AmbientLight();
196         lightA.setInfluencingBounds(
                getBoundingSphere());
198         bg.addChild(lightA);
        }
200

202     BoundingSphere getBoundingSphere()
        {
204         return new BoundingSphere();
        }
206

208     public void init()
        {
210         this.setLayout(new BorderLayout());
            GraphicsConfiguration config =
212             SimpleUniverse.getPreferredConfiguration();
            Canvas3D c = new Canvas3D(config);
214         this.add("Center", c);
            u = new SimpleUniverse(c);
216         u.getViewingPlatform().
                setNominalViewingTransform();
218
            u.addBranchGraph(this.createBackground());
220
            u.addBranchGraph(
222             this.createSceneGraph());

224     }

226
        public void destroy()
228     {
            u.cleanup();
230     }

232
        public static void main(String[] args)
234     {
```

```
        new MainFrame(
236             new Wachtel1(), 700, 700);
    }
238

240 /*
     * only test data from an other example
242  */
    float[] createCoordinateData()
244 {

246     float[] data = new float[160 * 3];

248     int i = 0;
        /*
250      * Front side
         */
252     data[i++] = 0.94f;
        data[i++] = 0.1f;
254     data[i++] = 0.01f;
        // 0
256     data[i++] = 0.67f;
        data[i++] = 0.2f;
258     data[i++] = 0.1f;
        // 1
260     data[i++] = 0.59f;
        data[i++] = -0.09f;
262     data[i++] = 0.1f;
        // 2
264     data[i++] = 0.43f;
        data[i++] = -0.25f;
266     data[i++] = 0.1f;
        // 3
268     data[i++] = 0.32f;
        data[i++] = -0.44f;
270     data[i++] = 0.1f;
        // 4
272     data[i++] = 0.32f;
        data[i++] = -0.65f;
274     data[i++] = 0.1f;
        // 5
276     data[i++] = 0.23f;
        data[i++] = -0.65f;
278     data[i++] = 0.1f;
        // 6
280     data[i++] = 0.2f;
        data[i++] = -0.44f;
282     data[i++] = 0.1f;
        // 7
284     data[i++] = 0.22f;
        data[i++] = -0.25f;
286     data[i++] = 0.1f;
        // 8
288     data[i++] = -0.14f;
        data[i++] = -0.22f;
290     data[i++] = 0.1f;
```

```
        // 9
292     data[i++] = -0.41f;
        data[i++] = -0.16f;
294     data[i++] = 0.1f;
        // 10
296     data[i++] = -0.67f;
        data[i++] = -0.4f;
298     data[i++] = 0.1f;
        // 11
300     data[i++] = -0.64f;
        data[i++] = -0.65f;
302     data[i++] = 0.1f;
        // 12
304     data[i++] = -0.75f;
        data[i++] = -0.65f;
306     data[i++] = 0.1f;
        // 13
308     data[i++] = -0.8f;
        data[i++] = -0.4f;
310     data[i++] = 0.1f;
        // 14
312     data[i++] = -0.71f;
        data[i++] = -0.3f;
314     data[i++] = 0.1f;
        // 15
316     data[i++] = -0.67f;
        data[i++] = -0.12f;
318     data[i++] = 0.1f;
        // 16
320     data[i++] = -0.71f;
        data[i++] = 0.03f;
322     data[i++] = 0.1f;
        // 17
324     data[i++] = -0.83f;
        data[i++] = -0.08f;
326     data[i++] = 0.05f;
        // 18
328     data[i++] = -1.04f;
        data[i++] = -0.09f;
330     data[i++] = 0.01f;
        // 19
332     data[i++] = -1.04f;
        data[i++] = -0.04f;
334     data[i++] = 0.01f;
        // 20
336     data[i++] = -0.86f;
        data[i++] = 0.02f;
338     data[i++] = 0.05f;
        //  21
340     data[i++] = -0.6f;
        data[i++] = 0.26f;
342     data[i++] = 0.1f;
        //  22
344     data[i++] = -0.4f;
        data[i++] = 0.31f;
346     data[i++] = 0.1f;
```

```
        // 23
348     data[i++] = 0.29f;
        data[i++] = 0.31f;
350     data[i++] = 0.1f;
        //  24
352     data[i++] = 0.67f;
        data[i++] = 0.55f;
354     data[i++] = 0.08f;
        // 25
356     data[i++] = 0.78f;
        data[i++] = 0.54f;
358     data[i++] = 0.08f;
        // 26
360     data[i++] = 0.88f;
        data[i++] = 0.34f;
362     data[i++] = 0.05f;
        // 27
364     data[i++] = 1.0f;
        data[i++] = 0.17f;
366     data[i++] = 0.01f;
        // 28
368     data[i++] = 0.94f;
        data[i++] = 0.1f;
370     data[i++] = 0.01f;
        //0
372
374     /*
         * Back side
376      */
        data[i++] = 0.94f;
378     data[i++] = 0.1f;
        data[i++] = -0.01f;
380     // 0
        data[i++] = 0.67f;
382     data[i++] = 0.2f;
        data[i++] = -0.1f;
384     // 1
        data[i++] = 0.59f;
386     data[i++] = -0.09f;
        data[i++] = -0.1f;
388     // 2
        data[i++] = 0.43f;
390     data[i++] = -0.25f;
        data[i++] = -0.1f;
392     // 3
        data[i++] = 0.32f;
394     data[i++] = -0.44f;
        data[i++] = -0.1f;
396     // 4
        data[i++] = 0.32f;
398     data[i++] = -0.65f;
        data[i++] = -0.1f;
400     // 5
        data[i++] = 0.23f;
402     data[i++] = -0.65f;
```

```
        data[i++] = -0.1f;
404     // 6
        data[i++] = 0.2f;
406     data[i++] = -0.44f;
        data[i++] = -0.1f;
408     // 7
        data[i++] = 0.22f;
410     data[i++] = -0.25f;
        data[i++] = -0.1f;
412     // 8
        data[i++] = -0.14f;
414     data[i++] = -0.22f;
        data[i++] = -0.1f;
416     // 9
        data[i++] = -0.41f;
418     data[i++] = -0.16f;
        data[i++] = -0.1f;
420     // 10
        data[i++] = -0.67f;
422     data[i++] = -0.4f;
        data[i++] = -0.1f;
424     // 11
        data[i++] = -0.64f;
426     data[i++] = -0.65f;
        data[i++] = -0.1f;
428     // 12
        data[i++] = -0.75f;
430     data[i++] = -0.65f;
        data[i++] = -0.1f;
432     // 13
        data[i++] = -0.8f;
434     data[i++] = -0.4f;
        data[i++] = -0.1f;
436     // 14
        data[i++] = -0.71f;
438     data[i++] = -0.3f;
        data[i++] = -0.1f;
440     // 15
        data[i++] = -0.67f;
442     data[i++] = -0.12f;
        data[i++] = -0.1f;
444     // 16
        data[i++] = -0.71f;
446     data[i++] = 0.03f;
        data[i++] = -0.1f;
448     // 17
        data[i++] = -0.83f;
450     data[i++] = -0.08f;
        data[i++] = -0.05f;
452     // 18
        data[i++] = -1.04f;
454     data[i++] = -0.09f;
        data[i++] = -0.01f;
456     // 19
        data[i++] = -1.04f;
458     data[i++] = -0.04f;
```

```
        data[i++] = -0.01f;
460     // 20
        data[i++] = -0.86f;
462     data[i++] = 0.02f;
        data[i++] = -0.05f;
464     //  21
        data[i++] = -0.6f;
466     data[i++] = 0.26f;
        data[i++] = -0.1f;
468     //  22
        data[i++] = -0.4f;
470     data[i++] = 0.31f;
        data[i++] = -0.1f;
472     // 23
        data[i++] = 0.29f;
474     data[i++] = 0.31f;
        data[i++] = -0.1f;
476     //  24
        data[i++] = 0.67f;
478     data[i++] = 0.55f;
        data[i++] = -0.08f;
480     // 25
        data[i++] = 0.78f;
482     data[i++] = 0.54f;
        data[i++] = -0.08f;
484     // 26
        data[i++] = 0.88f;
486     data[i++] = 0.34f;
        data[i++] = -0.05f;
488     // 27
        data[i++] = 1.0f;
490     data[i++] = 0.17f;
        data[i++] = -0.01f;
492     // 28
        data[i++] = 0.94f;
494     data[i++] = 0.1f;
        data[i++] = -0.01f;
496     //0

498     /*
         *  Ruecken 23--24
500      */
        data[i++] = -0.4f;
502     data[i++] = 0.31f;
        data[i++] = +0.1f;
504     // 23 Front
        data[i++] = -0.4f;
506     data[i++] = 0.31f;
        data[i++] = -0.1f;
508     // 23 Back
        data[i++] = 0.29f;
510     data[i++] = 0.31f;
        data[i++] = -0.1f;
512     //  24 Back
        data[i++] = 0.29f;
514     data[i++] = 0.31f;
```

```
        data[i++] = 0.1f;
516     //  24 Front
        data[i++] = -0.4f;
518     data[i++] = 0.31f;
        data[i++] = +0.1f;
520     // 23 Front

522     /*
         * Hals oben 24--25
524      */
        data[i++] = 0.29f;
526     data[i++] = 0.31f;
        data[i++] = 0.1f;
528     //  24 Front
        data[i++] = 0.29f;
530     data[i++] = 0.31f;
        data[i++] = -0.1f;
532     //  24 Back
        data[i++] = 0.67f;
534     data[i++] = 0.55f;
        data[i++] = -0.08f;
536     // 25 Back
        data[i++] = 0.67f;
538     data[i++] = 0.55f;
        data[i++] = 0.08f;
540     // 25 Front
        data[i++] = 0.29f;
542     data[i++] = 0.31f;
        data[i++] = 0.1f;
544     //  24 Front

546     /*
         * Kopf oben 25--26
548      */
        data[i++] = 0.67f;
550     data[i++] = 0.55f;
        data[i++] = 0.08f;
552     // 25 Front
        data[i++] = 0.67f;
554     data[i++] = 0.55f;
        data[i++] = -0.08f;
556     // 25 Back
        data[i++] = 0.78f;
558     data[i++] = 0.54f;
        data[i++] = -0.08f;
560     // 26 Back
        data[i++] = 0.78f;
562     data[i++] = 0.54f;
        data[i++] = 0.08f;
564     // 26 Front
        data[i++] = 0.67f;
566     data[i++] = 0.55f;
        data[i++] = 0.08f;
568     // 25 Front

570     /*
```

```
572          * Stoss 26--27
             */
             data[i++] = 0.78f;
574          data[i++] = 0.54f;
             data[i++] = 0.08f;
576          // 26 Front
             data[i++] = 0.78f;
578          data[i++] = 0.54f;
             data[i++] = -0.08f;
580          // 26 Back
             data[i++] = 0.88f;
582          data[i++] = 0.34f;
             data[i++] = -0.05f;
584          // 27 Back
             data[i++] = 0.88f;
586          data[i++] = 0.34f;
             data[i++] = 0.05f;
588          // 27 Front
             data[i++] = 0.78f;
590          data[i++] = 0.54f;
             data[i++] = 0.08f;
592          // 26 Front

594          /*
             * Schnauze oben 27--28
596          */
             data[i++] = 0.88f;
598          data[i++] = 0.34f;
             data[i++] = 0.05f;
600          // 27 Front
             data[i++] = 0.88f;
602          data[i++] = 0.34f;
             data[i++] = -0.05f;
604          // 27 Back
             data[i++] = 1.0f;
606          data[i++] = 0.17f;
             data[i++] = -0.01f;
608          // 28 Back
             data[i++] = 1.0f;
610          data[i++] = 0.17f;
             data[i++] = 0.01f;
612          // 28 Front
             data[i++] = 0.88f;
614          data[i++] = 0.34f;
             data[i++] = 0.05f;
616          // 27 Front

618          /*
             * Schnauze vorn 28--0
620          */
             data[i++] = 1.0f;
622          data[i++] = 0.17f;
             data[i++] = 0.01f;
624          // 28 Front
             data[i++] = 1.0f;
626          data[i++] = 0.17f;
```

```
        data[i++] = -0.01f;
628     // 28 Back
        data[i++] = 0.94f;
630     data[i++] = 0.1f;
        data[i++] = -0.01f;
632     // 0 Back
        data[i++] = 0.94f;
634     data[i++] = 0.1f;
        data[i++] = 0.01f;
636     // 0 Front
        data[i++] = 1.0f;
638     data[i++] = 0.17f;
        data[i++] = 0.01f;
640     // 28 Front

642     /*
         * Schnauze unten 0--1
644      */
        data[i++] = 0.94f;
646     data[i++] = 0.1f;
        data[i++] = 0.01f;
648     // 0 Front
        data[i++] = 0.94f;
650     data[i++] = 0.1f;
        data[i++] = -0.01f;
652     // 0 Back
        data[i++] = 0.67f;
654     data[i++] = 0.2f;
        data[i++] = -0.1f;
656     // 1 Back
        data[i++] = 0.67f;
658     data[i++] = 0.2f;
        data[i++] = 0.1f;
660     // 1 Front
        data[i++] = 0.94f;
662     data[i++] = 0.1f;
        data[i++] = 0.01f;
664     // 0 Front

666     /*
         * Hals unten 1--2
668      */
        data[i++] = 0.67f;
670     data[i++] = 0.2f;
        data[i++] = 0.1f;
672     // 1 Front
        data[i++] = 0.67f;
674     data[i++] = 0.2f;
        data[i++] = -0.1f;
676     // 1 Back
        data[i++] = 0.59f;
678     data[i++] = -0.09f;
        data[i++] = -0.1f;
680     // 2 Back
        data[i++] = 0.59f;
682     data[i++] = -0.09f;
```

```
            data[i++] = 0.1f;
684         // 2 Front
            data[i++] = 0.67f;
686         data[i++] = 0.2f;
            data[i++] = 0.1f;
688         // 1 Front

690         /*
             * Brust 2--3
692          */
            data[i++] = 0.59f;
694         data[i++] = -0.09f;
            data[i++] = 0.1f;
696         // 2 Front
            data[i++] = 0.59f;
698         data[i++] = -0.09f;
            data[i++] = -0.1f;
700         // 2 Back
            data[i++] = 0.43f;
702         data[i++] = -0.25f;
            data[i++] = -0.1f;
704         // 3 Back
            data[i++] = 0.43f;
706         data[i++] = -0.25f;
            data[i++] = 0.1f;
708         // 3 Front
            data[i++] = 0.59f;
710         data[i++] = -0.09f;
            data[i++] = 0.1f;
712         // 2 Front

714         /*
             * Brust unten 3--8
716          */
            data[i++] = 0.43f;
718         data[i++] = -0.25f;
            data[i++] = 0.1f;
720         // 3 Front
            data[i++] = 0.43f;
722         data[i++] = -0.25f;
            data[i++] = -0.1f;
724         // 3 Back
            data[i++] = 0.22f;
726         data[i++] = -0.25f;
            data[i++] = -0.1f;
728         // 8 Back
            data[i++] = 0.22f;
730         data[i++] = -0.25f;
            data[i++] = 0.1f;
732         // 8 Front
            data[i++] = 0.43f;
734         data[i++] = -0.25f;
            data[i++] = 0.1f;
736         // 3 Front

738         /*
```

```
         * Bauch vorn 8--9
740      */
        data[i++] = 0.22f;
742     data[i++] = -0.25f;
        data[i++] = 0.1f;
744     // 8 Front
        data[i++] = 0.22f;
746     data[i++] = -0.25f;
        data[i++] = -0.1f;
748     // 8 Back
        data[i++] = -0.14f;
750     data[i++] = -0.22f;
        data[i++] = -0.1f;
752     // 9 Back
        data[i++] = -0.14f;
754     data[i++] = -0.22f;
        data[i++] = 0.1f;
756     // 9 Front
        data[i++] = 0.22f;
758     data[i++] = -0.25f;
        data[i++] = 0.1f;
760     // 8 Front

762     /*
         * Bauch hinten 9--10
764      */
        data[i++] = -0.14f;
766     data[i++] = -0.22f;
        data[i++] = 0.1f;
768     // 9 Front
        data[i++] = -0.14f;
770     data[i++] = -0.22f;
        data[i++] = -0.1f;
772     // 9 Back
        data[i++] = -0.41f;
774     data[i++] = -0.16f;
        data[i++] = -0.1f;
776     // 10 Back
        data[i++] = -0.41f;
778     data[i++] = -0.16f;
        data[i++] = 0.1f;
780     // 10 Front
        data[i++] = -0.14f;
782     data[i++] = -0.22f;
        data[i++] = 0.1f;
784     // 9 Front

786     /*
         * Bauch ganz hinten 10--16
788      */
        data[i++] = -0.41f;
790     data[i++] = -0.16f;
        data[i++] = 0.1f;
792     // 10 Front
        data[i++] = -0.41f;
794     data[i++] = -0.16f;
```

```
            data[i++] = -0.1f;
796         // 10 Back
            data[i++] = -0.67f;
798         data[i++] = -0.12f;
            data[i++] = -0.1f;
800         // 16 Back
            data[i++] = -0.67f;
802         data[i++] = -0.12f;
            data[i++] = 0.1f;
804         // 16 Front
            data[i++] = -0.41f;
806         data[i++] = -0.16f;
            data[i++] = 0.1f;
808         // 10 Front

810         /*
             * Schnalle 16--17
812          */
            data[i++] = -0.67f;
814         data[i++] = -0.12f;
            data[i++] = 0.1f;
816         // 16 Front
            data[i++] = -0.67f;
818         data[i++] = -0.12f;
            data[i++] = -0.1f;
820         // 16 Back
            data[i++] = -0.71f;
822         data[i++] = 0.03f;
            data[i++] = -0.1f;
824         // 17 Back
            data[i++] = -0.71f;
826         data[i++] = 0.03f;
            data[i++] = 0.1f;
828         // 17 Front
            data[i++] = -0.67f;
830         data[i++] = -0.12f;
            data[i++] = 0.1f;
832         // 16 Front

834         /*
             * Rute unten 17--18
836          */
            data[i++] = -0.71f;
838         data[i++] = 0.03f;
            data[i++] = 0.1f;
840         // 17 Front
            data[i++] = -0.71f;
842         data[i++] = 0.03f;
            data[i++] = -0.1f;
844         // 17 Back
            data[i++] = -0.83f;
846         data[i++] = -0.08f;
            data[i++] = -0.05f;
848         // 18 Back
            data[i++] = -0.83f;
850         data[i++] = -0.08f;
```

```
        data[i++] = 0.05f;
852     // 18 Front
        data[i++] = -0.71f;
854     data[i++] = 0.03f;
        data[i++] = 0.1f;
856     // 17 Front

858     /*
         * Rute unten hinten 18--19
860      */
        data[i++] = -0.83f;
862     data[i++] = -0.08f;
        data[i++] = 0.05f;
864     // 18 Front
        data[i++] = -0.83f;
866     data[i++] = -0.08f;
        data[i++] = -0.05f;
868     // 18 Back
        data[i++] = -1.04f;
870     data[i++] = -0.09f;
        data[i++] = -0.01f;
872     // 19 Back
        data[i++] = -1.04f;
874     data[i++] = -0.09f;
        data[i++] = 0.01f;
876     // 19 Front
        data[i++] = -0.83f;
878     data[i++] = -0.08f;
        data[i++] = 0.05f;
880     // 18 Front

882     /*
         * Rute hinten 19--20
884      */
        data[i++] = -1.04f;
886     data[i++] = -0.09f;
        data[i++] = 0.01f;
888     // 19 Front
        data[i++] = -1.04f;
890     data[i++] = -0.09f;
        data[i++] = -0.01f;
892     // 19 Back
        data[i++] = -1.04f;
894     data[i++] = -0.04f;
        data[i++] = -0.01f;
896     // 20 Back
        data[i++] = -1.04f;
898     data[i++] = -0.04f;
        data[i++] = 0.01f;
900     // 20 Front
        data[i++] = -1.04f;
902     data[i++] = -0.09f;
        data[i++] = 0.01f;
904     // 19 Front

906     /*
```

```
            * Rute hinten oben 20--21
908         */
            data[i++] = -1.04f;
910         data[i++] = -0.04f;
            data[i++] = 0.01f;
912         // 20 Front
            data[i++] = -1.04f;
914         data[i++] = -0.04f;
            data[i++] = -0.01f;
916         // 20 Back
            data[i++] = -0.86f;
918         data[i++] = 0.02f;
            data[i++] = -0.05f;
920         //  21 Back
            data[i++] = -0.86f;
922         data[i++] = 0.02f;
            data[i++] = 0.05f;
924         //  21 Front
            data[i++] = -1.04f;
926         data[i++] = -0.04f;
            data[i++] = 0.01f;
928         // 20 Front

930         /*
            * Rute oben 21--22
932         */
            data[i++] = -0.86f;
934         data[i++] = 0.02f;
            data[i++] = 0.05f;
936         //  21 Front
            data[i++] = -0.86f;
938         data[i++] = 0.02f;
            data[i++] = -0.05f;
940         //  21 Back
            data[i++] = -0.6f;
942         data[i++] = 0.26f;
            data[i++] = -0.1f;
944         //  22 Back
            data[i++] = -0.6f;
946         data[i++] = 0.26f;
            data[i++] = 0.1f;
948         //  22 Front
            data[i++] = -0.86f;
950         data[i++] = 0.02f;
            data[i++] = 0.05f;
952         //  21 Front

954         /*
            * Kruppe 22--23
956         */
            data[i++] = -0.6f;
958         data[i++] = 0.26f;
            data[i++] = 0.1f;
960         //  22 Front
            data[i++] = -0.6f;
962         data[i++] = 0.26f;
```

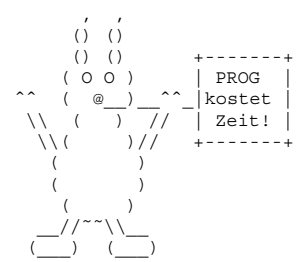
```
        data[i++] = -0.1f;
964     //  22 Back
        data[i++] = -0.4f;
966     data[i++] = 0.31f;
        data[i++] = -0.1f;
968     // 23 Back
        data[i++] = -0.4f;
970     data[i++] = 0.31f;
        data[i++] = 0.1f;
972     // 23 Front
        data[i++] = -0.6f;
974     data[i++] = 0.26f;
        data[i++] = 0.1f;
976     //  22 Front

978     return data;
    }
980
}
```

Protokoll `Wachtel1.log`

```
D:\bonin\prog\code>java -version
java version "1.5.0_04"
Java(TM) 2 Runtime Environment,
  Standard Edition (build 1.5.0_04-b05)
Java HotSpot(TM) Client VM
  (build 1.5.0_04-b05, mixed mode, sharing)

D:\bonin\prog\code>javac -deprecation
  de/unilueneburg/as/hound/Wachtel1.java
de/unilueneburg/as/hound/Wachtel1.java:73:
warning: Triangulator() in
  com.sun.j3d.utils.geometry.Triangulator
  has been deprecated
  Triangulator tr = new Triangulator();
                        ^
1 warning

D:\bonin\prog\code>java de.unilueneburg.as.hound.Wachtel1
```

Kapitel 6

Ausblick



Jeder Text von derartiger Länge und „Tiefe“ verlangt ein abschließendes Wort für seinen getreuen Leser. Es wäre nicht fair, nach 147 Seiten, die nächste aufzuschlagen und dann den Anhang zu finden. Daher zum Schluß ein kleiner Ausblick. Wenn man in Zukunft weit komplexere Aufgaben mit hinreichender Qualität programmieren will, dann **bedarf es eines neuen, oder zumindest fortentwickelten Paradigmas**.

Die Objekt-Orientierung in der klassischen Ausprägung von Java stößt schon heute oft an ihre Grenzen. Der hier gewählte Ansatz, die vielfältigen Konstrukte der Programmierung durch viele Beispiele aus dem Bereich Graphik zu verdeutlichen, möge Ihnen gefallen haben.

Einerseits wünsche ich Ihnen, dass Sie beim Durcharbeiten dieses Buches die Faszination der Programmierung selbst ausgiebig erfahren haben. Andererseits hoffe ich, dass Sie für die zukünftigen Paradigmen der Programmierung motiviert sind und ihnen zumindest positiv gegenüberstehen.

Anhang A

Übungen

Dieser Abschnitt enthält Aufgaben, die im Rahmen des Faches *Programmierung* im Studiengang *Wirtschaftsinformatik* der Fakultät III der Leuphana Universität Lüneburg gestellt wurden. Solche Aufgaben sind mit dem Zeichen † und anschließender Punktzahl *nP* gekennzeichnet.

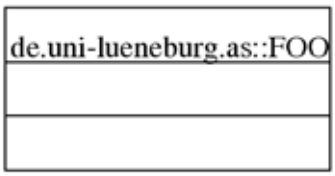
Bei einer Klausur von 120 Minuten sind 100 Punkte zu erreichen. Zum Bestehen einer Klausur müssen Sie mindestens die Hälfte der erreichbaren Punkte erzielen. Die Punktergebnisse werden wie folgt in die üblichen Noten umgerechnet:

- $\geq 95\% \equiv$ Note: $1,0$
- $\geq 92\% \equiv$ Note: $1,3$
- $\geq 89\% \equiv$ Note: $1,7$
- $\geq 80\% \equiv$ Note: $2,0$
- $\geq 77\% \equiv$ Note: $2,3$
- $\geq 74\% \equiv$ Note: $2,7$
- $\geq 65\% \equiv$ Note: $3,0$
- $\geq 62\% \equiv$ Note: $3,3$
- $\geq 59\% \equiv$ Note: $3,7$
- $\geq 50\% \equiv$ Note: $4,0$
- $< 50\% \equiv$ Note: nicht bestanden

Musterlösungen zu den Aufgaben finden Sie im Abschnitt B S. 179 ff. Eine dort notierte Lösung schließt andere, ebenfalls richtige Lösungen, natürlich nicht aus `:-)`.

A.1 UML-Klassensymbol programmieren

Programmieren Sie in PostScript das Klassensymbol von UML (*Unified Modeling Language*) und zwar in der Form wie es die Abbildung A.1 S. 150 zeigt.



Legende:
UML ≡ Unified Modeling Language

Abbildung A.1: UML Klassensymbol

A.2 PostScript-Kontur erläutern

Zeichnen Sie die Graphik, die durch den folgenden PostScript-Quellcode beschrieben ist.

Listing A.1: `halfcirclePS`

```
%!PS-Adobe-3.0
%%Creator: Hinrich E.G. Bonin
%%Title: Halber Kreis
%%CreationDate: 09-Oct-2005
%%EndComments
%%BeginProlog
/cm { 28.35 mul } def
/radius 4 cm def
%%EndProlog
5.0 cm 1.0 cm translate
0 cm  0 cm radius 0 180 arc
closepath
gsave
0.8 setgray
fill
grestore
0.6 setgray
1.4 cm setlinewidth
stroke
0 cm  0 cm radius 2 div 0 180 arc
closepath
0.1 setgray
fill
showpage
%%EOF
```

A.3 PostScript UMLClassSymbol interpretieren

Listing A.2: UMLClassSymbol

```
%!PS-Adobe-3.0
%%Creator: Emil Cody
%%Title: Nameless
%%CreationDate: 23-Jan-2006
%%EndComments
%%BeginProlog
/cm { 28.35 mul } def
%%EndProlog
1 cm 3 cm moveto
7 cm 3 cm lineto
1 cm 2 cm moveto
7 cm 2 cm lineto
1 cm 1 cm moveto
1 cm 4 cm lineto
7 cm 4 cm lineto
7 cm 1 cm lineto
closepath
stroke
/Times-Roman findfont
14 scalefont
setfont
1.1 cm 1.1 cm moveto
(getSlot() : String) show
1.1 cm 2.1 cm moveto
(slot : String="OK!") show
1.1 cm 3.1 cm moveto
(de.unilueneburg.as::Person) show
showpage
%%EOF
```

A.3.1 Graphik zeichnen

Der obige Quellcode in PostScript stellt eine Graphik dar. Zeichnen Sie diese Graphik (†15*P*).

A.3.2 Graphik klassifizieren

Kommt Ihnen diese Graphik bekannt vor? Wenn ja, dann geben Sie an um welche Notation und um welches Symbol es sich handelt (†15*P*).

A.4 PostScript `UMLInheritance` interpretieren

Listing A.3: `UMLInheritance`

```
%!PS-Adobe-3.0
%%Creator: Emil Cody
%%Title: Nameless
%%CreationDate: Feb-2008
%%EndComments
%%BeginProlog
/cm { 28.35 mul } def
%%EndProlog
2 cm 9 cm moveto
8 cm 9 cm lineto
8 cm 6 cm lineto
2 cm 6 cm lineto
2 cm 9 cm lineto
2 cm 8 cm moveto
8 cm 8 cm lineto
8 cm 7 cm moveto
2 cm 7 cm lineto
5 cm 6 cm moveto
4.5 cm 5.5 cm lineto
5.5 cm 5.5 cm lineto
5 cm 6 cm lineto
5 cm 5.5 cm moveto
5 cm 4 cm lineto
2 cm 3 cm moveto
8 cm 3 cm lineto
8 cm 2 cm moveto
2 cm 2 cm lineto
2 cm 1 cm moveto
2 cm 4 cm lineto
8 cm 4 cm lineto
8 cm 1 cm lineto
closepath
stroke
/Times-Roman findfont
14 scalefont
setfont
2.2 cm 3.2 cm moveto
(de.leuphana.as::Enduro) show
2.2 cm 8.2 cm moveto
(de.leuphana.as::Motorrad) show
2.2 cm 7.2 cm moveto
(hersteller : String) show
2.2 cm 6.2 cm moveto
(getHersteller() : String) show
showpage
%%EOF
```

A.4.1 Graphik zeichnen

Der obige Quellcode in PostScript stellt eine Graphik dar. Zeichnen Sie diese Graphik (†15*P*).[Hinweis: Achten Sie dabei auf Groß/Kleinschreibung und Leerräume.]

A.4.2 Graphik interpretieren

Die Graphik zeigt ein UML-Diagramm. Stellen Sie fest, welche der folgenden Aussagen für diese Graphik zutreffen und begründen Sie kurz ihre Antwort (†15*P*):

Aussage 1: Es handelt sich um ein Klassendiagramm.

Aussage 2: Es ist eine Vererbungsbeziehung zwischen `Motorrad` und `Enduro` abgebildet.

Aussage 3: `Motorrad` ist kein Objekt sondern eine Klasse.

Aussage 4: `Enduro` ist Unterklasse von `Motorrad`.

Aussage 5: Es handelt sich um keine Assoziationsbeziehung.

A.5 Java3D-Klasse `MyCylinder` erläutern

Listing A.4: `MyCylinder`

```
/**
 *  "Java 3D Example"
 *
 *@author       Emil Cody
 *@version      1.0
 */
package de.unilueneburg.as.figure3D;

import java.applet.Applet;
import java.awt.BorderLayout;
import java.awt.GraphicsConfiguration;
import com.sun.j3d.utils.applet.MainFrame;
import com.sun.j3d.utils.geometry.Primitive;
import com.sun.j3d.utils.geometry.Cylinder;
import com.sun.j3d.utils.universe.SimpleUniverse;
import javax.media.j3d.Appearance;
import javax.media.j3d.BranchGroup;
import javax.media.j3d.Canvas3D;
import javax.media.j3d.TransformGroup;
import javax.media.j3d.PolygonAttributes;


public class MyCylinder extends Applet
{
    public SimpleUniverse u = null;

    public BranchGroup createSceneGraph()
    {
        BranchGroup bg = new BranchGroup();

        TransformGroup tg = new TransformGroup();

        PolygonAttributes polyAtt =
            new PolygonAttributes();
        polyAtt.setPolygonMode(
            PolygonAttributes.POLYGON_LINE);
        polyAtt.setCullFace(
            PolygonAttributes.CULL_NONE);

        Appearance app = new Appearance();
        app.setPolygonAttributes(polyAtt);

        tg.addChild(new Cylinder(0.3f, 0.9f,
                    Primitive.GENERATE_NORMALS,
                    8,
                    3,
                    app));

        bg.addChild(tg);
        bg.compile();
        return bg;
    }
```

```
54      public void init()
        {
56          this.setLayout(new BorderLayout());
            GraphicsConfiguration config =
58              SimpleUniverse.getPreferredConfiguration();
            Canvas3D c = new Canvas3D(config);
60          this.add("Center", c);
            u = new SimpleUniverse(c);
62          u.getViewingPlatform().
                setNominalViewingTransform();
64          u.addBranchGraph(
                            this.createSceneGraph());
66      }

68      public void destroy()
        {
70          u.cleanup();
        }
72
        public static void main(String[] args)
74      {
            new MainFrame(
76                      new MyCylinder(),
                        400, 400);
78      }
    }
```

Log-Datei `MyCylinder.log`

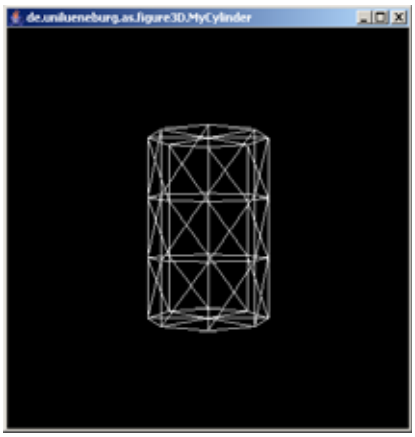
```
1  >java -version
2  java version "1.5.0_04"
3  Java(TM) 2 Runtime Environment, Standard Edition
4    (build 1.5.0_04-b05)
5  Java HotSpot(TM) Client VM
6    (build 1.5.0_04-b05, mixed mode)
7  >javac de/unilueneburg/as/figure3D/MyCylinder.java
8  >java de.unilueneburg.as.figure3D.MyCylinder
9
```

A.5.1 Konstruktor erläutern

In der Quellcodedatei `MyCylinder.java` (↪ Seite 154) wird der Konstruktor `MyCylinder()` appliziert. Ist dieser Konstruktor in der Quellcodedatei `MyCylinder.java` explizit deklariert? Wenn ja, geben Sie die Zeilennummer an. Wenn nein, dann erläutern Sie, warum das Programm trotzdem fehlerfrei (↪ Protokolldatei Seite 155) die Graphik (↪ Abbildung A.2 Seite 156) erzeugt (†10*P*).

A.5.2 Parameteränderung erläutern

Erläutern Sie die Änderung in der Graphik, wenn in Zeile 46 von `MyCylinder.java` (↪ Seite 154) der Wert von 3 auf 5 gesetzt wird (†10*P*).



Legende:
Java3D-Quellcode `MyCylinder.java` ↪ Seite 154
Protokolldatei `MyCylinder.log` ↪ Seite 155

Abbildung A.2: Java3D-Graphik: `MyCylinder`

Hinweis: Die Compilation und die Applikation sind mit diesem Wert erfolgreich.

A.6 Java3D-Klasse Leuphana erläutern

Listing A.5: Leuphana

```
/**
 *  "Java 3D Example" Leuphana
 *
 *@author       Emil Cody
 *@version      1.0
 */

package de.leuphana.iwi.figure3D;

import java.applet.Applet;
import java.awt.BorderLayout;
import java.awt.GraphicsConfiguration;
import java.awt.Font;

import com.sun.j3d.utils.applet.MainFrame;
import com.sun.j3d.utils.universe.SimpleUniverse;

import javax.media.j3d.Appearance;
import javax.media.j3d.BranchGroup;
import javax.media.j3d.Canvas3D;
import javax.media.j3d.Font3D;
import javax.media.j3d.FontExtrusion;
import javax.media.j3d.Material;
import javax.media.j3d.Shape3D;
import javax.media.j3d.Text3D;
import javax.media.j3d.TransformGroup;
import javax.media.j3d.Transform3D;

import javax.vecmath.AxisAngle4f;
import javax.vecmath.Color3f;
import javax.vecmath.Point3f;
import java.applet.Applet;

public class Leuphana extends Applet
{
    private SimpleUniverse u;

    private Leuphana(String message)
    {
        super();
        System.out.println(message);
    }

    public BranchGroup createSceneGraph()
    {
        BranchGroup bg = new BranchGroup();
        TransformGroup tg =
            new TransformGroup();

        Transform3D t3d = new Transform3D();
        t3d.setRotation(
              new AxisAngle4f(
```

```
            1f,
54          1f,
            0f,
56          (float) Math.toRadians(330)));
        t3d.setScale(0.45);
58      tg.setTransform(t3d);

60      Font3D font3D = new Font3D(
            new Font(
62            "Times", Font.PLAIN, 1
              ),
64          new FontExtrusion());
        Text3D textGeom = new Text3D(
66          font3D,
            new String("Leuphana"),
68          new Point3f(-0.6f, 0.0f, -0.9f),
            Text3D.ALIGN_CENTER,
70          Text3D.PATH_RIGHT);

72      Appearance app = new Appearance();
        Material m = new Material();
74      m.setEmissiveColor(
            new Color3f(0.8f, 0.8f, 0.8f));
76      app.setMaterial(m);
        tg.addChild(
78        new Shape3D(textGeom, app));

80      bg.addChild(tg);
        return bg;
82  }

84  public void init()
    {
86      setLayout(new BorderLayout());
        Canvas3D c = new Canvas3D(
88        SimpleUniverse.getPreferredConfiguration()
          );
90      add("Center", c);
        u = new SimpleUniverse(c);
92      u.getViewingPlatform().
            setNominalViewingTransform();
94      u.addBranchGraph(createSceneGraph());
    }
96
    public void destroy()
98  {
        u.cleanup();
100 }

102 public static void main(String[] args)
    {
104     new MainFrame(
            new Leuphana("new Leuphana(..)"),
106         800,
            200);
108 }
```

Legende:
Java3D-Quellcode `Leuphana.java` ↪ Seite 157
Protokolldatei `Leuphana.log` ↪ Seite 159

Abbildung A.3: Java3D-Graphik: `Leuphana`

}

Log-Datei `Leuphana.log`

```
 1  D:\bonin\prog\code>java -fullversion
 2  java full version "1.6.0_13-b03"
 3
 4  D:\bonin\prog\code>echo %CLASSPATH%
 5  .;C:\Programme\Java\jre6\lib\ext\QTJava.zip;
 6  c:\programme\aspectj1.6\lib\aspectjrt.jar;
 7  C:\Programme\Java\jdk1.6.0_13\lib\j3dutils.jar;
 8  C:\Programme\Java\jdk1.6.0_13\lib\j3dcore.jar;
 9  C:\Programme\Java\jdk1.6.0_13\lib\vecmath.jar
10
11  D:\bonin\prog\code>javac
12    de/leuphana/iwi/figure3D/Leuphana.java
13
14  D:\bonin\prog\code>java
15    de.leuphana.iwi.figure3D.Leuphana
16  new Leuphana(..)
17
18  D:\bonin\prog\code>
19
```

A.6.1 Code analysieren

Die Quellcodedatei `Leuphana.java` (↪ Seite 157) läuft fehlerfrei, wie die Protokolldatei (↪ Seite 159) und die Graphik (↪ Abbildung A.3 S. 159) zeigen. Hat der Programmierer *Emil Cody* einige Code-Zeilen zu viel angegeben? Nach kurzer Überlegung will er die beiden folgenden Zeilen streichen (†10*P*):

Fall 1: Zeile 32

Fall 2: Zeile 40

Eräutern Sie, ob oder warum nicht er den jeweiligen Fall streichen kann.

A.6.2 *Debug*-Ausgabe vermeiden

Emil Cody hat für Testzwecke in Zeile 41 eine Ausgabe codiert. Er möchte jetzt keine solche *Debug*-Ausgabe mehr erzeugen. Welche Änderungen sollte er in der Quellcodedatei `Leuphana.java` (↪ Seite 157) vornehmen. Geben Sie bei Ihrem Vorschlag stets die betroffenen Zeilennummern an (†10*P*).

A.7 Objekt & Referenz

Listing A.6: `Customer`

```
/**
 *  "Example Customer"
 *
 *@author       Emil Cody
 *@version      1.0
 */

public class Customer
{
    int id;
    String name;

    public String getName()
    {
        return name;
    }

    public Customer setName(String name)
    {
        this.name = name;
        return this;
    }

    public Customer(int id, String name)
    {
        this.id = id;
        this.setName(name);
    }

    public static void main(String[] args)
    {
        Customer c1 = new Customer(1, "Mustermann");
        Customer c2 = c1.setName("Musterfrau");

        System.out.println("c1: " + c1.getName());
        System.out.println("c2: " + c2.getName());

    }
}
```

Log-Datei `Customer.log`

```
 1  >java -version
 2  java version "1.5.0_04"
 3  Java(TM) 2 Runtime Environment, Standard Edition
 4    (build 1.5.0_04-b05)
 5  Java HotSpot(TM) Client VM
 6    (build 1.5.0_04-b05, mixed mode)
 7  >javac Customer.java
 8  >java Customer
 9  c1: Musterfrau
10  c2: Musterfrau
11  >
12
```

A.7.1 Kopieproblem erläutern

Der geniale (?) Programmierer *Emil Cody* hat in der `main`-Methode seiner Java-Applikation `Customer.java` (↪ Seite 160) erfolgreich ein Objekt der Klasse `Customer` erzeugt. Er ist der Ansicht, dass er von diesem Objekt `c1` mittels Gleichheitszeichen die Kopie `c2` erzeugen kann. Er wundert sich allerdings über das für ihn überraschende Ergebnis (↪ Protokolldatei Seite 161).

Erläutern Sie, warum in Zeile 33 keine Objektkopie entsteht. Geben Sie eine Korrektur für diese Zeile an, so dass ein zweites Objekt mit der Referenz `c2` entsteht (†10*P*).

A.7.2 Einhaltung von Notationsregeln prüfen

Üblicherweise haben sogenannte „*Setter*“ (also `set`-Methoden) eine Signatur ohne Rückgabewert, also mit der Angabe `void`. Prüfen Sie, ob in dieser Hinsicht die Java-Applikation `Customer.java` (↪ Seite 160) ordnungsgemäß codiert ist. Wenn nicht, formulieren Sie Ihre Korrektur unter Angabe der betroffenen Zeilennummern (†10*P*).

A.8 Interface & Inheritance

Listing A.7: `Baz`

```
/**
 *  "Example Inheritance"
 *
 *@author       Emil Cody
 *@version      1.0
 */

public interface Baz
{
    public String getSlot();
```

```
    public void setSlot(String slot);
}
```

Listing A.8: `Foo`

```
/**
 *  "Example Inheritance"
 *
 *@author       Emil Cody
 *@version      1.0
 */

public class Foo
{
    protected String slot ="";
    public static String global ="ENGLAND";

}
```

Listing A.9: `Bar`

```
/**
 *  "Example Inheritance"
 *
 *@author       Emil Cody
 *@version      1.0
 */

public class Bar extends Foo implements Baz
{
    private String id;

    public String getId()
    {
        return id;
    }

    public void setId(String id)
    {
        this.id = id;
    }

    public String getSlot()
    {
        return slot;
    }

    public void setSlot(String slot)
    {
        this.slot = slot;
    }

    public static void main(String[] args)
    {
        Bar b = new Bar();
        b.setId("007");
        b.setSlot(global);
```

```
        System.out.println("id:   " + b.getId());
38      System.out.println("slot: " + b.getSlot());

40    }
}
```

Log-Datei `Bar.log`

```
 1  >java -version
 2  java version "1.5.0_04"
 3  Java(TM) 2 Runtime Environment, Standard Edition
 4    (build 1.5.0_04-b05)
 5  Java HotSpot(TM) Client VM
 6    (build 1.5.0_04-b05, mixed mode)
 7  >javac Baz.java
 8  >javac Foo.java
 9  >javac Bar.java
10  >java Bar
11  id:   007
12  slot: ENGLAND
13
```

A.8.1 Interface implementieren

Die Java-Quellcodedatei `Baz.java` (↪ Seite 161) beschreibt ein Interface. In der Java-Quellcodedatei `Bar.java` (↪ Seite 162) wird dieses implementiert. Erläutern Sie welche Teile in `Bar.java` konkret die Vorgaben des Interfaces implementieren. Geben Sie dabei die betroffenen Zeilennummern an (†10*P*).

A.8.2 Zugriff auf Klassenvariable

In der Klasse `Foo` ist eine Klassenvariable angegeben. Nennen Sie diese und erläutern Sie, warum ohne Angabe der Klasse `Foo` innerhalb der Klasse `Bar` darauf zugegriffen werden kann (†10*P*).

A.9 Lokale Klasse

Listing A.10: `Think`

```
/**
2  *  "Example Think"
   *
4  *@author      Emil Cody
   *@version     1.0
6  */

8 public class Think
  {
```

```
10      private static final String slot = "outside";

12      String getSlot()
        {
14          return slot;
        }
16
        String think()
18      {
           class ThinkInside
20         {
               String slot = "inside";
22
               String getSlot()
24             {
                   return slot;
26             }
           }
28
           return (new ThinkInside()).getSlot();
30      }

32      public static void main(String[] args)
        {
34
            System.out.println(slot +
36                             " -> " +
                               (new Think()).think() +
38                             " -> " +
                               (new Think()).getSlot());
40      }
}
```

A.9.1 Lokale Klasse erläutern

Die Java-Quellcodedatei `Think.java` (↪ Seite 163) enthält eine lokale Klasse bezogen auf eine Methode. Nennen Sie die Methode und geben sie die betroffenen Zeilennummern an (†5*P*).

A.9.2 Ergebnis der Java-Applikation `Think`

Der Java-Quellcode `Think.java` wurde erfolgreich compiliert. Geben Sie exakt die Ausgabe auf der Console an, wenn die Klasse `Think` ausgeführt wurde, also folgendes Kommando abgearbeitet wurde:

```
>java Think
```

(†5*P*).

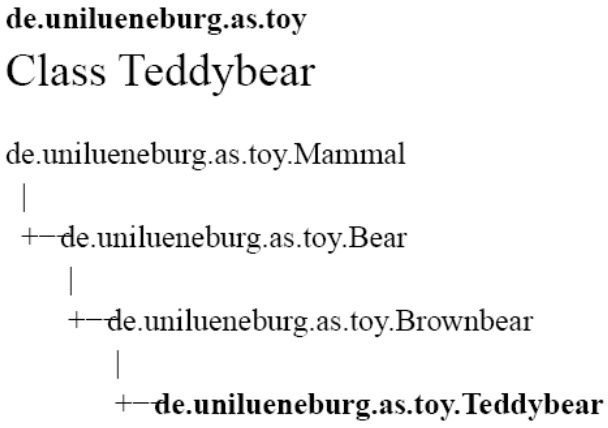


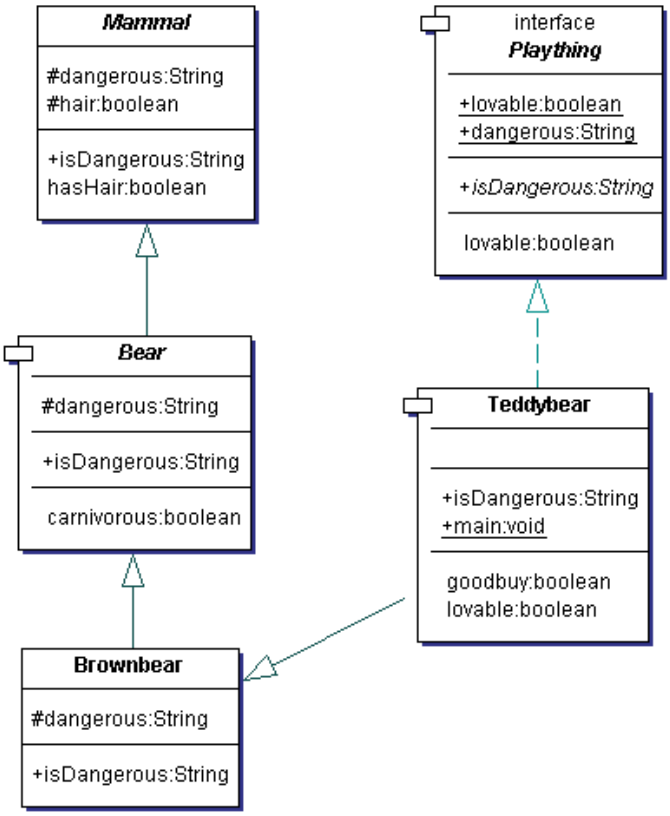
Abbildung A.4: Überblick zum Beispiel `Teddybear`

A.10 Konstanten aus Interface

Das Beispiel `Teddybear` verdeutlicht das Zusammenspiel von Vererbungsbeziehungen und der Vorgabe durch ein Inteface. Die Abbildung A.4 S. 165 gibt einen Überblick über die Vererbungsbeziehungen. Die Abbildung A.5 S. 166 zeigt das detailierte *Class Diagram* in UML-Notation generiert mit einem Werkzeug.

Listing A.11: `Teddybear`

```
/**
 *  Example Teddybear
 *
 *@author       Emil Cody
 *@version      1.0
 */

package de.unilueneburg.as.toy;

public class Teddybear extends Brownbear
    implements Plaything
{
    private final boolean goodbuy = true;

    public boolean isGoodbuy()
    {
        return goodbuy;
    }

    public boolean isLovable()
    {
        return lovable;
```



Legende:
Notation in *Unified Modeling Language* (UML) *Class Diagram*.
Hinweis: Gezeichnet mit *Borland Together Control Center*^{TM} 6.2.

Abbildung A.5: Klassendiagramm für `Teddybear`

```
    }

    public String isDangerous()
    {
        /* return dangerous;
         * reference to dangerous is ambiguous,
         * both variable dangerous in Brownbear
         * and Plaything match
         */

        return Plaything.dangerous;
    }

    public static void main(String[] args)
    {
        Teddybear teddy = new Teddybear();
        Brownbear bBear = new Brownbear();

        System.out.println
            (
             "My Teddy:" +

             "\n (1) has hair? " +
             teddy.hasHair() +

             "\n (2) is carnivorous? " +
             teddy.isCarnivorous() +

             "\n (3) is a good buy? " +
             teddy.isGoodbuy() +

             "\n (4) is lovable? " +
             teddy.isLovable() +

             "\n (5) is dangerous? " +
             teddy.isDangerous() +

             "\n (6) is dangerous? " +
             bBear.isDangerous() +

             "\n (7) is dangerous? " +
             ((Brownbear) teddy).isDangerous() +

             "\n (8) is dangerous? " +
             ((Brownbear) teddy).dangerous +

             "\n (9) is dangerous? " +
             ((Bear) teddy).isDangerous() +

             "\n(10) is dangerous? " +
             ((Bear) teddy).dangerous +

             "\n(11) is dangerous? " +
             ((Mammal) teddy).isDangerous() +

             "\n(12) is dangerous? " +
```

```
                ((Mammal) teddy).dangerous +
80
                "\n(13) is dangerous? " +
82              ((Plaything) teddy).isDangerous() +
                "\n(14) is dangerous? " +
84              ((Plaything) teddy).dangerous
                );
86      }
88  }
```

Listing A.12: `Brownbear`

```
    /**
2    *  Example  Teddybear
     *
4    *@author      Emil  Cody
     *@version     1.0
6    */

8   package de.unilueneburg.as.toy;

10  public class Brownbear extends Bear
    {
12      protected final String dangerous = "true";

14      public String isDangerous()
        {
16          return this.dangerous;
        }
18  }
```

Listing A.13: `Bear`

```
    /**
2    *  Example  Teddybear
     *
4    *@author      Emil  Cody
     *@version     1.0
6    */

8   package de.unilueneburg.as.toy;

10  public abstract class Bear extends Mammal
    {
12      protected final boolean carnivorous = true;
        protected final String dangerous = "unknown";
14
        public String isDangerous()
16      {
            return this.dangerous;
18      }

20      boolean isCarnivorous()
        {
22          return carnivorous;
        }
```

```
24 }
```

Listing A.14: `Mammal.java`

```
/**
 *  Example Teddybear
 *
 *@author       Emil Cody
 *@version      1.0
 */

package de.uniluebeburg.as.toy;

public abstract class Mammal
{
    protected final String dangerous = "dubious";
    protected final boolean hair = true;

    public String isDangerous()
    {
        return this.dangerous;
    }

    boolean hasHair()
    {
        return hair;
    }
}
```

Listing A.15: `Plaything`

```
/**
 *  Example Teddybear
 *
 *@author       Bonin
 *@version      1.0 26-Nov-2006
 */

package de.uniluebeburg.as.toy;

public interface Plaything
{
    public final boolean lovable = true;
    public final String dangerous = "false";

    public boolean isLovable();
    public String isDangerous();
}
```

A.10.1 Konstanten aus dem Interface

Die Java-Quellcodedatei `Plaything.java` (↪ Seite 169) enthält die Deklaration von Konstanten. Erläutern Sie wie auf diese Konstanten in der Klasse `Teddybear` (↪ Seite 165) zugegriffen werden kann. (†10*P*).

A.10.2 Ergebnis der Java-Applikation `Teddybear`

Der Java-Quellcode `Teddybear.java` (↪ Seite 165) wurde erfolgreich compiliert. Geben Sie exakt die Ausgabe auf der Console an, wenn die Klasse `Teddybear` ausgeführt wurde, also folgendes Kommando abgearbeitet wurde:

```
>java de.unilueneburg.as.toy.Teddybear
```

(†15*P*).

A.11 Seiteneffekt

Listing A.16: `Customer`

```
/**
 *  "Example Customer"
 *
 *@author       Emil Cody
 *@version      1.0
 */
package de.unilueneburg.as.sideeffect;

public class Customer
{
    int id = 0;
    Address address;

    public int getId()
    {
        return id;
    }

    public void setId(int id)
    {
        this.id = id;
    }

    public Address getAddress()
    {
        return address;
    }

    public void setAddress(Address address)
    {
        this.address = address;
        address.setText("Side effect");
    }

    public Customer(int id)
    {
        this.id = id;
```

```
38          id = 13;
          }
40
        public static void main(String[] args)
42      {
            Address a = new Address(
44            "Volgershall 1, D-21339 Lneburg"
              );
46          Customer c = new Customer(1);
            c.setAddress(a);
48
            System.out.println(
50            "c.getAddress().getText() = " +
               c.getAddress().getText()
52            );
            System.out.println(
54            "a.getText() =  " +
               a.getText()
56            );
            System.out.println(
58            "c.getId() =  " + c.getId()
              );
60      }
    }
```

Listing A.17: `Address`

```
    /**
2    *  "Example Customer"
     *
4    *@author       Emil Cody
     *@version      1.0
6    */

8   package de.uniluneburg.as.sideeffect;

10  public class Address
    {
12      private String text = "";

14      public String getText()
        {
16          return text;
        }
18
        public void setText(String text)
20      {
         this.text = text;
22      }

24      public Address(String text)
        {
26          this.text = text;
        }
28  }
```

```
D:\bonin\prog\code>java -version
```

```
java version "1.5.0_08"
Java(TM) 2 Runtime Environment,
  Standard Edition (build 1.5.0_08-b03)
Java HotSpot(TM) Client VM
  (build 1.5.0_08-b03, mixed mode)

D:\bonin\prog\code>javac
  de/unilueneburg/as/sideeffect/Customer.java

D:\bonin\prog\code>java
  de.unilueneburg.as.sideeffect.Customer
c.getAddress().getText() = Side effect
a.getText() =  Side effect
c.getId() =  1

D:\bonin\prog\code>
```

A.11.1 Erläuterung des Ergebnisses

Der geniale (?) Programmierer *Emil Cody* ist über das Ergebniss seiner Java-Applikation `Customer.java` (↪ Seite 170) völlig überrascht. Erläutern Sie, warum der Text:
`Volgershall 1, D-21339 Lüneburg`
keinmal erscheint. Kleiner Tipp: Trotz ähnlicher Konstruktion ist der Wert von id gleich 1 (†10*P*).

A.11.2 Programmänderung

Ändern Sie den Quellcode von `Customer.java` (↪ Seite 170) durch Streichung einer Zeile, so dass der Text:
`Volgershall 1, D-21339 Lüneburg`
ausgegeben wird. Geben Sie die Nummer der gestrichenen Zeile an und erläutern Sie das Ergebnis (†10*P*).

A.12 Rekursion

Listing A.18: `Person`

```
/**
 *  Beispiel "Rekursive Klasse"
 *
 *@since        14-July-2009
 *@author       Emil Cody
 *@version      1.0
 */
package de.leuphana.ics.duplicate;

import java.util.ArrayList;

class Person
```

```
{
    private String name;
    private Person ehegatte;
    private ArrayList<Person>
        kinder = new ArrayList<Person>();

    public Person(String name)
    {
        this.name = name;
    }

    public String getName()
    {
        return name;
    }

    public Person getEhegatte()
    {
        return ehegatte;
    }

    public Person getKind(int i)
    {
        return kinder.get(i - 1);
    }

    public Person setKind(Person kind)
    {
        kinder.add(kinder.size(), kind);
        System.out.println("Anzahl der Kinder: " + kinder.size());
        return this;
    }

    public Person heiratet(Person person)
    {
        this.ehegatte = person;
        person.ehegatte = this;
        return this;
    }

}
```

Listing A.19: `Application`

```
/**
 *  Beispiel "Rekursive Klasse"
 *
 *@since        14-July-2009
 *@author       Emil Cody
 *@version      1.0
 */
package de.leuphana.ics.duplicate;

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

```
14          Person frau = new Person("Ilse Meyer-Schulze");
            Person mann = new Person("Otto Meyer");
16
            Person kind1 = new Person("Klara Meyer");
18          Person kind2 = new Person("Ernst Meyer");
            Person kind3 = kind1;
20
            frau.heiratet(mann).setKind(kind1);
22          mann.setKind(kind2).setKind(kind3);

24          System.out.println(
             frau.getEhegatte().getKind(2).getName());
26      }
    }
```

A.12.1 Korrektheit feststellen

Die Java-Quellcodedateien `Application.java` (↪ S. 173) wird compiliert. Prüfen Sie, ob das Compilieren ohne Fehlermeldungen erfolgt. Wenn nicht, dann skizzieren Sie die von Ihnen angenommenen Fehler (†5*P*).

A.12.2 Ergebnis notieren

Geben Sie exakt das Ergebnis bei folgendem Statement an (†10*P*):

```
>java de.leuphana.ics.duplicate.Application
```

A.12.3 Ergebnis mit Modifikation notieren

Der Programmierer *Emil Cody* ändert in der Java-Application `Application.java` (↪ S. 173) die Zeile 24 wie folgt:

```
System.out.println(
  mann.getEhegatte().getKind(2).getName());
```

und compilert die so geänderte Datei fehlerfrei. Geben Sie nun erneut exakt das Ergebnis bei folgendem Statement an (†10*P*):

```
>java de.leuphana.ics.duplicate.Application
```

A.13 Abstrakte Klasse mit Konstruktor

Listing A.20: `Motorrad`

```
/**
2  *  "Example Abstract Class"
   *
4  *@author      Emil Cody
   *@version     1.0
6  */
package de.leuphana.ics.constructor;
```

```
8
   abstract class Motorrad
10 {
       private double power;
12     private double weight;

14     public int getPower()
       {
16         return (int) power;
       }
18
       public int getWeight()
20     {
           return (int) weight;
22     }

24     public Motorrad(double power, double weight)
       {
26         this.power = power;
           this.weight = weight;
28         System.out.println("Motorrad erzeugt!");
       }
30 }
```

Listing A.21: `Tourer`

```
   /**
2   *  "Example Abstract Class"
    *
4   *@author      Emil Cody
    *@version     1.0
6   */
   package de.leuphana.ics.constructor;
8
   class Tourer extends Motorrad
10 {
       private double cruisingRange;
12
       public Tourer (double power,
14                    double weight,
                      double cruisingRange)
16     {
         super(power, weight);
18       this.cruisingRange = cruisingRange;
       }
20
       public static void main(String[] args)
22     {
           System.out.println(
24           (new Tourer(98.99, 241.99, 360)).getPower());
       }
26 }
```

Listing A.22: `Sportler`

```
   /**
2   *  "Example Abstract Class"
```

```
 *
 *@author      Emil Cody
 *@version     1.0
 */
package de.leuphana.ics.constructor;

class Sportler extends Motorrad
{
    private double cubicCapacity;

    public Sportler (double power,
                     double weight,
                     double cubicCapacity)
    {
      super(power, weight);
      this.cubicCapacity = cubicCapacity;
    }

    public static void main(String[] args)
    {
        System.out.println(
          (new Sportler(167.99, 199.99, 650)).getWeight());
    }
}
```

Listing A.23: Sonstige

```
/**
 *  "Example Abstract Class"
 *
 *@author      Emil Cody
 *@version     1.0
 */
package de.leuphana.ics.constructor;

class Sonstige extends Motorrad
{
    private double cost;

    public static void main(String[] args)
    {
        System.out.println(
          (new Motorrad(98.0, 210.0)).getWeight());
    }
}
```

A.13.1 Ergebnis notieren

Geben Sie das Ergebnis bei den folgenden Statements an (†10*P*):

```
>java de.leuphana.ics.constructor.Tourer
>java de.leuphana.ics.constructor.Sportler
>javac de/leuphana/ics/constructor/Sonstige.java
```

A.13.2 Abstrakte Klasse erläutern

Skizzieren Sie die Aufgabe einer Java-Klasse mit der Kennzeichnung `abstract` (†10*P*).

Anhang B

Lösungen zu den Übungen

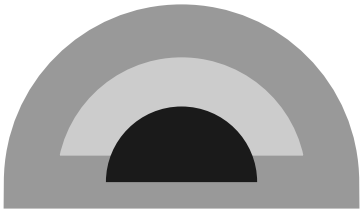
Lösung Aufgabe A.1 S. 149:

Listing B.1: `UMLKlassensymbol`

```
%!PS-Adobe-3.0
%%Creator: Hinrich E.G. Bonin
%%Title: UML Class Symbol
%%CreationDate: 08-Oct-2005
%%EndComments
%%BeginProlog
/cm { 28.35 mul } def
%%EndProlog
1 cm 3 cm moveto
7 cm 3 cm lineto
1 cm 2 cm moveto
7 cm 2 cm lineto
1 cm 1 cm moveto
1 cm 4 cm lineto
7 cm 4 cm lineto
7 cm 1 cm lineto
closepath
stroke
/Times-Roman findfont
16 scalefont
setfont
1.1 cm 3.1 cm moveto
(de.uni-lueneburg.as::FOO) show
showpage
%%EOF
```

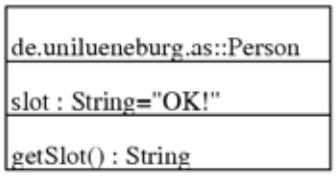
Lösung Aufgabe A.2 S. 150:

Der PostScript-Quellcode stellt zwei sich überdeckende Halbkreise dar, wobei die Kontur des äußeren Kreises mit ein Strichstärke von $1,4cm$ erzeugt wird (↪ Abbildung B.1 S. 180).



Legende:
PostSript-Quellcode ↪ S. 150

Abbildung B.1: Zwei sich überdeckende Halbkreise



Legende:
PostSript-Quellcode ↪ S. 151

Abbildung B.2: UML Klassensymbol

Lösung Aufgabe A.3 S. 151:

Lösung Aufgabe A.3.1 S. 151:
Das Ergebnis des PostSript-Quellcodes (↪ S. 151) zeigt Abbildung B.2 S. 180.

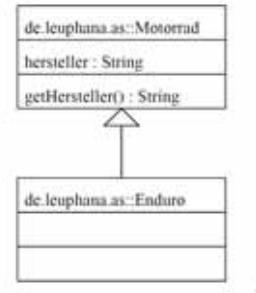
Lösung Aufgabe A.3.2 S. 151:
Die Abbildung B.2 S. 180 zeigt ein Klassensymbol in der Notation der *Unified Modeling Language* (UML).

Lösung Aufgabe A.4 S. 152:

Lösung Aufgabe A.4.1 S. 152:
Das Ergebnis des PostSript-Quellcodes (↪ S. 152) zeigt Abbildung B.3 S. 181.

Lösung Aufgabe A.4.2 S. 153:
Alle Aussagen treffen zu.

Lösung Aufgabe A.5 S. 154:



Legende:
PostSript-Quellcode ↪ S. 151

Abbildung B.3: UML Vererbung

Lösung Aufgabe A.5.1 S. 155:
In der Quellcodedatei `MyCylinder.java` (↪ Seite 154) ist kein Konstruktor `MyCylinder()` deklariert.

Das Programm erzeugt fehlerfrei (↪ Protokolldatei Seite 155) die Graphik (↪ Abbildung A.2 Seite 156) weil der Standardkonstruktor `MyCylinder()` in Java implizit deklariert ist.

Hinweis: Der implizit deklarierte Standardkonstruktur lässt sich explizit überschreiben:

```
public MyCylinder()
{
}
```

Lösung Aufgabe A.5.2 S. 155:
Der Zylinder wird in der vertikalen Einteilung in 5 Abschnitte geteilt, wenn in Zeile 46 von `MyCylinder.java` (↪ Seite 154) der Wert von 3 auf 5 gesetzt wird. Die Größe des Zylinders ändert sich nicht.

Lösung Aufgabe A.6 S. 157:

Lösung Aufgabe A.6.1 S. 159:

Fall 1: Eine Klasse „Applet“ wird schon im Programm importiert, daher kann das zweite `import ...` dieser Klasse entfallen.

Fall 2: Der Standardkonstruktor wird auch implizit aufgerufen, wenn er nicht explizit angegeben ist.

Lösung Aufgabe A.6.2 S. 160:

Zeile 38: Ändern in `private Leuphana()` — also ohne Parameter.

Zeile 41: Zeile Löschen.

Zeile 105: Ändern in `new Leuphana()` — also ohne Parameter.

Lösung Aufgabe A.7 S. 160:

Lösung Aufgabe A.7.1 S. 161:
In der Zeile 33 „`c2 = c1.setName(...)`“ referenziert das Objekt `c2` das veränderte Objekt `c1`, da die Methode `setName(...)` als Rückgabewert dasjenige Objekt hat, auf das sie angewendet wurde, also hier `c1`.

Ein zweites Objekt mit der Referenz `c2` entsteht durch Applikation des Konstruktors der Klasse `Customer`, also durch die neue Zeile 33:

```
Customer c2 = new Customer(2, "Musterfrau");
```

Lösung Aufgabe A.7.2 S. 161:
Die Methode `setName(...)` von Zeile 18 bis Zeile 22 entspricht nicht dem Java-Standard einer *Setter*-Methode. Folgende Korrektur ist erforderlich:

```
public void setName(String name)
{
    this.name = name;
}
```

Lösung Aufgabe A.8 S. 161:

Lösung Aufgabe A.8.1 S. 163:
Die Java-Quellcodedatei `Baz.java` (↪ Seite 161) beschreibt ein Interface. In der Java-Quellcodedatei `Bar.java` (↪ Seite 162) wird dieses Interface mit den beiden Methoden `getSlot()` und `setSlot(...)` in den Zeilen 22 – 30 implementiert.

Lösung Aufgabe A.8.2 S. 163:
In der Klasse `Foo` ist die Klassenvariable `global` angegeben. Auf `global` kann ohne Angabe der Klasse `Foo` innerhalb der Klasse `Bar` zugegriffen werden, weil `Bar` eine Unterklasse von `Foo` ist.

Hinweis: Die Klassenvariable `global` muss für diesen Zugriff nicht den Modifier `public` aufweisen. Mit dem Modifier `protected` wäre dieser Zugriff ebenfalls möglich, nicht jedoch mit `private`.

Lösung Aufgabe A.9 S. 163:

Lösung Aufgabe A.9.1 S. 164:
Die Java-Quellcodedatei `Think.java` (↪ Seite 163) enthält die lokale Klasse `ThinkInside` bezogen auf die Methode `think()`, notiert in den Zeilen 17–30.

Lösung Aufgabe A.9.2 S. 164:
Die Ausgabe auf der Console ist:

```
outside > inside > outside
```

Lösung Aufgabe A.10 S. 165:

Lösung Aufgabe A.10.1 S. 169:
In Java-Quellcodedatei `Teddybear.java` kann auf die Kontstanten des Interfaces `Plaything` direkt zugegriffen werden, wenn der Bezeichner der Konstanten in `Teddybear` eindeutig ist, also nicht durch Vererbung schon vorliegt. Daher gibt es einen Unterschied bei der Adressierung von `lovable` und `dangerous`. Letztere ist schon in der Klasse `Brownbear` deklariert und wird in die Klasse `Teddybear` vererbt. Es muss daher das Interface genannt werden, also `Plaything.dangerous`.

Lösung Aufgabe A.10.2 S. 170:
Die Ausgabe auf der Console zeigt das folgende Protokoll:

```
D:\bonin\prog\code>java -version
java version "1.5.0_08"
Java(TM) 2 Runtime Environment,
  Standard Edition (build 1.5.0_08-b03)
Java HotSpot(TM) Client VM
  (build 1.5.0_08-b03, mixed mode)

D:\bonin\prog\code>javac
  de/unilueneburg/as/toy/Teddybear.java

D:\bonin\prog\code>java
  de.unilueneburg.as.toy.Teddybear
My Teddy:
 (1) has hair? true
 (2) is carnivorous? true
 (3) is a good buy? true
 (4) is lovable? true
 (5) is dangerous? false
 (6) is dangerous? true
 (7) is dangerous? false
 (8) is dangerous? true
 (9) is dangerous? false
(10) is dangerous? unknown
(11) is dangerous? false
```

```
(12) is dangerous? dubious
(13) is dangerous? false
(14) is dangerous? false

D:\bonin\prog\code>
```

Lösung Aufgabe A.11 S. 170:

Lösung Aufgabe A.11.1 S. 172:
Im Fall des Wertes für den Parameter `address` handelt es sich um eine Referenz. Im Fall des Wertes für den Parameter `id` handelt es sich um einen primitiven Typ. Es ist zu unterscheiden zwischen „*call by value*“ (bei *Primitives*) und „*call by reference*“. Im zweiten Fall wird keine Copy des Argumentwertes an den Parameter gebunden, sondern der Wert der Speicheradresse (Referenz). Mit der Zeile „`address.setText("Side effect");`“ wird der Speicher an dieser Stelle geändert. Später wird mit einer Nachfrage nach dieser Referenz der geänderte Wert zurück gegeben.

Lösung Aufgabe A.11.2 S. 172:
Im Quellcode `Customer.java` ist die Zeile 32 zu streichen. Das Ergebnis nach Kompilation und Applikation ist dann:

```
c.getAddress().getText() = Volgershall 1, D-21339 Lüneburg
a.getText() =  Volgershall 1, D-21339 Lüneburg
c.getId() =  1
```

Lösung Aufgabe A.12 S. 172:

Lösung Aufgabe A.12.1 S. 174:
Das Compilieren erfolgt ohne Fehler.

Lösung Aufgabe A.12.2 S. 174:

```
Anzahl der Kinder: 1
Anzahl der Kinder: 1
Anzahl der Kinder: 2
Klara Meyer
```

Lösung Aufgabe A.12.3 S. 174:

```
Anzahl der Kinder: 1
Anzahl der Kinder: 1
Anzahl der Kinder: 2
Exception in thread "main"
 java.lang.IndexOutOfBoundsException:
 Index: 1, Size: 1
```

Lösung Aufgabe A.13 S. 174:

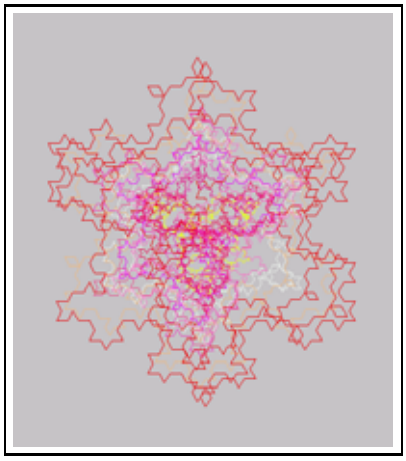
Lösung Aufgabe A.13.1 S. 176:

```
D:\bonin\prog\code>java
  de.leuphana.ics.constructor.Tourer
Motorrad erzeugt!
98
D:\bonin\prog\code>java
  de.leuphana.ics.constructor.Sportler
Motorrad erzeugt!
199
D:\bonin\prog\code>javac
  de/leuphana/ics/constructor/Sonstige.java
ERROR ...
de.leuphana.ics.constructor.Motorrad
is abstract; cannot be instantiated
```

Lösung Aufgabe A.13.2 S. 177:
Eine abstrakte Klasse dient zur Vererbung ihrer Eigenschaften ($\equiv$ Oberklassenfunktion). Aus ihr können keine Objekte (Instanzen) erzeugt werden.

Anhang C

Quellen



C.1 Literaturverzeichnis

Literaturverzeichnis

[Adobe99] Adobe Systems Incorporated; PostScript^{TM} — LANGUAGE REFERENCE,third edition, includes bibliographical references and index, ISBN 0-201-37922-8.

[Arnold/Gosling96] Ken Arnold / James Gosling; The Java Programming Language (Addison-Wesley) 1996.

[Alur/Crupi/Malks01] Deepak Alur / John Crupi / Dan Malks; Core J2EE Patterns, Sun Microsystems Press, Prentice Hall PTR, 2001, in deutsch von Frank Langenau; Core J2EE Patterns — Die besten Praxislösungen und Design-Strategien, 2002, ISBN 3-8272-6313-1.

[Bonin91b] Hinrich E. G. Bonin; Software-Konstruktion mit LISP, Berlin New York (Walter de Gruyter), 1991.

[Bonin92a] Hinrich E. G. Bonin; Arbeitstechniken für die Softwareentwicklung, (3. überarbeitete Auflage Februar 1994), FINAL, 2. Jahrgang Heft 2, 10. September 1992, [FINAL].

[Bonin04a] Hinrich E. G. Bonin; Aspect-oriented Softwaredevelopement — A Little Guidance to Better Java Applications —, aktuelle Fassung unter: `http://www.hegb.de`, begonnen 26-Jan-2002. {Hinweis: Beispiele primär in `AspectJ`.“}

[Bonin04b] Hinrich E. G. Bonin; Der Java^{TM}-Coach — Modellieren mit UML, Programmieren mit Java^{TM} 2 Plattform (J2SE & J2EE), Dokumentieren mit XHTML —, aktuelle Fassung unter: `http://www.hegb.de`, begonnen 5-Oct-1997. {Hinweis: Eine umfassende Einführung in die Objekt-Orientierung auf der Basis von Java^{TM}.}

[Deussen03] Oliver Deussen; Computergenerierte Pflanzen — Technik und Design digitaler Pflanzenwelten, Berlin u. a. (Springer Verlag), ISBN 3-540-43606-5. {Hinweis: Schöne Bilder mit fundierter Analyse von vielfältigen Pflanzenstrukturen.}

[Eckel02] Bruce Eckel; Thinking in Java — The Definitive Introduction to Object-Oriented Programming in the Language of the World-Wide-Web, Upper Saddle River, NJ 07458 (Prentice Hall PTR), 3rd Edition, ISBN 0-13-100287-2.

[Flanagan97] David Flanagan; Java in a Nutshell, Second Edition, updated for Java 1.1, Köln (O´Reilly), May 1997.

[FINAL] *Forum Informatics at Leuphana* (ursprünglich Fachhochschule Nordostniedersachsen, Informatik, Arbeitsberichte, Lüneburg) herausgegeben von Hinrich E. G. Bonin, ISSN 0939-8821, ↪ `http://www.leuphana.de/institute/iwi/final.html` (online 12-Feb-2010).

[Glover03] Dan Glover (Compiled by); Lila’s Child — An Inquiry into Quality; with introduction and annotations by Robert M. Pirsig, (1stBooks Library) 2003, ISBN 1-4033-5620-3. {Remark: Lila’s Child chronicles: Internet discussion group centered around Robert M. Pirsig’s novel.}

[Horstmann05a] Cay S. Horstmann / Gary Cornell; Core Java^{TM} 2, Volume I — Fundamentals, seventh edition, Sun Microsystems Press, a Prentice Hall Title, 2005, ISBN 0-13-148202-5. {Remark: *“A non-nonsens tutorial and reliable reference, this book features thoroughly tested real-world examples.”*}

[Horstmann05b] Cay S. Horstmann / Gary Cornell; Core Java^{TM} 2, Volume II — Advanced Features, seventh edition, Sun Microsystems Press, a Prentice Hall Title, 2005, ISBN 0-13-111826-9. {Remark: "*Revised and updated coverage of multithreading, collections, database programming, distributed computing and XML.*"}

[JavaSpec] James Gosling / Bill Joy / Guy Steele; The Java Language Specification, (Addison-Wesley) 1996;
`http://www.javasoft.com/docs/books/jls/html/index.html`
Änderungen für Java 1.1;
`http://www.javasoft.com/docs/books/jls/html/1.1Update.html`
(Zugriff: 20-Sep-1997)

[Mäckler00] Andreas Mäckler (Hrsg.); 1460 Antworten auf die Frage: Was ist Kunst? — Neuausgabe — Köln (DuMont), 2000, ISBN 3-7701-5420-7. {Hinweis: „Künstler nutzen die Zitate für Aktionen." }

[McGilton/Campione92] Henry McGilton / Mary Campione; PostScript by Example, Reading Massachusetts u. a. (Addison-Wesley) 1992, ISBN 0-201-632286-4.

[Nake03] Frieder Nake, space.color — Raum. Algorithmus. Farbe, in: [Rödiger03] S. 135–140. {Hinweis: „sätze, vorgetragen zur eröffnung der ausstellung von manfred mohr im museum für konkrete kunst in ingolstadt am 14. oktober 2001". (Im Orginal in kleinen Buchstaben geschrieben)}

[Nadin03] Mihai Nadin; Das Interessante als computationale Zielsetzung, in: [Rödiger03] S. 99–133. {Hinweis: Überarbeiteter Vortrag vom 16. Dezember 1998.}

[Rödiger03] Karl-Heinz Rödiger (Hrsg.); Algorithmik — Kunst — Semiotik, Hommage für Frieder Nake, Heidelberg (Synchron Wissenschaftsverlag) 2003, ISBN 3-935025-60. {Hinweis: Festschrift für Frieder Nake, einer der großen Pioniere der Computergraphik.}

[Schader+03] Martin Schader / Lars Schmidt-Thieme; Java — Eine Einführung, Berlin Heidelberg (Springer), 4. Auflage 2003, ISBN 3-540-00663-X. {Hinweis: Das Buch enthält gelungene Übungen mit Lösungen (auf der beigefügten CD-ROM).}

[Selman02] Daniel Selman; Java3D Programming, Greenwich CT 06830 (Manning Publications Co.), ISBN 1-930110-35-9. {Hinweis: "*Java 3D Programming* is a roadmap for application developers."}

[Shavor+03] Sherry Shavor / Jim D'Anjou / Scott Fairbrother / Dan Kehn / John Kellerman / Pat McCarty; The Java^{TM} Developer's Guide to Eclipse, Boston u. a. (Addison-Wesley), ISBN 0-321-15964-0. {Hinweis: "This Book does an excellent job of helping you learn Eclipse."}

[Ware04] Colin Ware; Information Visualization — Perception for Design, Amsterdam (Morgan Kaufmann / Elsevier) 2004, ISBN 1-55860-819-2. {Remark: "This book combines a strictly scientific approach to human perception with a practical concern for the rules governing the effective visual presentation of information."}

C.2 Web-Quellen

Java3D-Sun-Material
`http://java.sun.com/products/java-media/3D/`

Java3D-Material
`http://java3d.virtualworlds.de/`

J2SE-SDK-Dokumentation
`http://java.sun.com/docs/index.html`

Zur Geschichte von Java^{TM}
`http://java.sun.com/nav/whatis/storyofjava.html`

C.3 Werkzeuge zum Manuskript

Mit folgender Software wurde Faszination Programmierung erstellt:

Editor: GNU Emacs 21.3.1 (2002-03-19); jEdit 4.1 final

Layout: TeX, Version 3.14159 (Web2c 7.3.7x), LaTeX2e <2000/06/01>; Document Class: book 2001/04/21 v1.4e Standard LaTeX document class

Hardcopy: Corel CAPTURE 11; Corel PHOTO-PAINT 11 (version 10.427)

Figure: Microsoft Visio 2000 SR1 (6.0.2072)

Index: makeindex, version 2.13 [07-Mar-1997] (using kpathsea)

DVI→PS: LaTeX-File (Device Independent) to Postscript: dvips(k) 5.90a Copyright 2002 Radical Eye Software (www.radicaleye.com)

PS→PDF: Postscript file to PDF-File: Adobe Acrobat Distiller 8.0 Professional

Security: Adobe Acrobat 8.0 Professional (Version 5.0)

C.4 Liste der Fonts

Listing C.1: `fontPS`

```
%!PS-Adobe-3.0
%%Creator: McGilton-Campione-1992, p.554 (Bonin)
%%Title: List of fonts
%%CreationDate: 07-Oct-2005
%%EndComments
%%BeginProlog
/inch { 72 mul } def
%%EndProlog
/PrintSize 12 def
/Leading 14 def
/TopMargin 10.5 inch def
/BottomTopMargin 0.5 inch def
/LeftMargin 0.5 inch def
/Courier-Bold findfont
PrintSize scalefont
setfont
/whichSide 0 def
/BaseLine TopMargin def
/JunkString 256 string def
/ShowFontName {
  JunkString cvs
  LeftMargin whichSide 4.25 inch mul add BaseLine moveto
  show
  /whichSide 1 whichSide sub def
  whichSide 0 eq {
      /BaseLine BaseLine Leading sub def
      BaseLine BottomTopMargin lt {
```

```
FB                                      Fa
FT                                      Courier-Bold
Fv                                      Ff
Fh                                      Fg
FJ                                      FK
FW                                      F8
FL                                      Fq
F1                                      FD
FU                                      FE
F4                                      F5
FH                                      CMMI10
FZ                                      Fn
FA                                      FS
FC                                      F3
Ft                                      Gb
FG                                      Courier
F7                                      CMSY10
Helvetica-Bold                          Times-Bold
FQ                                      CMSY8
Helvetica                               FF
Fe                                      CMMI8
Fi                                      Times-Roman
FI                                      FX
F9                                      Fm
Fl                                      F0
CMR10                                   Ga
Fc                                      CMSY9
Fb                                      Times-Italic
Fr                                      Fs
Fu                                      F6
Fx                                      Helvetica-Oblique
Gc                                      Gd
Ge                                      FM
Fk                                      FP
Fp
```

Legende:

DVIPS(k) 5.90a Copyright 2002 Radical Eye Software ↪`http://www.radicaleye.com`
Die Idee für dieses Beispiel entnommen aus [McGilton/Campione92] p. 554; PostScript-Quellcode
↪ Seite 191.

Abbildung C.1: Liste der Fonts von `DVIPS`

Legende:
CorelDraw12 Version 12.0.0.458 Copyright 2003
Die Idee für dieses Beispiel entnommen aus [McGilton/Campione92] p. 554; PostScript-Quellcode ↪ Seite 191.

Abbildung C.2: Liste der Fonts von `CorelDraw12`

```
28              showpage
                /BaseLine TopMargin def
30          } if
        } if
32  } def
    FontDirectory
34  {
        pop
36      ShowFontName
    } forall
38  showpage
    %%EOF
```

C.5 Glossar

Alpha-Kanal Ein Bild wird üblicherweise über drei Farbwerte beschrieben: Rot, Grün und Blau (RGB) — beim Drucken : Gelb (*yellow*), Magenta (*magenta*) und Zyan (*cyan*). Zusätzlich wird noch eine Tranzparenzangabe benötigt. Sie gibt pro Bildpunkt den Grad der Durchsichtigkeit an. Diesen Tanspa-renzkanal bezeichnet man als *Alpha*-Kanal.

Appearance Aussehen, äußerer Schein

Bump-Mapping Mit einem Foto, gelegt auf die Objektoberfläche, erhöht man ohne großen

Aufwand den Realismus des Objektes (↪ Textur 194. Mit einem Bump-Mapping verändert man die Richtung der Oberflächennormalen. Beispielsweise ist man damit in der Lage, eine gewellte Oberfläche vorzutäuschen.

Branch Kante, Abzweigung

Keyframing Eine Animation wird durch prägende Teilszenen spezifiziert. Die dazwischenliegenden Teile werden durch Interpolation aller Bildwerte berechnet.

Level-of-Detail (LOD) Die Objektdarstellung wird, je nach der visuellen Größe auf dem Bildschirm, in ihrer Komplexität verändert. Ein entfernt befindliches Objekt besteht aus wenigen Daten, ein nahes aus vielen.

inward inner, curve inward ≡ Kurve nach innen

Polygon beschreibt eine Fläche im Raum, zum Beispiel durch ein Dreieck. Ein allgemeines Polygon kann eine beliebige Eckenanzahl haben. Als Polyeder bezeichnet man das über die Eckpunkte und Seitenflächen beschriebene Volumenelement.

Polyeder Volumenelement ↪ Polygon 194

Rendering ≈ Umwandlung, Transformation, Wiedergabe, (künstlerische) Interpretation. Die Eingabe in Form von Geometrie- und Beleuchtungsdaten wird in ein betrachtbares Bild „umgewandelt“.

Sphere Einflussbereich, Kugel

Textur ist ein Bild zur Projektion auf die Objektoberfläche, um das Objekt realistischer erscheinen zu lassen. Bei der Projektion ist es notwendig, jedem Bildpunkt der Textur einen Alpha-Wert (↪ Alpha-Kanal 193) mitzugeben. Er spezifiziert die Durchsichtigkeit des Bildpunktes.

Trigger auslösendes Ereignis, Auslöseimpuls

Vertex pl. *vertices* Scheitel(punkt), Spitze

C.6 Abkürzungen und Akronyme

API	Application programming interface
AWT	Abstract Windowing Toolkit
DAG	Directed acyclic graph
FHNON	Fachhochschule Nordost Niedersachsen
FOV	Fields of view
LOD	Level-of-Detail
UML	Unified Modeling Language

Abbildungsverzeichnis

Tabellenverzeichnis

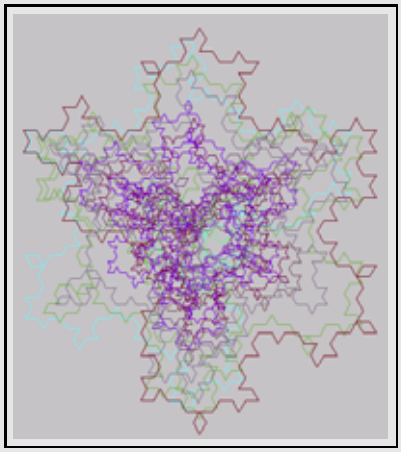
Listings

Anhang D

Index

Index


Programmieren
bleibt
schwierig!



* * *