Thomas Künneth

Java für Windows

Galileo Computing

Liebe Leserin, lieber Leser,

der Erfolg von Java erklärt sich nicht zuletzt durch die Plattformunabhängigkeit der Sprache. Damit die Lauffähigkeit auf verschiedenen Systemen gewährt bleiben kann, ist jedoch ein erheblicher Aufwand notwendig. Nicht immer wird man als Entwickler das gewünschte Resultat ohne Kompromisse erreichen können. Zum anderen gilt es Anforderungen an das jeweilige Wirtssystem im Hinblick auf Menüs, Schaltflächen, Installation, Softwareverteilung und Einbettung in das Betriebssystem zu berücksichtigen.

Dieses Buch hat die Java-Programmierung für Windows im Blick. Die meisten Java-Anwendungen werden auf der Plattform von Microsoft betrieben, daher liegt es nahe, zu schauen, welche Optimierungen hier vorgenommen werden können. Der Verdienst von Thomas Künneth ist es, hier ein Handbuch für die Java-Entwicklung für Windows vorgelegt zu haben. Er weiß, welche Kompromisse eventuell erforderlich sind und zeigt Lösungen für eine optimale Integration in Windows. Angefangen von der optimalen Installation, der Auswahl der richtigen Entwicklungsumgebung, der Anpassung von Swing-Anwendungen für Windows, Datenaustausch mit Windows-Anwendungen wie Word, Excel oder Access, den Umgang mit Java-COM-Brücken oder Kommunikationsschnittstellen bis hin zur Einbettung von Multimedia-Inhalten wird dieses Buch Ihnen ein zuverlässiger Begleiter sein.

Dieses Buch wurde mit großer Sorgfalt geschrieben, begutachtet, lektoriert und produziert. Sollte dennoch etwas nicht so funktionieren, wie Sie es erwarten, dann scheuen Sie sich nicht, sich mit mir in Verbindung zu setzen. Ihre freundlichen Anregungen und Fragen sind jederzeit willkommen.

Viel Vergnügen beim Lesen und Programmieren!

Stephan Mattescheck
Lektorat Galileo Computing

stephan.mattescheck@galileo-press.de
www.galileocomputing.de

Galileo Press • Rheinwerkallee 4 • 53227 Bonn

Auf einen Blick

Der Name Galileo Press geht auf den italienischen Mathematiker und Philosophen Galileo Galilei (1564–1642) zurück. Er gilt als Gründungsfigur der neuzeitlichen Wissenschaft und wurde berühmt als Verfechter des modernen, heliozentrischen Weltbilds. Legendär ist sein Ausspruch »Eppur se muove« (Und sie bewegt sich doch). Das Emblem von Galileo Press ist der Jupiter, umkreist von den vier Galileischen Monden. Galilei entdeckte die nach ihm benannten Monde 1610.

Lektorat Stephan Mattescheck
Korrektorat Claudia Falk, Holger Schmidt
Einbandgestaltung Barbara Thoben, Köln
Herstellung Iris Warkus
Titelbild zefa visual media
Satz Typographie & Computer, Krefeld
Druck und Bindung Koninklijke Wöhrmann B.V., Zutphen, Niederlande

Dieses Buch wurde gesetzt aus der Linotype Syntax (10/13,5 pt) in FrameMaker.
Gedruckt wurde es auf chlorfrei gebleichtem Offset-Papier.

Gerne stehen wir Ihnen mit Rat und Tat zur Seite:
stephan.mattescheck@galileo-press.de bei Fragen und Anmerkungen zum Inhalt des Buches
service@galileo-press.de für versandkostenfreie Bestellungen und Reklamationen
stefan.krumbiegel@galileo-press.de für Rezensions- und Schulungsexemplare

Bibliografische Information Der Deutschen Bibliothek
Die Deutsche Bibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über http://dnb.ddb.de abrufbar.

ISBN 3-89842-727-7

1. Auflage 2006

Für meine über alles geliebte Ehefrau Moni,
meine Eltern Rudolf und Gertraud
und meinen Bruder Andreas

Inhalt

Einleitung

Herzlich willkommen zu Java für Windows!

Microsoft Windows ist das derzeit am häufigsten verwendete Betriebssystem. Die meisten Java-Installationen werden also auf dieser Plattform betrieben. Eigentlich sollte man deshalb davon ausgehen, dass die beiden Hersteller *Sun Microsystems* und *Microsoft* an einer guten Zusammenarbeit interessiert sind.

Stationen eines schwierigen Miteinanders

In der Tat haben die beiden Firmenchefs Scott McNealy und Steve Ballmer in mehreren viel diskutierten Vereinbarungen ihren Willen zu einem konstruktiven Umgang der beiden Firmen miteinander betont.[1] Das war allerdings keineswegs immer so. Microsoft hatte Java von Sun lizenziert und in Form einer eigenen Implementierung auch in zahlreiche Produkte integriert, später dann aber die Sprache selbst sowie Kernklassen in – aus der Sicht von Sun – unzulässiger Weise verändert. Die Folge war ein langer, erbittert geführter Rechtsstreit.[2] Über Microsofts Gründe ist in der Vergangenheit viel philosophiert und spekuliert worden. Ich möchte mich an dieser Stelle nicht daran beteiligen. Tatsache ist, dass Microsoft mit dem .NET Framework seit einigen Jahren ein Produkt vermarktet, das mit Java in sehr vielen Punkten vergleichbar ist. Tatsache ist aber auch, dass Sun zwar eine Einigung mit Microsoft erzielen konnte,[3] es in der Folgezeit allerdings einige Mühe gekostet hat, Javas Position auf dem Desktop weiter zu festigen. So wollte man den Windows-Hersteller verpflichten, eine aktuelle Version der Sun Java Virtual Machine mit Windows XP auszuliefern.[4]

Konsequenzen für den Entwickler

Mittlerweile ist die Sprache aus fast allen Microsoft-Produkten verschwunden. Für den Anwender ist dies glücklicherweise kein Problem mehr. Musste Sun im Hinblick auf den Wegfall der Microsoft Virtual Machine einst mit Anzeigen für die Verbreitung ihrer Laufzeitumgebung trommeln,[5] ist die Technologie inzwischen so etabliert, dass zahlreiche Computerhersteller Java von sich aus installieren. Für Sie als Entwickler ist die Sache insofern komplizierter, als Sie

1 http://www.heise.de/newsticker/meldung/59550
2 http://www.heise.de/newsticker/meldung/1508
3 http://www.heise.de/newsticker/meldung/14721
4 http://www.heise.de/newsticker/meldung/40869
5 http://www.heise.de/newsticker/meldung/20291

Hilfestellung in Bezug auf das Zusammenspiel von Java und Windows zwar von Sun, nicht aber vom Hersteller des Betriebssystems erwarten können. Mit diesem Buch möchte ich Ihnen deshalb helfen, auf Windows-Systemen das Beste aus Java herauszuholen. In den folgenden Kapiteln werden Sie zahlreiche Klassenbibliotheken kennen lernen, die Ihnen Zugriff auf Funktionen gestatten, die mit Standard-Java unzugänglich bleiben. Aber ist dies eigentlich erstrebenswert?

Plattformunabhängigkeit und der verantwortungsvolle Umgang damit

Der Java-Erfinder *Sun Microsystems* hat mit dem Slogan »write once, run everywhere« stets die Plattformunabhängigkeit der Sprache und ihrer Kernklassen betont. Um die Lauffähigkeit auf allen bedeutenden Systemen sicherzustellen, ist allerdings ein zum Teil erheblicher Aufwand nötig. Beispielsweise hat sich bei den grafischen Komponenten des *Abstract Window Toolkits* der Ansatz des kleinsten gemeinsamen Nenners, also die Konzentration auf Bedienelemente, die die Benutzeroberflächen aller unterstützten Systeme zur Verfügung stellen, als nicht geeignet erwiesen, nutzerfreundliche Anwendungen zu realisieren. Deshalb greift die Weiterentwicklung *Swing* nicht mehr auf vorhandene native Elemente zurück, sondern implementiert alle zur Verfügung gestellten Komponenten vollständig selbst. Leider ist jedoch auch dieser Ansatz nicht frei von Problemen. Wie Sie diesen begegnen, zeige ich Ihnen übrigens in Kapitel 3, *Feintuning der Benutzeroberfläche*.

Zugegebenermaßen ist es aus der Sicht des Anwenders zweitrangig, welche Technologie für die Darstellung der Benutzeroberfläche eines Programms verwendet wird. Zumindest, solange er vertraute Elemente vorfindet und sich diese wie erwartet verhalten. Allerdings gehört zu diesem vertrauten Äußeren mehr als das »richtige« Aussehen der Menüs und Schaltflächen. Schon der Installationsvorgang, das Einrichten und Konfigurieren einer Anwendung, entscheidet darüber, ob sich ein Programm vertraut anfühlt oder eher fremdartig und ungewohnt wirkt. Bedauerlicherweise hat Java auch hier einige Defizite, die zum Teil durch die Plattformunabhängigkeit verursacht werden. So verfolgen die großen Systeme Linux, Mac OS X und Windows naturgemäß unterschiedliche Strategien im Hinblick auf Softwareverteilung, Installation und Einbettung in das Wirtssystem. Java geht mit *Web Start* (eine Technologie, die ich Ihnen ausführlich in Kapitel 10, *Deployment*, vorstelle) auch hier einen eigenen Weg, der allerdings zumindest in seiner jetzigen Form nicht vollständig zufrieden stellend ist. Dass an dieser Stelle noch einiges an Arbeit nötig ist, wird deutlich, wenn man Anwender nach ihren Erfahrungen mit Java auf dem Desktop befragt. Derzeit ist Java auf dem Desktop bei weitem noch nicht so erfolgreich wie auf dem Server. Ob dies nun, wie häufig kolportiert wird, an der feh-

lenden »Killer-Anwendung« liegt oder daran, dass die zur Verfügung stehenden Programme für den Anwender nicht vertraut genug wirken, mag der geneigte Leser selbst beantworten. Aus meiner Sicht ergibt sich hieraus aber als Hauptforderung an den Java-Entwickler, einfach zu bedienende Anwendungen zu schreiben, die sich allein schon durch ihr Äußeres und ihre Bedienung bekannt »anfühlen«. Wenn die zur Verfügung stehenden plattformunabhängigen Technologien hierfür nicht genügen, sollten stattdessen systemspezifische Mechanismen in Erwägung gezogen werden.

Augenmaß beim Einsatz plattformspezifischer Funktionen

In diesem Buch möchte ich Ihnen deshalb auch zeigen, was unter Windows von einer Anwendung erwartet wird und wie sich dies mit Java realisieren lässt. Bevor ich Ihnen den Inhalt der einzelnen Kapitel kurz vorstelle, möchte ich aber doch einigen möglicherweise aufkeimenden Missverständnissen vorbeugen. So glaube ich keineswegs, dass das Verwenden plattformspezifischer Funktionen automatisch zu guten Programmen führt. Das Paradigma »write once, run everywhere« hat viel Charme und ist einer der Gründe, warum Java gerade auf den Servern so erfolgreich wurde. Sie sollten deshalb auf jeden Fall versuchen, die Lauffähigkeit Ihrer Programme auf möglichst vielen Plattformen sicherzustellen. Deshalb meine Bitte: Machen Sie Kernfunktionen Ihrer Anwendung wenn möglich nicht von Bibliotheken abhängig, die nur für bestimmte Systeme angeboten werden. Prüfen Sie vor der Verwendung einer API, ob sie zur Verfügung steht. Java macht es Ihnen mit *Reflection* sehr einfach, zu ermitteln, ob bestimmte Klassen oder Methoden vorhanden sind. Die Erweiterung von Java durch zusätzliche Klassenbibliotheken verlangt also eine Menge Disziplin, erlaubt aber auf der anderen Seite eine für Java bisher nicht gekannte Nähe zum Wirtssystem.

Ein Streifzug durch das Buch

Eine verlässliche Entwicklungsplattform ist eine unverzichtbare Voraussetzung für das Schreiben guter Programme. Neben den richtigen Werkzeugen gehört hierzu eine optimal installierte Laufzeitumgebung. In Kapitel 1, *Installation und Konfiguration*, zeige ich Ihnen deshalb zunächst, was Sie beim Installieren und Einrichten des *Java Development Kits* und der Laufzeitumgebung beachten sollten. So erfahren Sie, warum es beim parallelen Betrieb mehrerer Java-Versionen zu Problemen kommen kann und wie Sie diesen begegnen. Außerdem stelle ich Ihnen die bisher kaum genutzten *Erweiterungsverzeichnisse* vor und erläutere, wie Sie Klassenbibliotheken gleichzeitig mehreren Java-Installationen zur Verfügung stellen können.

Kapitel 2, *Entwicklungswerkzeuge*, baut auf diesen Grundlagen auf und stellt für das Programmieren unverzichtbare Tools vor. Beispielsweise lernen Sie das noch recht neue, kostenlose Microsoft-Programm *SyncToy* kennen, mit dem Sie auf sehr bequeme Weise Sicherungskopien Ihrer Daten und Quelltexte anfertigen können. Außerdem zeige ich Ihnen, wie Sie unbekannten Klassendateien Informationen wie Methodennamen und Signaturen entlocken und bei Bedarf sogar zurück in Java-Quelltexte verwandeln können.

Kapitel 3, *Feintuning der Benutzeroberfläche*, steht ganz im Zeichen von *Swing*. Ich zeige Ihnen, was es mit dem Konzept des *Pluggable Look and Feel* auf sich hat und wie Sie es in Ihren Programmen einsetzen können. Beispielsweise ermöglicht der Aufruf einer einzigen Methode, eine Anwendung wie Office XP oder Office 2003 aussehen zu lassen. Außerdem lernen Sie, warum Swing-Programme manchmal nicht wie »echte« Windows-Anwendungen aussehen und wie Sie diesen Missstand beheben können.

Auch Kapitel 4, *Zusätzliche Komponenten für die Benutzeroberfläche*, beschäftigt sich mit dem Aussehen Ihrer Programme. Ich stelle Ihnen darin Klassenbibliotheken vor, mit denen Sie Ihre Anwendungen um Bedienelemente und -konzepte erweitern können, die nicht durch Swing-Klassen abgedeckt werden. Hierzu zählen kleine Infotäfelchen, die sich vom unteren Bildschirmrand nach oben schieben und den Anwender über Ereignisse wie den Eingang einer E-Mail oder den Abschluss der Virenprüfung informieren, sowie komplexe Komponenten wie Aufgabenleisten und Zeichensatzauswahl-Dialoge.

»Datenaustausch« ist das zentrale Thema von Kapitel 5, *Kommunikation mit Microsoft Office*. Wenn Sie Word- oder Excel-Dateien lesen oder schreiben müssen, finden Sie hier geeignete Klassenbibliotheken. Zudem wird erläutert, wie Sie auf Ihre Kontakte und Notizen in Outlook zugreifen können.

Kapitel 6, *Datenbanken*, widmet sich ausführlich der Datenbanksprache *SQL* und zeigt, wie Sie mit der *Structured Query Language* aus Ihren Java-Anwendungen auf Datenbanken zugreifen können. Außerdem lernen Sie, wie mit Windows-Bordmitteln Microsoft Access-kompatible Datenbanken angelegt werden und wie Sie auf diese zugreifen. Schließlich stelle ich ein äußerst nützliches Allround-Talent im Umgang mit den so genannten *relationalen Datenbanksystemen* vor.

Kapitel 7, *JDesktop Integration Components*, stellt die gleichnamige Open Source-Klassenbibliothek vor, die in mehreren Bereichen für eine bessere Integration von Java-Anwendungen in das Wirtssystem sorgt. Sie können mit ihrer Hilfe etwa Dateitypen registrieren, Symbole im Infobereich der Taskleiste able-

gen und sogar den Standard-Webbrowser als Komponente in Ihre Programme integrieren. Wie dies funktioniert, wird anhand zahlreicher Beispielanwendungen demonstriert.

Die *Registry* ist als zentrale Konfigurationsdatenbank eine Schlüsselkomponente von Windows. Dort legen nicht nur Anwendungsprogramme ihre Einstellungen ab, sondern auch Windows selbst speichert in der *Registrierung* den aktuellen Zustand des Systems. Der Zugang zur Registry öffnet viele Türen für Java. Welche dies sind und was Sie mit dem, was hinter diesen Türen liegt, anfangen können, zeige ich in Kapitel 8, *Zugriff auf die Registry*.

Auch Kapitel 9, *Java-COM-Brücken*, behandelt ein fundamentales Konzept der Windows-Welt. Das so genannte *Component Object Model* ist zwar in seinem Kern als sprachen- und plattformneutrale Technologie ausgelegt, hat sich im Laufe der Jahre aber zum Standard für das Bereitstellen von Funktionen über Anwendungsgrenzen hinweg etabliert. *Java-COM-Brücken* erlauben Java-Programmen den Datenaustausch und die Inanspruchnahme von Funktionen praktisch aller Windows-Anwendungen, die die COM-Technologie implementieren. Sie lernen beispielsweise, die Windows-eigene Sprachsynthese zu nutzen und Daten in Microsoft OneNote zu importieren.

Kapitel 10, *Deployment*, widmet sich der Frage, wie Sie Ihre Java-Programme auf bequeme Weise verteilen können und wie Sie für eine reibungslose Installation auf den Zielrechnern sorgen. Außerdem stelle ich Ihnen *Java Web Start* vor, eine Technologie, die das Installieren von Programmen radikal vereinfachen möchte, nämlich auf das Anklicken eines Links auf einer Webseite.

Die multimedialen Talente Ihres Rechners entfalten Sie nach Lektüre von Kapitel 11, *Multimedia*. In diesem Kapitel lernen Sie das *Java Media Framework* kennen, spielen MP3-Stücke ab und laden und speichern Grafiken in den gängigen Windows-Formaten.

Das abschließende Kapitel 12, *Kommunikation*, macht Sie mit den Grundlagen der Java Communications API vertraut und zeigt Ihnen, wie Sie mit Bluetooth-fähigen Endgeräten Daten austauschen. Außerdem erfahren Sie, wie Sie auf die Funktionen von so genannten Instant Messaging-Anwendungen zugreifen können.

Die einzelnen Kapitel dieses Buches sind in sich geschlossen, Sie können also nach Belieben zwischen den Themengebieten springen. Soweit es innerhalb eines Kapitels Abhängigkeiten gibt, weise ich Sie in der Einleitung ausdrücklich darauf hin.

Konventionen

Im Folgenden möchte ich Sie mit ein paar Konventionen vertraut machen, die Ihnen bei der Lektüre des Buches helfen sollen. Namen von neu eingeführten *Programmen*, *Produkten* und *Firmen* werden kursiv gesetzt. Treten diese Namen allerdings gehäuft auf, behalte ich die normale Darstellung bei, um den Lesefluss nicht zu stören.

Auch Dateinamen oder Teile davon erscheinen kursiv, beispielsweise *C:\Programme* oder *.jar*-Archiv. Texte, die auf dem Bildschirm erscheinen, beispielsweise Menüeinträge, Titel von Dialogen oder Fenstern erscheinen hingegen **halbfett**. Listings sowie Kommandofolgen, die Sie selbst eintippen müssen, werden `so dargestellt`.

Wenn ich Sie auf einen Sachverhalt besonders aufmerksam machen möchte, erscheint der Hinweis in einem eigenen, grau hinterlegten Kasten.

Software-Versionen

Die primäre Zielplattform dieses Buches ist Windows XP in seinen verschiedenen Ausgaben. Die Bildschirmfotos entstanden unter *Windows XP Home Edition* sowie *Windows Media Center Edition 2005*. Ältere Betriebssysteme bleiben in der Regel unberücksichtigt, da selbst Windows XP seit mehreren Jahren am Markt und dessen Nachfolger zumindest am Horizont zu sehen ist. Bei Java liegt mein Hauptaugenmerk auf der *Java Standard Edition 5*. Auch hier finden ältere Versionen keine explizite Erwähnung.

Danksagung

Dieses Buch hätte ohne die Hilfe und freundliche Unterstützung vieler Menschen nicht entstehen können. Ihnen allen gehört mein tief empfundener Dank. Dazu zählen die Mitarbeiterinnen und Mitarbeiter von Galileo Computing, namentlich mein Lektor Stephan Mattescheck, aber ganz besonders auch die Entwicklerinnen und Entwickler der Software, von der dieses Buch handelt. Sie alle leisten hervorragende Arbeit, und ich bin stolz und dankbar, ihre Programme hier verwerten zu können. Bedanken möchte ich mich auch bei allen, die mir während des Schreibens mit Rat und Tat zur Seite standen, für Lob und Tadel, für das unermüdliche Lesen und Kommentieren meiner Manuskripte.

Und ich danke meiner Familie für ihre Liebe und Unterstützung, ihre Geduld und ihr Verständnis: meiner Ehefrau Moni für das unermessliche Glück, das sie mir jeden Tag schenkt, meinen Eltern Rudolf und Gertraud für alles, was sie mir auf den Weg gegeben haben, und meinem Bruder Andreas für die vielen Dinge, die er mir ermöglicht hat.

1 Installation und Konfiguration

1 Installation und Konfiguration

Eine perfekt eingerichtete Entwicklungsplattform ist unabdingbar für reibungsloses und angenehmes Programmieren. Dieses Kapitel möchte Ihnen helfen, Ihren Computer optimal auf das Schreiben von Java-Anwendungen vorzubereiten.

Eine Java-Laufzeitumgebung ist schnell installiert, auch das Aufspielen des *Java Development Kits* erfordert nur wenige Mausklicks. Die Probleme einer solchen schnell »zusammengeschusterten« Plattform offenbaren sich meistens zu einem denkbar ungünstigen Zeitpunkt, nämlich mitten im Entwicklungsprozess. In diesem Kapitel möchte ich Sie zunächst auf potenzielle Fallen aufmerksam machen und Ihnen bei der Planung Ihrer Entwicklungsplattform helfen. Außerdem begleite ich Sie bei der Installation des Development Kits und zeige, wie Sie Ihre Java-Installation so »tunen«, dass sie sich harmonisch in die Windows-Umgebung einfügt.

1.1 SDK und JRE

Java-Programme werden innerhalb einer *Laufzeitumgebung* abgearbeitet (daneben ist auch der englische Begriff *Runtime Environment* sehr gebräuchlich). Der Compiler erzeugt aus den Quelltexten so genannte Klassendateien, die *Bytecode* enthalten. Hierbei handelt es sich um die Maschinensprache eines virtuellen Prozessors, der durch die *Java Virtual Machine* implementiert wird. Die Laufzeitumgebung übernimmt also das »Interpretieren« des Bytecodes und bindet das Java-Programm in das Wirtssystem ein, sodass die Anwendung Betriebsmittel wie Dateisystem, Schnittstellen, Maus, Tastatur und Bildschirm in Anspruch nehmen kann. Zu der Laufzeitumgebung gehört eine kaum überschaubare Anzahl von Kernklassen, die jedem Programm einen festen, wohl definierten Satz an Funktionalitäten zur Verfügung stellen. Die Java-Laufzeitumgebung existiert mittlerweile in zahlreichen Versionen von verschiedenen Herstellern. Allerdings gehe ich in diesem Buch nicht ausführlicher auf solche alternativen Implementierungen außerhalb der Sun-Produktlinie ein. Aus Kompatibilitätsgründen kann es sinnvoll sein, mehrere Versionen der Java-Laufzeitumgebung zu installieren und parallel zu betreiben, beispielsweise im Hinblick auf Änderungen an der physikalischen Struktur der Klassendateien, Änderungen im Sprachumfang oder an der Klassenbibliothek. Unabdingbare Grundlage für die Entwicklung von Java-Anwendungen ist zumindest ein Compiler, der Quelltexte in Klassendateien übersetzt. Das wahrscheinlich am häufigsten verwendete Tool, *javac*, ist Teil des von Sun zur Verfügung gestellten

Entwicklerpakets. Zwar existieren auch hier mittlerweile eine Reihe alternativer Compiler, allerdings bildet das *Java Development Kit* (das oft auch *Software Development Kit* genannt wird) einen Quasi-Standard, an den ich mich in diesem Buch halte.

Eine Java-Installation besteht also mindestens aus einer Java-Laufzeitumgebung. Sun bietet diese unter der Produktbezeichnung *J2SE Runtime Environment* (*JRE*) zum kostenlosen Download an.[1] Sie richtet sich an den Endanwender, der Java-Programme ohne großen Aufwand ausführen möchte. Um die Akzeptanz innerhalb dieses Anwenderkreises weiter zu erhöhen, pflegt Sun die Promotionsseite **java.com**[2], die zusätzlich eine sehr komfortable Web-gestützte Installation der Laufzeitumgebung erlaubt.

Abbildung 1.1 Suns Java-Promotionsseite java.com

Als weitere Komponente einer Java-Installation kommt für den Entwickler das bereits kurz angesprochene *Java Development Kit* (*JDK*) hinzu. Es enthält eine Reihe von Werkzeugen, die Sie für die Programmierung von Java-Anwendun-

1 http://java.sun.com/j2se/1.5.0/download.jsp
2 http://www.java.com/de/

gen benötigen, beispielsweise den Compiler *javac*, den *jarsigner* zum Signieren von *.jar*-Archiven und das Tool *packager* zum Erzeugen von *ActiveX*-Komponenten aus *JavaBeans*.

1.1.1 Überlegungen vor der Installation

Bevor Sie mit der Installation des JDK oder einer Java-Laufzeitumgebung beginnen, ist es ratsam, sich ein paar Gedanken über die entstehende Verzeichnisstruktur zu machen. Auf einem Entwicklungsrechner werden sehr wahrscheinlich verschiedene Versionen des Development Kits parallel installiert sein. Hinzu kommen möglicherweise mehrere Versionen des JRE. Da unter Windows (anders als beispielsweise unter Apples Mac OS X) eine Java-Installation nicht durch den Betriebssystemhersteller gepflegt wird, ist es im Hinblick auf etwaige Probleme besonders wichtig, zu wissen, welche Version in welches Verzeichnis installiert wurde. Generell gilt: Definieren Sie ein Java-Basisverzeichnis, das alle Versionen von JDK und JRE bündelt, beispielsweise *C:\Programme\Java*. Glücklicherweise schlagen aktuelle Java-Versionen dies von sich aus vor. Meiden Sie das Wurzelverzeichnis und korrigieren Sie solche Vorschläge der Installationsroutinen älterer Java-Ausgaben. Versuchen Sie ferner, alte Versionen zuerst zu installieren. Auf diese Weise vermeiden Sie, dass gemeinsam genutzte Komponenten, etwa das Modul *Java* der *Systemsteuerung* unbrauchbar werden. Und: Bitte überlegen Sie, ob Sie das komplette Development Kit einer bestimmten Version installieren müssen oder ob vielleicht eine Laufzeitumgebung ausreichend ist.

1.1.2 Die Installation des SDK

Sun stellt das Java Development Kit unter der Produktbezeichnung *J2SE Development Kit* zum kostenlosen Download bereit.[3] Am einfachsten zu handhaben ist trotz seiner beachtlichen Größe die so genannte *Offline Installation*. Sie bietet den Vorteil, durch das Brennen auf CD/DVD oder das Archivieren auf Festplatte jederzeit verfügbar zu sein. Nach dem Herunterladen der Installationsdatei (zum Zeitpunkt der Drucklegung ist dies *jdk-1_5_0_04-windows-i586-p.exe*) starten Sie diese. Achten Sie bitte darauf, als Benutzer mit Administratorrechten angemeldet zu sein. Nach dem Bestätigen der Lizenzvereinbarung haben Sie einige Einstellungsmöglichkeiten, die Sie in Abbildung 1.2 sehen. Die Pakete *Demos* und *Source Code* sollten Sie nur bei sehr wenig freiem Festplattenspeicherplatz nicht installieren. Anders verhält es sich mit *Public JRE*. Diese Option installiert zusätzlich eine vollständige Java-Laufzeitumgebung, wie sie auch Endkunden zum Download angeboten wird.

3 http://java.sun.com/j2se/1.5.0/download.jsp

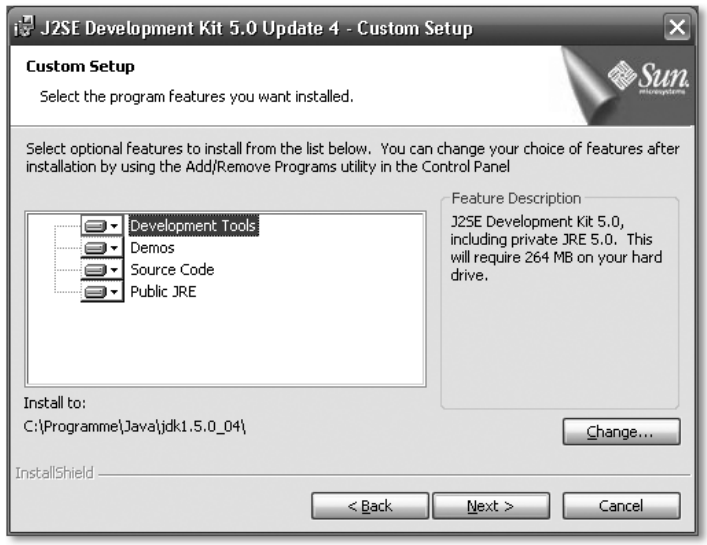

Abbildung 1.2 Einstellungen vor dem Beginn der Installation

Wenn Sie sich entscheiden, die *Public JRE* nicht zu installieren, erhalten Sie selbstverständlich ein vollständiges Entwicklerpaket. Allerdings haben Sie beispielsweise keine Möglichkeit, automatische Updates zu konfigurieren. Nähere Informationen hierzu finden Sie in Abschnitt 1.1.2. Außerdem wird die Laufzeitumgebung nicht in der Windows-Registrierung eingetragen. Für Programme, die dort nach installierten Java-Versionen suchen, bleibt sie folglich unsichtbar. Dies kann zu einem Problem werden, wenn Sie eine Anwendung installieren, deren Setup-Routine nach verfügbaren Laufzeitumgebungen sucht und abbricht, falls sie in der Windows-Registrierung keine findet.

Bevor Sie zum nächsten Schritt der Installation kommen, sollten Sie durch Anklicken der Schaltfläche **Change** das voreingestellte Installationsverzeichnis anpassen. Zwar hat das Setup-Programm korrekterweise *C:\Programme\Java* als Sammelverzeichnis für alle Java-Versionen vorgeschlagen, möchte die eigentliche Installation aber beispielsweise im Verzeichnis *jdk1.5.0_04* ablegen. Auf die derzeit dreistellige Versionsnummer folgt nach einem Unterstrich noch die Nennung, um welches Update es sich handelt. Da sich Java automatisch aktualisieren kann, hat der hier vermerkte Update-Stand wahrscheinlich nicht sehr lange Gültigkeit. Deshalb schlage ich vor, Updates bei der Benennung von Verzeichnissen außen vor zu lassen und stattdessen als Namen *jdk* gefolgt von der dreistelligen Versionsnummer zu vergeben, wie in Abbildung 1.3 zu sehen ist.

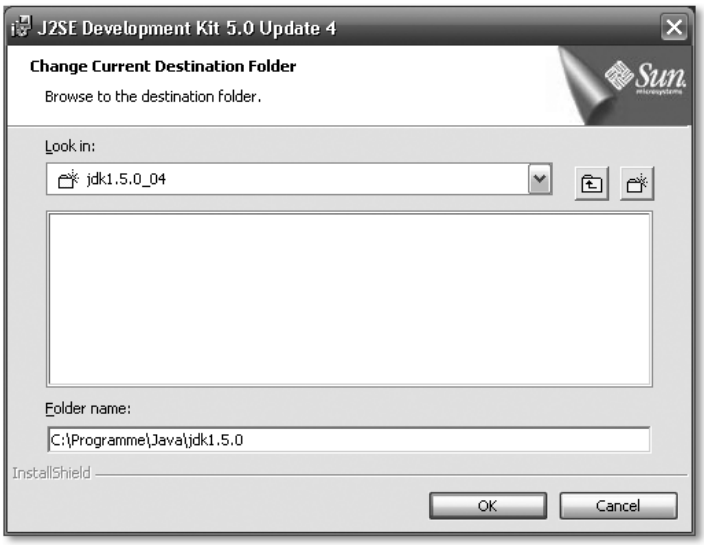

Abbildung 1.3 Änderung des Installationsverzeichnisses

Wenn Sie die Option *Public JRE* nicht deaktiviert haben, wird nach der Installation des Java Development Kits der Installationsassistent für diese zusätzliche Laufzeitumgebung aufgerufen. Auch hier haben Sie einige wenige Einstellungsmöglichkeiten, beispielsweise können Sie die Unterstützung für zusätzliche Sprachen aktivieren und einige weitere Schriftarten installieren. Abbildung 1.4 zeigt den entsprechenden Dialog. Wie schon beim Development Kit sollten Sie auch hier vor dem Anklicken der Schaltfläche **Next** auf **Change** klicken, um das Installationsverzeichnis der Laufzeitumgebung anzupassen.

Im nächsten Schritt des Installationsprozesses, den Sie in Abbildung 1.5 auf Seite 28 sehen, haben Sie die Möglichkeit, die Java-Integration in den Webbrowser zu konfigurieren. Das so genannte *Java Plug-In* ist beispielsweise für die Darstellung von Applets innerhalb von *Internet Explorer*, *Mozilla* und *Netscape* verantwortlich. Je nachdem, welches Programm Sie für den Zugriff auf Webseiten verwenden, sollten Sie das korrespondierende Häkchen auf jeden Fall setzen. Diese Einstellungen können Sie bei Bedarf aber auch nachträglich über das Modul *Java* der *Systemsteuerung* vornehmen.

Nach der Installation der Laufzeitumgebung und des Development Kits ist unter Umständen ein Neustart des Rechners nötig. Vor weiteren Einstellungen rund um Java sollten Sie einer solchen Aufforderung Folge leisten, um Windows die Möglichkeit zu geben, während des Starts neue Komponenten zu registrieren oder Einträge in der Registrierung vorzunehmen.

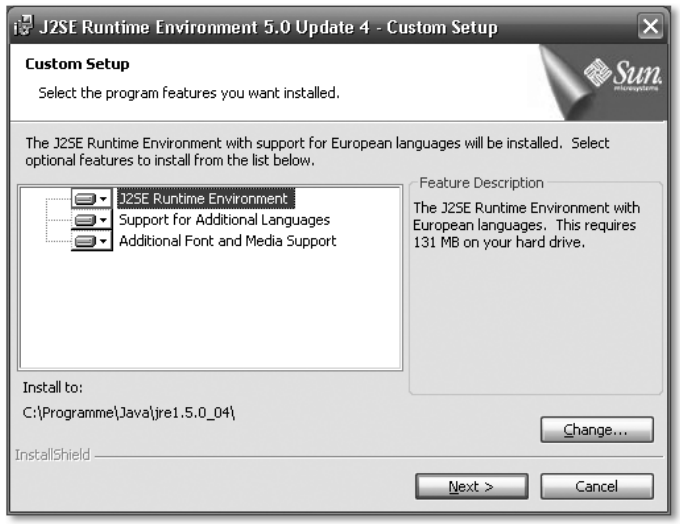

Abbildung 1.4 Einstellungen vor der Installation der JRE

Der nächste Schritt zu einer angenehmen und programmiererfreundlichen Entwicklungsplattform besteht im Herunterladen und Installieren der äußerst umfangreichen Dokumentation zu Java, der Laufzeitumgebung und der Klassenbibliothek. Sun stellt ein entsprechendes Paket kostenlos zum Download bereit.[4]

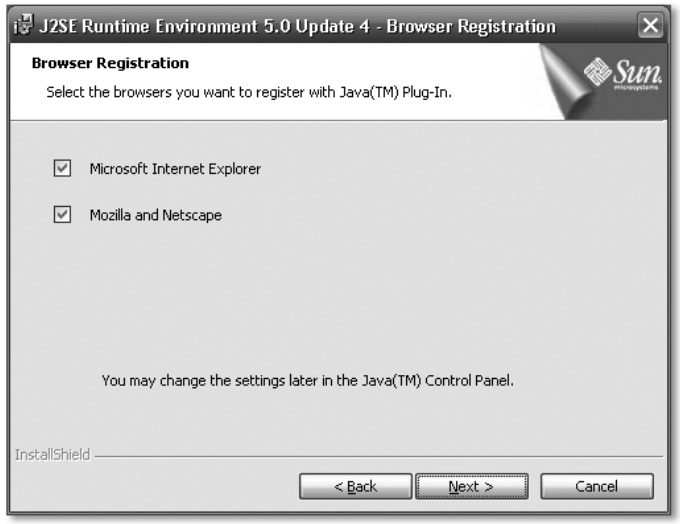

Abbildung 1.5 Einstellung der Java-Integration in den Webbrowser

4 http://java.sun.com/j2se/1.5.0/download.jsp

Wie schon das Entwicklerpaket selbst hat auch die Dokumentation eine beachtliche Größe, die zudem nach dem Entpacken nochmals massiv zunimmt. Deshalb mein Tipp: Anders als bei den Development Kits oder Laufzeitumgebungen ist es meistens nicht erforderlich, mehrere Versionsstände der Dokumentation auf dem Entwicklungsrechner vorzuhalten. Wenn Sie eine aktuelle Fassung installieren, können Sie ältere getrost löschen. Um unnötige Kopieraktionen zu vermeiden, rate ich ferner, als Zielverzeichnis für den Dateidownload gleich das Basisverzeichnis Ihres aktuellsten Java Development Kits anzugeben. Da die unzähligen Dateien der Dokumentation in einem Verzeichnis mit Namen *doc* abgelegt wurden, können Sie das Archiv nach dem Herunterladen mit einem geeigneten Tool gleich an der richtigen Stelle entpacken.

1.1.3 Ergänzende Installationsmaßnahmen

Nach der Installation des *Java Development Kits* und ggf. der *Public JRE* sind einige weitere Einstellungen an Ihrem System vorzunehmen, um die Integration wirklich rund zu machen. Welche dies sind, werde ich Ihnen im Folgenden zeigen. Beginnen möchte ich allerdings mit einer kleinen Bestandsaufnahme, was durch das Setup-Programm standardmäßig eingerichtet wurde.

Viele Java-Anwendungen, die in *.jar*-Archiven ausgeliefert werden, lassen sich mit einem Doppelklick starten. Sie können dies testen, indem Sie eines der mit dem Entwicklerpaket gelieferten Demos aufrufen, beispielsweise *SwingSet2.jar* im Verzeichnis *demo\jfc\SwingSet2* unterhalb des JDK-Basisverzeichnisses. Probleme kann es hierbei geben, weil einige Programme zum Entpacken von Archiven auch den Dateityp *.jar* registrieren. Ist dies der Fall, öffnet sich nach dem Doppelklick auf *SwingSet2.jar* das entsprechende Packprogramm. Gegebenenfalls müssen Sie in der Programmdokumentation nachlesen, wie Sie die Dateizuordnung aufheben. Alternativ können Sie solche Einstellungen aber auch auf der Registerkarte **Dateitypen** des Dialogs **Ordneroptionen** vornehmen, den Sie im *Windows Explorer* über **Extras · Ordneroptionen** erreichen. Wenn der Start durch Doppelklick bei einigen Programmen funktioniert, bei anderen hingegen nicht, ist in der Regel die Datei *Manifest.mf* nicht vorhanden oder beschädigt. Sie gehört zu einem *.jar*-Archiv und enthält unter anderem Informationen darüber, welche Klasse die Methode `main()` enthält. Nähere Informationen hierzu finden Sie in Abschnitt 1.2. Als weiterer Dateityp wird die Endung *.jnlp* mit der Java-Laufzeitumgebung verknüpft. Genauer gesagt startet ein Doppelklick auf solche Dateien die Anwendung *Java Web Start*. Hierbei handelt es sich um einen Programm-Manager, der über das Netz bezogene Java-Programme verwaltet. Anders als Applets werden Web Start-Anwendungen nicht im Kontext des Webbrowsers ausgeführt, sondern verhalten sich wie lokal installierte Programme. Ausführliche Informationen rund um *Java*

Web Start finden Sie in Kapitel 10, *Deployment*. Schließlich sorgt das Java Plug-In unter anderem dafür, dass Applets nicht mit einer möglicherweise noch installierten Microsoft Virtual Machine, sondern mit der Laufzeitumgebung einer aktuellen Java-Version von Sun ausgeführt werden.

Zahlreiche Einstellungen lassen sich mit dem in Abbildung 1.6 gezeigten Modul *Java* der *Systemsteuerung* vornehmen. Dies betrifft beispielsweise die Frage, ob die Netzwerkeinstellungen des Browsers übernommen werden, oder wie groß der Zwischenspeicher für aus dem Internet heruntergeladene Inhalte wie Applets sein soll. Außerdem können Zertifikate verwaltet und Sicherheitsprivilegien sowie Zugriffsrechte festgelegt werden. Besonderes Augenmerk verdient die Konfiguration automatischer Updates. Auf der Registerkarte **Aktualisierung** können Sie festlegen, ob und wann ein Update-Dienst nach Aktualisierungen für Java suchen soll.

Mein Tipp: Lassen Sie diese Option auf jeden Fall aktiviert. Aktuelle Java-Versionen haben sich zwar als sehr verlässlich und stabil erwiesen, dennoch kann es für den Entwickler eine große Arbeitserleichterung bedeuten, über die Verfügbarkeit von Programmaktualisierungen informiert zu werden.

Abbildung 1.6 Die Registerkarte »Allgemein« des Java-Moduls

Die Registerkarte **Erweitert** enthält eine ganze Reihe von Einstellmöglichkeiten, die auch für den Entwickler von Bedeutung sind. Beispielsweise lassen sich verschiedene Stufen von Debug-Ausgaben einschalten, außerdem kann die *Konsole* aktiviert werden. Diese Einstellungen betreffen allerdings nicht die Entwicklung von klassischen Anwendungen, sondern sind vor allem im Kontext der Applet-Entwicklung interessant. So landen Ausgaben, die ein Applet mittels `System.out.println()` ausgibt, in der eben vorgestellten Konsole.

Eine weitergehende Einbettung in das Wirtssystem ist derzeit nicht vorhanden und muss ggf. durch den Entwickler vorbereitet werden. Hierzu zählen unter anderem der Programmaufruf über das Start-Menü, die Einbettung in das *Software*-Modul der *Systemsteuerung* sowie die Registrierung anwendungsspezifischer Dateitypen. Viele dieser Themen greife ich in den folgenden Kapiteln auf.

Aber auch im Hinblick auf den Entwicklungsprozess selbst sind noch einige Vorarbeiten nötig. Praktisch alle modernen Betriebssysteme unterhalten eine Liste von Verzeichnissen, die häufig verwendete ausführbare Dateien enthalten. Solche Programme werden auch ohne Kenntnis ihres vollständigen Pfads gefunden. Dies ist beispielsweise bei der Arbeit mit Kommandozeileninterpretern wichtig. In diesem Fall genügt es, den Namen der zu startenden Anwendung einzugeben, statt einen vollständigen Pfad zu tippen. Windows speichert diese Verzeichnisliste in der Umgebungsvariablen `PATH`. Sie können sich den Wert der Variable anzeigen lassen, indem Sie in der *Eingabeaufforderung* `echo %PATH%` eintippen. Leider fügt die Installationsroutine des Java Development Kits das Verzeichnis mit den wichtigsten Tools wie *javac* oder *jar* `PATH` nicht hinzu. Konsequenterweise müssen Sie in der *Eingabeaufforderung* bei jedem Aufruf eines dieser Programme stets den vollständigen Pfad angeben, eine nervige und zudem fehlerträchtige Angelegenheit. Die Laufzeitumgebung selbst lässt sich hingegen direkt starten. Dies liegt allerdings daran, dass beispielsweise *java* und *javaw* zusätzlich nach *C:\WINDOWS\system32* kopiert werden. Dieses Verzeichnis ist in `PATH` enthalten. Logischerweise führt das Installieren mehrerer Java-Versionen dazu, dass in diesem Verzeichnis bereits vorhandene Java-Dateien überschrieben werden. Welche Version »überlebt«, ist schlicht von der Installationsreihenfolge abhängig. Deshalb mein Tipp: Ignorieren Sie *.exe*-Dateien, die zu einer Java-Installation gehören, wenn diese in *C:\WINDOWS\system32* liegen. Dies gelingt, indem Sie die Umgebungsvariable `PATH` so erweitern, dass das Verzeichnis mit den ausführbaren Java-Tools am besten an erster Stelle, in jedem Fall aber vor *C:\WINDOWS\system32* auftaucht. Öffnen Sie hierzu den Dialog **Systemeigenschaften**, den Sie über den Menüpunkt **Eigenschaften** des *Arbeitsplatz*-Kontextmenüs erreichen. Wechseln Sie bitte auf die Registerkarte **Erweitert** und klicken auf die Schaltfläche **Umgebungsvariablen**, woraufhin sich der in Abbildung 1.7 gezeigte Dialog öffnet. Bitte

suchen Sie in der Liste **Systemvariablen** nach `PATH`, markieren die Zeile und klicken auf **Bearbeiten**.

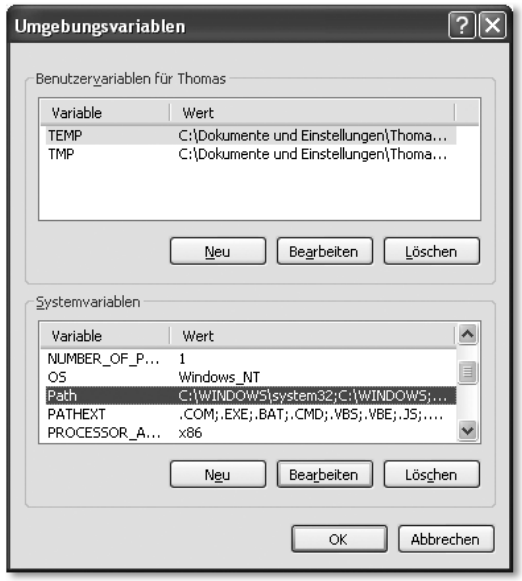

Abbildung 1.7 Der Dialog Umgebungsvariablen

Es wird ein weiterer Dialog geöffnet, der in Abbildung 1.8 zu sehen ist. In der Eingabezeile **Wert der Variablen** befinden sich Pfade, die durch den Strichpunkt voneinander getrennt sind. Diese Liste müssen Sie um das Verzeichnis erweitern, das die Tools des JDK enthält. Positionieren Sie hierzu den Cursor vor dem allerersten Zeichen und tippen dann `%JAVA_BIN%;`. Wenn Sie den Namen einer Umgebungsvariablen mit dem Prozentzeichen klammern, wird der Inhalt dieser Variablen eingesetzt. Sie erweitern also die Variable `PATH` um den Inhalt von `JAVA_BIN`. Noch existiert diese Umgebungsvariable nicht, deshalb werden wir sie gleich anlegen. Beenden Sie nun den Dialog **Systemvariable bearbeiten** durch Anklicken der Schaltfläche **OK**.

Abbildung 1.8 Der Dialog »Systemvariable bearbeiten«

Legen Sie eine neue Umgebungsvariable an, indem Sie auf die Schaltfläche **Neu** unterhalb der Liste der Systemvariablen klicken. Es öffnet sich der Dialog **Neue Systemvariable**, der in Abbildung 1.9 zu sehen ist.

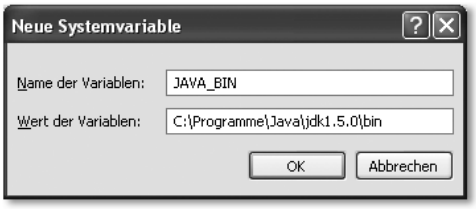

Abbildung 1.9 Der Dialog »Neue Systemvariable«

Tragen Sie bitte in der Eingabezeile **Name der Variablen** den Text `JAVA_BIN` ein. Welchen Pfad Sie bei **Wert der Variablen** eintragen, ist davon abhängig, welche Angaben Sie bei der Installation des Development Kits gemacht haben. Generell gilt: Die Java-Tools liegen im Verzeichnis *bin* unterhalb des Basisverzeichnisses einer Installation, beispielsweise *C:\Programme\Java\jdk1.5.0\bin*. Schließen Sie nun alle geöffneten Dialoge durch Anklicken der Schaltflächen **OK**. Sie können Ihre Einstellungen testen, indem Sie die *Eingabeaufforderung* öffnen und die in Abbildung 1.10 gezeigten Befehle eingeben.

Die Änderungen an der `PATH`-Umgebungsvariable gestatten ein schnelles und effizientes Arbeiten in der *Eingabeaufforderung*. Auch wenn Sie die meisten Ihrer Projekte in einer integrierten Entwicklungsumgebung verwalten, ist es oftmals schneller, kleine Hacks mit einem Texteditor zu erfassen und sie direkt zu übersetzen und auszuführen, anstatt erst ein Projekt anlegen zu müssen.

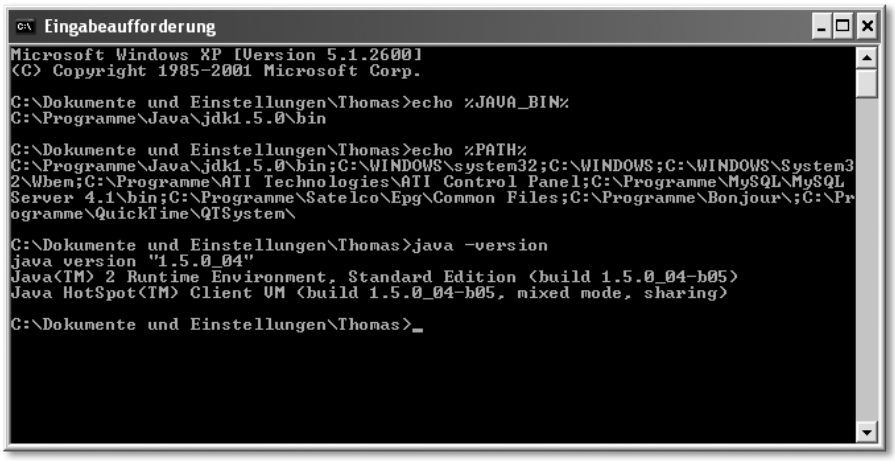

Abbildung 1.10 Tests, ob die Java-Tools gefunden werden

Damit sind die Installations- und Konfigurationsarbeiten abgeschlossen, soweit sie das Java Development Kit betreffen. Im nächsten Abschnitt beschäftige ich mich mit der Frage, wie Sie Java-Erweiterungen und Klassenbibliotheken von Drittherstellern installieren und verwalten.

1.2 Zusätzliche Bibliotheken

Teil der Java-Laufzeitumgebung ist eine äußerst umfangreiche Sammlung von Kernklassen. Als fester Bestandteil jeder Installation steht ihr Funktionsumfang allen Anwendungen zur Verfügung. Wenn man das erste Mal versucht, sich einen Überblick über die gebotenen Dienste zu verschaffen, mag der Eindruck entstehen, es gäbe kein Problem, das sich nicht mit Hilfe der angebotenen Klassen lösen lässt. Allerdings stelle ich Ihnen in diesem Buch eine ganze Reihe von Klassensammlungen vor, die fehlende Funktionen nachrüsten oder zusätzliche neue Möglichkeiten eröffnen. In der Tat kann keine Klassenbibliothek jemals vollständig sein (und soll es ja auch gar nicht). Wichtig ist in diesem Zusammenhang nur, inwieweit Java Mechanismen für die Erweiterung seines Funktionsumfangs zur Verfügung stellt bzw. wie Anwendungen auf die Klassen in solchen Bibliotheken zugreifen können. Hier gibt es zwei mögliche Vorgehensweisen, die ich Ihnen im Folgenden erläutere.

1.2.1 Der Klassenpfad

Java-Programme bestehen aus Klassen sowie weiteren Ressourcen, beispielsweise Text-, Grafik- oder Audiodaten. Damit die Laufzeitumgebung solche Elemente findet, muss sie wissen, wo die Dateien, die sie enthalten, gespeichert sind. Java verwaltet hierzu eine Liste mit Verzeichnissen sowie *.zip*- und *.jar*-Dateien, die *Klassenpfad* genannt wird. Dabei besteht dieser Klassenpfad aus drei individuellen Suchpfaden, die unabhängig voneinander modifiziert werden können. Der *Systemklassenpfad* verweist auf die bereits angesprochenen Kernklassen, die Teil der Laufzeitumgebung sind. Der *Erweiterungsklassenpfad* zeigt auf Klassenbibliotheken, die Java um zusätzliche Funktionen erweitern. Der *Anwenderklassenpfad* schließlich enthält alle übrigen benutzerdefinierten Klassen und Ressourcen. Übrigens wird der Begriff *Klassenpfad* oftmals verallgemeinernd für den *Anwenderklassenpfad* verwendet. Auch wenn es technisch möglich ist, besteht im Allgemeinen keine Notwendigkeit, den Systemklassenpfad oder den Erweiterungsklassenpfad zu verändern. Die Laufzeitumgebung aktueller Java-Versionen weiß, wo sie nach Klassen der drei genannten Gruppen suchen muss.[5] Aus diesem Grund ist auch die aus frühen Jahren bekannte Umgebungsvariable `CLASSPATH` nicht länger erforderlich. Sun schlägt daher

5 http://java.sun.com/j2se/1.5.0/docs/tooldocs/findingclasses.html

vor, sie nicht weiter zu verwenden, sofern sie von Anwendungen nicht explizit gefordert wird.[6] Der Benutzerklassenpfad umfasst standardmäßig das so genannte *aktuelle Verzeichnis*, das beim Aufruf der Laufzeitumgebung beliebig gesetzt werden kann. Einfacher ist allerdings die Anpassung des Benutzerklassenpfads selbst. Sie kann auf verschiedene Weise erfolgen. Häufig Verwendung findet die Option `-classpath` von *java.exe* und *javaw.exe*. Ihr folgt eine durch Strichpunkt getrennte Liste von Verzeichnissen sowie *.zip-* und *.jar*-Archiven. Wichtig ist hierbei, dass die Liste vollständig ist. Anwenderklassen werden ausschließlich in den angegebenen Orten gesucht. Eine weitere Möglichkeit, den Benutzerklassenpfad für eine Anwendung zu spezifizieren, ist es, die Liste der zu durchsuchenden Verzeichnisse und Archive durch Leerzeichen getrennt im Attribut `Class-Path:` der so genannten Manifest-Datei einer Anwendung einzutragen. Diese trägt den Namen *Manifest.mf* und wird im Verzeichnis *meta-inf* des *.jar*-Archivs erwartet. Ein weiteres Attribut, `Main-Class:`, enthält übrigens den voll qualifizierten Namen der Startklasse, die folglich eine `main`-Methode mit passender Signatur enthalten muss.

Damit Ihre Java-Anwendung die Klassen der Bibliothek eines Drittanbieters findet, können Sie also den Anwenderklassenpfad entsprechend erweitern. Sie müssen hierzu entweder Manifest-Dateien bearbeiten oder etwaige Startskripte anpassen, die *java.exe* bzw. *javaw.exe* mit geeigneten `-classpath`-Optionen versorgen. Eine weitaus elegantere Möglichkeit bietet der *Erweiterungsklassenpfad*, den ich Ihnen nun vorstelle.

1.2.2 Das Java-Erweiterungsverzeichnis

Der *Erweiterungsklassenpfad* ist einer von drei Suchpfaden, die zusammen den Klassenpfad bilden. Er verweist auf ein oder mehrere Verzeichnisse, in denen *optionale Pakete* abgelegt werden. Hierbei handelt es sich um Klassenbibliotheken, die den Funktionsumfang von Java erweitern. Optionale Pakete müssen hierzu keine besonderen Konventionen erfüllen. Die einzige Voraussetzung ist, dass die Klassen in einem *.jar*-Archiv gebündelt und in einem der so genannten Erweiterungsverzeichnisse abgelegt werden. Die Liste dieser Verzeichnisse kann über die System-Property `java.ext.dirs` abgefragt und bei Bedarf auch verändert werden. Standardmäßig wird *lib\ext* unterhalb des Java-Heimatverzeichnisses verwendet. Das Java-Heimatverzeichnis ist das Basisverzeichnis einer Java-Laufzeitumgebung, nicht jedoch einer Java-Installation. Es kann über die System-Property `java.home` abgefragt werden. Wenn Sie ein Entwicklerpaket beispielsweise in *C:\Programme\Java\jdk1.5.0* installiert haben, ist nicht dieses Verzeichnis das Java-Heimatverzeichnis, sondern dessen Unterver-

6 http://java.sun.com/j2se/1.5.0/docs/tooldocs/windows/classpath.html

zeichnis *jre*, also *C:\Programme\Java\jdk1.5.0\jre*. Aus frühen Java-Tagen stammt die Umgebungsvariable `JAVA_HOME`, die ebenfalls auf das Java-Heimatverzeichnis verweist. Moderne Java-Installationen kommen allerdings ohne sie aus, deshalb rate ich Ihnen, die Variable nur dann zu setzen, wenn eine von Ihnen verwendete Anwendung oder Klassenbibliothek dies ausdrücklich fordert.

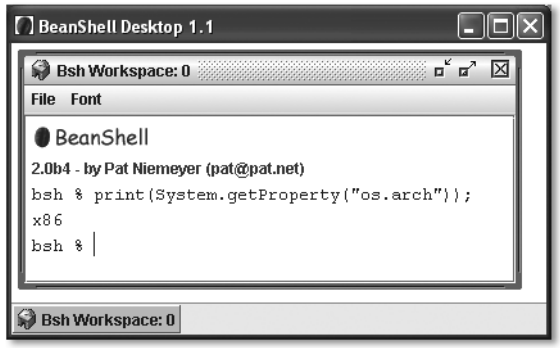

Abbildung 1.11 Die BeanShell als interaktiver Java-Interpreter

Um also allen Anwendungen, die durch eine Java-Laufzeitumgebung ausgeführt werden, die Verwendung der Klassenbibliothek eines Drittanbieters zu ermöglichen, müssen nur alle *.jar*-Archive der Bibliothek in ein Java-Erweiterungsverzeichnis kopiert werden, standardmäßig nach *<java.home>\lib\ext*. Wenn zu der Klassenbibliothek auch dynamisch gelinkte Bibliotheken oder ausführbare Programme gehören, muss sichergestellt sein, dass die Java-Laufzeitumgebung diese Komponenten findet. Deshalb gibt es auch für sie Standard-Verzeichnisse, beispielsweise *<java.home>\bin*. Besser geeignet ist aber ein Unterverzeichnis von *lib\ext*. Als Name wird der Wert der System-Property `os.arch` erwartet: *x86*. Abbildung 1.11 zeigt, wie Sie mit Hilfe der *BeanShell*, die ich Ihnen in Kapitel 2, *Entwicklungswerkzeuge*, ausführlicher vorstelle, solche Werte sehr schnell ermitteln können.

> Noch ein genereller Tipp: Folgen Sie niemals Ihren Annahmen über Dateinamen oder Verzeichnisse, ermitteln Sie die Werte stets mit den entsprechenden System-Properties.

Wenn Sie Klassenbibliotheken innerhalb einer Java-Laufzeitumgebung zur Verfügung stellen möchten, bieten sich Erweiterungsverzeichnisse aufgrund der äußerst einfachen Handhabung an. Außer dem Kopieren der Dateien sind keine weiteren Konfigurationsarbeiten notwendig. Zudem können sie über

System-Properties abgefragt werden und lassen sich deshalb sehr einfach in Installationsroutinen einbinden.

Der folgende Abschnitt beschäftigt sich mit Problemen, die sich aus der parallelen Nutzung mehrerer Java-Installationen ergeben können und präsentiert entsprechende Lösungsmöglichkeiten.

1.3 Parallele Nutzung mehrerer Java-Installationen

Aus der Sicht des Software-Entwicklers wäre es am bequemsten, nicht mehrere Java-Versionen auf dem Entwicklungsrechner vorhalten zu müssen. Zwar ist der Aspekt »Speicherplatz« in den letzten Jahren in den Hintergrund getreten, aber es erfordert zunehmend mehr Aufwand, die installierte Software auf dem neuesten Stand zu halten. Je weniger Programme installiert sind, umso leichter fällt die Wartung. Allerdings kann es notwendig sein, Komponenten ganz bewusst nicht zu aktualisieren, um das Verhalten der eigenen Anwendung auf »veralteten« Systemen zu simulieren. Zudem haben Sie als Entwickler sicherlich den Wunsch, die Lauffähigkeit Ihrer Programme zumindest unter verbreiteten Java-Versionen zu gewährleisten. Aber wie können Sie festlegen, wann eine bestimmte Version der Java-Laufzeitumgebung verwendet wird?

1.3.1 Wahl einer bestimmten Java-Version

Grundsätzlich ist die Installation und Nutzung unterschiedlicher Java-Versionen problemlos. In diesem Zusammenhang möchte ich nochmals auf die Anregungen aus Abschnitt 1.1.1 aufmerksam machen, beispielsweise sollten Sie, wenn möglich, alte Java-Versionen zuerst installieren. Denken Sie bitte auch daran, nur mit Programmen aus den Java-Installationsverzeichnissen zu arbeiten, weil aufeinander folgende Installationen unter Umständen Java-Dateien in System-Verzeichnissen überschreiben können. Die Installationsroutine der *Public JRE* registriert – wie Sie aus Abschnitt 1.1.3 bereits wissen – bestimmte Dateitypen. Folglich wird beispielsweise nach einem Doppelklick auf ein *.jar*-Archiv die zuletzt installierte Laufzeitumgebung verwendet.

Möchten Sie diese Zuordnung ändern, müssen Sie im in Abbildung 1.12 gezeigten Dialog **Ordneroptionen** des *Windows Explorer* auf der Registerkarte **Dateitypen** selbst die entsprechenden Einstellungen vornehmen. Wenn Sie Java-Programme häufiger mit unterschiedlichen Versionen der Laufzeitumgebung ausführen möchten, kann es sinnvoll sein, in diesem Dialog zusätzliche Aktionen zu definieren, etwa **Starte mit Java 1.4.2**. Markieren Sie hierzu in der Liste der registrierten Dateitypen den Dateityp *.jar* und klicken Sie anschließend auf die Schaltfläche **Erweitert**. Es öffnet sich der Dialog **Dateityp**

bearbeiten, der unter **Aktionen** nur *open* zeigt. Klicken Sie bitte auf dessen Schaltfläche **Neu**, woraufhin sich der Dialog **Neue Aktion** öffnet (Abbildung 1.13).

Abbildung 1.12 Der Dialog »Ordneroptionen«

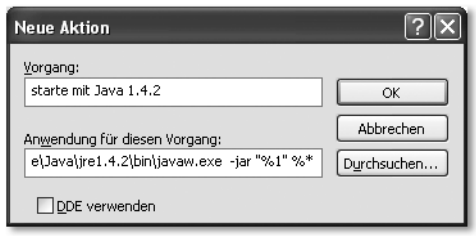

Abbildung 1.13 Der Dialog »Neue Aktion«

Unter **Vorgang** können Sie einen beliebigen Text eintragen, der die auszuführende Aktion möglichst knapp beschreiben sollte, beispielsweise *starte mit Java 1.4.2*. Die Eingabezeile **Anwendung für diesen Vorgang** nimmt das zu startende Programm sowie alle Parameter auf, die beim Start übergeben werden. Klicken Sie bitte auf **Durchsuchen**, um *javaw.exe* der gewünschten Laufzeitumgebung zu lokalisieren. Fügen Sie an den nun eingetragenen vollständigen Pfad nach einem Leerzeichen den in Abbildung 1.13 gezeigten Text an und beenden Sie den Dialog mit **OK**. Die neue Aktion erscheint im Dialog **Dateityp bearbeiten**, den Sie ebenfalls mit **OK** schließen können. Um sie auszuführen, klicken

Sie bitte mit der rechten Maustaste auf ein *.jar*-Archiv, um dessen Kontextmenü aufzurufen. Dort können Sie den entsprechenden Eintrag auswählen. Selbstverständlich ist die beschriebene Vorgehensweise nicht auf eine zusätzliche Laufzeitumgebung beschränkt. Bei Bedarf können Sie beliebig viele zusätzliche Aktionen eintragen.

Im Hinblick auf Applets und Java Web Start-Anwendungen ist auch die Frage interessant, welche Laufzeitumgebung für die Ausführung verwendet wird. In diesem Punkt zeigt sich Java komfortabel und bietet entsprechende Einstellmöglichkeiten auf der Registerkarte **Java** des Moduls *Java* der *Systemsteuerung* an. In Abbildung 1.14 sehen Sie den Dialog **Einstellungen für JNLP Runtime**, in dem Sie die Laufzeitumgebung für Web Start-Anwendungen bestimmen können.

Welche Version des Java-Compilers beim Aufruf aus der Eingabeaufforderung verwendet wird, können Sie durch Setzen der Umgebungsvariable `JAVA_BIN` selbst steuern. Wie Sie diese Variable setzen, zeige ich Ihnen in Abschnitt 1.1.3.

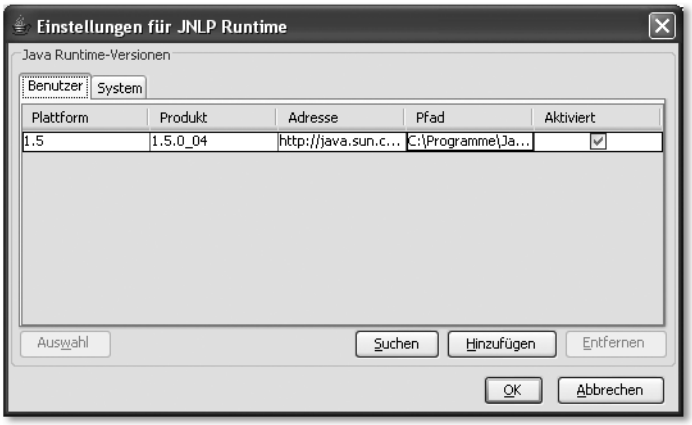

Abbildung 1.14 Der Dialog »Einstellungen für JNLP Runtime«

Nachdem ich Ihnen dargelegt habe, wie Sie die Java-Version, die zum Einsatz kommen soll, festlegen können, möchte ich im Folgenden einige Konsequenzen ansprechen, die sich aus der parallelen Installation und Nutzung unterschiedlicher Java-Versionen ergeben.

1.3.2 Konsequenzen aus der Nutzung mehrerer Java-Versionen

Jede Java-Installation ist mit Ausnahme einiger weniger Dateien, die in System-Verzeichnissen abgelegt werden, in sich geschlossen. Zahlreiche Konfigurationsdateien werden nicht im Heimatverzeichnis des aktuellen Benutzers, sondern innerhalb der Verzeichnishierarchie einer Installation abgelegt. Ein Beispiel

ist die Datei *swing.properties*, die Ihnen ausführlich in Kapitel 3, *Feintuning der Benutzeroberfläche*, vorgestellt wird. Sie enthält unter anderem Einstellungen, die das Aussehen von Swing-Komponenten beeinflussen. Da sich unterschiedliche Java-Versionen diese Datei nicht »teilen«, beziehen sich etwaige Änderungen stets auf genau eine Installation. Für Sie als Entwickler bedeutet dies unter Umständen einen erhöhten Wartungsaufwand, weil Sie Änderungen an solchen Konfigurationsdateien in allen Installationen nachvollziehen müssen.

Ähnlich verhält es sich mit den Erweiterungsverzeichnissen, die ich Ihnen in Abschnitt 1.2.2 vorgestellt habe. Auch sie sind nur für eine Java-Installation gültig. Um eine bestimmte Bibliothek in allen Installationen verfügbar zu haben, müssen Sie die zu ihr gehörenden Dateien in die Erweiterungsverzeichnisse aller Laufzeitumgebungen kopieren. Grundsätzlich stellt sich zwar die Frage, ob es überhaupt notwendig ist, jede Einstellung und jede Bibliothek auf alle installierten Java-Versionen zu übertragen, allerdings bietet beispielsweise die Java-Implementierung in Apples Mac OS X ein für alle Java-Installationen gültiges Erweiterungsverzeichnis an (und demonstriert damit die prinzipielle Machbarkeit). Deshalb zeige ich Ihnen im nächsten Abschnitt, wie Sie eine vergleichbare Funktionalität auch unter Windows realisieren können.

1.3.3 Gemeinsame Verwendung von Bibliotheken

Die angesprochene Implementierung eines gemeinsam genutzten Erweiterungsverzeichnisses lässt sich auf verschiedene Weise realisieren. Da sich beim Start der Java-Laufzeitumgebung durch die Option `-D` System-Properties setzen lassen, können Sie die in `java.ext.dirs` eingetragene Liste der gültigen Erweiterungsverzeichnisse einfach erweitern. Allerdings müssen Sie die entsprechende Option bei jedem Start einer Java-Anwendung angeben.

Bequemer wäre es, wenn die Erweiterungsverzeichnisse der verschiedenen Laufzeitumgebungen physikalisch auf ein Verzeichnis verweisen würden. Unter UNIX-artigen Betriebssystemen gibt es das Konzept der *symbolischen Links*, die genau dies leisten. Wenn Sie einen solchen symbolischen Link anlegen, teilen Sie dem Dateisystem mit, auf welche Datei oder Verzeichnis dieser Link zeigen soll. Bei einem Zugriff auf den Link verzweigt das System automatisch zu dessen Ziel. Was auf den ersten Blick aussieht wie diejenigen Verknüpfungen, die Sie unter Windows anlegen können, wird technisch gesehen grundlegend anders realisiert. Unter UNIX-artigen Systemen arbeiten Links auf der Ebene der Dateisysteme, wohingegen Windows-Verknüpfungen im Prinzip normale Datendateien sind. Die Auswertung der Verknüpfung geschieht nicht durch das Betriebssystem. Deshalb können sie für das Zusammenführen von Erweiterungsverzeichnissen nicht verwendet werden. Glücklicherweise kennt

aber das Dateisystem *NTFS* die so genannten *Junctions*[7], die einen mit symbolischen Links vergleichbaren Mechanismus implementieren. Junctions basieren auf den *Reparse Points*. Diese werden ebenfalls in Microsofts Platform SDK beschrieben.[8] Leider enthalten aktuelle Versionen von Windows keine Werkzeuge zum Anlegen von Junctions. Diese Lücke füllt ein kleines Freeware-Tool von Mark Russinovich, das Sie von der Seite **Sysinternals.com**[9] herunterladen können. Die zum Zeitpunkt der Drucklegung aktuelle Version finden Sie auch auf der Begleit-CD zum Buch. Nach dem Entpacken des Archivs ist das Programm ohne weitere Installation sofort einsatzbereit. Als klassisches Kommandozeilen-Tool rufen Sie es am besten aus der *Eingabeaufforderung* heraus auf. Um eine *Junction* anzulegen, übergeben Sie zwei Parameter, den Ursprung einer Verzweigung sowie deren Ziel. Mit Hilfe der Option `-d` können Sie eine nicht mehr benötigte Verzweigung löschen. Um ein von mehreren Laufzeitumgebungen genutztes Erweiterungsverzeichnis anzulegen, gehen Sie bitte wie folgt vor. Zunächst müssen Sie ein neues Verzeichnis anlegen, in das Sie später alle Klassenbibliotheken kopieren werden. Es bietet sich an, dieses Verzeichnis im Basisverzeichnis Ihrer Java-Installationen anzulegen, beispielsweise *C:\Programme\Java\Erweiterungen*. Als Nächstes benennen Sie das existierende Erweiterungsverzeichnis einer Installation um. Bitte löschen Sie es nicht, sondern geben ihm nur einen neuen Namen. Kopieren Sie dann den Inhalt dieses Verzeichnisses in das neu angelegte gemeinsame Erweiterungsverzeichnis. Der letzte Schritt besteht im Anlegen der Verzweigung mittels *junction.exe*. Geben Sie als ersten Parameter ihren Ursprung an, also den nun nicht mehr existierenden Ordner *ext*. Der zweite anzugebende Parameter ist das neu angelegte gemeinsame Erweiterungsverzeichnis. Jede Klassenbibliothek, die Sie in dieses Verzeichnis kopieren, wird nun von denjenigen Java-Installationen gefunden, für die Sie entsprechende Verzweigungen angelegt haben.

Gemeinsame Erweiterungsverzeichnisse erleichtern die Handhabung von Klassenbibliotheken bei gleichzeitiger Verwendung unterschiedlicher Java-Installationen. Bedenken Sie aber bitte, dass die Java-Laufzeitumgebung nicht damit rechnet, in physikalisch anderen Verzeichnissen zu operieren. Deshalb sollten Sie vor Änderungen an einer Installation oder vor dem Entfernen der Software etwaige Verzweigungen löschen und die umbenannten Originale der Erweiterungsverzeichnisse wiederherstellen.

7 http://msdn.microsoft.com/library/default.asp?url=/library/en-us/fileio/fs/hard_links_and_junctions.asp
8 http://msdn.microsoft.com/library/default.asp?url=/library/en-us/fileio/fs/reparse_points.asp
9 http://www.sysinternals.com/Utilities/Junction.html

2 Entwicklungswerkzeuge

2 Entwicklungswerkzeuge

Erst die richtige Auswahl an Werkzeugen macht Ihre Entwicklungsplattform wirklich komplett. Deshalb stellt Ihnen dieses Kapitel neben Java-Klassikern auch zahlreiche noch eher unbekannte Perlen vor, die Ihnen den Programmieralltag erleichtern.

Das *Java Development Kit*, dessen Installation und Konfiguration in Kapitel 1 beschrieben wird, ist eine sehr wichtige Säule im Entwicklungsprozess. Für eine effiziente und angenehme Programmierung benötigen Sie allerdings weitere Werkzeuge, beispielsweise einen Editor, mit dem Sie Ihre Quelltexte verfassen, Tools zur Performance-Messung und Code-Optimierung sowie Modellierungswerkzeuge und Dokumentationsgeneratoren. Diese stelle ich im Verlauf des Kapitels vor und zeige Ihnen, wie Sie zusätzliche Tools mit den Kernkomponenten verzahnen und sich so eine leistungsfähige, bequem zu bedienende Entwicklungsplattform schaffen.

2.1 Text- und Programmeditoren

Das *Java Development Kit* bringt im Prinzip alles mit, um Anwendungen zu erstellen. Kleine Programme, die nur aus sehr wenigen Klassen bestehen, lassen sich nämlich selbst mit dem in Abbildung 2.1 gezeigten Notepad erfassen, das im Unterverzeichnis *demo\jfc\Notepad* einer JDK-Installation zu finden ist. Allerdings bietet diese Demoanwendung keinerlei Komfort, sodass sie niemand ernstlich als Entwicklungstool in Erwägung ziehen wird. Grundsätzlich hat aber die Klassenerstellung mit Hilfe eines Texteditors und deren Übersetzung und Ausführung in der *Eingabeaufforderung* sehr wohl ihre Berechtigung, da hierbei keinerlei Vorarbeiten, beispielsweise das Erstellen von Projekten, notwendig sind. Geht es um das Testen eines Codefragments oder das Implementieren eines »schnellen Hacks«, ist der Start einer mächtigen integrierten Entwicklungsumgebung in den meisten Fällen zu zeitaufwändig. Deshalb gehört ein leistungsfähiger, schnell zur Verfügung stehender Texteditor zu einer Entwicklungsplattform in jedem Fall dazu.

Die Wahl des richtigen Editors ist für viele Programmierer eine Glaubensfrage. Entsprechende Diskussionen werden eher mit religiösem Eifer als mit sachlichen Argumenten geführt. Was als wichtige oder vollkommen unnötige Funktion angesehen wird, ist in diesem Zusammenhang ebenso von individuellen Vorlieben abhängig wie die seit *WordStar*-Tagen diskutierte Frage, welche Tastenkürzel welche Aktionen auslösen. Ich stelle Ihnen im Folgenden zwei ausgefeilte Text-

editoren vor: *Notepad++* und *jEdit*. Die beiden Anwendungen verfolgen insofern unterschiedliche Ansätze, als *jEdit* durch eine Vielzahl von Plugins erweiterbar ist, wohingegen *Notepad++* durch seine äußerst geringe Größe positiv auffällt.

Abbildung 2.1 Die JDK-Demoanwendung Notepad

2.1.1 Notepad++

Notepad++ ist eine äußerst schlanke, in C++ geschriebene Win32-Anwendung. Der Open Source-Editor kann von der Projekt-Homepage[1] heruntergeladen werden. Außerdem finden Sie die zum Zeitpunkt der Drucklegung aktuelle Version auf der Begleit-CD zum Buch. Die Installation ist selbsterklärend und in wenigen Minuten abgeschlossen. Nach dem ersten Programmstart sehen Sie einen leeren Eingabebereich. Vorhandene Dateien können Sie bequem durch Ziehen ihrer Symbole in das Programmfenster öffnen. Standardmäßig registriert sich die Anwendung nicht für bestimmte Dateitypen, allerdings können Sie solche Zuordnungen im *Windows Explorer* leicht selbst vornehmen. *Notepad++* verwaltet mehrere gleichzeitig geöffnete Dokumente als separate Registerkarten, zwischen denen Sie bequem durch Anklicken des Kartenreiters oder durch Drücken von Strg+Tab umschalten können. Ein Klick mit der rechten Maustaste öffnet ein Kontextmenü, das häufig verwendete Funktionen enthält. Diese Form der Dokumentenverwaltung hat durch ihre Verwendung in bekannten Webbrowsern stark an Popularität gewonnen und in vielen anderen Anwendungen das Konzept mehrerer gleichzeitig geöffneter Fenster abgelöst. Wie Sie in Abbildung 2.2 sehen, beinhaltet der Editor unter anderem auch die als *Syntax Highlighting* bekannte Funktion, Sprachkonstrukte und Schlüsselwörter von Programmiersprachen farblich zu kennzeichnen.

1 http://notepad-plus.sourceforge.net/

Neben Java erkennt *Notepad++* unter anderem auch *Perl*, *C/C++*, *HTML*, *Pascal* und *TeX*. Die für eine Sprache verwendeten Stile können Sie nach Belieben anpassen.

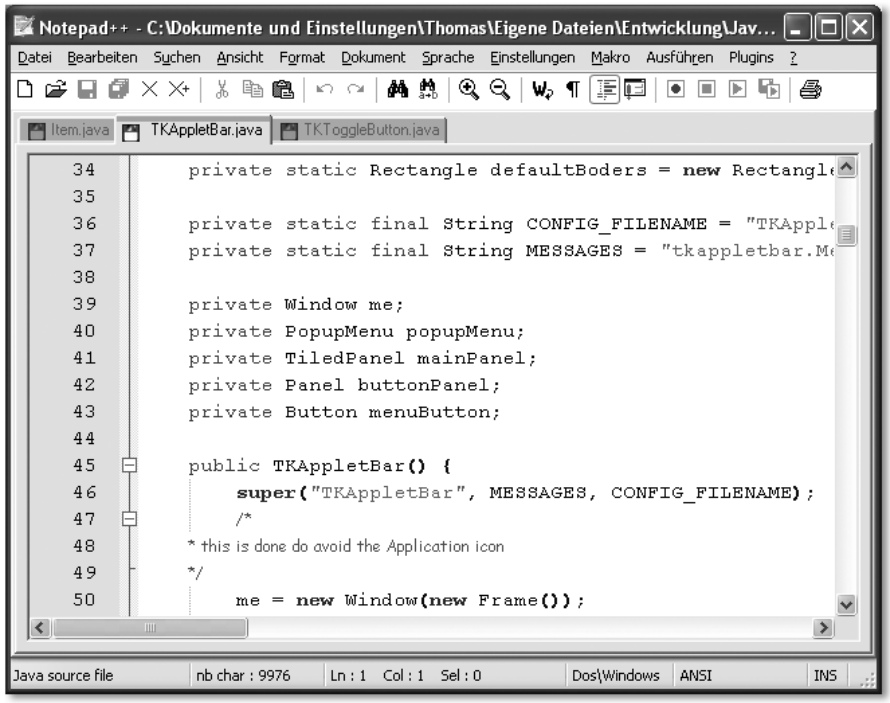

Abbildung 2.2 Das Hauptfenster von Notepad++

Äußerst interessant ist die Funktion, ein Dokument in mehreren Ansichten zu bearbeiten. Eine physikalische Datei ist hierbei in mehreren Registerkarten gleichzeitig zu sehen. Änderungen wirken dabei sofort auf alle »Kopien«. Wenn Sie häufig innerhalb eines großen Dokuments navigieren, kann Ihnen das Öffnen einer zusätzlichen Ansicht unter Umständen viel Blättern ersparen. *Notepad++* bietet eine Reihe weiterer nützlicher Funktionen, die sich dank einer aufgeräumten Oberfläche schnell erschließen. Erwähnenswert ist neben dem Makrorecorder, mit dem Sie Befehlsfolgen aufzeichnen können, beispielsweise noch das *Code Folding*, also das Platz sparende Einklappen von Teilen des Quelltextes.

Im Folgenden möchte ich Ihnen einen weiteren klassischen Text- und Programmeditor vorstellen: *jEdit*. Anders als *Notepad++* wurde er allerdings in Java geschrieben. Dies macht ihn insbesondere für Entwickler interessant, die unter verschiedenen Betriebssystemen programmieren müssen, aber gern eine möglichst einheitliche Entwicklungsumgebung vorfinden möchten.

2.1.2 jEdit

Auch *jEdit* ist Open Source und kann kostenlos von der Projekt-Homepage[2] bezogen werden. Sie finden die derzeit aktuelle Version auch auf der Begleit-CD. Die Installation erfordert nur sehr wenige Benutzereingaben und ist in wenigen Minuten beendet. Denken Sie allerdings bitte daran, vor dem Start von *jedit42install.exe* die Installation und Konfiguration zumindest einer aktuellen Java-Laufzeitumgebung abgeschlossen zu haben.

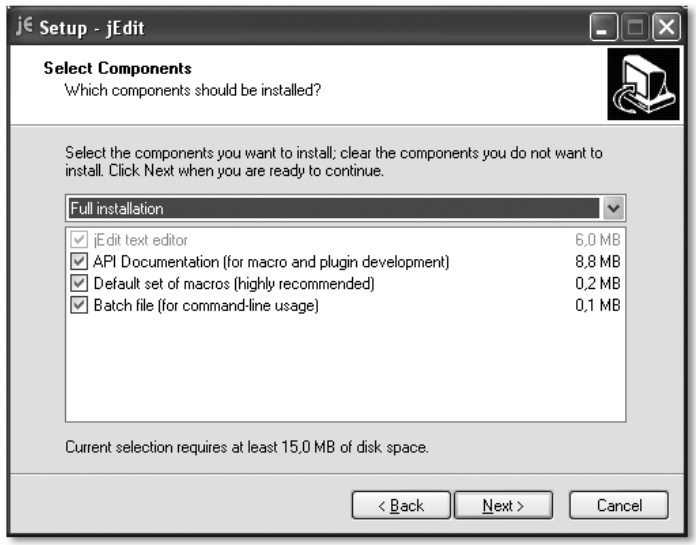

Abbildung 2.3 Die Installation von jEdit

Nach dem ersten Programmstart öffnet sich die Dokumentation zu *jEdit*. Das in Abbildung 2.4 gezeigte Hauptfenster enthält über dem Eingabebereich eine Klappbox, die die Liste aller geöffneten Dokumente beinhaltet. *jEdit* zeigt stets nur einen Text an, sodass nahezu das gesamte Programmfenster für die Eingabe zur Verfügung steht. Wenn Ihnen die in *Notepad++* verwendeten Registerkarten gefallen haben, wird es Sie freuen, dass *jEdit* ein vergleichbares Plugin bietet, das Sie anstelle der eben angesprochenen Klappbox verwenden können.

Starten Sie hierzu den *Plugin Manager*, den Sie über **Plugins · Plugin Manager** erreichen. Wechseln Sie bitte auf die Registerkarte **Install** und wählen die **Buffer Tabs** aus der Kategorie **File Management**. Durch Anklicken der Schaltfläche **Install** werden alle markierten Plugins geladen und auf der Benutzerebene installiert.

2 http://www.jedit.org/

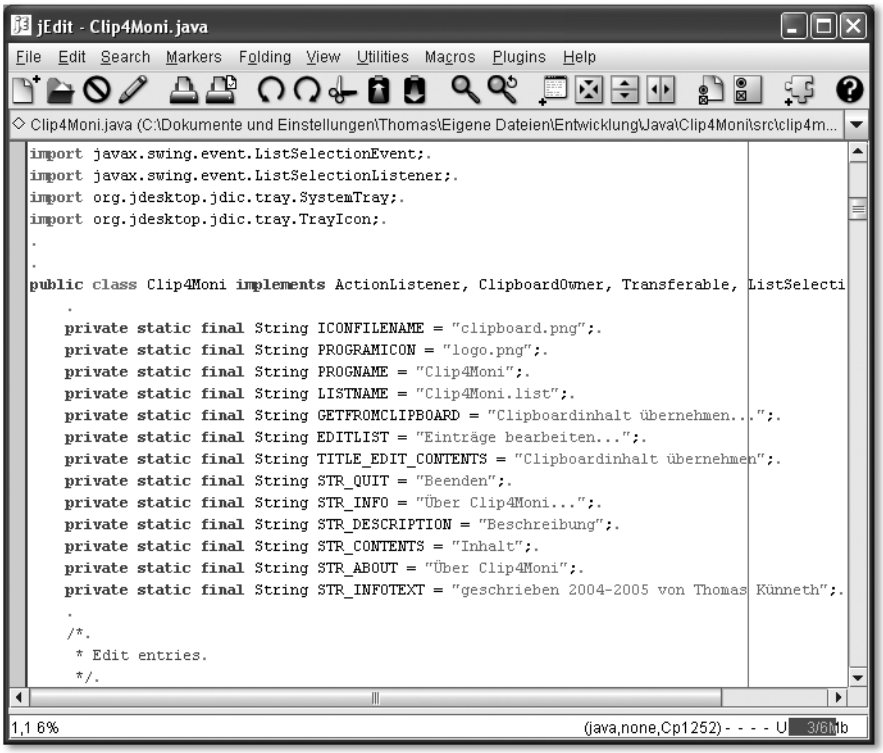

Abbildung 2.4 Das jEdit-Programmfenster

Wenn Sie möchten, dass Module unter allen Benutzerkonten zur Verfügung stehen, klicken Sie bitte auf die Schaltfläche **Download Options** und wählen für die Option *Install plugins in* **jEdit application directory**. Nachdem Sie den *Plugin Manager* beendet haben, wählen Sie bitte **Plugins · Plugin Options**, wählen das Plugin *Buffer Tabs* und setzen ein Häkchen vor **Enable Buffer Tabs by default**. Bei **Location of Buffer Tabs** sollten Sie **top** einstellen. Durch diese Einstellung wird das Modul bei jedem Programmstart aktiviert. Um *jEdit* nicht erst beenden und neu starten zu müssen, können Sie die *Buffer Tabs* im Menü **Plugins** auch manuell einschalten. Um die nun nicht mehr benötigte Klappbox unterhalb der Toolbar auszublenden, wählen Sie bitte **Utilities · Global Options**, klicken auf **View** und entfernen das Häkchen vor **Show buffer switcher**. Eine interessante Funktion von *jEdit* ist das automatische Laden von früher verwendeten Dokumenten. Wenn Sie dies nicht möchten, können Sie unter **Global Options** den Knoten **General** anwählen und dort die Häkchen vor **Restore previously open files on startup** und **even if the file names were specified on the command line** entfernen.

jEdit ist ein äußerst komfortabler Text- und Programmeditor. Seine besondere Stärke liegt in seiner Erweiterbarkeit durch Plugins, die in großer Zahl zur Verfügung stehen und die sich sehr komfortabel installieren und bei Bedarf auch wieder entfernen lassen. Beispielsweise gestattet die *Console* den Zugriff auf externe Programme. Sie funktioniert ganz ähnlich wie die *Eingabeaufforderung*. In diesem Kommandozeileninterpreter können Sie beispielsweise den Java-Compiler oder andere Tools aufrufen, ohne das *jEdit*-Fenster verlassen zu müssen. Im Vergleich zu *Notepad++* fällt allerdings die vergleichsweise lange Ladezeit auf. Der Hauptgrund hierfür ist, dass nicht nur *jEdit* selbst, sondern auch die Java-Laufzeitumgebung geladen werden muss. Dies lässt sich mindern, indem Sie einfach stets eine Instanz des Editors geöffnet haben. Eine weitere Möglichkeit besteht in der Verwendung eines *Application Managers*. Er sorgt dafür, dass nicht für jede Java-Anwendung eine eigene Laufzeitumgebung gestartet wird, sondern bündelt die Programme in einer Instanz.

2.2 Integrierte Entwicklungsumgebungen

Moderne Programmiereditoren bieten zahlreiche Funktionen, die den Entwickler beim Schreiben von Programmen unterstützen. Als Beispiele hierfür seien die visuelle Hervorhebung von Sprachelementen und die Möglichkeit genannt, Programmkonstrukte (Blöcke) bei Bedarf einzuklappen. Trotzdem behalten sie ganz bewusst ihren Charakter eines Allzweckwerkzeugs. Ein Editor soll eben nicht nur für das Schreiben von Programm-Quelltexten verwendet werden können, sondern auch für das Setzen eines Textdokumentes in LaTeX, für das Schreiben eines Briefes oder das Bearbeiten von Binärdateien in hexadezimaler Darstellung. Dies unterscheidet sie von so genannten integrierten Entwicklungsumgebungen, die ganz bewusst die Programmiertätigkeit in den Vordergrund stellen. Selbstverständlich bieten auch sie ausgefeilte Funktionen zur Texteingabe, ergänzen diese aber um Mechanismen des Projektmanagements und der Versionsverwaltung. Von besonderer Bedeutung ist für sie eine möglichst nahtlose Integration aller am Entwicklungsprozess beteiligten Programme und Tools.

2.2.1 JCreator

JCreator[3] ist eine in C++ geschriebene integrierte Java-Entwicklungsumgebung, die sich in Ihrem Erscheinungsbild an Microsofts *Visual Studio* anlehnt, und somit Entwicklern, die mit diesem Werkzeug vertraut sind, einen problemlosen Einstieg in die Java-Programmierung ermöglicht. Das von der niederländischen Firma *Xinox Software* entwickelte Programm wird in zwei Varianten vertrieben. Neben der Freeware-Version *JCreator LE* gibt es den kostenpflichtigen

3 http://www.jcreator.com/

JCreator Pro, der für 69 US-$ auf der Produkt-Homepage bestellt werden kann. Um *JCreator LE* herunterladen[4] zu können, ist eine Registrierung beim Hersteller erforderlich, wobei Sie allerdings nur Ihren Namen sowie Ihre E-Mail-Adresse angeben müssen. Sie erhalten daraufhin eine Nachricht, die den Link auf die herunterzuladende Datei enthält. Die zum Zeitpunkt der Drucklegung aktuelle Version finden Sie auch auf der Begleit-CD. Die Installation erfordert nur sehr wenige Benutzereingaben und ist in wenigen Minuten abgeschlossen. Folgen Sie hierzu bitte den Anweisungen des Programms *Setup.exe*, das in *jcrea350.zip* enthalten ist. Wichtig ist nur, dass Sie vor der Installation von *JCreator LE* ein aktuelles Java Development Kit installiert und eingerichtet haben.

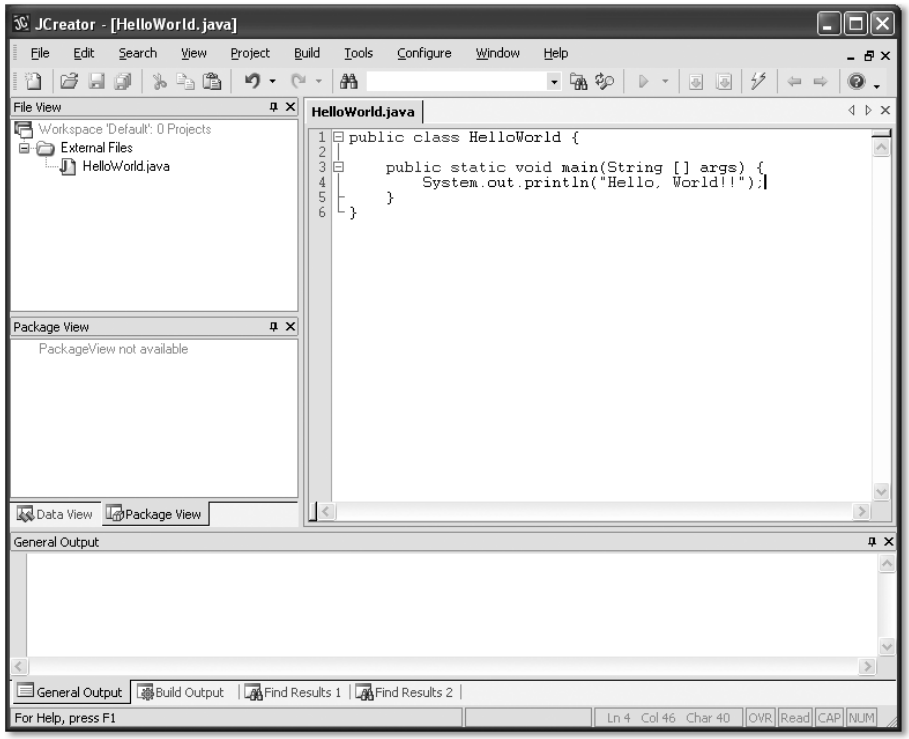

Abbildung 2.5 JCreator LE

Nach dem ersten Programmstart können Sie im Prinzip sofort mit der Entwicklung beginnen. Um eine Klasse zu realisieren, öffnen Sie mittels **File · New · File** den **File Wizard** und legen dort eine Java-Quelltextdatei, beispielsweise *HelloWorld.java* an. Nachdem Sie die Klasse vollständig implementiert haben, können Sie sie mit **Build · Compile File** übersetzen und anschließend mit **Build · Execute File** starten. Normalerweise werden Sie zusammengehörende Klassen aller-

4 http://www.jcreator.com/download.htm

dings in Projekten ablegen. Diese werden mit **File · New · Project** angelegt. Der **Project Wizard** bietet eine Reihe von Projektvorlagen, beispielsweise *Basic Java Application* und *Basic Java Applet*. Die in Abbildung 2.5 gezeigte Einteilung des JCreator-Programmfensters wird übrigens *Workspace* genannt. Sie können zum Beispiel Unter-Fenster beliebig anordnen sowie die Position und den Inhalt von Toolbars einstellen. Solche Einstellungen werden einem Workspace zugeordnet. Indem Sie mehrere solcher Workspaces vorhalten, können Sie mit wenigen Mausklicks zwischen unterschiedlichen Konfigurationen der IDE umschalten.

Trotz seines beachtlichen Funktionsumfangs ist *JCreator LE* äußerst kompakt und damit sehr schnell geladen. Durch die Programmierung in C++ ist die Anwendung zudem sehr performant.

2.2.2 NetBeans IDE

NetBeans startete 1996 als ein Projekt tschechischer Studenten unter dem Namen *Xelfi*. Ziel war damals, eine der integrierten Entwicklungsumgebung *Delphi* nachempfundene IDE für Java in Java zu implementieren. Nachdem Kapitalgeber auf die Studenten aufmerksam geworden waren, entstand die Idee, netzwerkfähige JavaBeans zu entwickeln, daher auch der Name *NetBeans*.

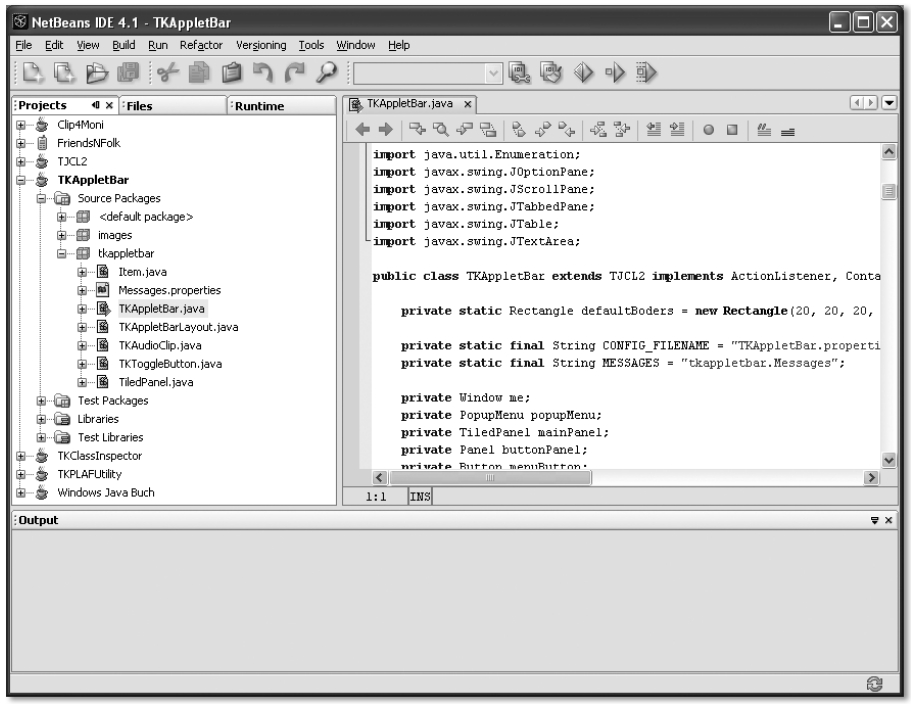

Abbildung 2.6 Die NetBeans IDE

1999 kaufte Sun die damalige Version der IDE, *NetBeans Developer X2*, und benannte sie in *Forté for Java* um. Bald entstand die Idee, hieraus quelloffene Software zu machen. So wurde das Produkt erneut umbenannt und ist seit 2000 als Open Source-IDE auf der Projekt-Homepage **www.netbeans.org** erhältlich. Neben der eigentlichen Entwicklungsumgebung *NetBeans IDE* steht das *NetBeans Mobility Pack* zur Verfügung, das auf Basis von Suns Wireless Toolkit die Entwicklung von J2ME-Anwendungen ermöglicht. Sie können *NetBeans IDE* von der Projekt-Homepage herunterladen.[5] Die zum Zeitpunkt der Drucklegung aktuelle Version 4.1 finden Sie aber auch auf der Begleit-CD. Bevor Sie die Installation durch Doppelklick auf *netbeans-4_1-windows.exe* starten, sollten Sie eine möglichst aktuelle Version des Java Development Kits installiert und eingerichtet haben.

NetBeans bietet eine *Ant*[6]-gestützte Projektverwaltung, erlaubt die Einbindung gängiger Versionierungssysteme und ist über Plugins praktisch beliebig erweiterbar. Auf der Projekt-Homepage finden Sie verfügbare Module nach Themen geordnet.[7] Sehr interessant ist unter anderem der *yWorks Ant Explorer*[8], der Ant-Build-Scripts grafisch aufbereitet und so die Abhängigkeiten im Build-Prozess visualisiert. Um ein NetBeans-Modul zu installieren, müssen Sie zunächst die entsprechende *.nbm*-Datei herunterladen und an beliebiger Stelle speichern. Öffnen Sie anschließend über **Tools · Update Center** den **Update Center Wizard** und wählen **Install Manually Downloaded Modules**. Im zweiten Schritt, *Select Modules to Install*, klicken Sie bitte auf die Schaltfläche **Add** und wählen das heruntergeladene Modul aus. Klicken Sie nun auf **Next**. Sie sehen eine Zusammenfassung der zu installierenden Plugins, die Sie mittels **Next** bestätigen. Anschließend müssen Sie der angezeigten Lizenzvereinbarung zustimmen. Im vierten und letzten Schritt können Sie durch Ankreuzen von **Global** festlegen, dass das zu installierende Modul in das NetBeans-Programmverzeichnis kopiert wird. Alternativ können Sie das Modul in Ihr Heimatverzeichnis installieren lassen. Wichtig ist noch, dass Sie durch Anklicken von **View Certificate** die Zustimmung für die Installation der Module gegeben haben.

Die Schaltfläche **Finish** schließt die Installation des Moduls ab. Um das Ant-Build-Script eines Ihrer Projekte zu betrachten, öffnen Sie bitte mit **Window · Files** das Fenster, das die Dateien Ihres Projektes anzeigt, und öffnen durch Klick mit der rechten Maustaste das Kontextmenü der Datei *build.xml*. Der Menüpunkt **Visualize** öffnet die grafische Darstellung, die Sie in Abbildung 2.8 sehen.

5 http://www.netbeans.info/downloads/download.php?type=4.1
6 http://ant.apache.org/
7 http://www.netbeans.org/catalogue/index.html
8 http://www.yworks.com/en/products_antexplorer_about.htm

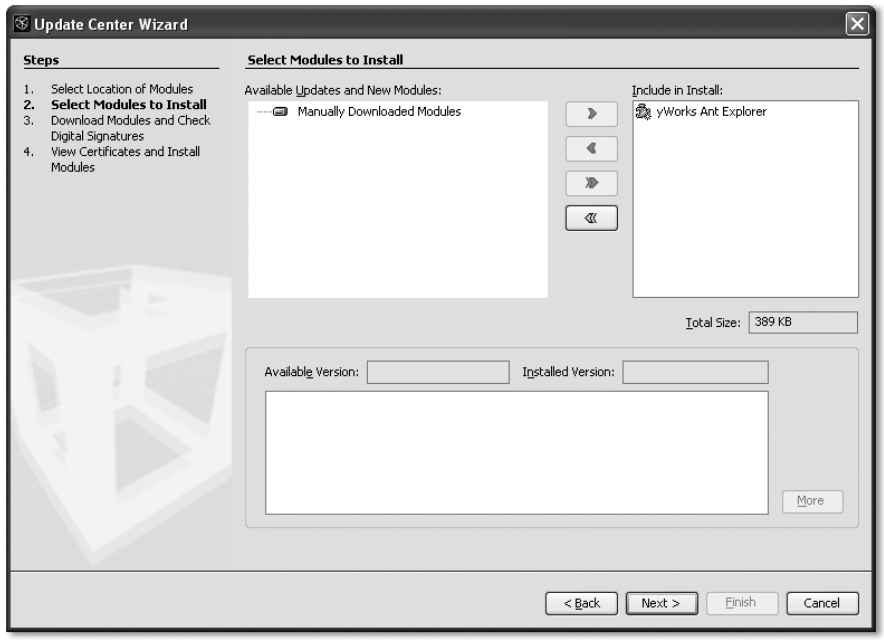

Abbildung 2.7 Der NetBeans Update Wizard

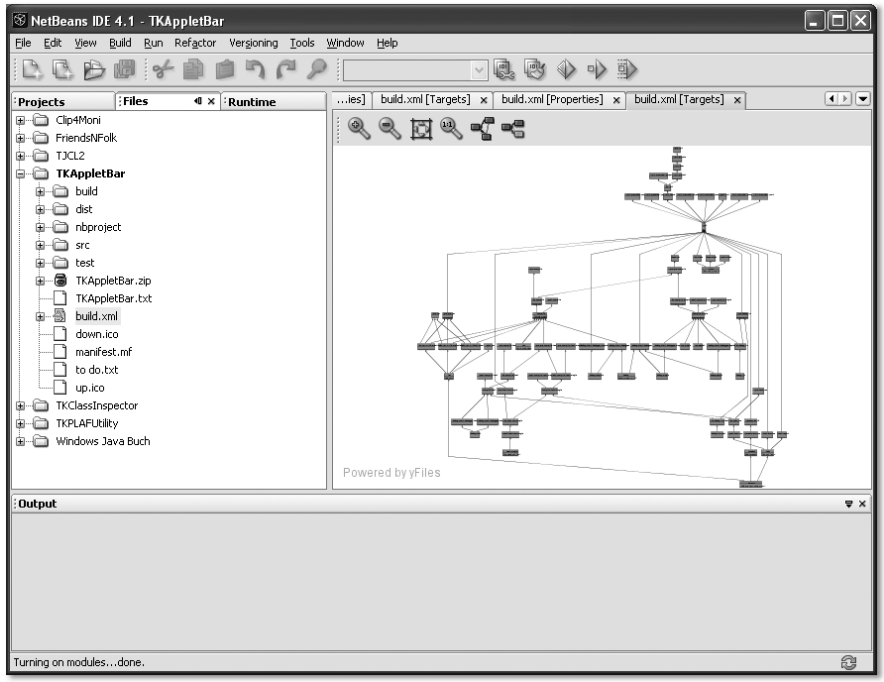

Abbildung 2.8 Die grafische Darstellung von Ant-Build-Scripts

Durch Doppelklick auf die blau eingefärbten Kästchen können Sie – übrigens schön animiert – in Teile des Baums hineinzoomen.

Eine weitere äußerst leistungsfähige Erweiterung der NetBeans IDE ist der *NetBeans Profiler*[9]. Derzeit müssen Sie dieses Modul noch zusätzlich installieren, allerdings ist es sehr wahrscheinlich, dass es in zukünftige Versionen der IDE integriert wird. Um es zu nutzen, müssen Sie das Installationsprogramm *profiler-m8v3-win.exe*[10] herunterladen. Bitte beachten Sie, dass der Profiler nur mit aktuellen Java-Versionen funktioniert. Wenn Sie mit dem Java Development Kit 5 arbeiten, sollten Sie mindestens das Update 4 installiert haben. Die eigentliche Installation ist nach wenigen Benutzereingaben abgeschlossen. Achten Sie aber bitte darauf, NetBeans vor dem Start zu beenden.

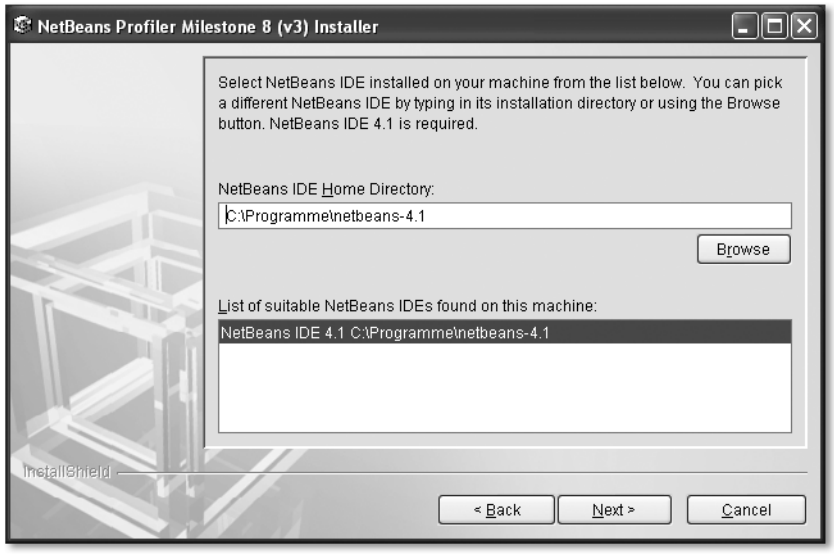

Abbildung 2.9 Die Installation des NetBeans Profilers

Nach einem Neustart der IDE steht der *Profiler* über ein eigenes **Profile**-Menü in der Menüleiste zur Verfügung. Bevor Sie Performance- oder Speicherverbrauchsmessungen durchführen können, müssen Sie die hierfür verwendete virtuellen Maschine einmalig kalibrieren. Die entsprechende Funktion erreichen Sie über **Profile · Advanced Commands · Run Profiler Calibration**. In Abbildung 2.10 sehen Sie den Informationsdialog, der Sie am Ende der Kalibrierung über deren Ausgang informiert.

9 http://profiler.netbeans.org/
10 http://profiler.netbeans.org/download.html

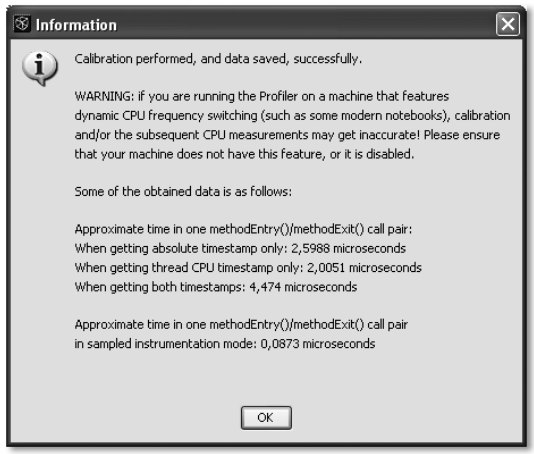

Abbildung 2.10 Das Ergebnis der Profiler-Kalibrierung

Anschließend können Sie über **Profile · Profile Main Project** die Geschwindigkeit Ihrer Anwendung oder ihren Speicherverbrauch messen. Wählen Sie hierzu im Dialog **Select Profiling Task** beispielsweise **Analyze Memory Usage** oder **Analyze Performance**. Durch Anklicken der Schaltfläche **Run** starten Sie den Analysevorgang. In Abbildung 2.11 sehen Sie, wie der Profiler seine gesammelten Informationen grafisch aufbereitet.

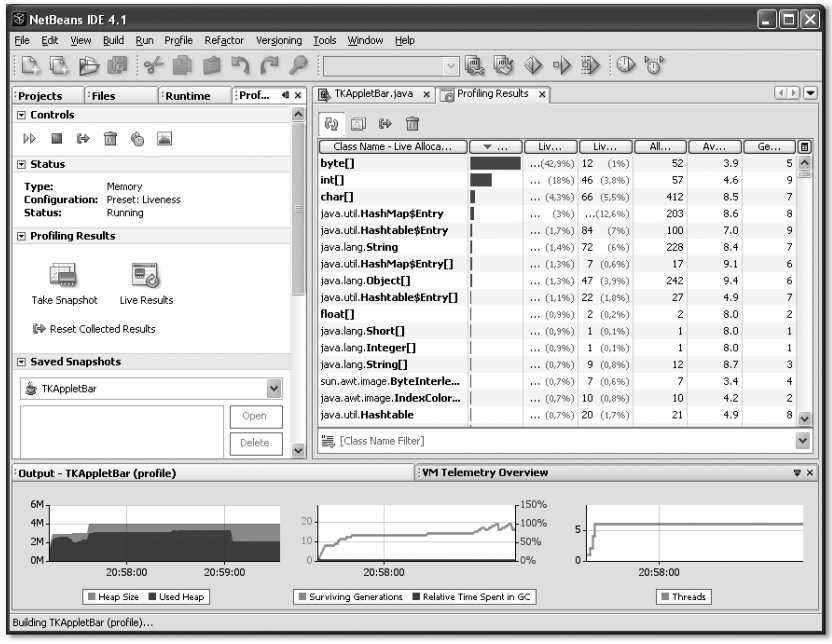

Abbildung 2.11 Der NetBeans-Profiler

Die *NetBeans IDE* ist eine ausgesprochen leistungsfähige Java-Entwicklungsumgebung, die sich durch eine Vielzahl von Modulen erweitern lässt. Allerdings ist ihr Speicherverbrauch im Vergleich zu beispielsweise *JCreator* beachtlich, zudem benötigt sie für den Start einige Zeit.

Im folgenden Abschnitt möchte ich Ihnen Werkzeuge vorstellen, die auf den ersten Blick für die Software-Entwicklung nicht unbedingt notwendig sind, Ihnen aber auf jeden Fall das Programmiererleben sehr erleichtern werden.

2.3 Sonstige Werkzeuge

Oft sind es kleine, unscheinbare Helfer, die Sie aus einer schier ausweglosen Lage befreien. Stellen Sie sich beispielsweise das Schreckensszenario vor, Sie haben ein Programmierprojekt praktisch abgeschlossen, wollen einen abschließenden Übersetzungslauf starten und wundern sich über die Meldung des Compilers, dass er eine Klasse nicht finden kann. Wenn Sie regelmäßig Ihre Entwicklungsarbeit auf ein sicheres externes Medium kopiert haben, können Sie die Situation schnell bereinigen. Was aber, wenn das rettende Medium nicht verfügbar ist, weil es zu Hause auf Ihrem Schreibtisch liegt, Sie aber auf einer längeren Dienstreise sind? Oder wenn Sie, aus welchen Gründen auch immer, kein Backup anfertigen konnten?

2.3.1 Microsoft SyncToy

Mit dem *SyncToy* stellt Microsoft ein weiteres Mitglied der *PowerToys*-Familie für die kostenlose Nutzung zur Verfügung. Hauptaufgabe dieses Programms ist es, Dateien und Verzeichnisse zu kopieren, umzubenennen und zu synchronisieren.

Somit ist es ideal für das schnelle und unkomplizierte Anfertigen von Sicherungen. Sie können *SyncToy* kostenlos von der Microsoft-Website herunterladen,[11] müssen allerdings den *Windows Genuine Advantage*-Test[12] durchlaufen, der sicherstellen soll, dass Sie mit Original-Windows-Software arbeiten. Nach der Installation der Software müssen Sie zunächst ein so genanntes *Folder Pair* anlegen. Klicken Sie hierzu auf den Link **Create New Folder Pair**, den Sie nach dem ersten Start von *SyncToy* im Programmfenster sehen.

11 http://www.microsoft.com/windowsxp/using/digitalphotography/prophoto/synctoy.mspx
12 http://www.microsoft.com/genuine/downloads/whyValidate.aspx

Abbildung 2.12 Das SyncToy-Programmfenster

Anschließend öffnet sich der in Abbildung 2.13 gezeigte Dialog **Create New Folder Pair**. In den ersten beiden Schritten wählen Sie im Prinzip ein Quellverzeichnis, das die zu kopierenden oder synchronisierenden Daten enthält, sowie ein Zielverzeichnis, das die Kopien aufnehmen soll. Da die Richtung, in der Dateien kopiert werden, erst in Schritt 3, den Sie in Abbildung 2.14 sehen, festgelegt wird, verwendet Microsoft die Begriffe **Left Folder** und **Right Folder**.

Mein Tipp: Tragen Sie unter **Left Folder** das Verzeichnis ein, das Ihre zu sichernden Daten enthält. Als **Right Folder** wählen Sie das Verzeichnis, das die zu sichernden Daten als Kopie aufnehmen soll. Idealerweise liegt es auf einem externen Medium, das beispielsweise am USB-Port Ihres Rechners angeschlossen ist. Ebenfalls möglich, aber natürlich nicht ganz so sicher, ist eine andere Partition auf der Festplatte.

Abbildung 2.13 Festlegen von Quell- und Zielverzeichnissen für SyncToy

Nicht raten würde ich Ihnen, die Sicherung auf dem gleichen Laufwerk abzulegen, auf dem auch die zu sichernden Dateien liegen. Geht diese Partition verloren, hilft Ihnen auch das Backup nicht mehr.

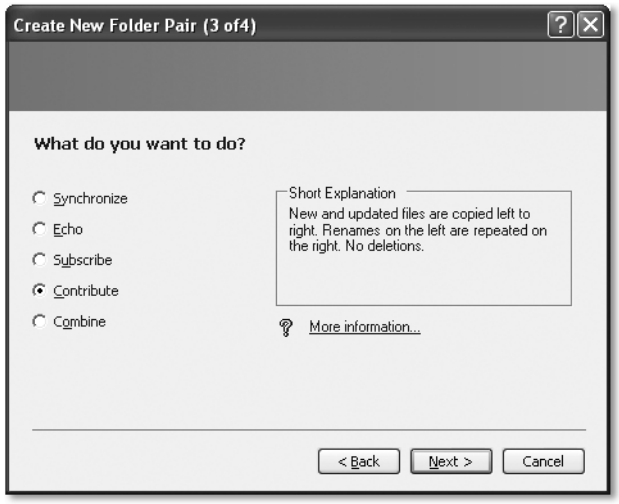

Abbildung 2.14 Wie sollen Dateien kopiert oder synchronisiert werden?

Im dritten Schritt legen Sie fest, wie die Dateien zwischen linkem und rechtem Verzeichnis kopiert werden sollen. Um ein Backup anzufertigen, ist **Contribute** eine gute Wahl. Neue und aktualisierte Dateien werden nämlich in das rechte Verzeichnis kopiert. Namensänderungen werden ebenfalls übernommen.

Dateien, die Sie aus Ihrem Quellverzeichnis gelöscht haben, bleiben im Zielverzeichnis aber erhalten. Wenn Sie also versehentlich eine Datei gelöscht haben, kann ein Aufruf des *SyncToys* keinen weiteren Schaden an Ihren gesicherten Dateien anrichten.

Im abschließenden vierten Schritt geben Sie Ihrem Verzeichnispaar einen möglichst aussagekräftigen Namen. Sie können jetzt jederzeit den Synchronisierungsvorgang starten. Hierzu müssen Sie *SyncToy* nicht unbedingt manuell starten, sondern können auch eine entsprechende Aufgabe in *Geplante Tasks* anlegen. Wie dies funktioniert, beschreibt die *SyncToy*-Online-Hilfe.

SyncToy ist ein äußerst flexibles, und dennoch leicht zu handhabendes Werkzeug, das das Anlegen von Backups sehr einfach macht. Sollte Ihnen aber trotz aller Vorsicht der Quelltext einer Ihrer Java-Klassen verloren gegangen sein, finden Sie im folgenden Abschnitt vielleicht Hilfe.

2.3.2 JAD

Der *Java Decompiler* von Pavel Kouznetsov kann aus Java-Klassendateien Quelltext generieren und nimmt so versehentlich gelöschten *.java*-Dateien ihren Schrecken. Die Verwendung des Tools ist für nichtkommerzielle Zwecke kostenlos. Sie können *JAD* von der Programm-Homepage[13] herunterladen. Zusätzlich finden Sie die zum Zeitpunkt der Drucklegung aktuelle Version auf der Begleit-CD zum Buch. Die Installation gestaltet sich äußerst einfach. Nach dem Entpacken des Archivs *jadnt158.zip* können Sie das Programm *jad.exe* prinzipiell in jedem beliebigen Verzeichnis ablegen. Damit Sie es in der *Eingabeaufforderung* ohne Angabe des absoluten Zugriffspfades aufrufen können, sollten Sie *JAD* aber in ein Verzeichnis kopieren, das in der `PATH`-Umgebungsvariable enthalten ist. Nähere Infos hierzu finden Sie in Kapitel 1, *Installation und Konfiguration*.

Um aus einer Klassendatei Java-Quelltext zu generieren, reicht es im Allgemeinen, den Namen der Bytecode-Datei als einzigen Parameter an *jad.exe* zu übergeben. Das Tool erzeugt daraufhin im aktuellen Verzeichnis eine Datei mit gleichem Namen, die allerdings auf *.jad* endet. Dies geschieht, um nicht versehentlich eine bereits existierende *.java*-Datei zu überschreiben. Nachdem Sie überprüft haben, dass keine solche Datei vorhanden ist, können Sie die *.jad*-Datei umbenennen und mit dem Java-Compiler ggf. neu übersetzen. Falls Sie häufiger *.class*-Dateien decompilieren müssen, bietet es sich an, eine entsprechende Aktion für den Dateityp *.class* zu definieren. Auf diese Weise kön-

13 http://www.kpdus.com/jad.html

nen Sie *JAD* über das Kontextmenü im Windows Explorer aufrufen. Wie Sie hierzu vorgehen können, zeigt Kapitel 1.

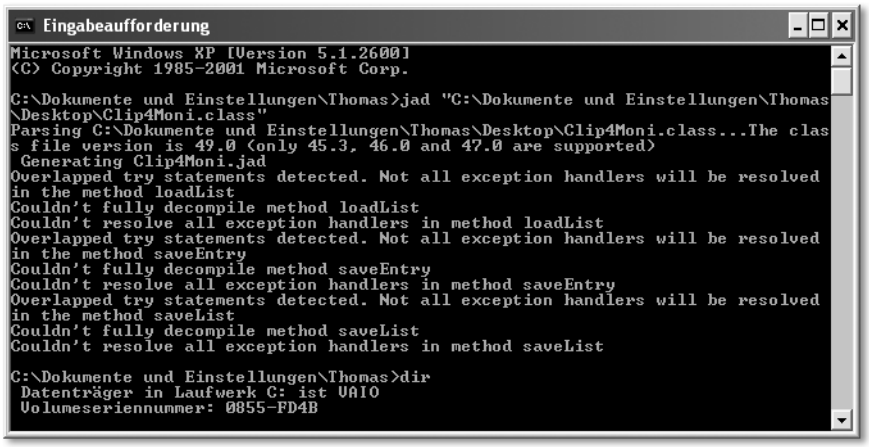

Abbildung 2.15 Ausgaben während eines Decompile-Laufs

Der nächste Abschnitt ist einem weiteren sehr nützlichen Werkzeug gewidmet: der *BeanShell*. Hierbei handelt es sich um Java-Interpreter und Skriptsprache in einem. Sie ermöglicht es, einfache Java-Ausdrücke und komplexe Klassen ohne vorheriges Übersetzen auszuführen. Beispielsweise ist es mit der *BeanShell* leicht, sich den Wert von System-Properties anzeigen zu lassen. Wie dies funktioniert, wurde in Kapitel 2 demonstriert.

2.3.3 Die BeanShell

Wenn Sie häufig wiederkehrende Aufgaben auf elegante Weise automatisieren möchten, empfehlen sich *Skriptsprachen*. Die auszuführenden Befehle werden in der Syntax der Sprache kodiert und bei Bedarf durch einen Interpreter ausgeführt. Im Java-Umfeld hat sich eine Reihe von Scriptsprachen etabliert, beispielsweise *Groovy*[14], *Rhino*[15] und *Jython*[16]. Ein weiteres, sehr interessantes Projekt ist die *BeanShell*, die Java-Anweisungen unmittelbar ausführbar macht, also ohne vorherige Übersetzung, und zudem die Java-Syntax um Annehmlichkeiten aus der Scripting-Welt erweitert. Die *BeanShell* kann kostenlos von der Projekt-Homepage[17] heruntergeladen werden. Außerdem finden Sie die zum Zeitpunkt der Drucklegung aktuelle Version auf der Begleit-CD zum Buch. Eine Installation ist nicht erforderlich, Sie können die Anwendung jederzeit durch

14 http://groovy.codehaus.org/
15 http://www.mozilla.org/rhino/
16 http://www.jython.org/
17 http://www.beanshell.org/

Doppelklick auf *bsh-2.0b4.jar* starten. In diesem Fall öffnet sich der in Abbildung 2.16 gezeigte *BeanShell Desktop*, in dem Sie neben mehreren *Workspaces* einen sehr nützlichen Klassen-Browser öffnen können. Ein Workspace nimmt Ihre Eingaben, also Anweisungen und Ausdrücke in Java-Syntax, entgegen und zeigt das Ergebnis der Ausführung an. Hierfür stehen alle Klassen bereit, die im Klassenpfad der *BeanShell* enthalten sind. Sie können diesen zur Laufzeit durch Aufruf der Methode `addClassPath()` erweitern. Der bereits angesprochene Klassen-Browser zeigt alle Konstruktoren, Methoden und Variablen einer Klasse an und hilft Ihnen so bei der Formulierung Ihrer Befehle.

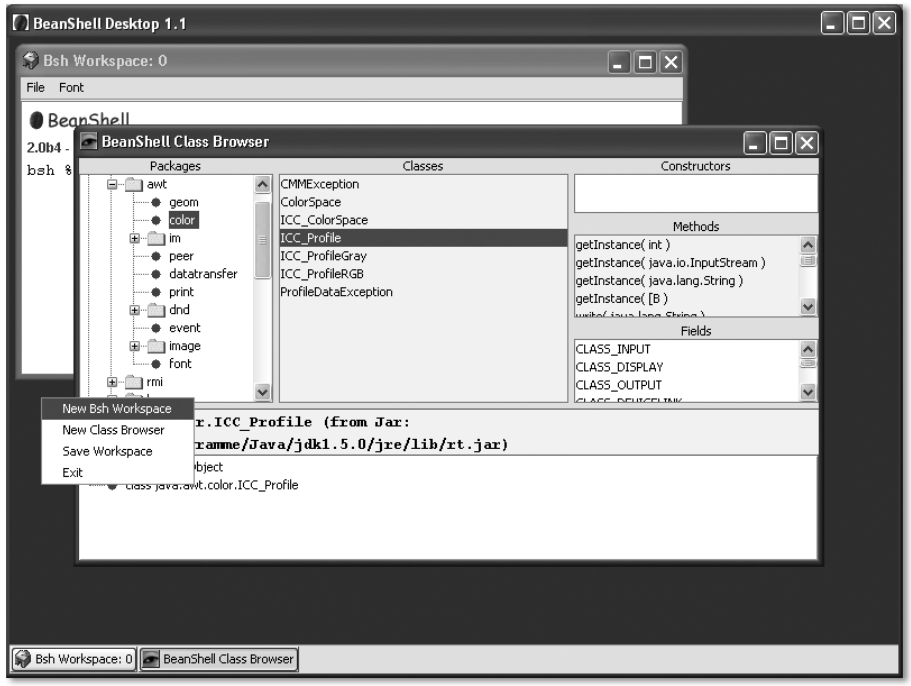

Abbildung 2.16 Der BeanShell Desktop

Alternativ können Sie die *BeanShell* auch in der *Eingabeaufforderung* starten. Da in der Manifest-Datei des *.jar*-Archivs der *BeanShell Desktop* als auszuführende Klasse eingetragen ist, müssen Sie folgendes Kommando eingeben:

```
java -cp C:\Programme\Java\BeanShell\bsh-2.0b4.jar
bsh.Interpreter
```

Bitte beachten Sie hierbei, dass Sie den Zugriffspfad an Ihre Umgebung anpassen müssen. Wie die Eingabe von Kommandos und die Ausgabe ihrer Ergebnisse im Textmodus aussieht, zeigt Abbildung 2.17. Wie Sie sehen, können Sie Variablenzuweisungen vornehmen, ohne erst deren Typ deklarieren zu müs-

sen. Wie in Java üblich werden Anweisungen mit dem Strichpunkt abgeschlossen. Um Werte auszugeben, verwenden Sie bitte die Methode `print()`. Außerdem stellt die *BeanShell* eine Reihe zusätzlicher Kommandos zur Verfügung, beispielsweise `pwd()` und `dir()`. Um eine Sitzung zu beenden, rufen Sie die Methode `exit()` auf. Die umfangreichen Möglichkeiten werden in der Dokumentation zur *BeanShell* ausführlich beschrieben. Pat Niemeyer hat auf der Projekt-Homepage[18] verschiedene Versionen bereitgestellt. Dort wird auch beschrieben, wie Sie den Interpreter in Ihre eigenen Programme einbinden und so um Scripting-Fähigkeiten erweitern können. Ein Beispiel hierfür ist der Editor *jEdit*, der in Abschnitt 2.1.2 vorgestellt wurde.

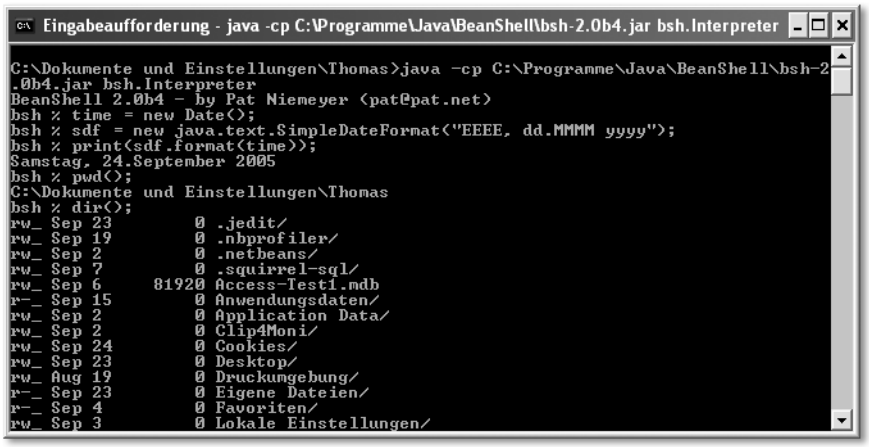

Abbildung 2.17 Die BeanShell in der Eingabeaufforderung

2.3.4 TKClassInspector

Die von mir entwickelte Anwendung *TKClassInspector* verwendet die *Reflection*-API, um ausführliche Informationen zu einer Klassendatei übersichtlich zu präsentieren. Das Programm steht unter der *GNU General Public Licence* und kann von der Projekt-Homepage[19] heruntergeladen werden. Selbstverständlich finden Sie die derzeit aktuelle Version auch auf der Begleit-CD zum Buch. Eine Installation ist nicht erforderlich. Um das Programm zu starten, genügt ein Doppelklick auf *TKClassInspector.jar*. Es öffnet sich eine Dateiauswahl, in der Sie beliebige Klassendateien markieren können. Diese werden anschließend analysiert und im Hauptfenster angezeigt (siehe Abbildung 2.18). Die Klappbox oberhalb der Registerkarten erlaubt die Auswahl der darzustellenden Klasse. Wenn Sie *TKClassInspector* über die Kommandozeile aufrufen, können Sie

18 http://www.beanshell.org/docs.html
19 http://www.moniundthomaskuenneth.de/tkclassinspector/

selbstverständlich gleich die zu analysierenden Klassendateien angeben. Noch mehr Komfort erreichen Sie, wenn Sie für den Dateityp *.class* in den **Ordneroptionen** des *Windows Explorers* eine neue Aktion definieren, die *TKClassInspector* aufruft. Sie erhalten dann einen zusätzlichen Eintrag im Kontextmenü von *.class*-Dateien. Wie Sie hierzu vorgehen, können Sie in Kapitel 1, *Installation und Konfiguration*, nachlesen.

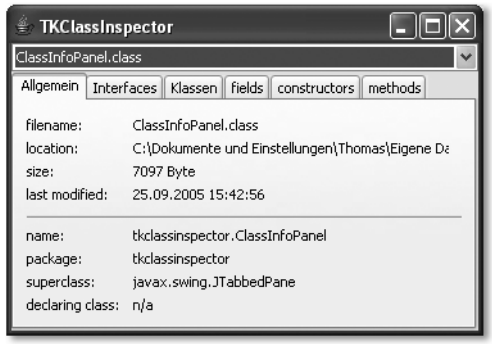

Abbildung 2.18 Das Programm TKClassInspector

Der Aufbau des *TKClassInspector*-Dialogs orientiert sich bewusst an den Dateiinfo-Dialogen von Windows. Das Programm enthält eine eigene Dialog-Unterklasse, die sich mit ganz wenigen Methoden-Aufrufen füllen lässt. Da ich TKClassInspector als Open Source freigegeben habe, können Sie die Klassen selbstverständlich in Ihre eigenen Programme übernehmen.

3 Feintuning der Benutzeroberfläche

3 Feintuning der Benutzeroberfläche

Die Benutzeroberfläche ist das Aushängeschild einer Anwendung. Leider wirken Java-Programme auf Windows-Benutzer oftmals fremdartig und ungewohnt. Dieses Kapitel zeigt Ihnen, wie Sie mit wenigen Handgriffen für ein gefälliges äußeres Erscheinungsbild Ihrer Anwendungen sorgen.

Gerade für unerfahrene Anwender ist es wichtig, sich an bekannten Elementen orientieren zu können. Beispielsweise sollten Schaltflächen stets dieselbe äußere Form haben und einheitlich beschriftet sein. Unter Windows weicht das Erscheinungsbild verschiedener Anwendungen allerdings zum Teil stark voneinander ab. Microsofts Office-Komponenten haben beispielsweise andere Menüleisten, Toolbars und Dialoge als die Windows-Kernkomponenten *Paint* oder *WordPad*. Diese wiederum sehen gänzlich anders aus als der *Windows Media Player* oder der *MSN Messenger*. Wenn schon das Wirtssystem die Tendenz zu einem uneinheitlichen Äußeren hat, ist es für eine Java-Anwendung unerlässlich, zumindest einem Standard konsequent zu folgen. Dieses Kapitel beschäftigt sich mit der Benutzeroberfläche von Java-Programmen. Ich möchte Ihnen zeigen, warum Windows-Anwender Java-Programme oftmals als hässlich und ungewohnt empfinden, und Ihnen dabei helfen, mit wenigen Handgriffen das Aussehen verändern zu können.

3.1 Swing und das Konzept Pluggable Look and Feel

Sehr viele Java-Programme mit einer grafischen Benutzeroberfläche verwenden Komponenten der Klassenbibliothek *Swing* für die Darstellung ihrer Bedienelemente. *Swing* ist Teil der *Java Foundation Classes* und seit Version 1.2 integraler Bestandteil einer Java-Installation.

3.1.1 Unterschiede zwischen AWT und Swing

Wie die seit Java 1.0 verfügbaren Komponenten des *Abstract Window Toolkits* werden *Swing*-Klassen unter anderem dazu verwendet, die Bedienoberfläche einer Java-Anwendung zu gestalten. Swing ist dabei keineswegs als Ersatz für das AWT gedacht, vielmehr erbt Swing zahlreiche durch AWT eingeführte Konzepte und Mechanismen, baut also auf seinem Vorgänger auf, ergänzt und erweitert ihn. Allerdings sind Swing-Komponenten weitaus mächtiger als ihre AWT-Pendants. Beispielsweise lassen sich die »alten« Bedienelemente nicht durchgängig mit der Tastatur bedienen. Abgesehen davon stellt Swing wichtige

Komponenten bereit, die im AWT keine Entsprechung haben. Beispiele hierfür sind Klassen für die Darstellung von Registerkarten, Tabellen oder Bäumen.

In einem Punkt unterscheiden sich AWT und Swing allerdings grundsätzlich voneinander. AWT-Elemente sind »nativ«, das heißt, sie werden (zumindest in allen bisher erschienenen Java-Versionen) durch die grafische Benutzeroberfläche des Wirtssystems dargestellt und verwaltet. Eine Schaltfläche sieht unter Windows nicht nur aus wie eine Schaltfläche einer nativen Windows-Anwendung, es ist auch eine. Im Gegensatz dazu werden Swing-Komponenten durch Java gezeichnet und kontrolliert. Man unterscheidet in diesem Zusammenhang zwischen *leichtgewichtigen* (lightweight) und *schwergewichtigen* (heavyweight) Komponenten. Da Swing-Komponenten keine nativen Elemente des Wirtssystems sind, muss Java sowohl für die grafische Darstellung, also das Zeichnen auf dem Bildschirm, als auch für die Behandlung von Benutzereingaben sorgen. Werden die grafischen Bedienelemente des Wirtssystems nachgebildet, sollten die »Nachbauten« ihren Vorbildern selbstverständlich weitestgehend entsprechen. Im Idealfall merkt der Anwender nicht, ob er eine »echte« Windows-Schaltfläche anklickt oder eine durch Swing simulierte. Leider ist dies in der Realität nicht immer so. Ausführliche Informationen hierzu finden Sie in Abschnitt 3.3.

Das Zusammenspiel von grafischer Darstellung und Behandlung von Benutzereingaben wird übrigens oft mit dem englischen Terminus *Look and Feel* umschrieben. Beispielsweise haben Schaltflächen unter Windows ein anderes Aussehen als vergleichbare Elemente in Mac OS X und sie reagieren auch anders auf Mausklicks und Mausbewegungen. Allerdings wird der Ausdruck *Look and Feel* oft ebenso in Zusammenhang mit dem Verhalten einer Anwendung als Ganzes verwendet. Beispielsweise erwartet man von einem Windows-Programm, dass es mittels Alt + F4 beendet werden kann und dass seine Menüleiste einem gewissen Schema folgt. In beiden Fällen geht es also um Aussehen und Verhalten, nur der Blickwinkel ist unterschiedlich – einmal von der Ebene der grafischen Bedienelemente, einmal von der Ebene der Anwendung aus.

3.1.2 Das Pluggable Look and Feel-Konzept

In Swing ist die Austauschbarkeit von Aussehen und Verhalten ein Designmerkmal. Die grafische Darstellung einer Komponente und deren Reaktion auf Benutzereingaben können ohne Änderungen am Programm ausgetauscht werden, sogar im laufenden Betrieb. Möglich wird dies durch einen *pluggable look and feel* genannten Mechanismus. Die ihm zugrunde liegende Idee ist, das »sich zeichnen« nicht der eigentlichen Komponente zu überlassen, sondern in spezi-

alisierte Klassen auszulagern. Sehen Sie sich hierzu bitte das Programm *SwingDemo1* an.

```
package javafuerwindows.plaf;
import java.awt.BorderLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.JButton;
import javax.swing.JComboBox;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.border.TitledBorder;
public class SwingDemo1 extends JFrame
                        implements ActionListener {
    private JComboBox comboBox;
    private JButton button;
    private JLabel label;
    public SwingDemo1() {
        super("SwingDemo");
        setDefaultCloseOperation(EXIT_ON_CLOSE);
        label = new JLabel("keine Auswahl getroffen");
        label.setBorder(new TitledBorder("SwingDemo"));
        comboBox = new JComboBox(new String [] {"Auswahl 1",
                                                "Auswahl 2",
                                               "Auswahl 3"});
        comboBox.addActionListener(this);
        button = new JButton("Ende");
        button.addActionListener(this);
        JPanel cp = new JPanel(new BorderLayout());
        JPanel p1, p2;
        p1 = new JPanel();
        p1.add(comboBox);
        p2 = new JPanel();
        p2.add(button);
        cp.add(label, BorderLayout.NORTH);
        cp.add(p1, BorderLayout.CENTER);
        cp.add(p2, BorderLayout.SOUTH);
        setContentPane(cp);
        pack();
        setLocation(100, 100);
```

```
        setVisible(true);
    }

    public void actionPerformed(ActionEvent e) {
        if (e.getSource() == button)
            System.exit(0);
        else if (e.getSource() == comboBox)
            label.setText((String)
                          comboBox.getSelectedItem());
    }

    public static void main(String [] args) {
        new SwingDemo1();
    }
}
```

Listing 3.1 SwingDemo1.java

Wenn Sie das Programm mit einer Java-Version ab 5.0 durch Eingabe von `java javafuerwindows.plaf.SwingDemo1` aufrufen, sehen Sie sehr wahrscheinlich das in Abbildung 3.1 gezeigte Fenster.

Abbildung 3.1 Das Java Look and Feel

Durch den nur geringfügig modifizierten Aufruf `java` **`-Dswing.defaultlaf=com.sun.java.swing.plaf.windows.WindowsLookAndFeel`** `javafuerwindows.plaf.SwingDemo1` erreichen Sie, dass die Bedienelemente wie diejenigen nativer Windows-Programme aussehen.

Abbildung 3.2 Das Windows XP Look and Feel

Sie können durch das Setzen der System Property `swing.defaultlaf` beeinflussen, wie eine Swing-Anwendung aussieht, ohne das Programm vorher neu übersetzen zu müssen.

3.1.3 Das Java Look and Feel

Wenn Sie unter Windows eine Java-Anwendung starten, sehen Sie meistens das in Abbildung 3.1 gezeigte *Java Look and Feel*. Da es auf jeder Plattform zur Verfügung stehen sollte, die Java unterstützt, wird es häufig auch *Cross Platform Look and Feel* genannt. In der Java-Klassenbibliothek befinden sich die zu diesem Look and Feel gehörenden Klassen im Paket `javax.swing.plaf.metal`. Deshalb hat es schnell den Spitznamen *Metal* erhalten. In der Java-Version 5 hat Sun das Aussehen etwas modernisiert und nennt diesen Look *Ocean*. Die »klassische« *Metal* Variante heißt *Steel*.

Abbildung 3.3 Steel, die klassische Metal-Variante

Wenn Sie sich *Steel* einmal ansehen möchten, steht Ihnen die System Property `swing.metalTheme` zur Verfügung. Um *SwingDemo1* zu starten, müssen Sie Folgendes eintippen:

```
java  -Dswing.defaultlaf=javax.swing.plaf.metal.MetalLookAndFeel
-Dswing.metalTheme=steel javafuerwindows.plaf.SwingDemo1
```

Wie Sie gesehen haben, können Sie über die Kommandozeile durch Setzen von System Properties Einfluss auf das von einem Programm verwendete Look and Feel nehmen. `swing.defaultlaf` legt fest, welches Look and Feel Swing standardmäßig für ein Programm verwenden soll.

Im nächsten Abschnitt erkläre ich Ihnen, wie Sie innerhalb Ihrer Java-Anwendung vorhandene Look and Feels ermitteln und zwischen ihnen umschalten können.

3.1.4 Die Klassen UIManager, LookAndFeel und LookAndFeelInfo

Die Klasse `javax.swing.UIManager` ist für den Programmierer die zentrale Schnittstelle im Hinblick auf Look and Feels. Sie bietet zahlreiche Auskunftsmethoden und erlaubt zudem das Umschalten auf ein bestimmtes Look and Feel. Zunächst möchte ich Ihnen zeigen, welche Informationen Sie bei `UIManager` erfragen können.

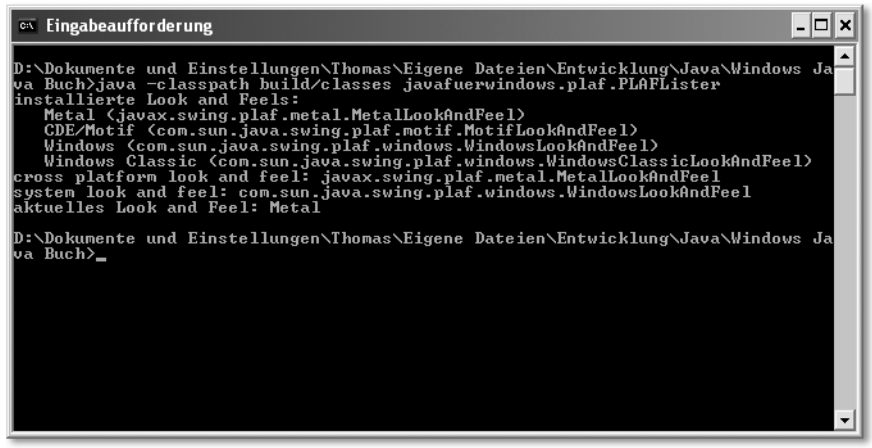

Abbildung 3.4 Ausgabe des Programms PLAFTester

Hierzu habe ich das Programm *PLAFTester* geschrieben, dessen Quelltext Sie im Folgenden sehen.

```
package javafuerwindows.plaf;
import javax.swing.UIManager;
import javax.swing.UIManager.LookAndFeelInfo;
public class PLAFLister {

    public static void main(String [] args) {
        LookAndFeelInfo [] lafs =
                        UIManager.getInstalledLookAndFeels();
        if (lafs != null) {
            System.out.println("installierte Look and
                                Feels:");
            LookAndFeelInfo laf;
            for (int i = 0; i < lafs.length; i++) {
                laf = lafs[i];
                System.out.println("   " + laf.getName() +
                                   " (" + laf.getClassName()
```

```
                                        + ")");
                }
            }
            System.out.println("cross platform look and feel: "
            + UIManager.getCrossPlatformLookAndFeelClassName());
            System.out.println("system look and feel: "
                   + UIManager.getSystemLookAndFeelClassName());
            System.out.println("aktuelles Look and Feel: "
                       + UIManager.getLookAndFeel().getName());
    }
}
```

Listing 3.2 PLAFTester.java

Die Methode `getInstalledLookAndFeels()` liefert ein Feld des Typs `UIManager.LookAndFeelInfo` und enthält alle installierten Look and Feels. Die Elemente des Feldes besitzen die Methoden `getName()` und `getClassName()`. Grundsätzlich wird ein Look and Feel durch einen voll qualifizierten Klassennamen und einen für den Anwender gedachten Namen (der in Menüs oder Dialogen angezeigt wird) eindeutig bestimmt. Wie Sie in Abbildung 3.4 sehen, hat beispielsweise das *Windows Look and Feel* den Klassennamen `com.sun.java.swing.plaf.windows.WindowsLookAndFeel`. Ferner gibt es unter anderem *CDE/Motif*. Die zugehörige Klasse heißt `com.sun.java.swing.plaf.motif.MotifLookAndFeel`. Bemerkenswert ist, dass *Ocean* und *Steel* nicht als eigenständige Look and Feels in Erscheinung treten. Die Erklärung hierfür ergibt sich aus ihrer technischen Umsetzung. Sie sind nämlich so genannte *Themes* (`javax.swing.plaf.metal.MetalTheme`), die durch die eigentliche *Metal* Look and Feel-Klasse `javax.swing.plaf.metal.Metal-LookAndFeel` angesprochen werden. Diese bietet die Methode `setCurrentTheme()` zum Umschalten zwischen Themes an.

Die beiden Methoden `getCrossPlatformLookAndFeelClassName()` und `getSystemLookAndFeelClassName()` liefern ebenfalls voll qualifizierte Klassennamen. Erstere liefert ein Look and Feel, das auf allen Java-Plattformen verfügbar sein sollte. Es handelt sich um das aus Abschnitt 3.1.3 bereits bekannte *Java Look and Feel*, also *Metal*. Der Rückgabewert von `getSystemLookAndFeelClassName()` ist auf Windows-, Mac OS- und Linux-Rechnern unterschiedlich. Die Methode liefert ein Look and Feel, das den nativen Bedienelementen des Wirtssystems möglichst genau entsprechen sollte.

Wie kann nun ein Java-Programm auf ein bestimmtes Look and Feel umschalten? `UIManager` bietet hierzu mehrere Varianten der Methode `setLookAndFeel()`. Um Ihnen zu verdeutlichen, wie man vorgeht, habe ich das Programm *SwingDemo2* geschrieben, das in Abbildung 3.5 zu sehen ist.

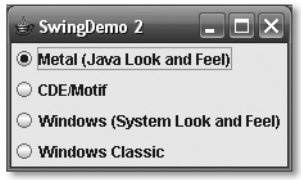

Abbildung 3.5 Das Programm SwingDemo2

Den vollständigen Quelltext finden Sie selbstverständlich auf der Begleit-CD zum Buch. Hier folgt nun ein Ausschnitt, der das Umschalten auf ein Look and Feel vornimmt.

```
public void actionPerformed(ActionEvent e) {
    try {
        UIManager.setLookAndFeel(e.getActionCommand());
        SwingUtilities.updateComponentTreeUI(this);
        pack();
        repaint();
    } catch (ClassNotFoundException ex) {
    } catch (InstantiationException ex) {
    } catch (IllegalAccessException ex) {
    } catch (UnsupportedLookAndFeelException ex) {
    }
}
```

Der Parameter, der an `setLookAndFeel()` übergeben wird, ist ein `String`, genauer gesagt der Rückgabewert der Methode `getActionCommand()`. Diese wiederum erhält den String von `JRadioButton`-Objekten, deren Action Commands an anderer Stelle im Programm mit den voll qualifizierten Klassennamen der Look and Feels belegt wurden, die sie repräsentieren. Der Aufruf von `SwingUtilities.updateComponentTreeUI()` und `repaint()` ist nur dann erforderlich, wenn Ihre Anwendung schon Swing-Komponenten dargestellt hat. Dies ist beispielsweise dann der Fall, wenn Sie dem Anwender die Möglichkeit geben möchten, das Look and Feel selbst zu bestimmen. Ihr Programm würde hierzu eine Liste der installierten Look and Feels anzeigen, entweder in einem eigenen Dialog oder in der Menüleiste.

Möchten Sie hingegen nur beim Programmstart ein bestimmtes Look and Feel setzen, genügt der Aufruf von `setLookAndFeel()`.

```
public static void main(String [] args) {
    try {
        UIManager.setLookAndFeel(
        UIManager.getSystemLookAndFeelClassName());
    } catch (Exception e) {
    }
    new SwingDemo3();
}
```

Mit `UIManager.setLookAndFeel` haben Sie die zweite Möglichkeit kennen gelernt, für Ihre Anwendung ein bestimmtes Look and Feel zu wählen. Indem Sie `UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName())` aufrufen, können Sie zudem sicherstellen, dass Ihre Anwendung unter jedem Betriebssystem ein Look and Feel verwendet, das die nativen Bedienelemente möglichst genau nachzubilden versucht.

Die Abschnitte 3.3.2 und 3.4 behandeln Alternativen zu den mit Java gelieferten Look and Feels. Wenn Sie, wie ich eben gezeigt habe, ein bestimmtes Look and Feel in Ihrem Programmquelltext vorgeben, nehmen Sie Ihren Anwendern die Freiheit, ein seiner Meinung nach besseres oder schöneres Aussehen zu verwenden. Aus diesem Grund zeige ich Ihnen im folgenden Abschnitt eine dritte Möglichkeit, das Look and Feel Ihrer Anwendung einzustellen.

3.2 Die Datei swing.properties

Bisher wurden zwei Wege erläutert, das von Ihrer Anwendung genutzte Look and Feel einzustellen, nämlich durch Setzen einer System Property als Kommandozeilenoption beim Start sowie durch Aufruf der Methode `UIManager.setLookAndFeel()`. Im Folgenden widmen wir uns einer dritten Variante, der Datei *swing.properties*.

3.2.1 Aufbau

swing.properties wird im Verzeichnis *lib* unterhalb des Java-Heimatverzeichnisses erwartet. Die Datei ist zeilenweise organisiert und kann aus Leerzeilen, durch # eingeleitete Kommentare sowie aus Schlüssel-Wert-Paaren bestehen. Je Zeile ist eine solche Zuweisung möglich. Sie hat die allgemeine Form `schlüssel=wert`.

```
#file written by TKPLAFUtility 0.22
#Wed May 25 22:14:59 CEST 2005
swing.installedlaf.motiflookandfeel.name=CDE/Motif
swing.installedlaf.windowslookand-
feel.class=com.sun.java.swing.plaf.windows.WindowsLookAndFeel
swing.instal-
ledlaf.office2003lookandfeel.class=org.fife.plaf.Office2003.Offi
ce2003LookAndFeel
swing.installedlaf.metallookandfeel.name=Metal
swing.installedlaf.motiflookand-
feel.class=com.sun.java.swing.plaf.motif.MotifLookAndFeel
swing.installedlaf.metallookand-
feel.class=javax.swing.plaf.metal.MetalLookAndFeel
swing.defaultlaf=org.fife.plaf.Office2003.Office2003LookAndFeel
swing.installedlaf.windowsclassiclookand-
feel.class=com.sun.java.swing.plaf.windows.WindowsClassic-
LookAndFeel
swing.auxiliarylaf=
swing.installedlafs=metallookandfeel,motiflookandfeel,windows-
lookandfeel,windowsclassiclookandfeel,office2003lookandfeel
swing.installedlaf.windowslookandfeel.name=Windows
swing.installedlaf.windowsclassiclookandfeel.name=Windows
swing.installedlaf.office2003lookandfeel.name=Office 2003
```

Listing 3.3 Beispiel für eine swing.properties-Datei

In *swing.properties* werden zahlreiche Aspekte des *Pluggable Look and Feels* konfiguriert. Besonders interessant ist der Schlüssel `swing.defaultlaf`. Sein Wert enthält den voll qualifizierten Klassennamen des Look and Feels, das verwendet wird, wenn kein anderes mittels System Property oder durch Aufruf von `UIManager.setLookAndFeel` gesetzt wird. Die Zeile `swing.defaultlaf=org.fife.plaf.Office2003.Office2003LookAndFeel` setzt beispielsweise das in Abschnitt 3.4.1 ausführlicher vorgestellte *OfficeLnFs*-Look and Feel als Standard.

Ferner werden in *swing.properties* alle installierten Look and Feels eingetragen. Sie erinnern sich, `UIManager.getInstalledLookAndFeels` liefert eine solche Liste. Für jedes Look and Feel werden hierzu zwei Schlüssel verwendet. Diese tragen die Namen `swing.installedlaf.`**`xyz`**`.name` und `swing.installedlaf.`**`xyz`**`.class`. Der Namensteil `xyz` ist bei zwei zusammengehörenden Schlüsseln gleich. Für jedes Look and Feel liefern die beiden Schlüssel also einen voll qualifizierten Klassennamen, der für das Laden des Look and Feels

benötigt wird, sowie einen Klartextnamen, der für die Anzeige in Menüs oder Dialogen verwendet werden kann. Der Namensteil `xyz` wird zudem im Schlüssel `swing.installedlafs` eingetragen. Anhand dieses Schlüssels erkennt Swing, welche Look and Feels vorhanden sind, und kann die Namen derjenigen Schlüssel ermitteln, die den Klassen- und Klartextnamen enthalten. In der Beispieldatei wurde das *OfficeLnFs*-Look and Feel eingetragen. Zu ihm gehören die Schlüssel `swing.installedlaf.office2003lookandfeel.name` und `swing.installedlaf.office2003lookandfeel.class`. Zudem findet sich der Namensbestandteil `office2003lookandfeel` in `swing.installedlafs`.

Auch so genannte *Auxiliary Look and Feels* werden in *swing.properties* eingetragen. Bei diesen handelt es sich nicht um vollwertige Look and Feels, sondern um Ergänzungen, die zusätzlich zu einem vollständigen Look and Feel verwendet werden. Beispielsweise könnte ein Auxiliary Look and Feel die Ansteuerung einer Braille-Lesezeile übernehmen oder in Verbindung mit einem Text to Speech-System wichtige Texte der Benutzeroberfläche vorlesen. Leider gibt es derzeit nur sehr wenige Auxiliary Look and Feels. Dennoch möchte ich Ihnen kurz zeigen, wie Sie ein solches in *swing.properties* einbinden können. Die voll qualifizierten Klassennamen der zu verwendenden Auxiliary Look and Feels werden dem Schlüssel `swing.auxiliarylaf` als durch Kommata getrennte Liste zugewiesen. Die weiter oben beschriebene Aufteilung in Schlüssel für Klassen- und Klartextnamen sowie Nennung des gemeinsamen Namensbestandteils in einem Sammelschlüssel wird hier nicht verwendet.

swing.properties ist eine relativ einfach aufgebaute Datei, die mit jedem Texteditor bearbeitet werden kann. Allerdings ist das Eintragen neuer Look and Feels auf diesem Wege aufwändig und damit fehleranfällig. Ein Programm namens *TKPLAFUtility* kann Ihnen den Umgang mit *swing.properties* so einfach wie möglich machen.

3.2.2 TKPLAFUtility

TKPLAFUtility ist Freeware. Sie können mein Programm entweder von seiner Homepage[1] herunterladen oder die zum Zeitpunkt der Drucklegung aktuelle Version von der Begleit-CD zum Buch installieren. Die Installationsroutine basiert auf dem äußerst leistungsfähigen Programm *IzPack*, das in Kapitel 10, *Deployment*, ausführlich vorgestellt wird.

Mit diesem Tool können Sie das standardmäßig verwendete Look and Feel aus einer Liste wählen und gleich in Aktion sehen. Zudem haben Sie die Möglich-

1 http://www.moniundthomaskuenneth.de/tkplafutility/

keit, neue Look and Feels in *swing.properties* einzutragen und nicht mehr benötigte zu löschen.

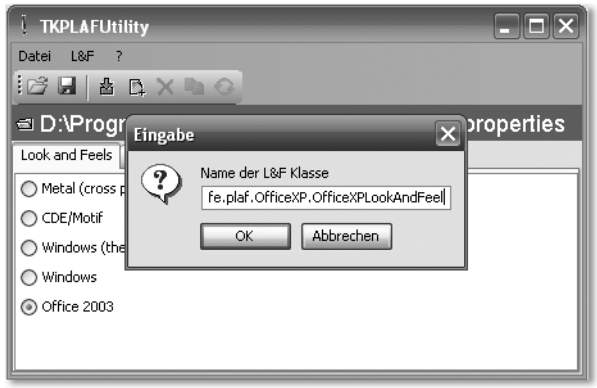

Abbildung 3.6 Hinzufügen des Office XP Look and Feels in TKPLAFUtility

Sie müssen hierfür nur den voll qualifizierten Klassennamen des neuen Look and Feels in einem Dialog eintragen. Diesen entnehmen Sie bitte der Dokumentation des Look and Feels. TKPLAFUtility ermittelt automatisch den dazu gehörenden Klartextnamen und erzeugt geeignete Schlüssel in *swing.properties*.

3.2.3 Hinweise zum Umgang mit swing.properties

Die Datei *swing.properties* gestattet das einfache Setzen des Standard Look and Feels. Bei Bedarf kann sie leicht durch eine Anwendung gelesen und geparst werden. Allerdings ist die Datei in allen bisher verfügbaren Java-Versionen global. Da sie im Java-Installationsverzeichnis abgelegt wird, wirkt sich das Setzen eines Standard Look and Feels auf alle Benutzer eines Rechners aus. Zudem haben unter vielen Systemen nur Administratoren Schreibrechte auf das Java-Verzeichnis und damit die Datei *swing.properties*. Um solche Probleme zu beseitigen, müsste Sun in zukünftigen Java-Versionen benutzerspezifische *swing.properties*-Dateien einführen, die dann im Heimatverzeichnis des Benutzers abgelegt werden.

3.3 Das Windows Look and Feel

Sun liefert mit `com.sun.java.swing.plaf.windows.WindowsLookAndFeel` ein Look and Feel, das das Aussehen von Windows XP nachbildet. Im Großen und Ganzen ist diese Simulation auch gelungen, gerade in Details zeigen sich aber bis in die derzeit aktuelle Version 5.0 störende Unstimmigkeiten.

3.3.1 Unterschiede zum Vorbild

Um Ihnen diese Unstimmigkeiten in der Umsetzung zu verdeutlichen, greife ich auf die in Abbildung 3.7 gezeigte Anwendung *SwingSet2* zurück, die Sun mit dem Java Development Kit ausliefert. Sie finden sie im Verzeichnis `demo\jfc\SwingSet2` des SDK-Installationsverzeichnisses.

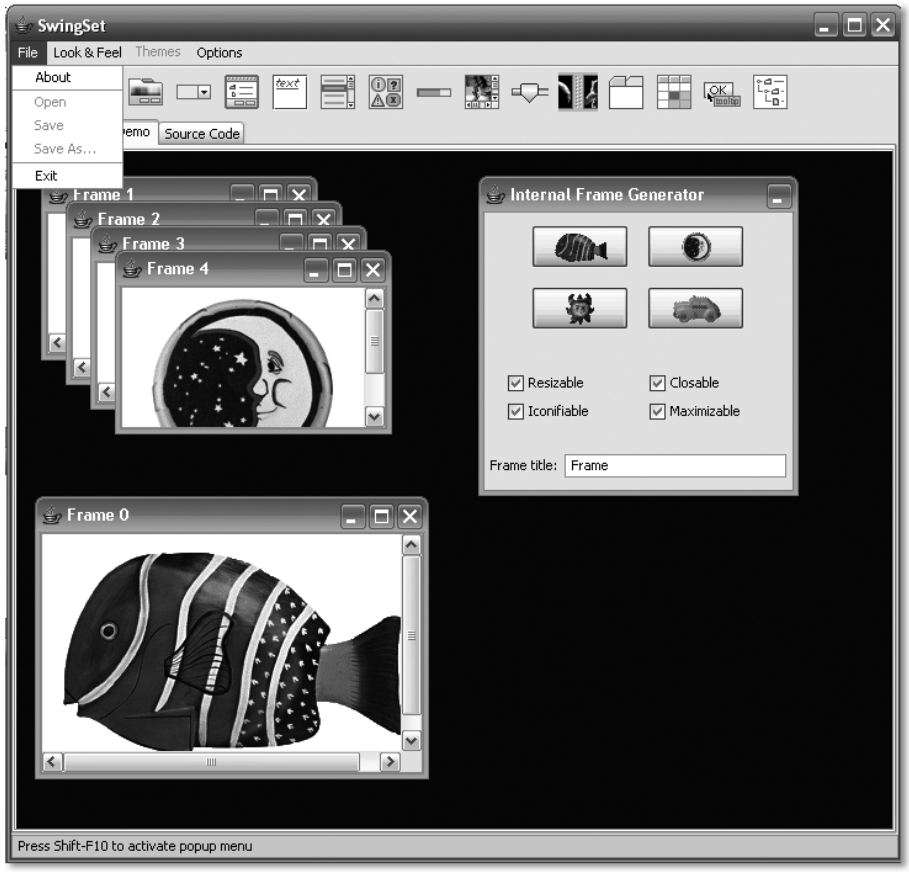

Abbildung 3.7 Die Beispiel-Anwendung SwingSet2

Um die Probleme am eigenen Rechner nachzuvollziehen, schalten Sie bitte zunächst im Menü **Look & Feel** auf **Windows Style Look & Feel** um. Wenn Sie anschließend mit dem Mauszeiger über nicht anwählbare Menüeintrage fahren, werden diese nicht farbig unterlegt. Native Windows-Anwendungen zeigen hier ein anderes Verhalten. Wie Sie in Abbildung 3.8 sehen, ist der Eintrag **Von Scanner oder Kamera** blau unterlegt, obwohl der Menüeintrag nicht ausgewählt werden kann.

Abbildung 3.8 Ein nicht anwählbarer Menüeintrag in Microsoft Paint

Eine weitere Diskrepanz zeigt sich, wenn Sie einen nicht anwählbaren Menüeintrag anklicken. In *Paint* bleibt das Menü geöffnet, wohingegen es sich in *SwingSet2* schließt.

Abbildung 3.9 Ein Kontextmenü nach Klick auf einen Slider

Abbildung 3.9 zeigt ein Kontextmenü, das jede native Windows-Anwendung bietet, wenn Sie mit der Maus über einen Schiebebalken oder Scrollpfeil eines

Fensters fahren und die rechte Maustaste drücken. Leider zeigen Swing-Anwendungen dieses nützliche Verhalten nicht.

Auf den ersten Blick mögen die vorgestellten Unstimmigkeiten eher unbedeutend wirken. Und natürlich sollte die eigentliche Funktionalität Ihrer Anwendung im Vordergrund stehen. Auf der anderen Seite kann unerwartetes oder ungewohntes Verhalten eines Programms den Arbeitsfluss signifikant stören. Ganz sicher wird jeder Anwender, der sich an eine Funktion gewöhnt hat (beispielsweise das in Abbildung 3.9 gezeigte Kontextmenü) ihr Fehlen bald bemerken. Aus diesem Grund möchte ich Ihnen im nächsten Abschnitt ein Projekt vorstellen, das eine ganze Reihe von Fehlern im Windows Look und Feel beseitigt und das Sie zudem mit minimalem Aufwand in Ihren eigenen Programmen verwenden können.

3.3.2 Das WinLAF-Projekt

Mit dem *WinLAF*-Projekt hat es sich dessen Initiator Brian Duff zum Ziel gesetzt, die Unzulänglichkeiten *des Windows Look and Feels* aufzudecken und zu beseitigen. Auf der Projekt-Homepage[2] steht ein *.jar*-Archiv zum Download bereit, das ein auf dem *Windows Look and Feel* basierendes, aber um viele Fehler bereinigtes Look and Feel beinhaltet.

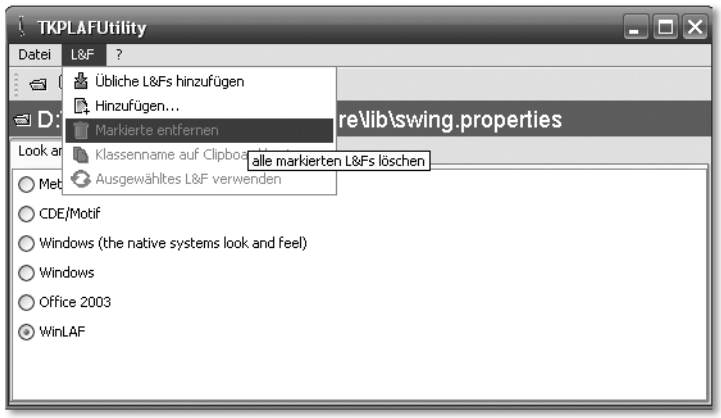

Abbildung 3.10 WinLAF stellt nicht anwählbare Menüeinträge richtig dar.

Beispielsweise werden bei Verwendung des *WinLAF*-Look and Feels nicht anwählbare Menüeinträge richtig dargestellt. Ein Mausklick darauf blendet das Menü zudem nicht mehr aus. Des Weiteren verhilft *WinLAF* den Schiebebalken und Scrollpfeilen von Fenstern zu einem Kontextmenü.

2 https://winlaf.dev.java.net/

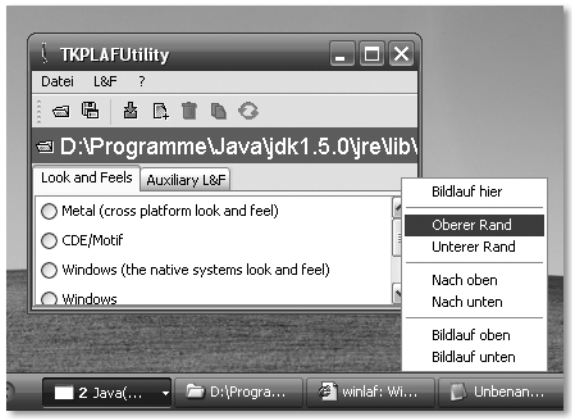

Abbildung 3.11 Fensterelemente von Swing-Anwendungen erhalten durch WinLAF ein Kontextmenü

Um *WinLAF* in eigenen Programmen zu verwenden, müssen Sie nur dafür sorgen, dass die Java-Laufzeitumgebung die Klassen des *.jar*-Archivs *winlaf-0.5.1.jar* findet. Ausführliche Hinweise hierzu gibt Kapitel 1, *Installation und Konfiguration*. Anschließend erweitern Sie *swing.properties* um entsprechende Einträge (beispielsweise mit TKPLAFUtility oder wie in Abschnitt 3.2.1 beschrieben) oder rufen in Ihrem Programm `UIManager.setLookAndFeel()` auf.

In den folgenden Abschnitten werden einige weitere Look and Feels vorgestellt, die Sie in Ihre eigene Programme integrieren können.

3.4 Weitere interessante Look and Feels

Um Look and Feels in Ihren Programmen zu verwenden, stehen Ihnen die drei in Abschnitt 3.1 gezeigten Vorgehensweisen zur Verfügung, nämlich festes Verdrahten des Look und Feels in der Anwendung durch den Aufruf von `UIManager.setLookAndFeel()`, das Belegen von `swing.defaultlaf` in der Datei *swing.properties* oder aber das Setzen der System Property `swing.defaultlaf`. Unabhängig davon, welche Variante Sie wählen, müssen Sie allerdings noch dafür sorgen, dass die Java-Laufzeitumgebung die Look and Feel-Klassen auch findet. Hier können Sie zwischen den zwei in Kapitel 1, *Installation und Konfiguration*, dargestellten Alternativen wählen, nämlich dem Kopieren der betreffenden *.jar*-Archive in das Java-Erweiterungsverzeichnis oder aber das explizite Hinzufügen zum Klassenpfad Ihrer Anwendung.

3.4.1 Die OfficeLnFs Look and Feels

Soll Ihre Anwendung wie Office XP oder 2003 aussehen, steht Ihnen das *OfficeLnFs*-Projekt[3] auf *Sourceforge* zur Verfügung. Das herunterladbare *.jar*-Archiv enthält zwei Look and Feels, die die Benutzeroberfläche der beiden Office-Versionen nachbilden. Eine dritte Variante simuliert die Komponenten von *Visual Studio 2005*. Des Weiteren sind die wichtigsten Icons aus den Toolbars von Word, Excel und Co. enthalten. *OfficeLnFs* basiert auf dem in Swing enthaltenen *Windows Look and Feel*. Leider erbt es damit auch einige der in Abschnitt 3.3.1 geschilderten Probleme.

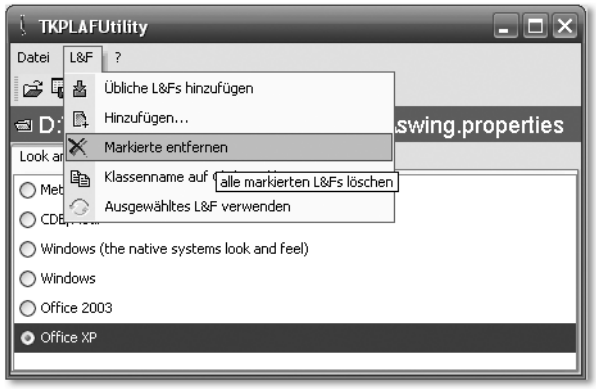

Abbildung 3.12 Das Office XP Look and Feel

In Abbildung 3.12 sehen Sie TKPLAFUtility unter Verwendung des *Office XP Look and Feels*. Der voll qualifizierte Klassenname lautet `org.fife.plaf.OfficeXP.OfficeXPLookAndFeel`.

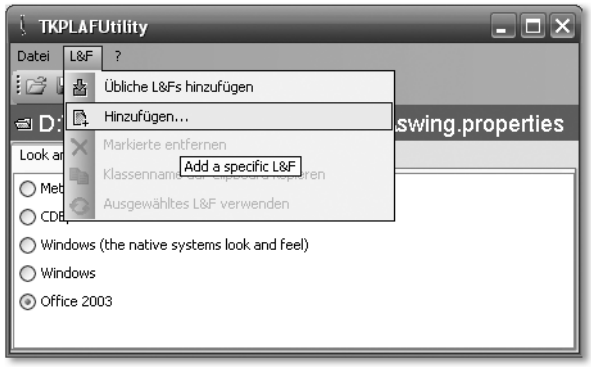

Abbildung 3.13 Das Office 2003 Look and Feel

3 http://officelnfs.sourceforge.net/

Abbildung 3.13 zeigt das *Office 2003 Look and Feel*. Um es in eigenen Programmen zu verwenden, müssen Sie als voll qualifizierten Klassennamen `org.fife.plaf.Office2003.Office2003LookAndFeel` angeben. Möchten Sie das *Visual Studio 2005 Look and Feel* in Ihren Anwendungen einsetzen, tragen Sie als Klassennamen bitte `org.fife.plaf.VisualStudio2005.VisualStudio2005LookAndFeel` ein. Abbildung 3.1 zeigt TKPLAFUtility im Gewand des *Visual Studio 2005 Look and Feels*.

TKPLAFUtility wechselt passend zum ausgewählten Look und Feel auch die Icons seiner Toolbar und der Menüleiste. Dabei erkennt es ein installiertes *OfficeLnFs* und greift auf dessen eingebettete Grafiken zu. Um diese in Ihre Anwendung zu integrieren, können Sie folgendermaßen vorgehen.

```
URLClassLoader cl = (URLClassLoader)getClass().getClassLoader();
      URL url = cl.findResource("org/fife/plaf/OfficeXP/images/
cut.gif");
      JButton cutButton = new JButton(new ImageIcon(url));
```

Das Beispiel erzeugt ein `JButton`-Objekt, welches das zur Aktion **Ausschneiden** passende Bild enthält. *TKPLAFUtility* verwendet hierfür die von mir entwickelte Klasse `de.thomaskuenneth.toolbaricons.ToolbarIcons`. Diese enthält eine ganze Reihe von `getxyzIcon()`-Methoden, die Sie verwenden können, um die Toolbars Ihrer Anwendungen zu füllen. Ausführliche Informationen zu der Klasse `ToolbarIcons` und der unterstützten Look and Feels entnehmen Sie bitte der Dokumentation, die sich mit den Quelltexten auf der Begleit-CD zum Buch befindet.

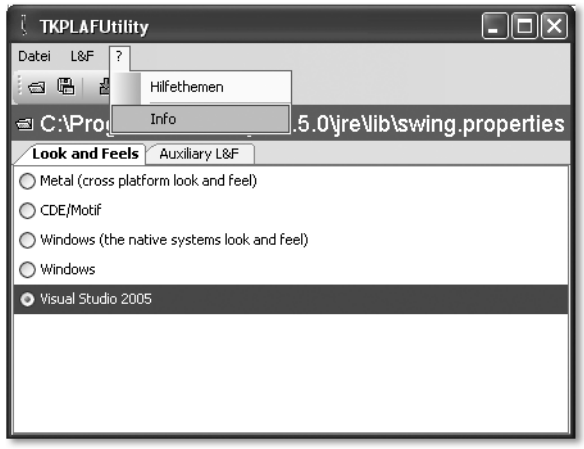

Abbildung 3.14 Das Visual Studio 2005 Look and Feel

3.4.2 JGoodies Looks

JGoodies Looks[4] von Karsten Lentzsch besteht aus insgesamt vier Look and Feels, einem *Windows Look and Feel* sowie der *Plastic*-Familie. Das *Looks .jar*-Archiv kann im Prinzip wie gewohnt im Java-Erweiterungsverzeichnis abgelegt werden. Allerdings rät dessen Autor ausdrücklich davon ab und empfiehlt stattdessen, den Klassenpfad der eigenen Anwendung entsprechend zu erweitern.

Plastic steht in drei Varianten zur Verfügung, *Plastic*, *Plastic3D* sowie *PlasticXP*. Alle drei unterstützen Farbthemen und bieten eine Vielzahl an Konfigurationsmöglichkeiten. Beispielsweise lässt sich das Aussehen von Registerkarten verändern und bei den Fokusfarben können Sie zwischen kontrastreicher und -armer Darstellung wählen.

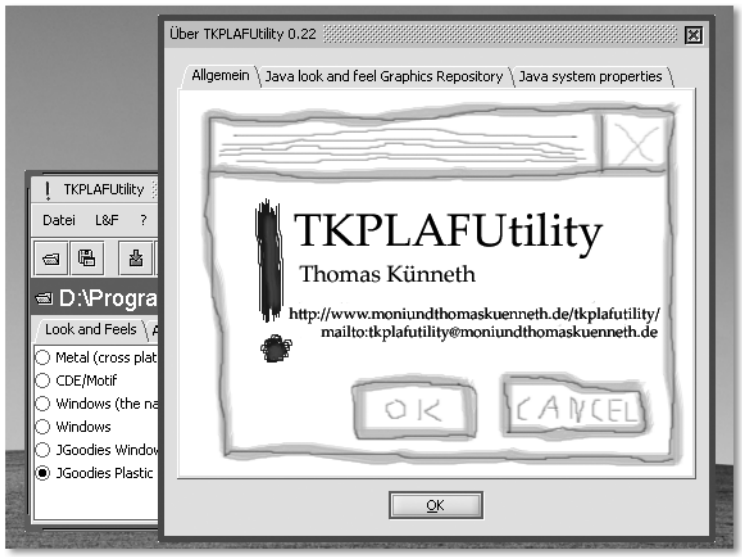

Abbildung 3.15 JGoodies Plastic 3D

Wie Sie am Beispiel von *OfficeLnFs* gesehen haben, machen erst die passenden Grafiken in Toolbars und Menüleisten die Illusion einer nativen Anwendung im Stile von *Office XP* oder *Visual Studio 2005* komplett. Generell runden schöne Icons das optische Erscheinungsbild eines Programms ab. Für Kontinuität und für Vertrautheit sorgen *Icon Sets*, die ich Ihnen im folgenden Abschnitt vorstelle.

4 http://www.jgoodies.com/freeware/looks/

3.5 Icon Sets

Icon-Sammlungen enthalten Bilder für häufig verwendete Funktionen, beispielsweise Ausschneiden, Kopieren, Laden oder Drucken. Diese Grafiken werden in Toolbars und Menüs verwendet. Da Toolbars oft benutzte Befehle enthalten sollten, ist der Wiedererkennungswert der eingesetzten Symbole hier besonders wichtig. Aber auch in Menüs profitieren Anwender von vertrauten Bildern. Deshalb sollten Sie Ihre Programme möglichst mit bekannt wirkenden Icons versehen.

3.5.1 Java look and feel Graphics Repository

Das *Java look and feel Graphics Repository* ist eine Sammlung von Toolbar-Grafiken, die zwar speziell für das »alte« *Java Look and Feel* – also *Metal* – entwickelt wurden, aber natürlich ebenso gut in Verbindung mit anderen Look und Feels verwendet werden können. Sun hat das Paket unter **java.sun.com/developer/techDocs/hi/repository/** zum kostenlosen Download bereitgestellt. Die gezippte Datei *jlfgr-1_0.zip* enthält ein *.jar*-Archiv, das Sie mit einem geeigneten Tool oder über die Kommandozeile (mit *jar.exe* des Java Development Kits) entpacken können, um sich einen Überblick über die enthaltenen Grafiken zu verschaffen.

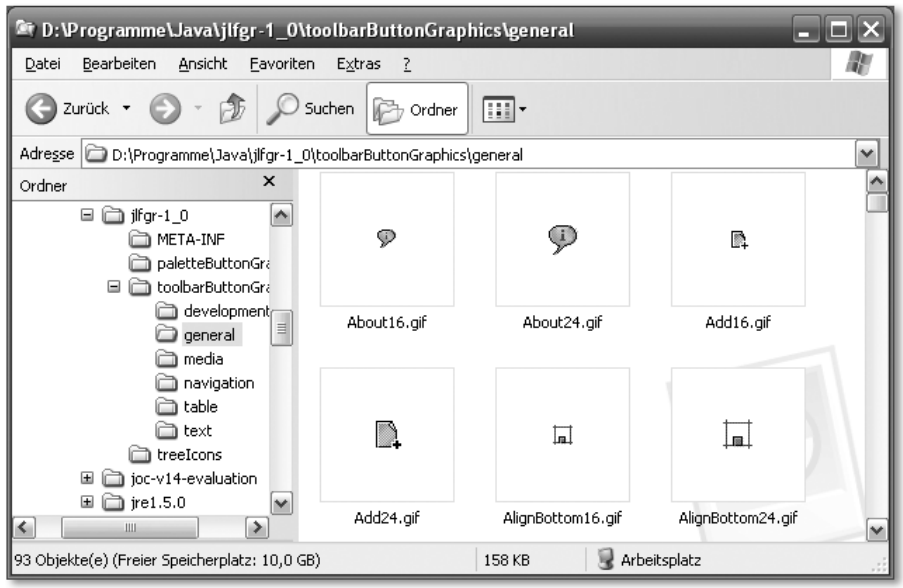

Abbildung 3.16 Das Java look and feel Graphics Repository

Die Grafiken liegen in zwei Größen vor: 16 x 16 und 24 x 24 Pixel. Welche Variante Sie wählen sollten, ist davon abhängig, wo Sie die Bilder einsetzen. In Toolbars machen sich größere Icons sehr gut, bei der Verwendung in Menüleisten sollten Sie aber auf die kleinere Variante zurückgreifen. Die Praxis zeigt nämlich, dass zahlreiche Look and Feels bei der Darstellung von Menüs Schwierigkeiten mit zu großen Icons haben.

Die in Abschnitt 3.4.1 vorgestellte Klasse `ToolbarIcons` kann auf die Icons des *Java look and feel Graphics Repository* zugreifen. Hierzu muss nur das *.jar*-Archiv im Klassenpfad Ihrer Anwendung oder im Java Erweiterungsverzeichnis abgelegt werden. Um den Problemen mit bestimmten Look and Feels vorzubeugen, liefert `ToolbarIcons` übrigens immer die 16 x 16 Pixel großen Icons.

3.5.2 Icon Collection auf Sourceforge

Eine weitere Quelle für sehr brauchbare Toolbar-Icons ist *Dean S. Jones' Icon Collection*[5]. Die Grafiken liegen in unterschiedlichen Größen und Formaten vor, allerdings ist nicht jedes Bild in allen Größen verfügbar.

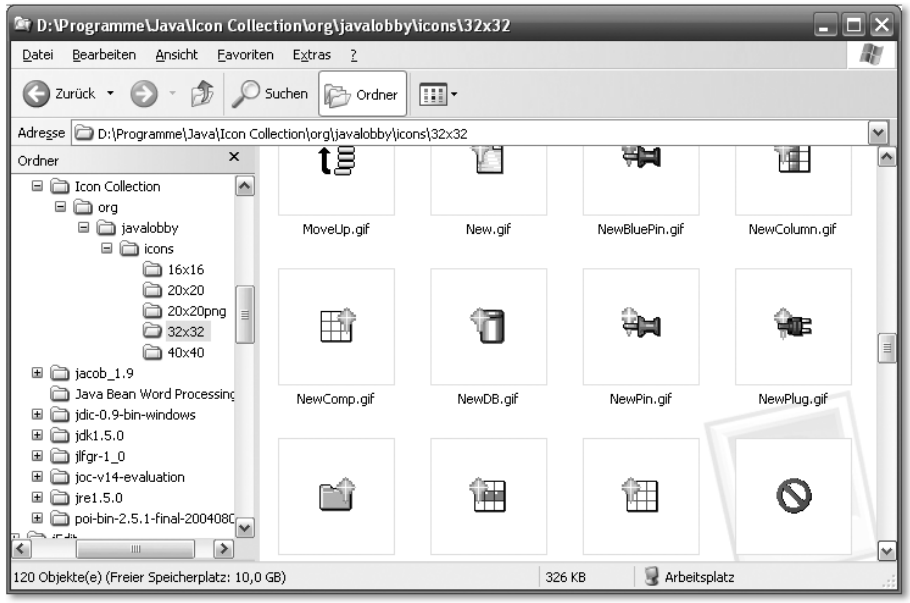

Abbildung 3.17 Dean S. Jones' Icon Collection

5 http://sourceforge.net/projects/icon-collection/

4 Zusätzliche Komponenten für die Benutzeroberfläche

4 Zusätzliche Komponenten für die Benutzeroberfläche

Swing stellt eine beeindruckende Zahl an Klassen zur Verfügung. Dennoch lassen sich manche Windows-Elemente nur mit erheblichen Aufwand nachbilden. Ich zeige Ihnen Bibliotheken, die diese Arbeit übernehmen.

Mit den *Java Foundation Classes* hat Sun den Entwicklern eine Vielzahl an Klassen an die Hand gegeben, um grafische Benutzeroberflächen zu implementieren. Viele Komponenten, die man als Programmierer im *Abstract Window Toolkit* schmerzlich vermisst hat, wurden seiner Zeit durch das *Projekt Swing* zur Verfügung gestellt. Einige Beispiele unter vielen sind `JTable`, `JTree` und `JTabbedPane`. Zweifellos lassen sich mit Swing Anwendungen mit modernem, ansprechendem Äußeren entwickeln. Ausführliche Tipps hierzu finden Sie in Kapitel 3, *Feintuning der Benutzeroberfläche*. Allerdings wurden seit der Einführung der *Java Foundation Classes* nicht nur unter Windows neue Bedienelemente und Konzepte eingeführt. Zu nennen sind hier unter anderem vieleckige oder gerundete Fenster ohne Randelemente, Komponenten zum Auswählen von Zeichensätzen und Verzeichnissen sowie »Tipp des Tages«-Dialoge. Solche Funktionen sind mit Swing durchaus machbar, erfordern aber erheblichen Aufwand, den jeder Entwickler erneut betreiben muss. Es sei denn, er greift auf Klassenbibliotheken zurück, die die beschriebenen Elemente und Konzepte zur Verfügung stellen. In den folgenden Abschnitten werde ich Ihnen einige solcher Bibliotheken vorstellen und zeigen, wie deren Klassen eingesetzt werden.

4.1 Nachrichtenfenster mit JToaster

In Kapitel 7, *JDesktop Integration Components*, zeige ich Ihnen, wie Sie kleine Sprechblasen erscheinen lassen, deren Spitzen auf ein Icon im Infobereich der Taskleiste zeigen. Eine andere Möglichkeit Informationen einzublenden ist es, Nachrichtenfenster vom unteren Bildschirmrand aus langsam nach oben zu schieben. Beispielsweise machen Microsofts Instant Messaging-Programme *Windows Messenger* und *MSN Messenger* hiervon Gebrauch. Wie dies aussieht, zeigt Abbildung 4.1.

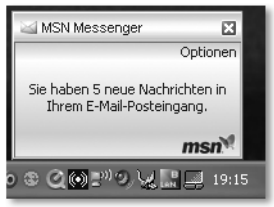

Abbildung 4.1 Infomeldung des MSN Messeners

Mittlerweile präsentieren viele Anwendungen Informationen auf diese Weise, zum Beispiel Benachrichtigungen über erfolgreiche Aktualisierungen von Viren-Signaturen oder Titelinformationen des gerade begonnenen Stücks. Der Charme dieser Vorgehensweise ist es, den Anwender nicht in seiner momentanen Tätigkeit zu unterbrechen. Damit er die Nachricht trotzdem zur Kenntnis nimmt, wird während der Einblendung oft zusätzlich ein Jingle abgespielt. Im Falle der Musikwiedergabe ist eine zusätzliche akustische Rückmeldung natürlich nicht nötig, da der Benutzer den Titelwechsel von sich aus bemerkt.

4.1.1 Download und Installation

Das Projekt *JToaster*[1] ermöglicht auch Java-Programmen, solche Infotäfelchen anzuzeigen. Laden Sie hierzu bitte die Datei *jtoaster-1.0.4.jar* herunter.[2] Die zum Zeitpunkt der Drucklegung aktuelle Version finden Sie auch auf der Begleit-CD zum Buch. Um *JToaster* in Ihren Programmen zu verwenden, müssen Sie dafür sorgen, dass die Java-Laufzeitumgebung die Datei *jtoaster-1.0.4.jar* findet. Sie können hierzu den Klassenpfad Ihrer Anwendung entsprechend erweitern oder das Archiv in das Erweiterungsverzeichnis kopieren. Ausführliche Informationen hierzu finden Sie in Kapitel 1, *Installation und Konfiguration*.

4.1.2 JToaster in der Praxis

Wie Sie *JToaster* verwenden, zeige ich Ihnen anhand des Programms *JToasterDemo1*. Hier der entsprechende Quelltext.

```
package javafuerwindows.gui;
import com.nitido.utils.toaster.Toaster;
import java.awt.Color;
import java.net.URL;
import javax.swing.ImageIcon;
```

1 http://jtoaster.sourceforge.net/
2 http://sourceforge.net/projects/jtoaster/

```
public class JToasterDemo1 {

    public JToasterDemo1() {
        URL url = ClassLoader.getSystemResource(
                            "javafuerwindows.png");
        ImageIcon ii = new ImageIcon(url);
        Toaster toaster = new Toaster();
        String text = "Anzeigedauer: " +
                toaster.getDisplayTime() +
                "\nRand: " + toaster.getMargin() +
                "\nSchrittzeit: " + toaster.getStepTime() +
                "\nSchrittweite: " + toaster.getStep();
        toaster.setDisplayTime(10000);
        toaster.setMessageColor(Color.red);
        toaster.setToasterColor(Color.lightGray);
        toaster.setBorderColor(Color.blue);
        toaster.showToaster(ii, text);
    }

    public static void main(String[] args) {
        new JToasterDemo1();
    }
}
```

Listing 4.1 JToasterDemo1.java

Die gesamte Funktionalität wird durch die Klasse `com.nitido.utils.toaster.Toaster` zur Verfügung gestellt. Nachdem Sie eine Instanz erzeugt haben, können Sie das Erscheinungsbild des Nachrichtenfensters Ihren Vorstellungen anpassen, indem Sie beispielsweise andere Farben vorgeben oder den für die Darstellung verwendeten Zeichensatz ändern. Sie starten die Einblendung durch Aufruf der Methode `showToaster()`. In Abbildung 4.2 sehen Sie das durch *JToasterDemo1* erzeugte Infotäfelchen.

Abbildung 4.2 JToaster-Infotäfelchen

Sofern sie sinnvoll eingesetzt werden, sind die durch *JToaster* realisierten Infotäfelchen eine Bereicherung für viele Programme. Bitte prüfen Sie bei einem Einsatz der Klasse aber, ob während der Anzeige ein akustischer Hinweis auf die Nachricht angebracht ist.

Im folgenden Abschnitt stelle ich Ihnen eine Klassenbibliothek vor, die eine ganze Reihe leistungsfähiger Komponenten zur Verfügung stellt. Mit diesen können Sie, wie Sie gleich sehen werden, das Erscheinungsbild Ihrer Anwendung ganz erheblich aufwerten.

4.2 L2FProd.com Common Components

Die Komponenten des Projekts *L2FProd.com Common Components*[3] sind Open Source und stehen unter der *Apache- License*-Version 2.0, können also kostenlos in eigenen Programmen eingesetzt und weitergegeben werden. Laden Sie zunächst bitte die Datei *l2fprod-common-0.2-dev-20050918.zip* von der Projekt-Homepage herunter und entpacken Sie sie in einem beliebigen Verzeichnis. Die zum Zeitpunkt der Drucklegung aktuelle Version finden Sie auch auf der Begleit-CD zum Buch. Um die Klassenbibliothek in Ihren Programmen zu verwenden, müssen Sie dafür sorgen, dass die Java-Laufzeitumgebung die *.jar*-Archive des Verzeichnisses *lib* findet. Ausführliche Hinweise hierzu finden Sie in Kapitel 1, *Installation und Konfiguration*.

Bitte beachten Sie, dass Sie anstelle des Sammelarchivs *l2fprod-common-all.jar* bei Bedarf auch nur diejenigen Archive verwenden können, die die benötigten Komponenten enthalten, beispielsweise *l2fprod-common-outlookbar.jar* für die Klasse `JOutlookBar`. Denken Sie auch daran, die Dokumentation im Javadoc-Format in Ihrer Entwicklungsumgebung zu registrieren, damit Sie bei Bedarf darauf zugreifen können.

4.2.1 Tipp des Tages-Dialoge

In diesem Abschnitt stelle ich Ihnen die Klasse `com.l2fprod.common.swing.JTipOfTheDay` vor, mit der Sie so genannte »Tipp des Tages«-Dialoge realisieren können. Die Idee dahinter ist, den Anwender Schritt für Schritt mit interessanten Funktionen eines Programms vertraut zu machen. Nach jedem Start öffnet sich ein kleines Fenster, das verschiedene Aspekte in der Art »*Wussten Sie, dass ...*« vorstellt. Wie dies aussehen kann, zeigt Abbildung 4.3.

3 http://common.l2fprod.com/

Abbildung 4.3 »Tipp des Tages«-Dialog

Der in Abbildung 4.3 gezeigte Dialog wird durch das Programm *L2ProdDemo1* erzeugt. Hier sehen Sie den Quelltext.

```
package javafuerwindows.gui;
import com.l2fprod.common.swing.JTipOfTheDay;
import com.l2fprod.common.swing.TipModel;
public class L2FProdDemo1 implements TipModel {
    private static final String [] tips = {
        "Das Buch Java für Windows
         enthält eine Begleit-CD.",
               "Java ist auch eine Insel",
               "Das Buch ist im Verlag
                Galileo Computing erschienen."
    };
    private int param;

    public L2FProdDemo1() {
        JTipOfTheDay t = new JTipOfTheDay(this);
        t.showDialog(null);
    }

    public static void main(String [] args) {
        new L2FProdDemo1();
    }

    /*
     * TipModel interface
     */
```

```
    public TipModel.Tip getTipAt(int param) {
        this.param = param;
        return new TipModel.Tip() {
            public Object getTip() {
                return L2FProdDemo1.this.tips
                                    [L2FProdDemo1.this.param];
            }

            public String getTipName() {
                return "Tip Nr. " + L2FProdDemo1.this.param;
            }
        };
    }

    public int getTipCount() {
        return tips.length;
    }
}
```

Listing 4.2 L2FProdDemo1.java

Um einen »Tipp des Tages«-Dialog anzuzeigen, instanziieren Sie zunächst ein Objekt der Klasse `com.l2fprod.common.swing.JTipOfTheDay` und rufen deren Methode `showDialog()` auf. Welche Tipps zur Verfügung stehen, erfährt `JTipOfTheDay` von einem Objekt, dass das Interface `com.l2fprod.common.swing.TipModel` implementiert. Dessen Methoden `getTipCount()` und `getTipAt()` liefern die Zahl der Tipps bzw. die Tipps selbst. Tipps wiederum werden durch Objekte dargestellt, die das Interface `TipModel.Tip` implementieren. Sie können entweder für eine eigene Implementierung sorgen (analog zu *L2FProdDemo1*) oder auf die Klasse `com.l2fprod.common.swing.tips.DefaultTip` zurückgreifen.

> Eine weitere interessante Hilfsklasse ist `com.l2fprod.common.swing.tips.TipLoader`, mit deren Hilfe Sie Tipps aus Instanzen von `java.util.Properties` beziehen können.

Bei vernünftigem Einsatz sind »Tipp des Tages«-Dialoge eine sinnvolle Erweiterung für viele Anwendungen, die zudem sehr einfach und elegant realisiert werden können. Auch im folgenden Abschnitt stelle ich Ihnen ein Konzept vor, das sehr oft in Windows-Anwendungen anzutreffen ist.

4.2.2 Verzeichnisse auswählen

Häufig soll in einem Programm keine Datei, sondern ein Verzeichnis ausgewählt werden. Ein Beispiel hierfür ist **Datei kopieren** aus den **Datei- und Ordneraufgaben** des *Windows Explorers*. Wenn Sie diese Funktion aufrufen, sehen Sie den in Abbildung 4.4 gezeigten Dialog *Elemente kopieren*. Zwar bietet die Swing-Klasse `JFileChooser` die Möglichkeit, ausschließlich Verzeichnisse auszuwählen, allerdings entspricht das Aussehen weiterhin einer normalen Dateiauswahl. Die *L2FProd.com Common Components* bieten mit `com.l2fprod.common.swing.JDirectoryChooser` eine von `JFileChooser` abgeleitete Klasse, die dieses Problem behebt.

Abbildung 4.4 Der Dialog Elemente kopieren des Windows Explorers

Lassen Sie uns zunächst einen Blick auf die Funktionsweise dieser Klasse werfen. Sehen Sie sich hierzu bitte das äußerst kurze Programm *L2FProdDemo2* an.

```
package javafuerwindows.gui;
import com.l2fprod.common.swing.JDirectoryChooser;
public class L2FProdDemo2 {
    public static void main(String [] args) {
        JDirectoryChooser dc = new JDirectoryChooser();
        dc.setDialogTitle("bitte ein Verzeichnis wählen");
        if (dc.showOpenDialog(null) ==
                        JDirectoryChooser.APPROVE_OPTION) {
            System.out.println(dc.getSelectedFile());
        }
    }
}
```

Listing 4.3 L2FProdDemo2.java

Da `JDirectoryChooser` von `JFileChooser` ableitet, stehen uns alle Methoden der Elternklasse zur Verfügung. Ich verwende beispielsweise `setDialogTitle()`, um den Dialog-Titel zu setzen. Auch das Anzeigen der Auswahl verläuft analog, hier durch Aufruf von `showOpenDialog()`. In Abbildung 4.5 sehen Sie, wie ein durch `JDirectoryChooser` erzeugter Dialog aussieht.

Abbildung 4.5 Der durch L2FProdDemo2 erzeugte Dialog

Auch `JDirectoryChooser` ist eine sehr interessante Klasse. Mit ihrer Hilfe wirken Java-Anwendungen noch ein Stückchen vertrauter, da der Windows-Anwender bei der Auswahl von Verzeichnissen einen solchen Dialog erwartet, nicht aber eine normale Dateiauswahl.

4.2.3 Aufgaben mit JTaskPane

In diesem Abschnitt beschäftige ich mich mit so genannten *Aufgaben*. Gruppen von zusammen gehörenden Aktionen werden hierzu in einem Element kombiniert, das sich bei Bedarf auf- und zuklappen lässt.

Beispiele für solche Aufgaben sind die *Datei- und Ordneraufgaben*, *Andere Orte* und *Details* des Windows Explorers, die Sie in Abbildung 4.6 sehen. Diese ergeben zusammen eine Aufgaben-Leiste.

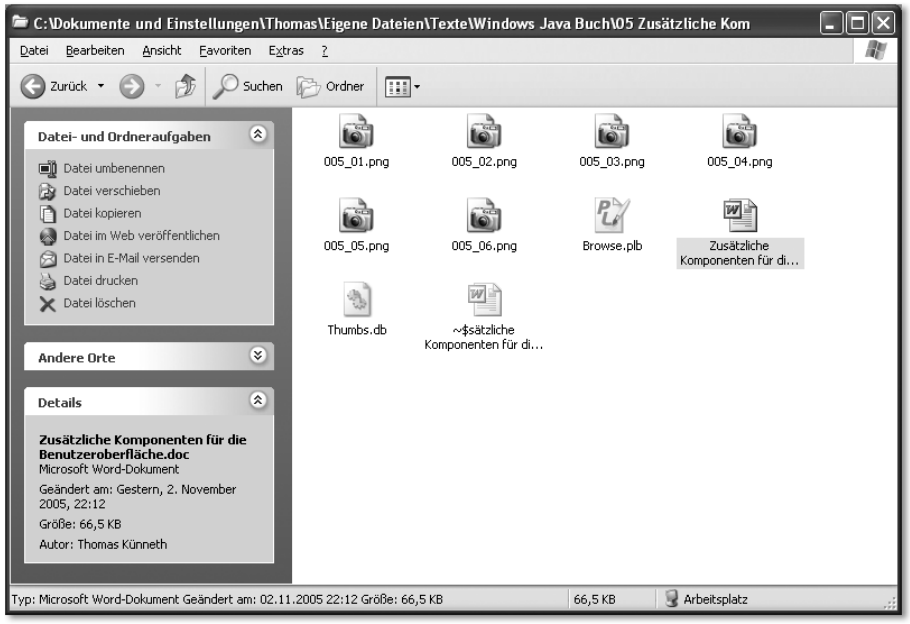

Abbildung 4.6 Aufgaben in einem Fenster des Windows Explorers

Die *L2FProd.com Common Components* stellen mit den Klassen `com.l2fprod.common.swing.JTaskPane` und `l2fprod.common.swing.JTaskPaneGroup` eine Implementierung dieses Bedienelements zur Verfügung. Das Programm *L2FProdDemo3* zeigt, wie Sie die Klassen in Ihren Anwendungen einsetzen.

```
package javafuerwindows.gui;
import com.l2fprod.common.swing.JTaskPane;
import com.l2fprod.common.swing.JTaskPaneGroup;
import java.awt.BorderLayout;
import java.net.URL;
import javax.swing.ImageIcon;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JScrollPane;
public class L2FProdDemo3 {
    public L2FProdDemo3() {
        JFrame f = new JFrame("L2FProdDemo3");
        JTaskPane tp = new JTaskPane();
        JTaskPaneGroup tpg = new JTaskPaneGroup();
        tpg.setTitle("Datei-Aktionen");
```

```
        URL url = ClassLoader.getSystemResource(
                                    "javafuerwindows.png");
        ImageIcon ii = new ImageIcon(url);
        tpg.add(new JLabel("Datei kopieren",
                           ii, JLabel.LEFT));
        tp.add(tpg);
        f.add(new JScrollPane(tp), BorderLayout.CENTER);
        f.pack();
        f.setVisible(true);
    }

    public static void main(String [] args) {
        new L2FProdDemo3();
    }
}
```

Listing 4.4 L2FProdDemo3.java

Zunächst erzeugen Sie eine Instanz der Klasse `JTaskPane`. Diese nimmt beliebig viele `JTaskPaneGroup`-Objekte auf. Mit deren Methode `add()` können Sie der *Aufgabe* schließlich Aktionen zuweisen. In Abbildung 4.7 sehen Sie die durch *L2FProdDemo3* erzeugte Aufgabenleiste. Sie besteht aus einer Aufgabe mit einer Aktion.

Abbildung 4.7 Die durch L2FProdDemo3 erzeugte Aufgaben-Leiste.

4.2.4 Outlook-Leisten

In diesem Abschnitt zeige ich Ihnen eine Komponente, deren Vorbild wahrscheinlich das erste Mal in Outlook implementiert wurde. Es handelt sich um eine schmale Leiste, die Schaltflächen stapelförmig angeordnet. Zwischen zwei oder oberhalb aller Schaltflächen befindet sich ein Bereich, der beliebig genutzt werden kann. Seine Größe ist abhängig von den Ausmaßen der Leiste und der Zahl der Schaltflächen. Dabei ist stets nur ein solcher Bereich zu sehen, alle anderen bleiben ausgeblendet. Das Anklicken einer Schaltfläche blendet den ihr zugeordneten Bereich ein. In Abbildung 4.8 sehen Sie die in Outlook 2003

implementierte Outlook-Leiste. Sie blendet die Bereiche einer Schaltfläche stets oberhalb des Schaltflächen-Stapels ein.

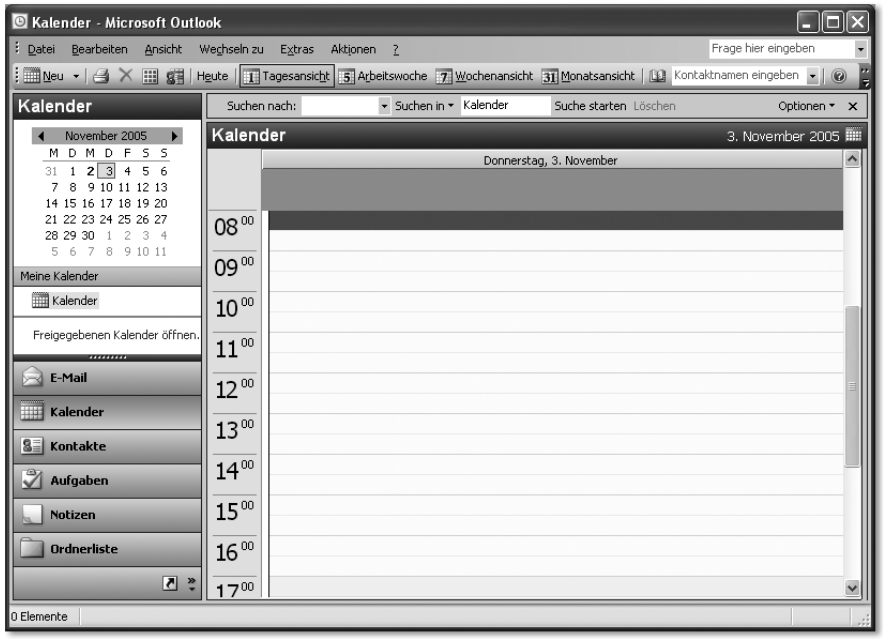

Abbildung 4.8 Eine Leiste mit Schaltflächen in Outlook 2003

Die Klasse `com.l2fprod.common.swing.JOutlookBar` stellt ein ähnliches Konzept auch für Java-Programme zur Verfügung. Sie leitet von `JTabbedPane` ab und wird analog zu dieser Klasse verwendet.

```
package javafuerwindows.gui;
import com.l2fprod.common.swing.JOutlookBar;
import java.awt.BorderLayout;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;
public class L2FProdDemo4 {
    public static void main(String [] args) {
        JOutlookBar b = new JOutlookBar();
        JFrame f = new JFrame("Test");
        for (int i = 1; i < 4; i++) {
            JPanel p = new JPanel();
            p.add(new JLabel("hallo, ich bin JPanel #"
                             + i));
```

```
            b.addTab("Ordner " + i, p);
        }
        f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        f.getContentPane().add(b, BorderLayout.WEST);
        f.pack();
        f.setBounds(100, 100, 500, 300);
        f.setVisible(true);
    }
}
```

Listing 4.5 L2FProdDemo4.java

Der erste Schritt besteht im Instanziieren der Klasse `com.l2fprod.common.swing.JOutlookBar`. Anschließend fügen Sie diesem Objekt beliebig viele neue Elemente hinzu. Wie bei der Elternklasse können Sie hierfür die Methode `addTab()` verwenden, der Sie bei Bedarf sogar ein Icon übergeben können. Abbildung 4.9 zeigt die durch *L2FProdDemo4* erzeugte `JOutlookBar`.

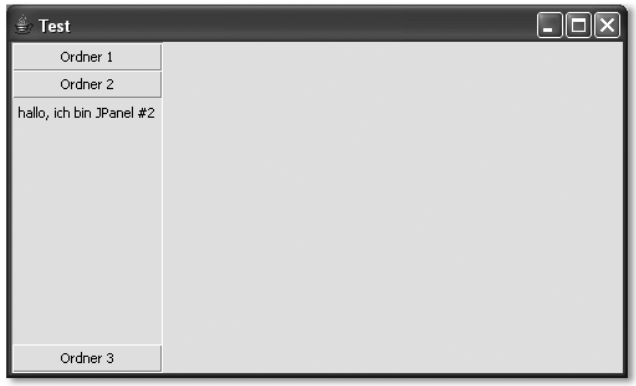

Abbildung 4.9 Die durch L2FProdDemo4 erzeugte Outlook-Leiste

Outlook-Leisten und Registerkarten verfolgen dasselbe Ziel, nämlich möglichst viele Informationen auf begrenztem Raum unterzubringen. Durch die Form ihrer grafischen Darstellung eignet sich `JOutlookBar` besonders für hierarchische Strukturen, beispielsweise Bäume oder Leisten mit Schaltflächen. Übrigens implementieren die *L2FProd.com Common Components* auch dieses Konzept in Form der Klasse `JButtonBar`. Ausführliche Beschreibungen zu dieser und allen übrigen Klassen finden Sie in der Dokumentation.

Noch ein Tipp: Das Archiv *l2fprod-common-all.jar* enthält eine Demonstration, die alle Komponenten vorstellt. Um sie zu starten, öffnen Sie bitte das Archiv durch Doppelklick.

Im nächsten Abschnitt beschäftigen wir uns mit einer fundamentalen Designfrage im Hinblick auf die Benutzeroberfläche Ihrer Anwendungen. Grundsätzlich haben Sie die Möglichkeit, mehrere Dokumente in gleichberechtigten Fenstern zu verwalten oder aber ein Hauptfenster anzulegen, das neben der Menüleiste die Dokumente enthält und diese in untergeordneten Fenstern anzeigt. Solche Programme werden *Multiple Document Interface*-Anwendungen genannt.

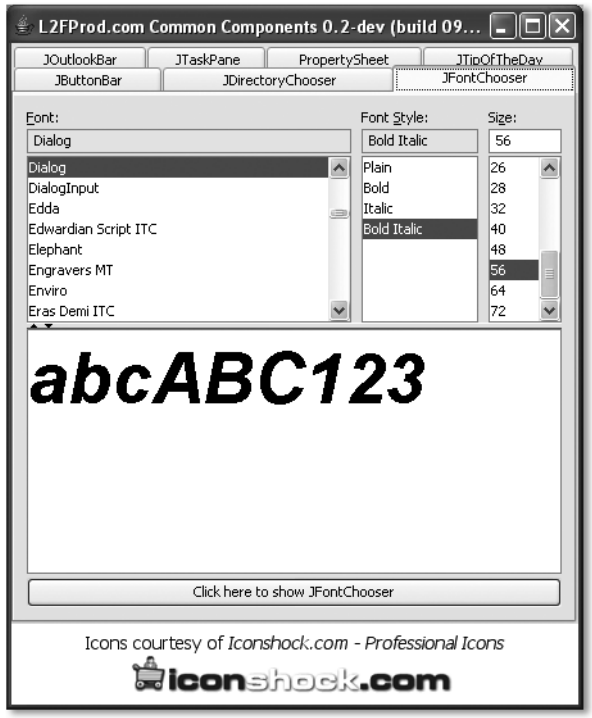

Abbildung 4.10 Die Zeichensatzauswahl der L2FProd.com Common Components

4.3 MDI-Anwendungen

Das *Multiple Document Interface* fasst die Dokumente einer Anwendung in einem Hauptfenster zusammen. Im Gegensatz dazu sind beim *Single Document Interface* alle Fenster eines Programms gleichberechtigt. Der Vorteil von SDI liegt darin, die Dokumente beliebig auf dem Bildschirm verschieben zu kön-

nen. Dies ist zwar auch bei MDI-Anwendungen möglich, aber nur in den Grenzen des Hauptfensters. Nimmt dies den gesamten Bildschirm ein, steht zwar für die Unterfenster ausreichend Platz zur Verfügung, allerdings bleiben Fenster anderer Anwendungen sowie der Desktop vollständig verdeckt.

Aus diesem Grund sind MDI-Anwendungen in den letzten Jahren aus der Mode gekommen. Etwas zu Unrecht, wie ich finde, denn es hat sich gezeigt, dass ab einer gewissen Zahl offener Fenster auch die Arbeit mit SDI-Anwendungen unübersichtlich wird. Deshalb wird MDI in einer abgewandelten Form, dem *Tabbed Document Interface*, wieder häufiger eingesetzt. Waren dessen Registerkarten zunächst vor allem in Browsern populär, greifen auch andere Anwendungen zunehmend auf diese Form der Benutzeroberfläche zurück.

Als Beispiel unter vielen sei der Editor *Notepad++* genannt, den ich Ihnen in Kapitel 2, *Entwicklungswerkzeuge*, ausführlich vorstelle. Grund genug also, auch MDI wieder mehr Beachtung zu schenken.

4.3.1 MDI-Anwendungen in Swing

Um in Swing eine Anwendung mit MDI zu realisieren, sind nur wenige Klassen nötig. Sehen Sie sich zunächst den Quelltext von *SwingMDIDemo* an.

```
package javafuerwindows.gui;
import javax.swing.JDesktopPane;
import javax.swing.JFrame;
import javax.swing.JInternalFrame;
public class SwingMDIDemo extends JFrame {
    private JDesktopPane dp;
    public SwingMDIDemo() {
        super("SwingMDIDemo");
        setDefaultCloseOperation(EXIT_ON_CLOSE);
        dp = new JDesktopPane();
        setContentPane(dp);
        setBounds(100, 100, 400, 300);
        neuesFenster("Fenster 1", 40, 40, 200, 100);
        neuesFenster("Fenster 2", 60, 60, 220, 120);
        neuesFenster("Fenster 3", 80, 80, 220, 120);
        setVisible(true);
    }
```

```
    private void neuesFenster(String titel,
                              int x, int y, int w, int h) {
        JInternalFrame iframe = new JInternalFrame(titel,
                                    true, true, true, true);
        iframe.setBounds(x, y, w, h);
        iframe.setVisible(true);
        dp.add(iframe);
    }

    public static void main(String [] args) {
        new SwingMDIDemo();
    }
}
```

Listing 4.6 SwingMDIDemo.java

Das Hauptfenster wird mit der Klasse `JFrame` realisiert. Die Dokumentfenster sind Instanzen der Klasse `JInternalFrame`. Sie werden von einem `JDesktop-Pane`-Objek verwaltet. Ein Problem der Swing-Implementierung des MDI zeigt sich, wenn Sie eines der Dokumentfenster auf seine maximale Größe bringen.

Abbildung 4.11 Das Programm SwingMDIDemo

Wie Sie in Abbildung 4.11 sehen, behält das Dokumentfenster seine Randelemente. Als Windows-Anwender erwarten Sie allerdings ein anderes Verhalten, das Abbildung 4.12 zeigt.

Die Dokumentfenster haben keine Titelzeile. Außerdem sind die Kontrollelemente zum Schließen, Minimieren und Wiederherstellen der ursprünglichen Fenstergröße in die Menüleiste gewandert. Die Klassenbibliothek *AceMDI* bildet dieses Verhalten nach. Ein weiteres Schmankerl: Sobald Sie ein Fenster auf seine maximale Größe bringen, erscheinen die Dokumente Ihrer Anwendung als Registerkarten.

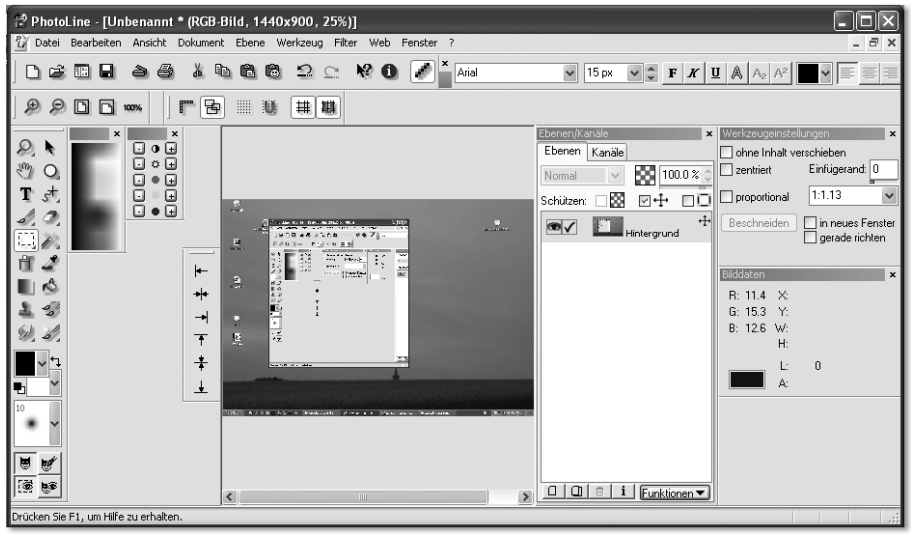

Abbildung 4.12 Die MDI-Anwendung Photoline

4.3.2 AceMDI in der Praxis

AceMDI[4] wird von Pritam G. Barhate entwickelt. Die Klassenbibliothek steht unter der *GNU Lesser General Public License*, kann also kostenlos von der Projekt-Homepage herunter geladen, verwendet und weiter gegeben werden. Die zum Zeitpunkt der Drucklegung aktuelle Version finden Sie auch auf der Begleit-CD zum Buch. Bitte entpacken Sie die Datei *acemdi.zip* in ein beliebiges Verzeichnis. Um die Klassenbibliothek in Ihren Programmen zu verwenden, kopieren Sie *acemdi1_1.jar* in das Java-Erweiterungsverzeichnis oder fügen die *.jar*-Datei dem Klassenpfad Ihrer Anwendung hinzu.

Lassen Sie uns nun einen Blick auf die Funktionsweise von *AceMDI* werfen. Im Folgenden sehen Sie den Quelltext von *AceMDIDemo*. Dieses Programm ist bewusst ähnlich aufgebaut wie das in Abschnitt 4.3.1 vorgestellte *SwingMDIDemo*, damit Sie die beiden leichter vergleichen können.

```
package javafuerwindows.gui;
import javax.swing.JFrame;
import ui.acemdi.MDIFrame;
import ui.acemdi.MDIFrameListener;
import ui.acemdi.MDIView;
import ui.acemdi.MDIFrameEvent;
public class AceMDIDemo extends MDIFrame
```

4 https://acemdi.dev.java.net/

```
                        implements MDIFrameListener {
    public AceMDIDemo() {
        super("AceMDIDemo");
        addMDIFrameListener(this);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setBounds(100, 100, 400, 300);
        neuesFenster("Fenster 1", 40, 40, 200, 100);
        neuesFenster("Fenster 2", 60, 60, 220, 120);
        setVisible(true);
    }

    private void neuesFenster(String titel, int x, int y,
                              int w, int h) {
        MDIView iframe = new MDIView(this, null,
                                     titel, null);
        iframe.setBounds(x, y, w, h);
        add(iframe);
    }

    public static void main(String [] args) {
        new AceMDIDemo();
    }

    public void viewPaneChanged(MDIFrameEvent event) {
        System.out.println(event);
    }
}
```

Listing 4.7 AceMDIDemo.java

Grundlage des Programms ist eine Instanz der Klasse `ui.acemdi.MDIFrame`. Ihr werden Instanzen von `ui.acemdi.MDIView` durch Aufruf der Methode `add()` hinzugefügt. Sie repräsentieren die Dokumente Ihrer Anwendung. Um auf Größenveränderungen reagieren zu können, sollten Sie das Interface `ui.acemdi.MDIFrameListener` implementieren.

Abbildung 4.13 AceMDIDemo mit Dokumentfenstern als Registerkarten

AceMDI kombiniert die Vorteile des *Multiple Document Interface* mit denen des Tabbed Document Interface. Wenn Sie erwägen, Ihre Anwendung nicht in der Form einzelner, gleichberechtigter Dokumentfenster zu realisieren, ist *AceMDI* in jedem Fall einen Blick wert als mögliche Alternative zu Swings MDI-Implementierung.

5 Kommunikation mit Microsoft Office

5 Kommunikation mit Microsoft Office

In diesem Kapitel zeige ich Ihnen, wie Sie Ihre Software ohne großen Aufwand um die Möglichkeit erweitern, Microsoft Office-Dateiformate zu lesen und zu schreiben sowie mit Excel, Outlook und Co. zu kommunizieren.

Microsofts Office-Suite beinhaltet einige der am häufigsten eingesetzten Windows-Programme überhaupt. Keine Textverarbeitung kann es sich derzeit leisten, auf Import- und Exportfilter für *Word*-Dokumente zu verzichten. Selbst auf anderen Systemen, beispielsweise Linux oder Mac OS X, ist es notwendig, mit *.doc*-Dateien umgehen zu können. Dass es zu den Microsoft-Programmen äußerst leistungsfähige Alternativen gibt, ist selbstverständlich unstrittig. Gerade im Hinblick auf die Verbreitung von *Word*, *Powerpoint* oder *Access* ist es für solche Konkurrenzprodukte zwingend erforderlich, die Daten der »Vorbilder« verarbeiten zu können. Natürlich ist der Umgang mit Office-Dateiformaten nicht nur dann interessant, wenn Sie eine neue Textverarbeitung schreiben möchten. Beispielsweise können Sie Messdaten, die Ihre Anwendung erfasst, mit sehr geringem Aufwand in einem *Excel*-Arbeitsblatt ablegen. Und der lesende und schreibende Zugriff auf Ihre in *Outlook* gespeicherten Kontakte, Termine und Notizen erlaubt Ihnen sogar, die Funktionalität der entsprechenden Anwendungen erweitern. Stellen Sie sich ein Programm vor, das Ihre Adressen als HTML-Dateien speichert und automatisch mittels *Bluetooth* an Ihr Handy übermittelt. Eine andere Idee ist, Notizen als schicke kleine Klebezettelchen zu präsentieren. In diesem Kapitel stelle ich Ihnen Klassenbibliotheken vor, mit deren Hilfe Sie solche Programme sehr leicht realisieren können.

5.1 Excel

Tabellenkalkulationen sind ideal für das Präsentieren und Auswerten langer Zahlenkolonnen oder listenartiger Daten. Beispielsweise könnte Ihre Anwendung Kurse von Wertpapieren abfragen und am Ende des Börsentags eine Aufstellung der Kursverläufe Ihrer Aktien erzeugen. Da sich in *Excel* die Zellen eines Arbeitsblattes auf vielfältige Weise formatieren lassen, werden solche Anwendungen aber auch häufig für einfache grafische Gestaltungen verwendet, beispielsweise von Terminplänen oder Inventarlisten. Im Folgenden stelle ich Ihnen die Klassenbibliothek *POI* vor, mit der Sie nahezu beliebig komplexe Excel-Dokumente erstellen sowie bereits vorhandene Arbeitsmappen einlesen können.

5.1.1 Jakarta POI – ein Überblick

Das Projekt *Jakarta* der *Apache Software Foundation* besteht aus einer Vielzahl quelloffener Java-Projekte. Eines davon, *POI*, bietet sowohl lesenden als auch schreibenden Zugriff auf Microsofts *OLE 2 Compound Document Format*, welches die Grundlage für einige Microsoft Office-Dateiformate bildet. So ist es mit Hilfe der POI APIs möglich, Excel-Arbeitsmappen und Word-Dokumente anzulegen, zu lesen, zu verändern und zu speichern.

Der Name *POI* kann übrigens auf zwei Weisen interpretiert werden. In der ersten Lesart stehen die drei Buchstaben für *Poor Obfuscation Implementation* (auf Deutsch etwa »schlecht implementierter Verschleierungsalgorithmus«). Zudem hat eine hawaiianische Delikatesse denselben Namen. In beiden Fällen machen sich die Entwickler über Microsofts Dateiformat lustig, das sie für schlecht erweiterbar und obendrein anfällig für Fragmentierung halten. Wenn Sie des Englischen mächtig sind und darüber hinaus ein Faible für flapsige Sprüche haben, rate ich Ihnen, sich den entsprechenden Abschnitt in der POI-Dokumentation einfach selbst durchzulesen.

Das *POI*-Projekt bildet das Dach für eine Reihe von Teilprojekten. *POIFS* ist ein solches Unterprojekt, das ganz allgemein Zugriff auf das OLE 2 Compound Document Format bereitstellt. *HSSF* ermöglicht speziell das Lesen und Schreiben von Excel 97–2002 Dateien. *HWPF*, ein weiteres Teilprojekt, konzentriert sich auf Word-Dokumente und *HPSF* liest so genannte Property Sets. Das Bemerkenswerte an den genannten Projekten ist, dass sie ohne Zugriff auf native Ressourcen auskommen, also auch dort funktionieren, wo kein Microsoft Office installiert ist. Um *Jakarta POI* in Ihren Programmen verwenden zu können, müssen Sie sich zunächst von der Projekt-Homepage[1] die aktuellste Fassung herunterladen. Es stehen mehrere Archivtypen zur Verfügung, wobei die gezippte Version sicher am leichtesten zu handhaben ist. Selbstverständlich finden Sie die zum Zeitpunkt der Drucklegung aktuelle Fassung auch auf der Begleit-CD. Diese können Sie in einem beliebigen Verzeichnis entpacken. Anschließend sollten Sie dafür sorgen, dass Java die *.jar*-Archive findet. Die Vorgehensweise sowie Tipps für einen bequemen Umgang mit zusätzlichen Bibliotheken habe ich in Kapitel 1, *Installation und Konfiguration*, zusammengestellt. Wenn Sie Ihre Programme in einer integrierten Entwicklungsumgebung wie *NetBeans* oder *Eclipse* erstellen, denken Sie bitte daran, die im Javadoc-Format vorliegende Dokumentation zu POI einzubinden, damit Sie bei Bedarf schnell die benötigten Informationen finden.

1 http://jakarta.apache.org/poi/

5.1.2 Erzeugen von Excel-Arbeitsmappen

Die *POI*-Komponente *HSSF* ist für das Erzeugen, Lesen und Schreiben von Excel-Arbeitsmappen zuständig. Die vier Buchstaben stehen für *Horrible Spreadsheet Format* (auf Deutsch etwa »schreckliches Tabellenkalkulationsformat«) und sind, wie der Name des gesamten Projekts, ein Seitenhieb auf Microsoft. Die Arbeitsmappe bildet in *HSSF* die größte Einheit. Sie enthält eine oder mehrere Tabellen (Arbeitsblätter), die wiederum aus mindestens einer Zeile bestehen. Jede Zeile beinhaltet mindestens eine Zelle. Deren Aussehen und Inhalt kann auf vielfältige Weise beeinflusst werden. Diese hierarchische Beziehung spiegelt sich in den Klassen der *HSSF*-Bibliothek wider und bildet – wie Sie gleich sehen werden – die Grundlage für den Aufbau *POI*-basierter Programme. Im Folgenden zeige ich Ihnen, wie Sie mit *POI* eine einfache Weinliste erzeugen, die Sie in Excel jederzeit weiter bearbeiten können. In Abbildung 5.1 sehen Sie die durch das Programm *ExcelDemo1* generierte Arbeitsmappe *ExcelDemo1.xls*.

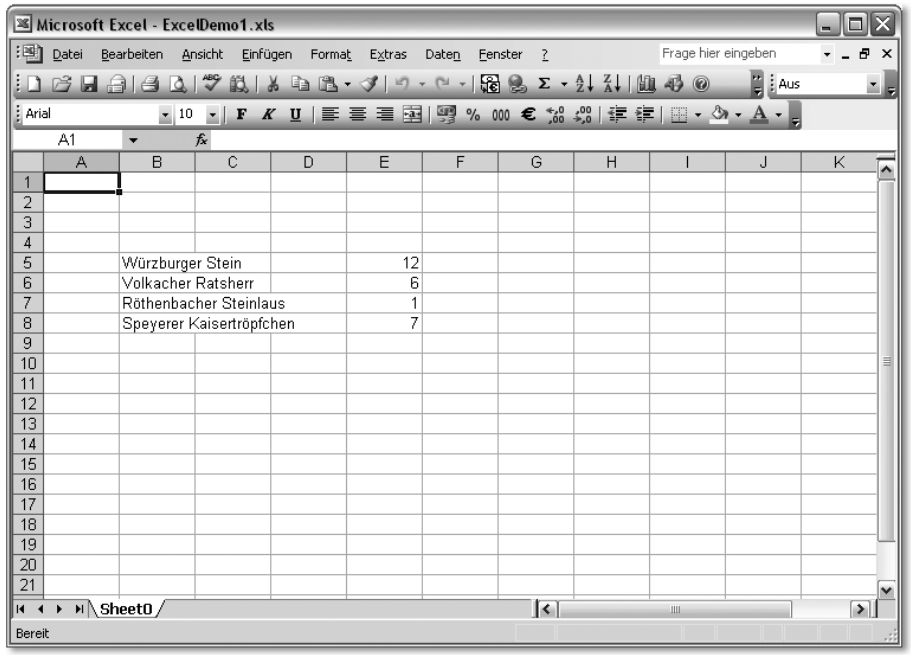

Abbildung 5.1 Die von ExcelDemo1 erzeugte Weinliste

Die Mappe beinhaltet ein Arbeitsblatt, das den Namen *Sheet0* trägt. In den Zellen B5 bis B8 stehen Namen von Weinen, E5 bis E8 beinhalten die Zahl der gelagerten Flaschen. Das Programm, das dieses Excel-Dokument erzeugt, sieht folgendermaßen aus.

```
package javafuerwindows.poi;
import java.io.FileOutputStream;
import org.apache.poi.hssf.usermodel.HSSFCell;
import org.apache.poi.hssf.usermodel.HSSFRow;
import org.apache.poi.hssf.usermodel.HSSFSheet;
import org.apache.poi.hssf.usermodel.HSSFWorkbook;
public class ExcelDemo1 {

    public ExcelDemo1() {
        /*
         * 1. Schritt: Anlegen einer Arbeitsmappe
         */
        HSSFWorkbook mappe = new HSSFWorkbook();
        /*
         * 2. Schritt: ein Arbeitsblatt wird angelegt
         */
        HSSFSheet blatt = mappe.createSheet();
        /*
         * 3. Schritt: Festlegen der Daten,
         *    die das Arbeitsblatt aufnehmen soll
         */
        String [] weine = {"Würzburger Stein",
                "Volkacher Ratsherr",
                "Röthenbacher Steinlaus",
                "Speyerer Kaisertröpfchen"};
        int [] flaschen = {12, 6, 1, 7};
        /*
         * 4. Schritt: Erzeugen der Zeilen
         */
        HSSFRow [] zeilen = new HSSFRow[4];
        for (int i = 0; i < 4; i++)
            zeilen[i] = blatt.createRow(4 + i);
        /*
         * 5. Schritt: Erzeugen der Zellen
         *             und Füllen mit den Daten
         */
        HSSFCell [] zellen = new HSSFCell[8];
        for (int i = 0; i < 4; i++) {
            zellen[i] = zeilen[i].createCell((short) 1);
            zellen[i].setCellValue(weine[i]);
```

```
        }
        for (int i = 0; i < 4; i++) {
            zellen[i + 4] = zeilen[i].createCell((short) 4);
            zellen[i + 4].setCellValue(flaschen[i]);
        }
        /*
         * 6. Schritt: Speichern des Dokuments
         */
        try {
            FileOutputStream stream =
                 new FileOutputStream("ExcelDemo1.xls");
            mappe.write(stream);
        } catch (Exception e) {
            System.err.println(e);
        }
    }

    public static void main(String [] args) {
        new ExcelDemo1();
    }
}
```

Listing 5.1 ExcelDemo1.java

Wie Sie sehen, lässt sich mit wenigen Programmzeilen eine Excel-Arbeitsmappe erzeugen und als Datei speichern. In den Schritten 1 und 2 wird eine leere Mappe erzeugt und darin ein zunächst leeres Arbeitsblatt abgelegt. In den Schritten 3 bis 5 werden ihm vier Zeilen hinzugefügt. Jede Zeile besteht aus zwei Zellen, dem Namen eines Weines und der Zahl der gelagerten Flaschen. Schritt 6 zeigt, wie die fertige Arbeitsmappe gespeichert wird. Die Struktur der Datei ist sehr schön im Programm zu erkennen.

Bei der Verwendung von HSSF gibt es nur sehr wenige Dinge zu beachten. Wichtig ist, dass *POI* bei 0 anfängt zu zählen, wohingegen Zellen ab A1 hochgezählt werden. Dies betrifft beispielsweise die Methoden zum Erzeugen von Zeilen und Zellen. Etwas gewöhnungsbedürftig ist ferner, dass POI bei bestimmten Werten ausdrücklich `short`-Datentypen verlangt. Wenn beim Übersetzen also eine offenkundig richtig aussehende Programmzeile moniert wird, sehen Sie bitte in der Dokumentation nach, ob ein Parameter ausdrücklich als `short` angegeben ist.

Die durch *ExcelDemo1*.java erzeugte Weinliste wurde ganz bewusst einfach gehalten. Natürlich sind Excel und *POI* weitaus mächtiger. Deshalb zeige ich Ihnen in einem zweiten Beispiel, wie Sie das Arbeitsblatt mit Überschriften versehen können und wie Sie mit Formeln arbeiten. Abbildung 5.2 zeigt das durch *ExcelDemo2* erzeugte Ergebnis.

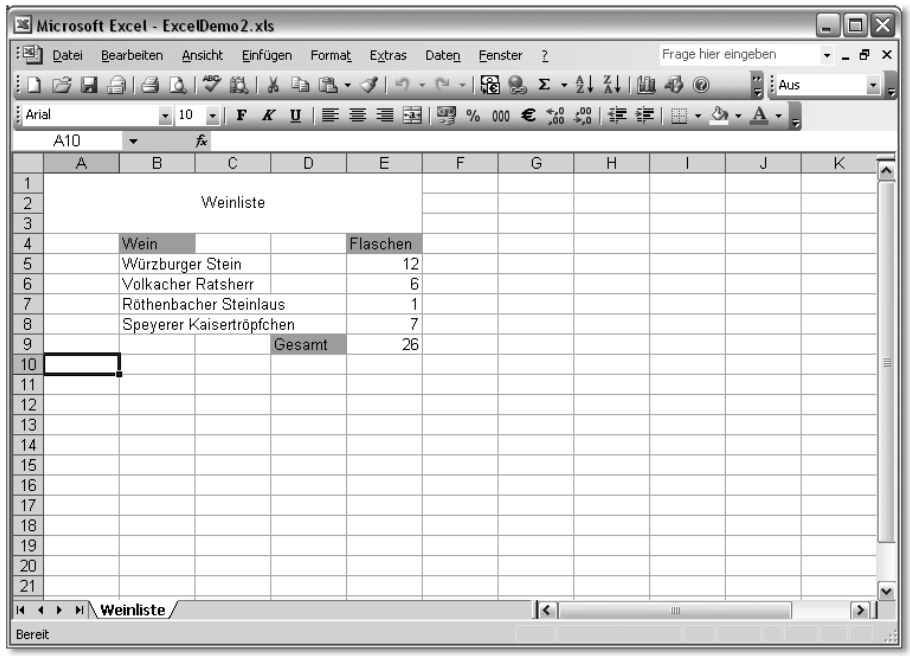

Abbildung 5.2 Durch ExcelDemo2 erzeugte erweiterte Version der Weinliste

Die Grundstruktur des Programms *ExcelDemo2* ist mit der des ersten Beispiels identisch. Deshalb folgen hier nur diejenigen Programmteile, die sich gegenüber *ExcelDemo1* verändert haben oder neu hinzugekommen sind. Selbstverständlich finden Sie beide Listings auf der Begleit-CD zum Buch.

```
/*
 * 2. Schritt: Ein Arbeitsblatt wird angelegt;
 *             anders als in ExcelDemo1 wird hier ein
 *             Name für das Arbeitsblatt vergeben
 */
HSSFSheet blatt = mappe.createSheet("Weinliste");
```

Anstelle des wenig aussagekräftigen Standardnamens *Sheet0* habe ich eine etwas sprechendere Variante gewählt, die den Zweck oder Inhalt des Arbeitsblattes ausdrückt.

```
/*
 * Schritt 2a: Erzeugen einer Überschrift; sie soll
 *             den Bereich A1 bis E3 umfassen
 */
 HSSFRow zeileUeberschrift = blatt.createRow(0);
 Region regionUeberschrift = new Region(0, (short)0,
                                        2, (short)4);
 blatt.addMergedRegion(regionUeberschrift);
 HSSFCell zelleUeberschrift =
                     zeileUeberschrift.createCell((short)0);
 zelleUeberschrift.setCellValue(mappe.getSheetName(0));
 HSSFCellStyle stilUeberschrift = mappe.createCellStyle();
 stilUeberschrift.setAlignment(HSSFCellStyle.ALIGN_CENTER);
stilUeberschrift.setVerticalAlignment(HSSFCellStyle.VERTICAL_
CENTER);
 zelleUeberschrift.setCellStyle(stilUeberschrift);
```

Im neu hinzugekommenen Schritt 2a wird eine Überschrift erzeugt, die sich über den Bereich A1 bis E3 erstreckt. Dieser wird durch eine Instanz der Klasse `Region` definiert. Die Zelle, die die Überschrift enthält, wird auf bereits bekannte Weise erzeugt: durch Aufruf der Methode `createCell()`. Interessant ist in diesem Zusammenhang die Auskunftsmethode `getSheetName()`, mit der sich der Name eines Arbeitsblattes ermitteln lässt. Die Formatierung von Zellen wird durch Instanzen der Klasse `HSSFCellStyle` realisiert. Zunächst erzeugen Sie durch Aufruf von `createCellStyle()` eine Instanz, die Sie dann nach Belieben anpassen können. Das Beispiel zeigt, wie die horizontale und vertikale Ausrichtung der Zelle eingestellt wird. Die Methode `setCellStyle()` von `HSSFCell`-Instanzen schließlich übernimmt einen Zellstil für die entsprechende Zelle.

```
/*
 * Schritt 2b: Erzeugen und Formatieren
 *             weiterer Überschriften
 */
 HSSFCellStyle stilSpalte = mappe.createCellStyle();
 stilSpalte.setFillForegroundColor(HSSFColor.AQUA.index);
 stilSpalte.setFillPattern(HSSFCellStyle.SOLID_FOREGROUND);
 HSSFRow zeileSpalten = blatt.createRow(3);
 HSSFCell zelleWein = zeileSpalten.createCell((short)1);
 zelleWein.setCellValue("Wein");
 zelleWein.setCellStyle(stilSpalte);
```

```
HSSFCell zelleFlaschen = zeileSpalten.createCell((short)4);
zelleFlaschen.setCellValue("Flaschen");
zelleFlaschen.setCellStyle(stilSpalte);
```

Der ebenfalls neue Schritt 2b zeigt weitere Möglichkeiten, Zellen zu formatieren. So ist es möglich, Zellen mit einem Füllmuster zu versehen und eine Füllfarbe festzulegen. Schließlich möchte ich Ihnen noch zeigen, wie Sie Formeln erzeugen können. Das Vorgehen ist im Schritt 2c zu sehen.

```
/*
 * Schritt 2c: Hinterlegen einer Formel
 */
HSSFRow zeileSumme = blatt.createRow(8);
HSSFCell zelleSummeSpalte = zeileSumme.createCell((short)3);
zelleSummeSpalte.setCellValue("Gesamt");
zelleSummeSpalte.setCellStyle(stilSpalte);
HSSFCell zelleSumme = zeileSumme.createCell((short)4);
zelleSumme.setCellType(HSSFCell.CELL_TYPE_FORMULA);
zelleSumme.setCellFormula("SUM(E5:E8)");
```

Die einzige Neuerung ist, dass Sie mit `setCellTyp()` festlegen müssen, dass eine Zelle eine Formel enthält. Der Zellinhalt wird in diesem Fall auch nicht wie sonst üblich mit `setCellValue()` gesetzt, sondern mit der speziellen Methode `setCellFormula()`.

Der folgende Abschnitt erläutert nun, wie Sie mit POI beliebige vorhandene Arbeitsmappen mit Ihren Java-Programmen verarbeiten können.

5.1.3 Lesen bestehender Excel-Dokumente

Auch das Lesen existierender Excel-Dokumente ist mit *POI* ein einfaches Unterfangen. Prinzipiell kommen die gleichen Mechanismen zum Einsatz wie beim Erstellen eigener Arbeitsmappen. Bei der Implementierung Ihrer Leseroutinen sollten Sie allerdings defensiver vorgehen als beim Schreiben: Wenn Sie selbst eine Datei anlegen, behalten Sie die volle Kontrolle über den Aufbau und Inhalt. Beim Einlesen bestehender Dokumente wissen Sie nicht zwangsläufig, was Sie erwartet. Bevor Sie beispielsweise den Inhalt einer Zelle verarbeiten, müssen Sie prüfen, ob er in einem Format vorliegt, mit dem Ihre Anwendung umgehen kann. Wie so etwas aussehen kann, zeige ich Ihnen im Programm *ExcelDemo3*, ein sehr einfaches Analyseprogramm für bestehende Excel-Dokumente. Hier zunächst der Quelltext.

```
package javafuerwindows.poi;
import java.io.FileInputStream;
import java.io.IOException;
import org.apache.poi.hssf.usermodel.HSSFCell;
import org.apache.poi.hssf.usermodel.HSSFRow;
import org.apache.poi.hssf.usermodel.HSSFSheet;
import org.apache.poi.hssf.usermodel.HSSFWorkbook;
import org.apache.poi.poifs.filesystem.POIFSFileSystem;

public class ExcelDemo3 {
    public ExcelDemo3(String name) {
        FileInputStream fin = null;
        POIFSFileSystem poi = null;
        HSSFWorkbook mappe = null;
        HSSFSheet blatt;
        HSSFRow zeile;
        HSSFCell zelle;
        String inhalt, wert;
        try {
            fin = new FileInputStream(name);
            poi = new POIFSFileSystem(fin);
            mappe = new HSSFWorkbook(poi);
            for (int i = 0; i < mappe.getNumberOfSheets();
                                    i++) {
                /*
                 * Namen des Arbeitsblattes ausgeben
                 */
                System.out.println("Arbeitsblatt " +
                    (i + 1) + ": " + mappe.getSheetName(i));
                blatt = mappe.getSheetAt(i);
                for (int zeilenZaehler
                     = blatt.getFirstRowNum();
                     zeilenZaehler <= blatt.getLastRowNum();
                     zeilenZaehler++) {
                    zeile = blatt.getRow(zeilenZaehler);
                    if (zeile == null)
                        continue;
                    inhalt = "";
                    for (short spaltenZaehler
                       = zeile.getFirstCellNum();
```

```
                spaltenZaehler <= zeile.getLastCellNum();
                spaltenZaehler++) {
                    zelle =
                    zeile.getCell(spaltenZaehler);
                    if (zelle == null)
                        continue;
                    if (zelle.getCellType() ==
                    HSSFCell.CELL_TYPE_FORMULA)
                        wert = zelle.getCellFormula();
                    else if (zelle.getCellType() ==
                    HSSFCell.CELL_TYPE_NUMERIC)
                        wert =
            Double.toString(zelle.getNumericCellValue());
                    else
                        wert =
                        zelle.getStringCellValue();
                    inhalt += (wert + " ");
                }
                System.out.println(inhalt);
            }
        }
    } catch (IOException e) {
        System.err.println(e);
    } finally {
        if (fin != null)
            try {
                fin.close();
            } catch (IOException e) {
            }
    }
}

public static void main(String [] args) {
    if (args.length != 1) {
        System.err.println("Aufruf:
                            ExcelDemo3 Dateiname");
        System.exit(0);
    }
    new ExcelDemo3(args[0]);
```

```
    }
}
```

Listing 5.2 ExcelDemo3.java

Das Programm erwartet beim Aufruf den Namen eines zu analysierenden Excel-Dokuments als Parameter. Abbildung 5.3 zeigt, wie Sie es starten und welche Ausgaben es erzeugt.

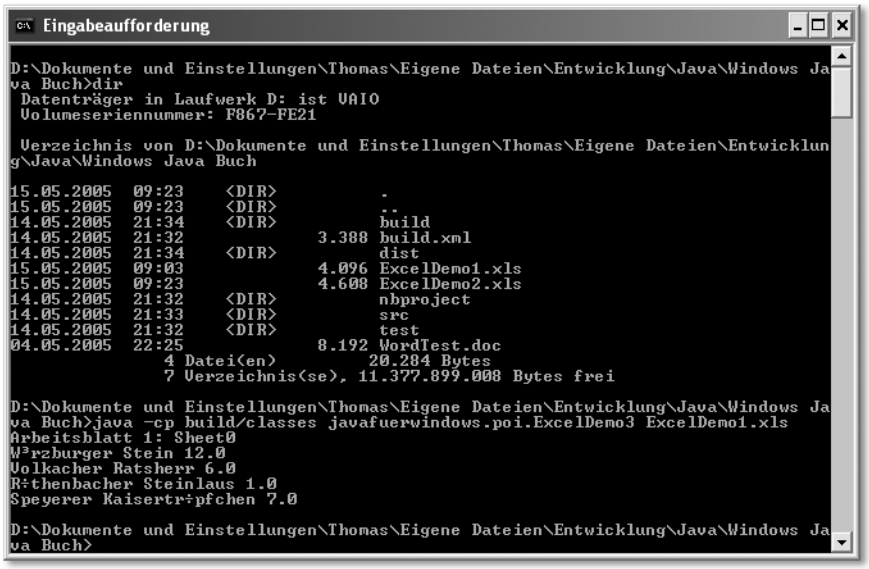

Abbildung 5.3 Ausgabe des Programms ExcelDemo3

Ein interessanter Aspekt an diesem Programm ist zunächst sicher die Verwendung von `org.apache.poi.poifs.filesystem.POIFSFileSystem`. Eine Instanz dieser Klasse wird an den Konstruktor `HSSFWorkbook` übergeben, was dazu führt, dass keine leere Arbeitsmappe erzeugt, sondern aus einer bereits bestehenden Datei geladen wird. Welche dies ist, wurde `POIFSFileSystem` selbst im Konstruktor übermittelt. Anschließend haben Sie durch die Methode `getSheetAt()` Zugriff auf ein Arbeitsblatt. Wenn Ihr Programm unbekannte Excel-Dokumente verarbeiten muss, werden Sie sicher die Auskunftsmethoden der Klassen `HSSFWorkbook`, `HSSFSheet` und `HSSFCell` schätzen lernen, beispielsweise `getFirstRowNum()`, `getLastRowNum()`, `getFirstCellNum()` und `getLastCellNum()`.

Dies ist vor allem deshalb wichtig, weil im Gegensatz zu relationalen Datenbanken mehrere Zeilen eines Excel-Arbeitsblattes keineswegs immer die gleiche Anzahl von Zellen aufweisen müssen. Auch rate ich Ihnen dringend, vor dem Aufruf von `getStringCellValue()` stets mit `getCellType()` den Zelltyp zu ermitteln.

Bisher habe ich Ihnen gezeigt, wie Sie Excel-Dokumente lesen und erzeugen können. Für die Darstellung wurde Excel verwendet. Wenn es vor allem um das Anzeigen einer Arbeitsmappe geht, ist es eigentlich nicht sinnvoll, hierfür stets die gesamte Anwendung zu laden. Außerdem ist die Plattformunabhängigkeit ein Aushängeschild von POI. Es sollte daher möglich sein, Tabellen auch ohne Excel darzustellen. Wie dies geht, zeige ich Ihnen im folgenden Abschnitt.

5.1.4 Anzeigen von Arbeitsmappen

Mit der Klasse `org.apache.poi.hssf.contrib.view.SViewerPanel` steht eine von `JPanel` abgeleitete Komponente zum Anzeigen und Bearbeiten von Instanzen der Klasse `HSSFWorkbook` zur Verfügung. Das Laden einer bestehenden Arbeitsmappe kennen Sie bereits aus Abschnitt 5.1.3. Eine Instanz der Klasse `HSSFWorkbook` wird anschließend dem Konstruktor von `SViewerPanel` übergeben. Mit dem zweitem Parameter, einem `boolean`-Wert, legen Sie fest, ob die neu erzeugte Komponente editierbar sein soll oder ob die Zellen der Arbeitsmappe unveränderlich sind. Klassen aus `contrib`-Paketen sind noch nicht »offizieller« Bestandteil von POI und sollten deshalb umsichtig verwendet werden.

Im nächsten Abschnitt wenden wir uns der zweiten großen Anwendung aus Microsofts Office-Suite zu, der Textverarbeitung *Word*.

5.2 Word

Die Fähigkeit, das Dateiformat *.doc* lesen und schreiben zu können, ist aus zahlreichen Gründen interessant. Beispielsweise ermöglichen Sie den Anwendern Ihrer Programme den Datenaustausch mit einem riesigen Benutzerkreis, sogar über Systemgrenzen hinweg. Außerdem können Sie indirekt den Funktionsumfang von *Word* vergrößern, ohne sich in die Makroprogrammierung mit *Visual Basic for Applications* einarbeiten zu müssen. Ihr Java-Programm realisiert die benötigten Funktionen und schreibt das Ergebnis in ein Word-Dokument. Ein Beispiel für eine solche Vorgehensweise stelle ich Ihnen in Abschnitt 5.2.2 vor.

5.2.1 Lesen vorhandener Word-Dokumente mit HWPF

Das Projekt *POI* bietet neben dem Lesen und Schreiben von Excel-Arbeitsmappen auch Zugriff auf Word-Dokumente. Im Gegensatz zu *HSSF* befindet sich das für Word zuständige Teilprojekt *HWPF* derzeit allerdings in einem vergleichsweise frühen Stadium. Dennoch lassen sich einfachere Word 97-Dokumente schon jetzt problemlos lesen. Sehen Sie sich hierzu bitte den folgenden Quelltext des Programms *WordDemo1* an.


```
package javafuerwindows.poi;
import java.io.CharArrayWriter;
import org.apache.poi.hdf.extractor.WordDocument;

public class WordDemo1 {

    public WordDemo1(String name) {
        WordDocument dokument = null;
        CharArrayWriter puffer = null;
        try {
            puffer = new CharArrayWriter();
            dokument = new WordDocument(name);
            dokument.writeAllText(puffer);
            System.out.println(puffer.toString());
        } catch (Exception e) {
            System.err.println(e);
        }
    }

    public static void main(String [] args) {
        if (args.length != 1) {
            System.err.println("Aufruf:
                                WordDemo1 Dateiname");
            System.exit(0);
        }
        new WordDemo1(args[0]);
    }
}
```

Listing 5.3 WordDemo1.java

Das Programm *WordDemo1* gibt den Inhalt eines Word-Dokuments ohne jegliche Textauszeichnungen auf dem Bildschirm aus. Fast die komplette hierfür

benötigte Funktionalität wird durch die Klasse `org.apache.poi.hdf.extractor.WordDocument` zur Verfügung gestellt. Nach dem Instantiieren dieser Klasse wird deren Methode `writeAllText()` aufgerufen, die den reinen Text (also ohne Formatierungen oder Auszeichnungen) der dem Konstruktor übergebenen Datei an die abstrakte Klasse `java.io.Writer` weitergibt. Im Beispiel wird `java.io.CharArrayWriter` als von `Writer` abgeleitete Klasse verwendet. Deren Methode `toString()` macht den Inhalt des Word-Dokumentes zugänglich.

Im Vergleich zur Implementierung des Excel-Dateiformats steht *HWPF* noch am Anfang seiner Entwicklung. Auch ist die API noch im Fluss, wie die Verwendung des alten Projekttitels *HDF* (*Horrible Document Format*) in Paketnamen `org.apache.poi.hdf.extractor` vermuten lässt. Durch die Möglichkeit, den eigentlichen Text auszulesen und weiterzuverarbeiten, haben Sie in Ihren Programmen aber dennoch schon rudimentären Zugang zu Word-Dokumenten.

5.2.2 Java Bean Word Processing

Eine Möglichkeit, Word-Dokumente in Java zu erstellen, bietet die kostenlose *Java Bean Word Processing*[2] der Stuttgarter Firma *Müller und Stein Software*. Die Komponente setzt auf eine direkte, native Schnittstelle zu Word, ist also nur auf Rechnern mit installiertem Word einsetzbar. Die Grundidee des Programms ist es, auf Basis einer Textvorlage zunächst ein neues Dokument zu erzeugen und in diesem an gezielten Stellen, so genannten Textmarken, Text hinzuzufügen. Der Java-Teil, der aus der Quelltextdatei *WordProcessing.java* besteht, bereitet hierzu eine Liste von Anweisungen vor, die in einer Datei namens *WordInp.txt* zwischengespeichert und anschließend durch *WordAPI.exe* abgearbeitet werden. Sie erstellen diese Anweisungsliste mit Ihrem eigenen Programm durch Aufrufe bestimmter Methoden der Klasse `WordProcessing`. Sehen Sie sich bitte das Programm *JBWPDemo1* an, das die Verwendung von *Java Bean Word Processing* demonstriert.

```
package javafuerwindows.jbwp;
import de.must.util.WordProcessing;
public class JBWPDemo1 {
    public JBWPDemo1(String vorlage, String ausgabe) {
WordProcessing.createNewDocumentFromTemplate(vorlage);
        WordProcessing.typeTextAtBookmark("AddressLine1",
                                          "Firma Mustermann");
```

2 http://www.java400.de/default.html?Javactp.htm

```
        WordProcessing.typeTextAtBookmark("AddressLine2",
                                          "Herrn Mustermann");
        WordProcessing.typeTextAtBookmark("AddressLine3",
                                          "Beispielstr. 1");
        WordProcessing.typeTextAtBookmark("AddressLine4",
                                          "12345 Stadt");
        WordProcessing.typeTextAtBookmark("Salutation",
                           "Sehr geehrter Herr Mustermann,");
        WordProcessing.saveDocumentAsAndClose(ausgabe);
        WordProcessing.exec();
    }

    public static void main(String [] args) {
        if (args.length != 2) {
            System.out.println("JBWPDemo1 <Vorlagendatei>
                                          <Ausgabedatei>");
            System.exit(0);
        }
        new JBWPDemo1(args[0], args[1]);
    }
}
```

Listing 5.4 JBWPDemo1.java

Das Beispiel erwartet zwei Parameter, den Namen der zu verwendenden Word-Vorlage sowie den Dateinamen des zu erzeugenden Dokuments. Wenn Sie das Programm mit der Vorlage *SampleTemplate.dot* von *Müller und Stein Software* aufrufen, entsteht das in Abbildung 5.4 gezeigte Dokument.

Der erste Schritt ist das Erzeugen eines Dokumentes durch Aufruf der Methode `createNewDocumentFromTemplate()`. Anschließend können Sie mit `typeTextAtBookmark()` an nahezu beliebiger Stelle Text einfügen. Die Position innerhalb des Dokumentes wird hierbei durch den ersten Parameter festgelegt. Bitte beachten Sie, dass in der Dokumentvorlage eine gleichnamige Textmarke definiert sein muss. Nachdem Sie eine Vorlage, beispielsweise *SampleTemplate.dot*, in Word geöffnet haben, können Sie durch Aufruf von **Einfügen · Textmarke** vorhandene Textmarken anzeigen und bearbeiten sowie neue hinzufügen.

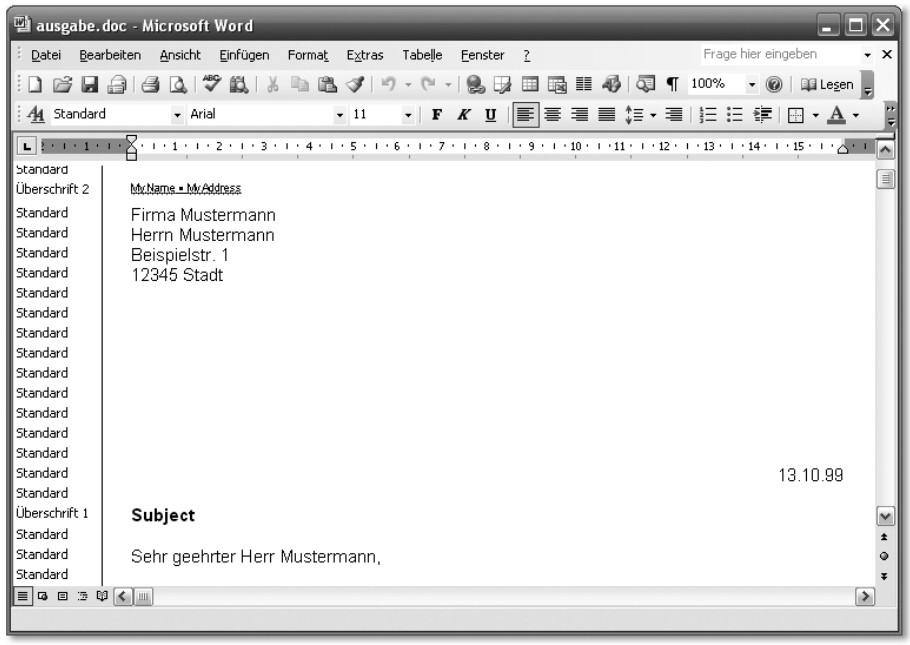

Abbildung 5.4 Das durch JBWPDemo1 erzeugte Word-Dokument

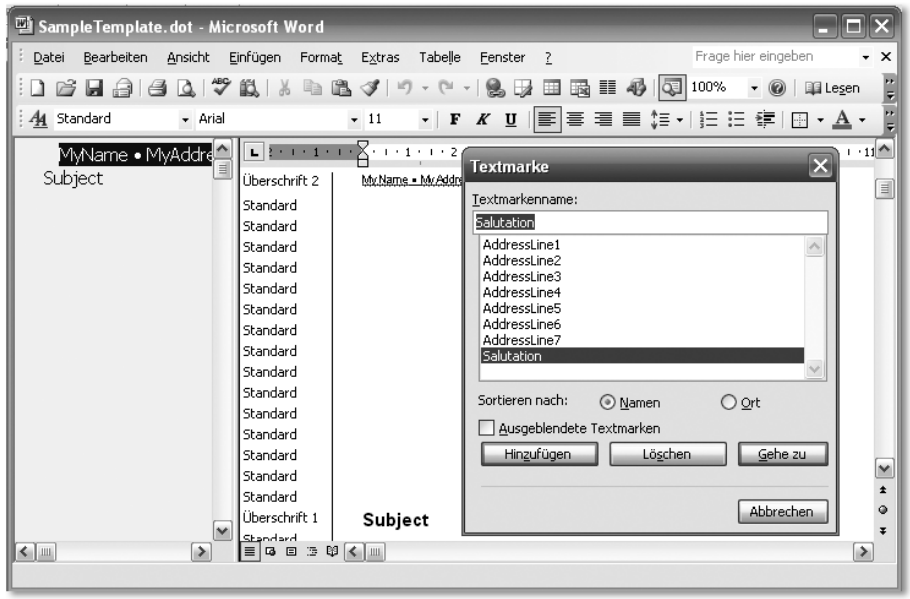

Abbildung 5.5 Anzeigen und Bearbeiten von Textmarken

Nachdem Sie in Ihrem Programm an allen Textmarken Inhalt eingefügt haben, können Sie das Dokument durch `saveDocumentAsAndClose()` speichern. Um die Anweisungskette abzuarbeiten, rufen Sie bitte die Methode `exec()` auf. Die Klasse `WordProcessing` startet daraufhin mittels `Runtime.getRuntime().exec()` die native Komponente *WordAPI.exe*.

5.3 Outlook

Der *Java Outlook Connector* (*JOC*) der Schweizer Firma Kova Solutions gestattet sowohl lesenden als auch schreibenden Zugriff auf die Outlook-Versionen 2000, 2002 (XP) und 2003. Anders als beispielsweise *POI* arbeitet diese Klassenbibliothek allerdings nicht auf Datendateien, im Fall von Outlook etwa *.pst*, sondern kommuniziert unter Verwendung von JNI direkt mit der Anwendung. Konsequenterweise lässt sie sich nur auf Systemen einsetzen, auf denen Outlook in einer unterstützten Version installiert ist.

Java Outlook Connector ist kostenpflichtig. Kova Solutions bietet unterschiedliche Lizenzmodelle an, wobei die günstigste Ein-Entwickler-Variante derzeit für 189 Euro zu haben ist. Um sich mit den Funktionen der Bibliothek vertraut zu machen, ist die kostenlose Testversion, die der Hersteller auf seiner Homepage zum Download[3] anbietet, allerdings völlig ausreichend. Um *JOC* in Ihren Programmen verwenden zu können, müssen Sie dafür sorgen, dass die Java-Laufzeitumgebung sowohl die *.jar*-Archive als auch die native DLL findet. Tipps hierzu sind in Kapitel 1, *Installation und Konfiguration*, zusammengestellt. Bitte denken Sie auch daran, die mitgelieferte Dokumentation in Ihrer Entwicklungsumgebung einzutragen, sodass Sie sich bei Bedarf über Parameter und Bedeutung der einzelnen Klassen informieren können.

5.3.1 Anzeigen von Outlook-Kontakten

Mit dem Programm *JOCDemo1* zeige ich Ihnen, wie leicht Sie mit Hilfe des *Java Outlook Connector*s auf Outlook-Daten zugreifen können. Bitte beachten Sie, dass das Programm nur funktioniert, wenn Sie eine von *JOC* unterstützte Version von Outlook installiert haben. Hier zunächst der Quelltext.

```
package javafuerwindows.joc;
import ch.kova.connector.exception.ComponentObjectModelException;
import ch.kova.connector.exception.ItemNotFoundException;
import ch.kova.connector.exception.LibraryNotFoundException;
import ch.kova.connector.ms.outlook.Outlook;
import ch.kova.connector.ms.outlook.contact.OutlookContact;
```

3 http://www.kova-solutions.com/joc/

```
import ch.kova.connector.ms.outlook.folder.FolderType;
import ch.kova.connector.ms.outlook.folder.OutlookFolder;
import ch.kova.connector.ms.outlook.item.ItemsCollection;
import ch.kova.connector.ms.outlook.item.ItemsIterator;
import ch.kova.connector.ms.outlook.item.OutlookItem;
import java.text.SimpleDateFormat;
import java.util.Date;

public class JOCDemo1 extends Outlook {
    private OutlookFolder ordner;
    private ItemsCollection elemente;

    public JOCDemo1() throws ComponentObjectModelException,
                             LibraryNotFoundException,
                             ItemNotFoundException {
        /*
         * der erste Schritt ist das Erzeugen eines
         * Outlook-Objektes; da wir die Klasse ableiten,
         * reicht es, den Konstruktor der Elternklasse
         * aufzurufen
         */
        super();

        /*
         * zweiter Schritt:
         * den Kontakte-Ordner holen
         */
        ordner = this.getDefaultFolder(FolderType.CONTACTS);

        /*
         * dritter Schritt:
         * die Elemente des Kontakte-Ordners
         * einlesen
         */
        elemente = ordner.getItems();

        /*
         * vierter Schritt:
         * alle Kontakte anzeigen; wir verwenden eine
         * Instanz von java.text.SimpleDateFormat
```

```
 * zur formatierten Ausgabe des Geburtsdatums
 */
SimpleDateFormat sdfGeburtstag =
       new SimpleDateFormat("dd. MMMM yyyy");
ItemsIterator iterator = elemente.iterator();
OutlookItem element;
OutlookContact kontakt;
String zeile, strNachname, strVorname, strGebDatum;
Date dateGebDatum;
while (iterator.hasNext()) {
    element = iterator.nextItem();

    /*
     * wenn es ein Kontakt ist, Namen, Vornamen
     * und Geburtstag anzeigen
     */
    if ((element != null) &&
       (element instanceof OutlookContact)) {
        kontakt = (OutlookContact) element;
        /*
         * Nachnamen ermitteln
         */
        strNachname = kontakt.getLastName();
        if (strNachname == null)
            strNachname = "unbekannt";
        /*
         * Vornamen ermitteln
         */
        strVorname = kontakt.getFirstName();
        if (strVorname == null)
            strVorname = "unbekannt";
        /*
         * Geburtsdatum ermitteln
         * und als String formatieren
         */
        dateGebDatum = kontakt.getBirthday();
        if (dateGebDatum != null)
            strGebDatum =
                 sdfGeburtstag.format(dateGebDatum);
        else
```

```
                    strGebDatum = "unbekannt";
                zeile = strNachname + ", " + strVorname
                        + ": " + strGebDatum;
                System.out.println(zeile);
            }
        }
    }

    /*
     * Einstieg in das Programm
     */
    public static void main(String[] args) {
        try {
            new JOCDemo1();
        } catch (Exception e) {
            System.out.println(e);
        }
        System.exit(0);
    }
}
```

Listing 5.5 JOCDemo1.java

Wenn Sie das Programm aufrufen, wird eine Liste Ihrer Outlook-Kontakte ausgegeben. Die JOC-Demoversion zeigt – wie in Abbildung 5.6 zu sehen ist – einen entsprechenden Hinweis, den Sie mit **OK** bestätigen können. Allerdings beendet sich das Programm mittels `System.exit(0)`, nachdem es die Liste ausgegeben hat, sodass der Dialog automatisch geschlossen wird. Dieser explizite Aufruf ist nötig, weil aufgrund der Verwendung von Swing ein Thread erzeugt wird, der das automatische Beenden nach Verlassen von `main()` verhindert.

Abbildung 5.6 Hinweis auf die Verwendung einer Demoversion von Java Outlook Connector

Zentrales Element bei der Kommunikation mit Outlook ist eine Instanz der Klasse `ch.kova.connector.ms.outlook.Outlook`. Sie können in Ihren Programmen eine eigene Klasse definieren, die von `Outlook` ableitet, oder direkt eine Instanz erzeugen und mit Referenzen arbeiten. In beiden Fällen entsteht das Bindeglied zwischen Ihrer Anwendung und dem Microsoft-Programm. Anschließend können Sie, wie im Quelltext zu sehen ist, gezielt Outlook-Ordner ansprechen. Hierfür wird die Methode `getDefaultFolder()` verwendet, der Sie den Typ des zu ermittelnden Standard-Ordners übergeben. Die Klasse `FolderType` definiert diese Typen, beispielsweise `CONTACTS`, `CALENDAR` und `NOTES`.

Nachdem Sie einen Ordner ermittelt haben, können Sie mit der Methode `getItems()` auf seine Elemente zugreifen. Der Zugriff auf sie erfolgt über eine Instanz der Klasse `ch.kova.connector.ms.outlook.item.ItemsCollection`, die zum Beispiel das Hinzufügen neuer (`createNew()`) und das Löschen bestehender (`deleteItem()`) Elemente gestattet. Die Methode `iterator()` liefert eine Instanz der Klasse `ch.kova.connector.ms.outlook.item.ItemsIterator`, mit der auf alle Elemente des Ordners zugegriffen werden kann.

5.3.2 Schreiben von Notizen

Java Outlook Connector kann auch schreibend auf Outlook zugreifen. Im Folgenden sehen Sie ein Programm, das eine kleine Eingabezeile (siehe Abbildung 5.7) öffnet, in der Sie beliebigen Text eingeben können. Dieser Text wird anschließend als Notiz an Outlook übergeben.

Abbildung 5.7 JOCDemo2 erwartet die Eingabe einer Notiz

Der Aufbau des Programms ist im Wesentlichen identisch mit *JOCDemo1*. Deshalb sehen Sie unten nur den Konstruktor, in dem die gesamte Arbeit verrichtet wird.

```
public JOCDemo2() throws ComponentObjectModelException, LibraryN
otFoundException, ItemNotFoundException {
/*
         * der erste Schritt ist das Erzeugen eines
         * Outlook-Objektes; da wir die Klasse ableiten,
```

```
         * reicht es, den Konstruktor der Elternklasse
         * aufzurufen
         */
        super();

        /*
         * zweiter Schritt:
         * den Kontakte Notizen holen
         */
        ordner = this.getDefaultFolder(FolderType.NOTES);
```

Die ersten beiden Schritte, Erzeugen einer Instanz der Klasse `Outlook` und Ermitteln eines Standard-Ordners, kennen Sie bereits.

```
        /*
         * im dritten Schritt wird ein
         * kleiner Eingabedialog dargestellt
         */
        String strEingabe = JOptionPane.showInputDialog(null,
                            "Bitte eine Notiz eingeben");
        if (strEingabe == null)
            return;
```

Der dritte Schritt, das Öffnen einer kleinen Eingabezeile, ist nicht *JOC*-spezifisch. Hier wird eine statische Methode der Klasse `javax.swing.JOptionPane` aufgerufen.

```
        /*
         * im vierten Schritt erzeugen wir
         * eine neue Notiz
         */
        ItemsCollection elemente = ordner.getItems();
        OutlookItem notiz = elemente.createNew(ItemType.NOTE);
        if (notiz instanceof OutlookNote) {
            ((OutlookNote) notiz).setBody(strEingabe);
            notiz.save();
        }
    }
```

Der vierte und letzte Schritt besteht aus vier Teilen: Erzeugen einer Instanz von `ItemsCollection` durch Aufruf von `getItems()`, Anlegen einer neuen Notiz mit `createNew()`, dem Setzen des Notiz-Textes mit `setBody()` und schließlich dem Speichern der Notiz mit `save()`.

5.3.3 Überblick über die JOC-Klassen

Java Outlook Connector bildet die hierarchische Ordnerstruktur von Outlook ab. Jeder Ordner hat einen bestimmten Typ, beispielsweise `TASKS`, `OUTBOX` oder `DRAFTS`, der durch die Klasse `FolderType` abgebildet wird. Die Ordner selbst sind Instanzen der Klasse `OutlookFolder`. In einem Ordner abgelegte Elemente dagegen sind Instanzen von Klassen, die von `OutlookItem` ableiten, beispielsweise `OutlookAppointment` oder `OutlookAttachment`. Jedes Element entspricht einem Typ, der durch die Klasse `ItemType` beschrieben wird. Der Zugriff auf Daten ist mehrstufig realisiert: In den meisten Fällen werden Sie mittels `getDefaultFolder()` und `getItems()` eine `ItemsCollection` erzeugen, die Ihnen dann als Grundlage für weitere Lese- und Schreiboperationen dient.

5.4 Weitere Office-Komponenten

Bisher habe ich Ihnen Klassenbibliotheken für die Kommunikation mit Word, Excel und Outlook vorgestellt. Für einige Office-Komponenten, beispielsweise Visio und Publisher, sind mir derzeit keine Bibliotheken bekannt, die den Zugriff auf die Anwendungen mittels Java ermöglichen. Die Zusammenarbeit mit Access und OneNote wird aus thematischen Gründen in anderen Kapiteln beschrieben. Welche dies sind, verrate ich Ihnen im Folgenden.

5.4.1 Access

Access arbeitet mit *.mdb*-Datenbankdateien, auf die mit dem *Jet*-Datenbanktreiber über die so genannte ODBC-Datenbankschnittstelle zugegriffen werden kann. Eine ausführliche Darstellung des Themenkomplexes Datenbanken finden Sie in Kapitel 6, *Datenbanken*.

5.4.2 OneNote

Die noch recht neue Anwendung *OneNote* dient dazu, Informationen und Textschnipsel, URLs, Töne und Bilder zu sammeln und zu verwalten. Erst mit dem Service Pack 1 hat Microsoft eine Schnittstelle vorgesehen, mit der Anwendung zu kommunizieren. Die hierfür verwendete Technologie, das Component Object Model, wird ausführlich in Kapitel 9, *Java-COM-Brücken*, vorgestellt. Dort zeige ich Ihnen, wie Sie Daten in OneNote importieren können.

6 Datenbanken

6 Datenbanken

Dieses Kapitel zeigt Ihnen, wie Sie in Java mit gängigen Windows-Datenbanksystemen kommunizieren. Außerdem erfahren Sie, wie Sie auf Excel-Arbeitsmappen und Access-Datenbanken zugreifen, ohne die jeweilige Anwendung installieren zu müssen.

Datenbanken spielen in unserer Gesellschaft eine sehr bedeutsame Rolle. Unternehmen speichern in ihnen Kunden- und Personaldaten, Inventar- und Artikellisten. So genannte Expertensysteme enthalten Spezialwissen aus den verschiedensten wissenschaftlichen Disziplinen. Die öffentliche Verwaltung reduziert nach und nach ihre Aktenberge aus Papier und stellt auf elektronische Datenhaltung um. Auch im privaten Umfeld haben wir ständig mit Datenbanken zu tun: das Adressbuch im Handy oder Organizer, die Medienbibliothek auf dem PC oder Notebook, die Rezepte- und Cocktail-Sammlung. Es gibt unzählige Anwendungen, die sich mit Hilfe einer Datenbank realisieren lassen. Viele Jahre waren solche Systeme allerdings eine Domäne der Großrechner. Ihre immensen Anforderungen an Rechenleistung und Speicherplatz ließen einen Einsatz auf Personalcomputern nicht zu. Zudem waren sie aufwändig zu administrieren und umständlich in der Bedienung. Dies hat sich mittlerweile grundlegend geändert. Es entstanden Datenbanksysteme mit zunächst abgespecktem Leistungs- und Funktionsumfang. Heute konkurrieren diese klassischen Desktop-Anwendungen, zu denen beispielsweise *Access* als Komponente von Microsofts Office-Suite zählt, mit mächtigen Client-Server-Lösungen. In diese Kategorie gehört unter anderem der *SQL Server* von Microsoft, aber auch das Open Source-Produkt *MySQL*.

Praktisch alle Datenbanksysteme bieten neben dem interaktiven Zugriff auf ihren Bestand auch eine Schnittstelle für externe Anwendungen. Aus der Sicht des Java-Entwicklers ist es in diesem Zusammenhang wichtig, inwieweit sich das System mittels Java ansprechen lässt. In diesem Kapitel stelle ich Ihnen zwei Schnittstellen für den Zugriff auf Datenbanken vor, *JDBC* und *ODBC*. Letztere ist eine im Prinzip sprachunabhängige und plattformneutrale Funktionsbibliothek, die vor allem unter Windows äußerst weite Verbreitung gefunden hat. JDBC ist Javas Bindeglied zu Datenbanken. Beiden gemeinsam ist eine strikte Trennung zwischen einem datenbankspezifischen Treiber und einer allgemein gültigen Schicht, mit der die Anwendungen kommunizieren. Damit eine Java-Anwendung mittels JDBC auf ein Datenbanksystem zugreifen kann, bedarf es eines entsprechenden JDBC-Treibers. Gelänge es zusätzlich, ODBC für Java verfügbar zu machen, stünden automatisch alle gängigen Windows-Daten-

banken zur Verfügung. Hierzu ist eine so genannte Brücke zwischen JDBC und ODBC erforderlich, die zum Lieferumfang der Java Standard Edition gehört. Diese stelle ich Ihnen in Abschnitt 6.1.3 vor. Außerdem erfahren Sie in Abschnitt 6.2.2, wie Sie auf Access-Datenbankdateien zugreifen.

6.1 Grundlagen von Datenbanken

Ein *Datenbankmanagementsystem* ermöglicht die permanente Speicherung von Informationen in einer Datenbank. Es ist für den Zugriff auf die Daten, deren Darstellung sowie gegebenenfalls ihre Manipulation zuständig. Die Gesamtheit der in einer Datenbank gespeicherten Informationen wird *Datenbasis* genannt. Datenbank und Datenbankmanagementsystem wiederum bilden zusammen ein *Datenbanksystem*. Diese können die zu speichernden Informationen auf verschiedene Weise in der Datenbank ablegen. Die am häufigsten eingesetzte Vorgehensweise basiert auf dem *relationalen Datenmodell*. Es wurde 1970 von E.F. Codd in seinem berühmten Aufsatz *A Relational Model of Data for Large Shared Data Banks*[1] beschrieben. Unter *Relation* ist dabei im Wesentlichen ein mathematisches Modell für eine Tabelle zu verstehen. Informationen werden in zweidimensionalen Tabellen verwaltet, die über *Schlüssel* miteinander verknüpft werden können. Der Zugriff auf Datenbanken erfolgt zumeist mit Hilfe einer *Datenbanksprache*. Am weitesten Verbreitung gefunden hat die *Structured Query Language*.

6.1.1 Structured Query Language (SQL)

SQL ist eine Sprache für relationale Datenbanken. Ihre vergleichsweise einfache Syntax ist an die englische Umgangssprache angelehnt. Neben der Definition von Daten und Datenstrukturen kann die *Structured Query Language* auch für das Anlegen, Ändern und Suchen von Informationen verwendet werden. Sie hat sich im Laufe der Zeit zu einem Quasi-Standard entwickelt. Auch wenn die Hersteller von Datenbanksystemen immer wieder produktspezifische Erweiterungen und Anpassungen vornehmen, sind SQL-Anweisungen doch in gewissen Grenzen universell verwendbar. Wie solche Kommandos aussehen, möchte ich Ihnen anhand einiger Beispiele zeigen, die selbstverständlich nur ein Appetithappen sein können. Wenn Sie noch nicht mit SQL vertraut sind, aber mehr über die faszinierende Welt der Datenbanksysteme erfahren möchten, bietet das Buch *Einstieg in SQL*[2] einen guten Start für weitere Erkundungen.

1 Edgar F. Codd: A Relational Model of Data for Large Shared Data Banks. In: Communications of the ACM. 6/13/Juni 1970, ACM Press, New York, S. 377–387

2 Marcus Throll, Oliver Bartosch: Einstieg in SQL, Galileo Computing

In einer relationalen Datenbank werden die zu speichernden Informationen in *Tabellen* abgelegt. Vereinfacht gesagt legen hierbei die *Spalten* der Tabelle fest, *was* gespeichert wird. Die *Zeilen* hingegen bilden den Inhalt der Datenbank. Jede Zeile ist also ein *Datensatz*, der aus der stets gleichen Anzahl von *Feldern* besteht. Diese Felder, die Spalten der Tabelle, können Sie sich als Name-Wert-Paare vorstellen, die Elemente eines Datensatzes eindeutig beschreiben. Jedes Feld hat einen tabellenweit eindeutigen Namen sowie einen Datentyp, durch den festgelegt wird, welche Art von Daten ein Feld aufnehmen kann. Hierzu ein Beispiel: Nehmen wir an, Sie möchten in einer kleinen Datenbank Informationen zu Homecomputer-Modellen ablegen. Hierzu zählen die Produktbezeichnung, der Name des Herstellers sowie die Größe des Arbeitsspeichers. Als ersten Schritt müssen Sie eine Tabelle anlegen, die diese Daten aufnehmen kann. SQL definiert hierzu die Schlüsselwort-Abfolge `CREATE TABLE`. Eine entsprechende Anweisung enthält neben dem Namen der zu erzeugenden Tabelle auch Informationen über deren Felder, also Name und Datentyp. Abbildung 6.1 zeigt die konkrete Anwendung. Die Tabelle `computer` soll aus den drei Feldern `name`, `hersteller` und `speicher` bestehen. Die beiden ersten Felder können Zeichenketten mit maximal 32 Zeichen aufnehmen, `speicher` hingegen akzeptiert nur ganze Zahlen. Um leichter zwischen Schlüsselworten und Inhalten unterscheiden zu können, bietet es sich übrigens an, SQL-Elemente großzuschreiben, Namen von Tabellen und Fehlern hingegen klein. Dies ist allerdings reine Geschmackssache.

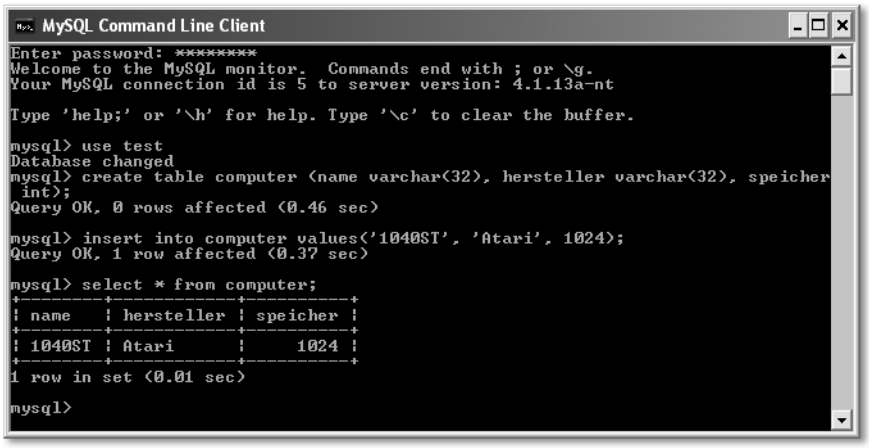

Abbildung 6.1 Beispiele für CREATE TABLE, INSERT INTO und SELECT

In einem zweiten Schritt müssen der Tabelle `computer` Inhalte hinzugefügt werden. Dies geschieht mit der SQL-Anweisung `INSERT INTO`. Wie in Abbildung 6.1 zu sehen ist, leitet das Schlüsselwort `VALUES` den einzufügenden

Datensatz ein. Zeichenketten (Herstellername und Produktbezeichnung) werden in Hochkommas eingeschlossen, der vollständige Datensatz in runde Klammern. Um Daten anzuzeigen (bzw. um nach ihnen zu suchen), wird der Befehl `SELECT` verwendet. Es handelt sich hierbei um eine äußerst mächtige Anweisung, mit deren Hilfe beispielsweise Daten aus verschiedenen Tabellen zueinander in Beziehung gesetzt oder verknüpft werden können. `SELECT` ist ferner dazu in der Lage, Daten zu aggregieren, umzuformen oder zusammenzufassen. Abbildung 6.1 zeigt die einfachste Anwendungsform dieses Befehls, die den vollständigen Inhalt einer Tabelle wiedergibt, ohne die Ausgabe zu sortieren.

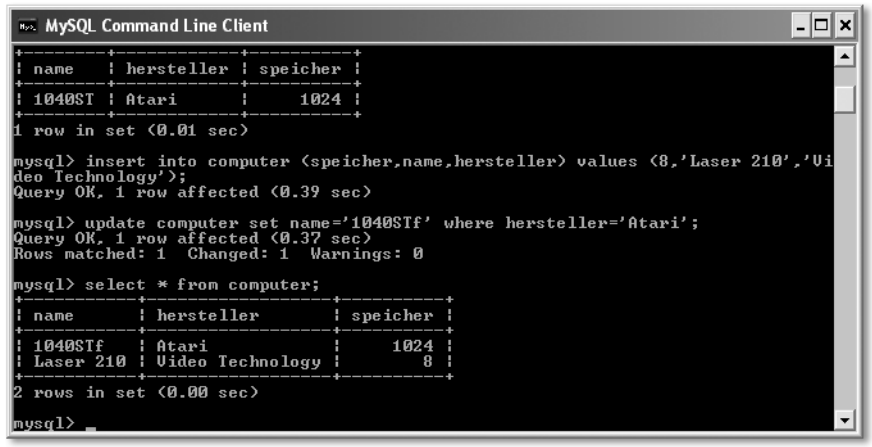

Abbildung 6.2 Die Anweisungen INSERT INTO, UPDATE und SELECT

In Abbildung 6.2 sehen Sie die `INSERT INTO`-Anweisung in einer leicht komplexeren Variante. Vor dem `VALUES`-Schlüsselwort findet sich eine in Klammern gesetzte Liste, die angibt, welche Felder durch die folgende `VALUES`-Klausel befüllt werden und in welcher Reihenfolge dies geschieht. Dies ist beispielsweise sinnvoll, wenn beim Anlegen eines Datensatzes nicht alle Informationen vorliegen. Selbstverständlich lassen sich einmal in der Datenbank abgelegte Werte jederzeit verändern. Dies geschieht mit Hilfe des `UPDATE`-Befehls. Dem Schlüsselworte `SET` folgt eine Liste von Name-Wert-Paaren, die den neuen Inhalt von Datensatzfeldern festlegt. Welche Datensätze verändert werden, wird durch eine *Bedingung* gesteuert. Sie wird durch `WHERE` eingeleitet. `UPDATE` kann übrigens auch für das »Nachtragen« fehlender Informationen verwendet werden.

Für die Eingabe und Ausführung der Beispiele habe ich das Datenbanksystem *MySQL*[3] verwendet, zweifellos einer der profiliertesten Vertreter der Open

3 http://www.mysql.com/

Source-Bewegung, der sich im Verlauf seiner Entwicklung zu einem äußerst stabilen und leistungsfähigen System entwickelt hat. Neben *MySQL Command Line Client*, einem einfachen textbasierten Frontend, das Teil der Standard-Installation von *MySQL* ist, gibt es selbstverständlich auch Werkzeuge mit grafischer Benutzeroberfläche. Abbildung 6.6 zeigt das Programm *MySQL Query Browser*, das von der Homepage des Datenbank-Herstellers heruntergeladen werden kann.[4] Ausführliche Informationen zu *MySQL* finden Sie in Marcus Throlls *MySQL 4*[5], Tipps für die Installation habe ich in Anhang A zusammengestellt. Sowohl *MySQL Command Line Client* als auch *MySQL Query Browser* nehmen Benutzereingaben entgegen und senden sie an den Datenbankserver, der die Eingaben interpretiert und ausführt, sofern es sich um gültige SQL-Kommandos handelt. Die Datenbank-Frontends wiederum nehmen die Ergebnisse der gesendeten Anweisungen entgegen und geben sie auf dem Bildschirm aus. Es findet also eine Kommunikation zwischen zwei Programmen statt, die ein herstellerabhängiges, also proprietäres Protokoll verwenden oder auf standardisierten Mechanismen basieren kann. In den folgenden beiden Abschnitten stelle ich Ihnen zwei Schnittstellen vor, die beliebigen Programmen die Kommunikation mit einem Datenbanksystem ermöglichen.

6.1.2 ODBC

Open Database Connectivity (*ODBC*) ist eine standardisierte API für den Zugriff auf Datenbanken. Die Bibliothek ist im Prinzip programmiersprachen-unabhängig und plattformneutral. Allerdings basieren konkrete Implementierungen immer auf nativem Programmcode, sodass der Zugriff nur mittels Sprachen erfolgen kann, die Einsprünge in diese Bibliotheken gestatten. *ODBC* ist auch unter UNIX, OS/2 und Mac OS verfügbar, hat aber auf Windows-basierten Systemen die größte Verbreitung gefunden. Der eigentliche Zugriff auf ein bestimmtes relationales Datenbankmanagementsystem wird nicht durch die Funktionsbibliothek selbst, sondern einen spezialisierten Treiber realisiert, der die Rolle eines Vermittlers zwischen den Anfragen, die ein Programm an die Bibliothek richtet, und dem Datenbanksystem spielt. Auf diese Weise sind Programme, die *ODBC* für die Kommunikation mit Datenbanken nutzen, nicht an ein bestimmtes Produkt gebunden. Jedes RDBMS, für das es einen entsprechenden Treiber gibt, lässt sich über die API ansprechen. Dies ist für sehr viele der unter Windows verfügbaren Datenbanksysteme der Fall. Für die Verwaltung von ODBC-Treibern und Datenbankverbindungen gibt es den zentralen Konfigurationsdialog **ODBC-Datenquellen-Administrator**. Er befindet sich unter **Datenquellen (ODBC)** im Modul **Verwaltung der Systemsteuerung**.

4 http://dev.mysql.com/downloads/query-browser/1.1.html
5 Marcus Throll: MySQL 4, Galileo Computing

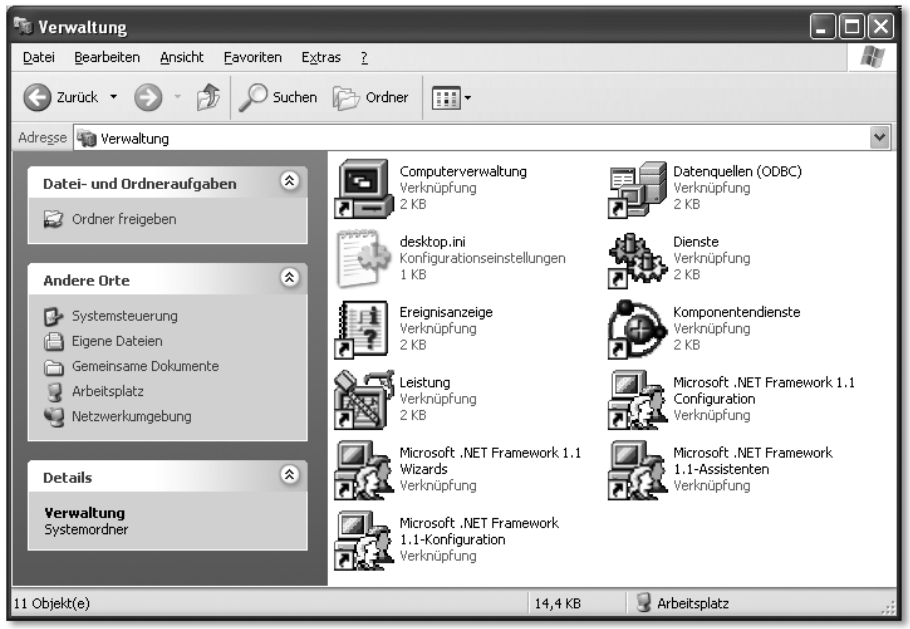

Abbildung 6.3 Das Modul »Verwaltung der Systemsteuerung«

Nach dem Öffnen des Dialogs **ODBC-Datenquellen-Administrator** können Sie auf der Registerkarte **Treiber** einen Blick auf alle installierten ODBC-Treiber werfen. Diese werden normalerweise zusammen mit dem eigentlichen Datenbanksystem installiert. Allerdings bringt auch Windows zahlreiche dateibasierte Treiber mit. Beispielsweise können Sie Access-Datenbanken anlegen und bearbeiten, ohne den Office-Bestandteil installiert zu haben. Näheres hierzu finden Sie in Abschnitt 6.2.2. In der ODBC-Terminologie greifen Programme auf *Datenquellen* zu. Diese können auf den Registerkarten **Benutzer-DSN**, **System-DSN** und **Datei-DSN** angelegt und bearbeitet werden. Zu einer Datenquelle gehören ein Treiber, ein Datenquellenname, eine optionale Beschreibung sowie etwaige treiberspezifische Einstellungen. Durch die Wahl des Treibers wird letztlich festgelegt, mit welchem Datenbanksystem und welcher Datenbank eine Anwendung kommuniziert. Beispiele für das Anlegen einer Datenquelle finden Sie in den Abschnitten 6.2.2 und 6.2.3.

ODBC ist eine Datenbankschnittstelle, die zwar für verschiedene Plattformen verfügbar ist, letztlich aber aus nativen Bibliotheken besteht. Sie kann folglich nur in Verbindung mit Programmiersprachen genutzt werden, die den Aufruf solcher APIs unterstützen. Für den Java-Entwickler ist ODBC äußerst reizvoll, weil die API den Zugang zu Datenbanksystemen gewährt, die mangels nativer JDBC-Treiber sonst verschlossen bleiben.

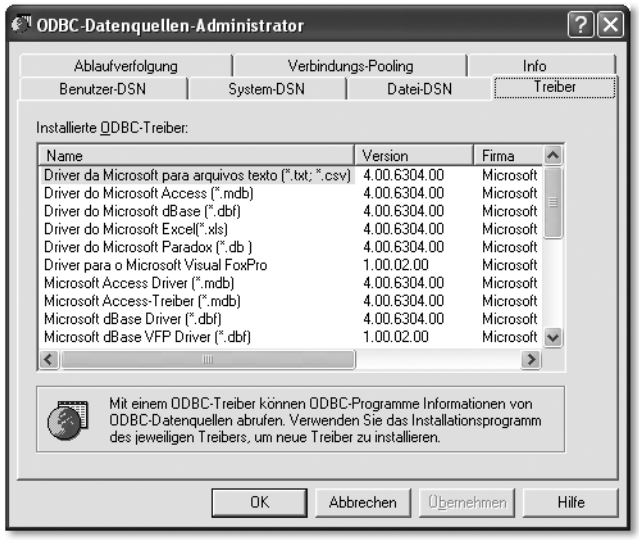

Abbildung 6.4 Die Registerkarte »Treiber«

6.1.3 JDBC und die JDBC-ODBC-Brücke

Auch Java stellt leistungsfähige Mechanismen für die Kommunikation mit Datenbanksystemen bereit. Die zur Java Standard Edition gehörende API *Java Database Connectivity* (*JDBC*) bildet wie *ODBC* eine für Anwendungen einheitliche Schnittstelle, die Aufrufe an datenbankspezifische Treiber delegiert. In Abschnitt 6.2.1 zeige ich Ihnen, wie *JDBC* funktioniert und wie Sie mittels SQL mit Datenbanken kommunizieren. Die *JDBC-ODBC-Bridge* ist ein spezieller JDBC-Treiber, der mittels *ODBC* auf ein Datenbanksystem zugreift. Auf diese Weise lassen sich in Java alle Datenbanken ansprechen, für die ein ODBC-Treiber vorhanden ist. Die Brücke wandelt hierzu JDBC-Methodenaufrufe in geeignete ODBC-Funktionsaufrufe um. Naturgemäß ist eine solche Vorgehensweise weniger performant als die Verwendung eines optimierten JDBC-Treibers für ein bestimmtes Datenbanksystem. Sie sollten die Brücke deshalb vor allem dann einsetzen, wenn kein passender JDBC-Treiber zur Verfügung steht.

6.2 JDBC im praktischen Einsatz

In diesem Abschnitt erfahren Sie, wie Sie JDBC verwenden, um auf unterschiedliche Datenbanken zuzugreifen. Sie lernen, wie ein JDBC-Treiber geladen wird und wie man Verbindungen zu einer Datenbank herstellt und trennt. Ferner zeige ich Ihnen, wie Sie SQL-Befehle absetzen und deren Ergebnisse auswerten können. Naturgemäß sind die kurzen Beispiele nur ein erster Schritt. JDBC ist eine spannende, aber auch komplexe Technologie. Wenn Sie sich aus-

führlicher mit der Materie befassen möchten, empfehle ich spezielle Lehrbücher zu Java und Datenbanken. Ein umfassendes englischsprachiges Tutorial speziell zu JDBC bietet *JDBC API Tutorial and Reference*[6]. Das deutschsprachige Werk *Datenbanken und Java*[7] stellt nicht nur JDBC vor, sondern widmet sich auch Alternativen des Java-gestützten Datenbankzugriffs, beispielsweise *SQLJ* (die Einbettung von SQL-Anweisungen unmittelbar in den Java-Quelltext).

Zunächst zum grundsätzlichen Ablauf einer JDBC-gestützten Datenbankkommunikation: Ich verwende hierfür *MySQL*, da für dieses System ein nativer JDBC-Treiber zur Verfügung steht, der zudem leicht zu installieren und zu handhaben ist. Um das Beispiel an Ihrem PC nachvollziehen zu können, sollten Sie zumindest die MySQL-Kernkomponenten auf Ihrem System eingerichtet haben. Wie Sie hierbei vorgehen, können Sie gegebenenfalls in Anhang A nachlesen.

6.2.1 MySQL

In *MySQL* werden so genannte Konnektoren für die Kommunikation von Anwendungen und Tools mit dem Datenbankserver verwendet. Der *MySQL Connector/J* bindet Java-Anwendungen mittels *JDBC* an *MySQL* an. Sie können ihn von der Homepage des Datenbankherstellers herunterladen.[8] Die Installation des Treibers ist äußerst einfach. Nach dem Entpacken des Archivs müssen Sie nur dafür sorgen, dass die Java-Laufzeitumgebung das entsprechende *.jar*-Archiv findet. Hierzu können Sie Ihren Klassenpfad um den voll qualifizierten Dateinamen ergänzen oder das *.jar*-Archiv in das Java-Erweiterungsverzeichnis kopieren. Eine ausführliche Anleitung hierzu finden Sie in Kapitel 1, *Installation und Konfiguration*.

Mit dem Programm *MySQLDemo1* zeige ich Ihnen, wie Sie in MySQL eine Tabelle mit Namen `personen` anlegen, die aus den beiden Feldern `name` und `gebdt` besteht.

```
package javafuerwindows.jdbc;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
import java.sql.Statement;

public class MySQLDemo1 {
```

6 Jonathan Bruce, Jon Ellis, Maydene Fisher: JDBC API Tutorial and Reference, Addison-Wesley Professional
7 Gunter Saake, Kai-Uwe Sattler: Datenbanken und Java, Dpunkt Verlag
8 http://dev.mysql.com/downloads/connector/j/3.1.html

```
    public static void main(String [] args) {
        try {
            Class.forName("com.mysql.jdbc.Driver");
            Connection conn =
        DriverManager.getConnection("jdbc:mysql://localhost/
                          test?user=batch&password=geheim");
            Statement statement = conn.createStatement();
            statement.executeUpdate("CREATE TABLE
                  personen (name varchar(32), gebdt int)");
            statement.close();
            conn.close();
        } catch (ClassNotFoundException e) {
            System.err.println(e);
        } catch (SQLException e) {
            System.err.println(e);
        }
    }
}
```

Listing 6.1 MySQLDemo1.java

Das Programm legt die Tabelle `personen` in der Datenbank `test` an, die im Verlauf der Installation von MySQL automatisch bereitgestellt wird. Damit *MySQLDemo1* funktioniert, müssen Sie vor dem Programmstart einen MySQL-Benutzer *batch* mit dem Passwort *geheim* angelegt haben und diesen Benutzer mit den nötigen Rechten versehen, damit er beispielsweise Tabellen anlegen kann. Wie dies funktioniert, können Sie in Anhang A, *Installation von MySQL*, nachlesen. In Abbildung 6.5 sehen die durch *MySQLDemo1* erzeugte Tabelle im Dienstprogramm *MySQL Administrator*.

Das Erzeugen der Tabelle `personen` beginnt mit dem Laden des JDBC-Treibers, der die Kommunikation mit dem Datenbanksystem übernehmen wird. Hierzu rufen Sie die Methode `forName()` der Klasse `Class` mit dem voll qualifizierten Klassennamen des Treibers als Argument auf. Welche Klasse hier anzugeben ist, entnehmen Sie der Dokumentation des entsprechenden JDBC-Treibers. Falls Java den Treiber nicht finden kann, wird die Ausnahme `ClassNotFoundException` ausgeworfen. Die Klasse `DriverManager` ist für das Herstellen von Datenbankverbindungen verantwortlich. Deshalb wird in einem zweiten Schritt ihre Klassenmethode `getConnection()` aufgerufen.

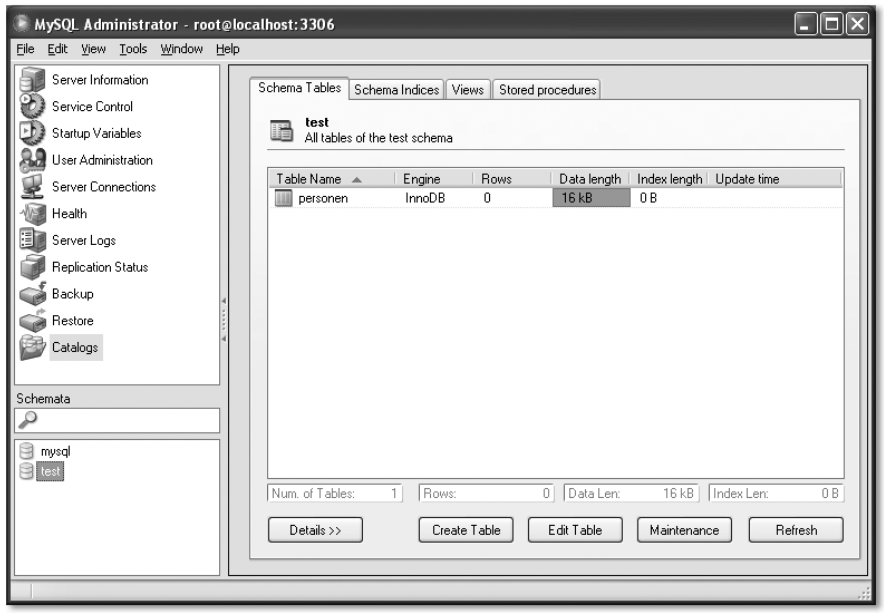

Abbildung 6.5 Die erzeugte Tabelle im MySQL Administrator

Der übergebene `String` *url* nennt den zu verwendenden Treiber, die anzusprechende Datenbank sowie Benutzername und Passwort. Allerdings ist sein Aufbau letztlich vom verwendeten JDBC-Treiber abhängig. Nach der Zeichenkette `jdbc:` folgt der Name des Treibers. Alle weiteren Angaben sind treiberabhängig und müssen in der Dokumentation des JDBC-Treibers nachgelesen werden. Falls eine Verbindung zur Datenbank hergestellt werden konnte, liefert `getConnection()` eine Instanz der Klasse `java.sql.Connection`. Dieses Objekt wiederum stellt die Instanzmethode `createStatement()` zur Verfügung, die Objekte des Typs `java.sql.Statement` zurückgibt. An solche `Statement`-Objekte werden die eigentlichen SQL-Anweisungen übergeben, beispielsweise durch Aufruf ihrer Instanzmethode `executeUpdate()`.

Nachdem Sie mit dem Programm *MySQLDemo1* die Tabelle `personen` angelegt haben, können Sie dort Namen und Geburtstage eintragen. Dies geht sehr einfach mit dem *MySQL Query Browser*. Wenn Sie diese Anwendung starten, werden Sie nach einigen Verbindungsdaten gefragt. Was Sie dort eintragen können, sehen Sie in Abbildung 6.6. Der Query Browser ist die benutzerfreundliche Variante des *MySQL Command Line Client*. Mit seiner Hilfe können Sie Anfragen formulieren, aber auch bestehende Felder ändern oder Datensätze löschen. Um in der Tabelle `personen` einen Namen und ein Geburtsdatum einzutragen, geben Sie in der Eingabezeile im oberen Bereich des Programmfensters bitte den folgenden SQL-Befehl ein: `insert into per-`

sonen values ('Max Mustermann', 19650304). Nach dem Absetzen der Anweisung durch Anklicken des grünen **Execute**-Pfeils finden Sie in der Statuszeile die Meldung `1 row affected by the last command, no resultset returned.`. Der Begriff *Resultset* wird Ihnen auch bei der JDBC-Programmierung sehr häufig begegnen, ich komme gleich auf ihn zurück.

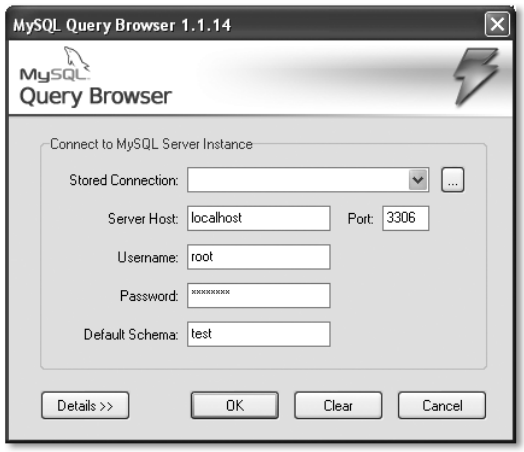

Abbildung 6.6 Der Startdialog des MySQL Query Browsers

Mit Hilfe des Programms *MySQLDemo2* möchte ich Ihnen zeigen, wie Sie in Java Tabellen abfragen können. Sehen Sie sich hierzu zunächst den Quelltext des Beispiels an.

```
package javafuerwindows.jdbc;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;

public class MySQLDemo2 {
    public static void main(String [] args) {
        try {
            Class.forName("com.mysql.jdbc.Driver");
            Connection conn =
                  DriverManager.getConnection("jdbc:mysql://
                localhost/test?user=batch&password=geheim");
            Statement statement = conn.createStatement();
            ResultSet resultSet =
           statement.executeQuery("SELECT * FROM personen");
```

```
            if (resultSet != null) {
                int anzSpalten =
                resultSet.getMetaData().getColumnCount();
                while ( resultSet.next() ) {
                    for ( int i = 1 ; i <= anzSpalten ;
                          i++ ) {
                        System.out.println( "Spalte " + i
                         + " = " + resultSet.getObject(i) );
                    }
                }
            }
            statement.close();
            conn.close();
        } catch (ClassNotFoundException e) {
            System.err.println(e);
        } catch (SQLException e) {
            System.err.println(e);
        }
    }
}
```

Listing 6.2 MySQLDemo2.java

Das Laden des Treibers und das Herstellen einer Datenbankverbindung kennen Sie bereits. Allerdings wird die SQL-Anweisung nicht durch die Methode `executeUpdate()`, sondern `executeQuery()` des `Statement`-Objekts ausgeführt. Sie liefert eine Instanz der Klasse `java.sql.ResultSet`. Stellen Sie sich ein Resultset als eine Tabelle vor, die die Ergebnisse einer Datenbankabfrage enthält. Mit Hilfe der Instanzmethode `next()` können Sie diese »Tabelle« zeilenweise abarbeiten. Vereinfacht ausgedrückt ist immer eine ihrer Zeilen aktiv. Auf die Elemente dieser Zeile können Sie mit verschiedenen Methoden zugreifen, beispielsweise `getObject()`. Der übergebene Parameter gibt an, für welche Spalte Sie sich interessieren. Abbildung *6.7* zeigt die Bildschirmausgabe von *MySQLDemo2* in *NetBeans*.

Nachdem Sie nun mit einigen Grundlagen der JDBC-Programmierung vertraut sind, widmen wir uns im Folgenden den ODBC-Datenquellen. Wie Sie bereits wissen, ist der Java-seitige Zugriff auf sie besonders dann interessant, wenn es keine speziellen JDBC-Treiber gibt. Dies ist bei den meisten klassischen Windows-Datenbanken, beispielsweise Microsofts *Access* oder *Visual FoxPro* der Fall.

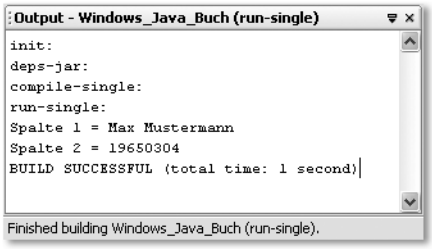

Abbildung 6.7 MySQLDemo2 nach der Programmausführung in NetBeans

6.2.2 Microsoft Access

Access ist Bestandteil von Microsofts Office-Suite und damit neben *Word*, *Excel* und *Powerpoint* wahrscheinlich eine der am häufigsten eingesetzten Anwendungen überhaupt. Gegenstand dieses Abschnittes ist aber nicht die Arbeit mit Access, sondern die Frage, wie Sie in Ihren Java Programmen auf bestehende Access-Datenbankdateien zugreifen können. Dies ist besonders deshalb interessant, weil jede Windows XP-Version alles Notwendige mitbringt, um mit Bordmitteln Access-Datenbanken zu erzeugen und zu bearbeiten.

Bevor Sie mittels Java und JDBC auf eine Access-Datenbank zugreifen können, müssen Sie eine neue Datenquelle anlegen. Hierfür verwenden Sie den Dialog **ODBC-Datenquellen-Administrator**, den ich Ihnen in Abschnitt 6.1.2 bereits vorgestellt habe. Öffnen Sie bitte zunächst die **Systemsteuerung**, dann **Verwaltung** und anschließend **Datenquellen (ODBC)**.

Abbildung 6.8 Der ODBC-Datenquellen-Administrator

Die Registerkarte **Benutzer-DSN** zeigt alle vorhandenen Benutzerdatenquellen an. Hierbei handelt es sich um Datenquellen, die nur für denjenigen Benutzer sichtbar sind, der sie angelegt hat. Wenn Sie stattdessen systemweit sichtbare Datenquellen anlegen möchten, klicken Sie bitte auf **System-DSN**. Klicken Sie anschließend auf **Hinzufügen**, um eine neue Datenquelle zu erstellen.

Abbildung 6.9 Der Dialog »Neue Datenquelle erstellen«

Der in Abbildung 6.9 gezeigte Dialog **Neue Datenquelle erstellen** enthält eine Liste aller verfügbaren ODBC-Datenbanktreiber. Da Sie mit Access-Datenbankdateien arbeiten möchten, wählen Sie **Microsoft Access-Treiber** und klicken auf **Fertig stellen**. Es erscheint erneut ein Dialog: **ODBC Microsoft Access Setup**. Er ist in Abbildung 6.10 zu sehen. In diesem Dialog bestimmen Sie die physikalisch zu verwendende Access-Datenbankdatei. Sie können hierzu eine neue, leere Datei erzeugen oder auf bereits bestehende Dateien zugreifen. Um eine neue Datei anzulegen, wählen Sie **Erstellen**.

Abbildung 6.10 Der Dialog »ODBC Microsoft Access Setup«

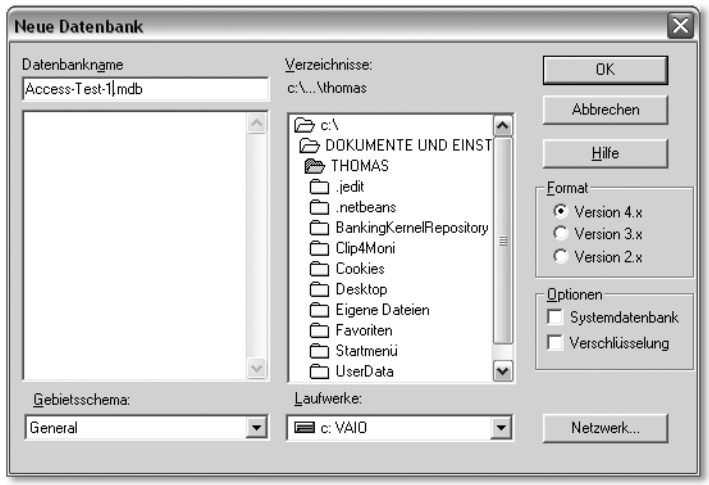

Abbildung 6.11 Der Dialog »Neue Datenbank«

Nachdem Sie eine neue Access-Datenbankdatei erzeugt haben, müssen Sie im Dialog **ODBC Microsoft Access Setup** noch einen Datenquellennamen und (optional) eine Beschreibung der Datenquelle vergeben. Um die Aktion abzuschließen, beenden Sie die noch geöffneten Dialoge jeweils mit **OK**. Der von Ihnen vergebene Datenquellenname ist sehr wichtig, weil er innerhalb von ODBC als ein Verweis auf die von Ihnen ausgewählte physikalische Datendatei behandelt wird. Wenn Sie mittels ODBC auf eine Datenbank zugreifen, müssen Sie immer eine Datenquelle angeben. Wie Sie gleich sehen werden, trifft dies auch auf die *JDBC-ODBC-Bridge* zu.

```
package javafuerwindows.jdbc;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.ResultSetMetaData;
import java.sql.Statement;

public class AccessDemo1 {

    private static final String treiber =
                "sun.jdbc.odbc.JdbcOdbcDriver";

    private static final String quelle =
                              "jdbc:odbc:Access-Test";
```

```
private static final String sqlCreateTable =
      "CREATE TABLE kunden (" +
        "KundenID INTEGER NOT NULL," +
        "Nachname TEXT(50) NOT NULL," +
        "Vorname TEXT(50) NOT NULL)";

private static final String sqlInsert =
        "INSERT INTO kunden " +
        "(KundenID, Nachname, Vorname) " +
        "VALUES (1, \'Künneth\', \'Thomas\')";

private static final String sqlSelect =
              "SELECT * FROM kunden";

public static void main( String[] args ) {
    Connection conn = null;
    Statement statement = null;
    ResultSet rs = null;
    ResultSetMetaData metaData = null;
    try {
        Class.forName(treiber);
        conn = DriverManager.getConnection(quelle);
        statement = conn.createStatement();
        statement.executeUpdate(sqlCreateTable);
        statement.executeUpdate(sqlInsert);
        rs = statement.executeQuery(sqlSelect);
        if (rs != null) {
            metaData = rs.getMetaData();
            int cols = metaData.getColumnCount();
            String text = "";
            for (int i = 1; i <= cols; ++i)
                text += metaData.getColumnName(i) + " ";
            text += "\n";
            System.out.println(text);
            int colCount = 1;
            while (rs.next() == true) {
                text = "";
                for (int i = 1; i <= cols; i++)
                    text += rs.getString(i) + " ";
                text += "\n";
```

```
                    System.out.println(colCount++ + ": "
                                       + text);
                }
            }
            statement.close();
            conn.close();
        } catch (Exception ex) {
            System.out.println( ex.toString() );
        }
    }
}
```

Listing 6.3 AccessDemo1.java

Das Programm *AccessDemo1* legt die Tabelle `kunden` an, trägt einen Namen und eine Kunden-ID ein und selektiert anschließend »alle« Einträge. Abbildung 6.12 zeigt seine Bildschirmausgaben.

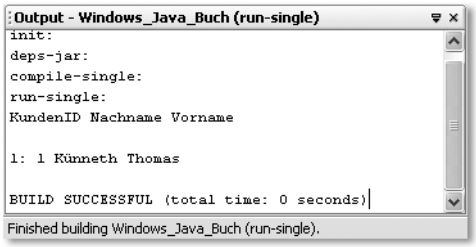

Abbildung 6.12 Die Bildschirmausgabe von AccessDemo1 in NetBeans

Damit *AccessDemo1* funktioniert, müssen Sie im Dialog **ODBC-Datenquellen-Administrator** eine Datenquelle mit dem Namen `Access-Test` angelegt haben. Als Treiber wird die *JDBC-ODBC-Bridge* verwendet, deren voll qualifizierter Klassenname `sun.jdbc.odbc.JdbcOdbcDriver` an `Class.forName()` übergeben wird. Interessant ist in diesem Zusammenhang auch der *url*-Parameter, der an die Methode `DriverManager.getConnection()` übergeben wird: Auf das obligatorische `jdbc:` folgt `odbc:`. Hiermit wird die *JDBC-ODBC-Bridge* als zu verwendender JDBC-Treiber festgelegt. Schließlich folgt noch der Datenquellenname, der zunächst an die Brücke und von ihr dann an den Access-Treiber weitergegeben wird.

Der Zugriff auf ODBC-Datenquellen durch die JDBC-ODBC-Brücke unterscheidet sich – wie Sie gesehen haben – in keiner Weise vom Zugriff auf Datenbanken, die mittels nativer JDBC-Treiber an Java angebunden sind. Selbstverständlich müssen Sie die Namen der anzusprechenden Treiber nicht hart in Ihren

Anwendungen kodieren, sondern können diese zur Laufzeit durch den Anwender auswählen lassen. Dies gilt auch für die *Quelle*, die an `Class.forName()` übergeben wird.

Im Folgenden stelle ich Ihnen eine weitere Verwendungsmöglichkeit der JDBC-ODBC-Brücke vor, den Zugriff auf Excel-Arbeitsmappen.

6.2.3 Excel als Datenbank

ODBC gestattet nicht nur den Zugriff auf »klassische« Datenquellen wie Access- oder dBASE-Datenbankdateien. Vielleicht ist Ihnen beim Stöbern in der Treiberliste des Dialogs **Neue Datenquelle erstellen** der Eintrag **Microsoft Excel-Treiber** oder sein englisches Pendant aufgefallen. Mit ihm ist es möglich, auf Excel-Arbeitsmappen und -blätter mit SQL-Anweisungen fast genauso zuzugreifen wie auf eine herkömmliche relationale Datenbank. Zwar steht, wie später in Kapitel 5, *Kommunikation mit Microsoft Office*, gezeigt wird, mit *POI* bzw. *HSSF* eine mächtige und obendrein plattformunabhängige Schnittstelle zu Excel zur Verfügung. Allerdings ist der von POI verfolgte Ansatz naturgemäß zellbasiert und stellt keine Mechanismen bereit, einzelne Teile einer Excel-Tabelle zu anderen in Beziehung zu setzen. Sie können beispielsweise nicht ohne weiteres alle Zeilen filtern, die ein bestimmtes Kriterium erfüllen. Eine solche Auswahl müssten Sie explizit in Ihrem Programm implementieren. Im Gegensatz dazu genügt unter Verwendung von SQL eine entsprechende `SELECT`-Anweisung.

Um mittels JDBC auf Excel-Arbeitsblätter zuzugreifen, müssen Sie zunächst – wie schon bei Access – eine neue Datenquelle anlegen, die die zu bearbeitende Mappe referenziert. Hierzu öffnen Sie die **Systemsteuerung**, wählen **Verwaltung** und anschließend **Datenquellen (ODBC)**. Im Ihnen mittlerweile vertrauten Dialog **ODBC-Datenquellen-Administrator** klicken Sie bitte auf die Registerkarte **Benutzer-DSN**. Die Schaltfläche **Hinzufügen** öffnet den Dialog **Neue Datenquelle erstellen**, in dem Sie den **Microsoft Excel-Treiber** auswählen.

Im nun angezeigten Dialog **ODBC Microsoft Excel Setup** vergeben Sie einen möglichst aussagekräftigen Datenquellennamen und eine (optionale) Beschreibung. Durch Klicken auf **Arbeitsmappe auswählen** erhalten Sie eine Dateiauswahl angezeigt, in der Sie die mittels JDBC zu bearbeitende Excel-Arbeitsmappe auswählen. Die Schaltfläche **Optionen** macht zwei zusätzliche Einstellmöglichkeiten sichtbar. Zum einen können Sie festlegen, wie viele Zeilen eines Arbeitsblattes durch den ODBC-Treiber berücksichtigt werden sollen. Zum anderen können Sie den schreibenden Zugriff auf die Datei einschalten oder verhindern.

Abbildung 6.13 Der Dialog »ODBC Microsoft Excel Setup«

Bei der Verwendung des Excel-ODBC-Treibers sind einige technische Restriktionen zu beachten, auf die Microsoft in einem Knowledgebase-Eintrag hinweist.[9] Leider kann der Treiber keine `UPDATE`-, `DELETE`- und `ALTER TABLE`-Anweisungen verarbeiten, `INSERT` hingegen ist möglich. Zudem sollten Ihnen die folgenden grundlegenden konzeptionellen Unterschiede zwischen Datenbanktabellen und den Arbeitsblättern einer Tabellenkalkulation bewusst sein. Oberflächlich betrachtet sehen beide genau gleich aus. Allerdings kann der Inhalt einer Excel-Tabelle weitaus freier gestaltet werden, als dies das spaltenbasierte Format einer Datenbanktabelle erlaubt. Hier haben alle Werte einer Spalte zwingend den gleichen Typ. Nicht so in einem Excel-Arbeitsblatt: Der Inhalt jeder einzelnen Zelle ist beliebig. Aus diesem Grund muss Ihre Anwendung beim Abarbeiten eines Resultset vor dem Zugriff auf einen Wert diesen sehr genau prüfen, um keine Typverletzung zu provozieren.

Ein Beispiel für den Zugriff auf eine Excel-Arbeitsmappe finden Sie im Programm *ExcelDemo1*, dessen vollständiger Quelltext sich auf der Begleit-CD befindet. Ein Unterschied im Vergleich zu meinen bisherigen JDBC-Beispielen ergibt sich im Hinblick auf die Angabe von Tabellennamen. Excel-Arbeitsmappen können mehrere Arbeitsblätter enthalten. Jedes dieser Arbeitsblätter entspricht hierbei einer eigenständigen Tabelle. Der Tabellenname entspricht zwar dem Namen des Arbeitsblattes, wird aber in `[ $]` geklammert. Hierzu ein kurzer Ausschnitt aus *ExcelDemo1.java*. Der in Excel angezeigte Name des Arbeitsblattes lautet *Sheet0*.

```
private static final String sqlSelect = "SELECT * FROM
                                        [Sheet0$]";
```

9 http://support.microsoft.com/kb/178717/en-us

Mit dem Excel-ODBC-Treiber ist es trotz gewisser Einschränkungen möglich, unter Verwendung von JDBC sowohl lesend als auch schreibend auf Excel-Arbeitsmappen zuzugreifen. Da er mit allen derzeit gängigen Windows-Versionen ausgeliefert wird, kann er unter bestimmten Voraussetzungen als Alternative zu *POI* in Betracht gezogen werden, beispielsweise wenn die Installation zusätzlicher Klassenbibliotheken auf dem Zielsystem problematisch ist.

Im nächsten Abschnitt stelle ich Ihnen Werkzeuge vor, die die Arbeit mit bestimmten Datenbanksystemen erleichtern.

6.3 Werkzeuge und Bibliotheken

Die Hersteller von Datenbanksystemen bieten zu ihren Produkten meistens Werkzeuge für die Administration und Datenpflege an. Beispielsweise haben Sie bereits den *MySQL Query Browser* sowie den *MySQL Administrator* kennen gelernt. Im Folgenden stelle ich Ihnen weitere hilfreiche Geister vor, beispielsweise ein datenbankunabhängiges Frontend sowie Tools zu Microsoft Access.

6.3.1 SQirreL SQL Client

Für spontane Datenbank-Recherchen ist das Schreiben von Programmen natürlich zu aufwändig. In solchen Fällen greift man zu so genannten Datenbank-Frontends, die eine unmittelbare Kommunikation mit der Datenbank gestatten. Das Internet bietet unzählige, zumeist kostenlose Programme, die die Arbeit mit beliebigen Datenquellen erlauben. Ein sehr empfehlenswertes Programm ist *SQuirreL SQL Client*[10], eine Swing-basierte Anwendung, die einen komfortablen Zugriff auf JDBC-Datenquellen bietet. Die Installation verläuft IzPack-gestützt und ist mit einigen wenigen Mausklicks erledigt.

Nach dem ersten Programmstart müssen Sie Ihre Datenquellen konfigurieren. Wie Sie hierzu vorgehen, zeige ich Ihnen anhand des Access-ODBC-Treibers. Um auf eine Datenquelle zugreifen zu können, muss ein geeigneter JDBC-Treiber vorhanden sein. Die Konfiguration dieser Treiber erfolgt mit Hilfe der Palette **Drivers**. Um nicht benötigte Treiber auszublenden, klicken Sie auf das Icon am rechten Rand der Palette. Die Ihnen bereits bekannte *JDBC-ODBC-Bridge* sollte weiterhin zu sehen sein. Nach einem Doppelklick auf einen Treibernamen öffnet sich das Fenster **Change Driver**, mit dessen Hilfe ein Treiber konfiguriert werden kann. Richtig konfigurierte Treiber sind an einem Häkchen vor ihrem Namen zu erkennen.

10 http://squirrel-sql.sourceforge.net/

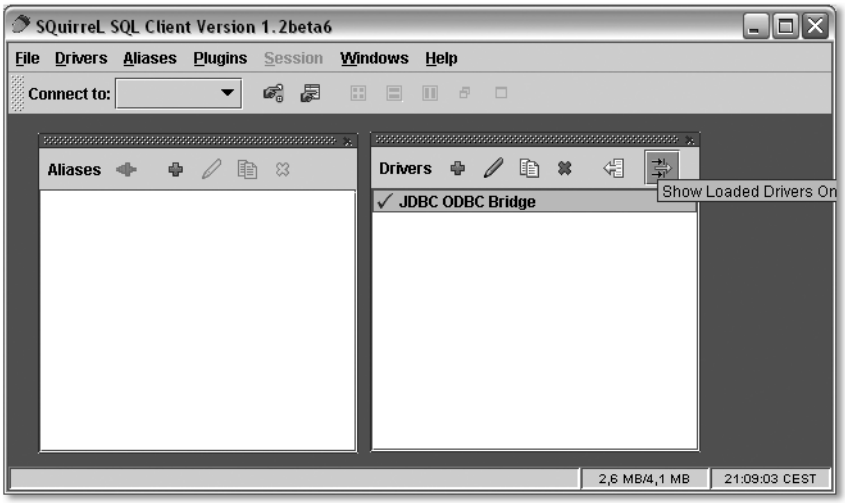

Abbildung 6.14 SQuirreL SQL Client nach dem ersten Programmstart

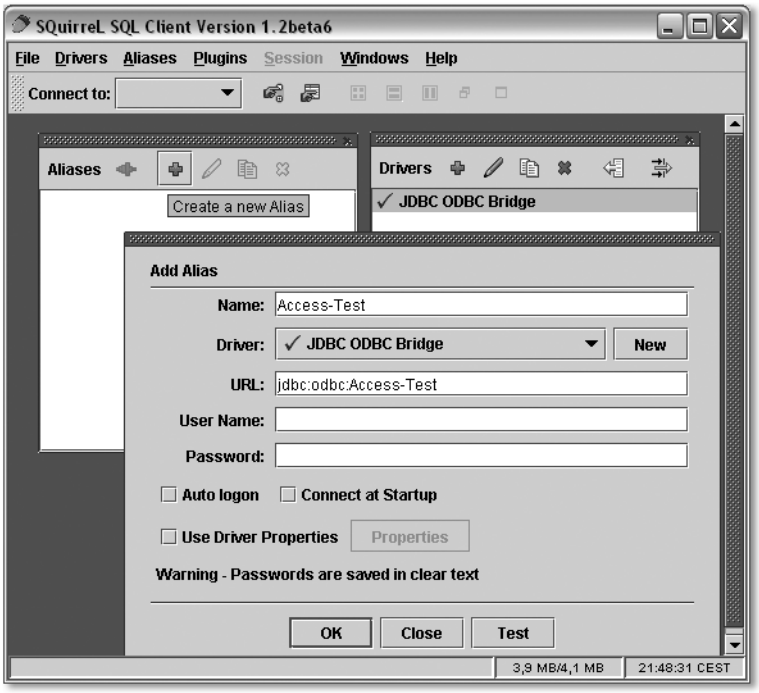

Abbildung 6.15 Das Fenster »Add Alias«

Verbindungen zu Datenquellen werden mit der Palette **Aliases** verwaltet. Um eine neue Verbindung zu erstellen, klicken Sie auf das Pluszeichen, woraufhin sich das Fenster **Add Alias** öffnet. Unter **Name** können Sie einen beliebigen

Namen vergeben, aus Gründen der Übersichtlichkeit bietet es sich aber an, den gleichen Namen zu verwenden, den Sie auch im Dialog **ODBC Microsoft Access Setup** vergeben haben. Bei **Driver** wählen Sie bitte die JDBC-ODBC-Bridge aus. Bei **URL** tragen Sie bitte `jdbc:odbc: ein`, gefolgt vom Namen der Datenquelle, so wie Sie ihn im Dialog **ODBC Microsoft Access Setup** eingetragen haben. Nachdem Sie **Add Alias** mit **OK** beendet haben, erscheint automatisch der Dialog **Connect To:**. Sie können auch jederzeit in der Toolbar aus der Klappbox eine Verbindung auswählen. Wenn Sie beim Anlegen der Datenbank kein Passwort vergeben haben, können Sie die Felder **User** und **Password** leer lassen. **Connect** stellt die Verbindung zur Datenquelle her.

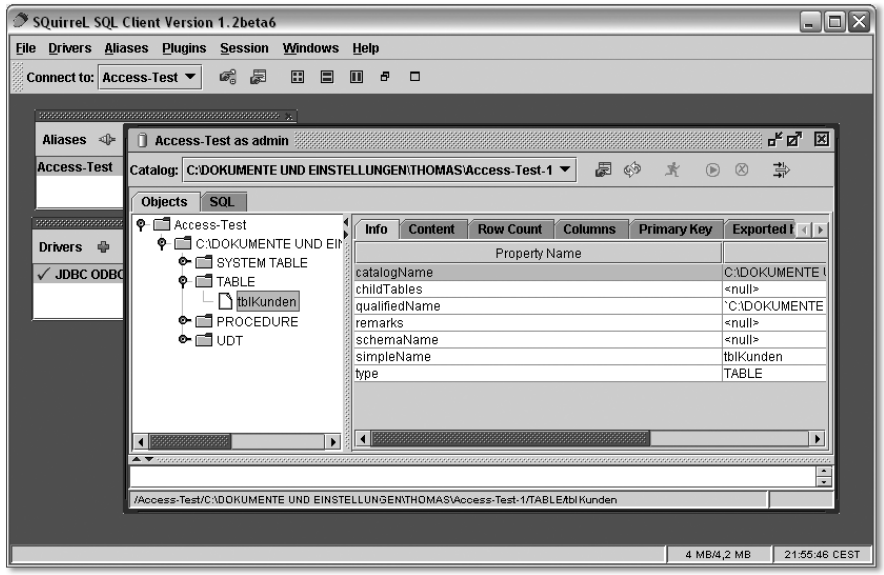

Abbildung 6.16 SQuirreL SQL Client mit geöffneter Datenbank

Verbindungen zu Datenbanken mittels nativer JDBC-Treiber stellen Sie auf die gleiche Weise her. So ist es etwa leicht möglich, *SQuirreL SQL Client* anstelle des *MySQL Query Browsers* zu verwenden.

6.3.2 Jackcess

Jackcess ist eine Klassenbibliothek, die ohne Zuhilfenahme der JDBC-ODBC-Brücke auf Access-Datenbankdateien zugreifen kann. Wie *POI* ist *Jackcess* plattformunabhängig, kann also auch auf Nicht-Windows-Systemen eingesetzt werden. Die Software steht unter der GNU Lesser General Public Licence und kann kostenlos von der Projekt-Homepage[11] heruntergeladen werden.

11 http://jackcess.sourceforge.net/

Außerdem finden Sie die zum Zeitpunkt der Drucklegung aktuelle Version auf der Begleit-CD. Die Installation der API erfolgt wie bei praktisch allen in diesem Buch vorgestellten Bibliotheken durch Hinzufügen des *.jar*-Archivs zum Klassenpfad oder das Kopieren der Datei in das Java-Erweiterungsverzeichnis.

Eine Besonderheit von *Jackcess* ist, dass die API in ihrer Verwendung deutlich von *JDBC* abweicht. Allerdings sind Klassen- und Methodennamen intuitiv gewählt, sodass eine Einarbeitung nicht schwer fällt. Im folgenden Beispiel zeige ich Ihnen, wie Sie die durch das Programm *AccessDemo1* erzeugte Tabelle `kunden` aus Abschnitt 6.2.2 mit *Jackcess* auslesen können.


```
package javafuerwindows.jackcess;
import com.healthmarketscience.jackcess.Column;
import com.healthmarketscience.jackcess.Database;
import com.healthmarketscience.jackcess.Table;
import java.io.File;
import java.util.Iterator;
import java.util.List;
import java.util.Map;
import java.util.Set;

public class JackcessDemo1 {

    public static void main(String [] args) {
        try {
            String name = "C:\\Dokumente und
               Einstellungen\\Thomas\\Access-Test1.mdb";
            Database db = Database.open(new File(name));
            Set set = db.getTableNames();
            Iterator iter = set.iterator();
            while (iter.hasNext()) {
                name = (String) iter.next();
                System.out.println("zeige Tabelle: "
                        + name);
                Table tabelle = db.getTable(name);
                List spalten = tabelle.getColumns();
                System.out.print("Spaltennamen: ");
                for (int i = 0; i < spalten.size(); i++) {
                    Column spalte = (Column) spalten.get(i);
                    System.out.print(spalte.getName() +
                        " ");
```

```
                }
                System.out.println();
                Map zeile;
                int zeilennummer = 1;
                while ((zeile = tabelle.getNextRow())
                          != null) {
                    System.out.println("Zeilennummer: "
                           + zeilennummer);
                    for (int i = 0; i < spalten.size();
                                i++) {
                        Column spalte =
                          (Column) spalten.get(i);
                        String spaltenname =
                           spalte.getName();
             System.out.print(zeile.get(spaltenname) + " ");
                    }
                    System.out.println();
                }
            }
        } catch (Exception e) {
            System.err.println(e);
        }
    }
}
```

Listing 6.4 JackcessDemo1.java

Derzeit nennt die Projekt-Homepage noch eine Reihe von Abhängigkeiten[12] zu anderen Open Source-Projekten, unter anderem den *Jakarta Commons*[13]. Um *JackcessDemo1* ausführen zu können, müssen Sie die Komponenten *Logging*, *Lang* und *Collections* von der *Commons*-Homepage herunterladen und Ihrem Klassenpfad hinzufügen bzw. in das Java-Erweiterungsverzeichnis kopieren. Alternativ können Sie die Versionen installieren, die sich auf der Begleit-CD zum Buch befinden. Der Programmablauf lässt sich in einigen wenigen Schritten zusammenfassen. Zunächst wird mittels `Database.open()` eine Instanz der Klasse `com.healthmarketscience.jackcess.Database` erzeugt, die die Datenbank repräsentiert. Die Namen der enthaltenen Tabellen sowie die Tabellen selbst können mit den Methoden `getTableNames()` und `getTable()` ermittelt werden. Tabellen werden in Instanzen der Klasse `com.healthmar-`

12 http://jackcess.sourceforge.net/dependencies.html
13 http://jakarta.apache.org/commons/

`ketscience.jackcess.Table` gespeichert. Das Lesen der eigentlichen Daten erfolgt mit Hilfe der Methode `getNextRow()`, die eine Tabellenzeile in Instanzen von `java.util.Map` abbildet.

Jackcess bietet eine vollständig plattformunabhängige Möglichkeit, Access-Datenbanken zu lesen und zu schreiben. Die API empfiehlt sich deshalb vor allem für Nicht-Windows-Systeme, kann aber auch dann eingesetzt werden, wenn die JDBC-ODBC-Brücke nicht verwendet werden kann oder soll.

7 Die JDesktop Integration Components

7 Die JDesktop Integration Components

Die JDesktop Integration Components sind eine Sammlung von Klassen, die für eine bessere Integration von Java-Anwendungen in das Wirtssystem sorgen. Sie erlauben beispielsweise, Dokumenttypen zu registrieren oder E-Mails zu versenden.

Windows-Anwender sind gewohnt, dass nach dem Doppelklick auf eine Datei die für den jeweiligen Dokumenttyp zuständige Anwendung gestartet und das Dokument geladen wird. Ein weiterer häufig genutzter Mechanismus ist das Anzeigen von Icons im Infobereich der Taskleiste. Schließlich nutzen zahlreiche Programme die Rendering Engine des Internet Explorers für das Darstellen von HTML-Seiten. Ganz sicher würden auch Java-Anwendungen von solchen Funktionen profitieren. Das Open Source-Projekt *JDesktop Integration Components* (JDIC) möchte genau dies leisten. In den folgenden Abschnitten werde ich Ihnen die wichtigsten *JDIC*-Pakete vorstellen und zeigen, wie Sie diese in Ihren eigenen Anwendungen ansprechen können.

7.1 Dateitypen

Die Funktionalität, durch Doppelklick auf ein Dokument die ihm zugeordnete Anwendung zu starten, findet sich in praktisch allen modernen grafischen Benutzeroberflächen.

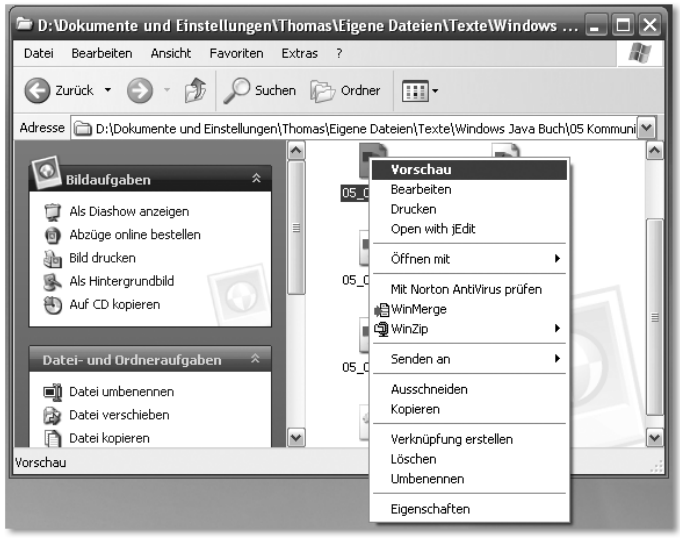

Abbildung 7.1 Ein Kontextmenü mit verschiedenen dateibezogenen Aktionen

Diese häufig **Öffnen** genannte Funktion wird in Windows durch Kontextmenüs erweitert. Diese enthalten zusätzliche Aktionen, die auf eine Datei angewendet werden können, beispielsweise **Drucken** oder **Bearbeiten**. Aktionen beziehen sich im Allgemeinen nicht auf bestimmte einzelne Dateien, sondern *Dateitypen*.

Ein Dateityp besteht hierbei aus einer Beschreibung, mindestens einer Dateiendung, einem Mime-Typ sowie einem Icon. Im Prinzip können für ihn beliebig viele Aktionen registriert werden. Jede dieser Aktionen beinhaltet eine Beschreibung, ein Kommando sowie ein Verb. Die Klassen des Paketes `org.jdesktop.jdic.filetypes` bilden diese durch Windows vorgegebene logische Struktur sehr genau nach. Mit ihnen ist es möglich, Dateitypen und Aktionen zu ermitteln, zu registrieren sowie diese gegebenenfalls wieder aufzuheben.

7.1.1 Ermitteln von Dateitypen und Aktionen

In einem ersten Beispiel sehen Sie, wie Sie Informationen zu einem Dateityp erfragen können. Hier der Quelltext des Programms *FiletypeDemo1*.

```
package javafuerwindows.jdic;
import org.jdesktop.jdic.filetypes.Association;
import org.jdesktop.jdic.filetypes.AssociationService;
public class FiletypesDemo1 {
    public static void info(Association assoc) {
        System.out.println("Name         : " +
                           assoc.getName());
        System.out.println("Beschreibung : " +
                           assoc.getDescription());
        System.out.println("Mime-Typ     : " +
                           assoc.getMimeType());
        System.out.println("Icondatei    : " +
                           assoc.getIconFileName());
        System.out.println("Dateiendungen: " +
                           assoc.getFileExtList());
        System.out.println("-----------------------------");
    }

    public static void main(String [] args) {
        AssociationService service =
                    new AssociationService();
        info(service.getFileExtensionAssociation(".txt"));
```

```
        info(service.getMimeTypeAssociation("text/html"));
    }
}
```

Listing 7.1 FiletypesDemo1.java

Um mit Dateitypen und Aktionen arbeiten zu können, müssen Sie zunächst die Klasse `AssociationService` instanziieren. Durch deren Methoden `getFileExtensionAssociation()` und `getMimeTypeAssociation()` erhalten Sie Objekte des Typs `Association`. Ein solches Objekt entspricht einem bestimmten Dateityp, den Sie entweder durch seine Endung, beispielsweise *.txt*, oder seinen Mime-Typ, zum Beispiel *text/html*, spezifiziert haben. `Association`s enthalten diverse Auskunftsmethoden, beispielsweise `getDescription()` und `getIconFileName()`.

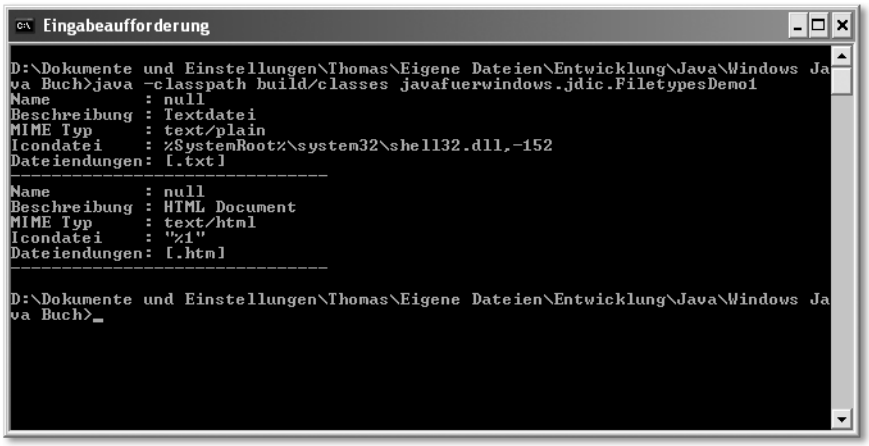

Abbildung 7.2 Bildschirmausgabe von FiletypesDemo1

Vielleicht ist Ihnen aufgefallen, dass in Abbildung 7.2 keine einzige Aktion aufgelistet ist. In der Tat habe ich Ihnen eine sehr interessante Auskunftsmethode bisher vorenthalten. `getActionList()` liefert eine Liste von Aktionen. Diese werden in `Action`-Objekten abgebildet. Im Folgenden erfahren Sie, wie Sie mit `getActionList()` arbeiten können.

```
package javafuerwindows.jdic;
import java.util.Iterator;
import java.util.List;
import org.jdesktop.jdic.filetypes.Action;
import org.jdesktop.jdic.filetypes.Association;
import org.jdesktop.jdic.filetypes.AssociationService;
```

```
public class FiletypesDemo2 {
    public static void main(String [] args) {
        AssociationService service =
              new AssociationService();
        Association assoc =
              service.getMimeTypeAssociation("text/html");
        List liste = assoc.getActionList();
        Iterator iter = liste.iterator();
        Action action;
        while (iter.hasNext()) {
            action = (Action) iter.next();
            System.out.println("Beschreibung: " +
                               action.getDescription());
            System.out.println("Verb        : " +
                               action.getVerb());
            System.out.println("Befehl      : " +
                               action.getCommand());
            System.out.println("--------------");
        }
    }
}
```

Listing 7.2 FiletypesDemo2.java

`getActionList()` liefert ein Objekt des Typs `java.util.List`. Es fasst alle Aktionen eines Dateityps zusammen. Wie Sie bereits gesehen haben, sind Aktionen Instanzen der Klasse `Action`. Ein Dateityp wird durch die Klasse `Association` repräsentiert. Wenn Sie der Reihe nach alle Aktionen durchgehen möchten, bietet sich `java.util.Iterator` an. Alternativ können Sie mit `getActionByVerb()` auch direkt auf eine Aktion zugreifen, allerdings müssen Sie dann deren *Verb* kennen, das als Parameter an die Methode übergeben wird. Was es mit solchen Verben auf sich hat, sehen Sie gleich.

Wie Sie Abbildung 7.3 entnehmen können, gehören zu einer Aktion drei Schlüsselelemente, eine *Beschreibung*, ein *Verb* sowie ein *Befehl*. Die Beschreibung beinhaltet eine Erläuterung dessen, was die Aktion bewirkt, beispielsweise *Im selben Fenster öffnen*. Das Verb ist ebenfalls eine Art von Beschreibung, allerdings sollten Sie hier nach Möglichkeit die Grundform eines englischen Verbs nehmen, beispielsweise *open*, *edit* oder *print*. Diesen drei Verben kommt übrigens eine besondere Bedeutung zu, die ich Ihnen in Abschnitt 7.2 demonstrieren werde. Der Befehl schließlich enthält ein vollständiges Kom-

mando, das normalerweise den absoluten Pfad zur auszuführenden Anwendung nebst allen zu übergebenden Parametern enthält.

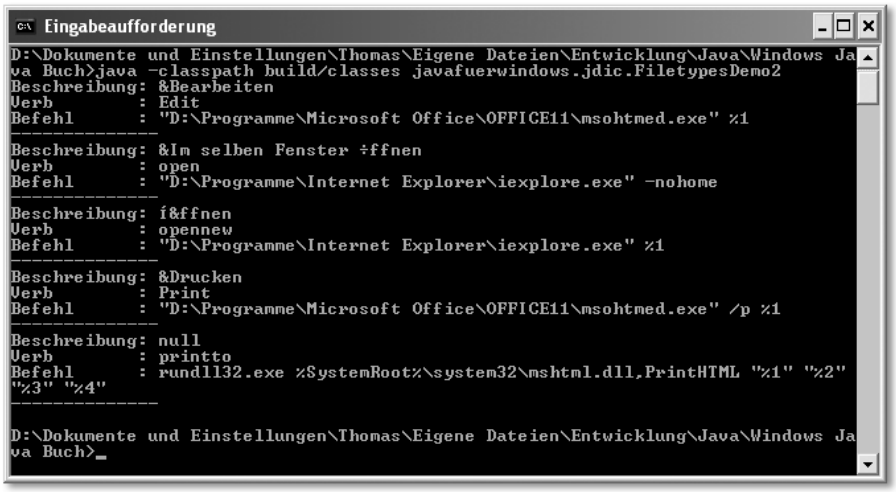

Abbildung 7.3 Bildschirmausgabe des Beispiels FiletypesDemo2

Wie stellt Windows nun die Informationen aus Abbildung 7.3 dar? *FiletypesDemo2* ermittelt den zum Mime-Typ *text/html* gehörenden Dateityp, ein HTML-Dokument. Im Windows Explorer erreichen Sie unter **Extras · Ordneroptionen** den gleichnamigen Dialog. Dessen Registerkarte **Dateitypen** zeigt eine Liste der registrierten Dateitypen, in der zweimal **HTML Document** auftauchen sollte. Bitte markieren Sie einen dieser Einträge und klicken auf **Erweitert**.

Abbildung 7.4 Der Dialog »Dateityp bearbeiten«

Der Dialog **Dateityp bearbeiten** zeigt die Aktionen, die Sie schon aus der Bildschirmausgabe des Programms *FiletypesDemo2* kennen. Wenn Sie **Bearbeiten** anklicken, öffnet sich ein weiterer Dialog **Vorgang bearbeiten für Typ**, dem Sie die zu startende Anwendung nebst Parameter entnehmen können.

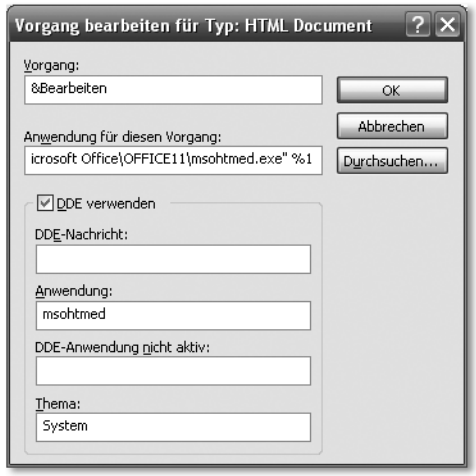

Abbildung 7.5 Der Dialog »Vorgang bearbeiten für HTML Document«

In diesem Abschnitt haben Sie gesehen, welche Werte `Association-` und `Action-`Objekte aufnehmen können und auf welche Weise sie miteinander in Beziehung stehen. Im Folgenden zeige ich Ihnen, wie Ihre Anwendung eigene Dateitypen und Aktionen registrieren und bei Bedarf wieder entfernen kann.

7.1.2 Registrieren von Dateitypen und Aktionen

Am Beginn dieses Kapitels habe ich erwähnt, dass Windows-Anwender gewohnt sind, Datendateien durch Doppelklick öffnen zu können. Wenn Ihr Programm ein eigenes Dateiformat liest und schreibt, sollten Sie Ihren Anwendern auf jeden Fall diesen Komfort bieten. Und selbst wenn Sie »nur« Fremdformate lesen, kann es sich lohnen, dem Benutzer die Möglichkeit zu geben, Ihr Programm über das Kontextmenü einer Datei zu starten. Wie dies funktioniert, sehen Sie jetzt.

Für die Registrierung von Dateitypen stellt `AssociationService` die Methoden `registerSystemAssociation()` und `registerUserAssociation()` bereit. Um eine bereits erfolgte Registrierung aufzuheben, gibt es `unregisterSystemAssociation()` und `unregisterUserAssociation()`. Sie haben also prinzipiell die Wahl, Dateitypen auf Benutzer- oder Systemebene zu registrieren. Allerdings bieten Versionen vor Windows 2000 diese Unterscheidung

nicht an. Im Hinblick auf eine einheitliche Funktionsweise Ihrer Anwendung sollten Sie daher erwägen, Dateitypen grundsätzlich auf Systemebene zu registrieren. Dies bedeutet, dass die angemeldeten Aktionen allen Benutzern zur Verfügung stehen. Wie Sie die Methoden anwenden, zeige ich Ihnen anhand des Programms *FiletypesDemo3*. Es erzeugt einen neuen Dateityp, der für alle auf *.abc* endende Dateien gilt. Er beinhaltet drei Aktionen, **.abc öffnen**, **.abc bearbeiten** und **.abc drucken**. Nach dem Programmstart können Sie eine beliebige Datei anlegen, die auf *.abc* enden sollte. Wenn Sie mit der Maus über das Datei-Icon fahren und die rechte Maustaste drücken, sehen Sie im Kontextmenü die drei neu angelegten Aktionen.

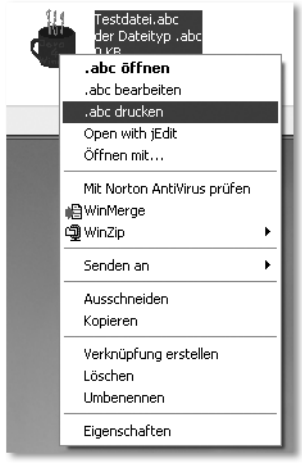

Abbildung 7.6 Ein durch FiletypesDemo3 angelegter Dateityp

FiletypesDemo3 öffnen nach dem Erzeugen des neuen Dateityps einen kleinen Dialog. Sobald Sie diesen bestätigen, wird die systemweite Registrierung des neuen Typs wieder rückgängig gemacht. Eine von Ihnen angelegte Datei wird zwar weiter das Kaffeetassen-Icon zeigen, die zusätzlichen Einträge im Kontextmenü sind aber wieder verschwunden. Damit das Programm ein eigenes Icon registrieren kann, muss dessen absoluter Pfad unter Umständen an Ihre Systemumgebung angepasst werden. Bevor Sie das Programm übersetzen und ausführen, sollten Sie die Pfadangabe im Quelltext entsprechend korrigieren.

```
package javafuerwindows.jdic;
import javax.swing.JOptionPane;
import org.jdesktop.jdic.filetypes.Action;
import org.jdesktop.jdic.filetypes.Association;
import org.jdesktop.jdic.filetypes.AssociationAlreadyRegistered
Exception;
```

```
import org.jdesktop.jdic.filetypes.AssociationNotRegisteredExcep
tion;
import org.jdesktop.jdic.filetypes.AssociationService;
import org.jdesktop.jdic.filetypes.RegisterFailedException;

public class FiletypesDemo3 {
    public static void main(String [] args) {
        AssociationService service =
                          new AssociationService();
        /*
         * den Dateityp definieren
         */
        Association assoc = new Association();
        assoc.setDescription("der Dateityp .abc");
        /*
         * Achtung: Bitte Pfad anpassen!!!
         */
        assoc.setIconFileName("D:\\Dokumente und
                        Einstellungen\\Thomas\\Eigene
                        Dateien\\Entwicklung\\Java\\Windows
                        Java Buch\\src\\Buchtasse.ico");
        assoc.setMimeType("text/plain");
        assoc.setName(".abc");
        assoc.addFileExtension(".abc");
        /*
         * drei Aktionen erzeugen und dem
         * Dateityp hinzufügen
         */
        assoc.addAction(new Action("open",
               "%SystemRoot%\\system32\\NOTEPAD.EXE \"%1\"",
               ".abc öffnen"));
        assoc.addAction(new Action("edit",
               "%SystemRoot%\\system32\\NOTEPAD.EXE \"%1\"",
               ".abc bearbeiten"));
        assoc.addAction(new Action("print",
            "%SystemRoot%\\system32\\NOTEPAD.EXE /p \"%1\"",
            ".abc drucken"));
        try {
            /*
             * Dateityp systemweit registrieren
```

```
             */
            service.registerSystemAssociation(assoc);
            /*
             * eine kleine Pause einlegen
             */
            JOptionPane.showMessageDialog(null,
                        "Registrierung wird aufgehoben");
        } catch (AssociationAlreadyRegisteredException e) {
            System.err.println(e);
        } catch (RegisterFailedException e) {
            System.err.println(e);
        } finally {
            try {
                /*
                 * Registrierung aufheben
                 */
                service.unregisterSystemAssociation(assoc);
            } catch (AssociationNotRegisteredException e) {
                System.err.println(e);
            } catch (RegisterFailedException e) {
                System.err.println(e);
            }
        }
    }
}
```

Listing 7.3 FiletypesDemo3.java

Das Registrieren eines neuen Datentyps besteht aus nur wenigen Schritten. Zum einen müssen Sie die eigentlichen Aktionen definieren. Diese bestehen – wie Sie bereits wissen – aus einem Verb, einem Kommando und einer Beschreibung. Zum anderen wird der eigentliche Datentyp erzeugt. Er beinhaltet einen Mime-Typ, eine Liste von Dateiendungen, die Aktionen, einen Namen sowie eine Bilddatei. Bitte denken Sie daran, für diese stets einen absoluten Pfad zu übergeben.

Mit den Klassen des Paketes `org.jdesktop.jdic.filetypes` ist es sehr einfach, eigene Dateitypen zu registrieren sowie eine bereits erfolgte Registrierung wieder aufzuheben. In den folgenden Abschnitten erfahren Sie, wie Sie aus Ihren Programmen die Standardanwendungen für beliebige Dateitypen aufrufen können. In diesem Zusammenhang gibt es auch ein Wiedersehen mit den Aktionen **Öffnen**, **Drucken** und **Bearbeiten**.

7.2 Die Klassen Desktop und Message

Die Klassen des Paketes `org.jdesktop.jdic.desktop` ermöglichen es Ihnen, aus Ihren Anwendungen heraus beliebige Dokumente zu öffnen, zu bearbeiten und zu drucken. Für jede dieser Aktionen wird das für den entsprechenden Dateityp festgelegte Programm aufgerufen. Außerdem ist es möglich, mit dem Standard-Webbrowser eine beliebige HTML-Seite anzuzeigen. Dies bietet sich an, wenn Sie eine kleine Online-Hilfe, eine Versionshistorie oder ein README anzeigen möchten. Schließlich können Sie mit dem Standard-E-Mail-Client Nachrichten versenden, wobei Ihre Anwendung Teile der Nachricht vorgeben kann.

7.2.1 Zugriff auf Standardanwendungen

Wie Sie in Abschnitt 7.1.2 gesehen haben, können Sie für einen Dateityp beliebige Aktionen definieren. **Öffnen**, **Bearbeiten** und **Drucken** sind hierbei für die meisten Datentypen verfügbar. Mit der Klasse `org.jdesktop.jdic.desktop.Desktop` können Sie diese drei Aktionen für eine beliebige Datei auslösen. Wie dies funktioniert, demonstriere ich Ihnen mittels *DesktopDemo1*.

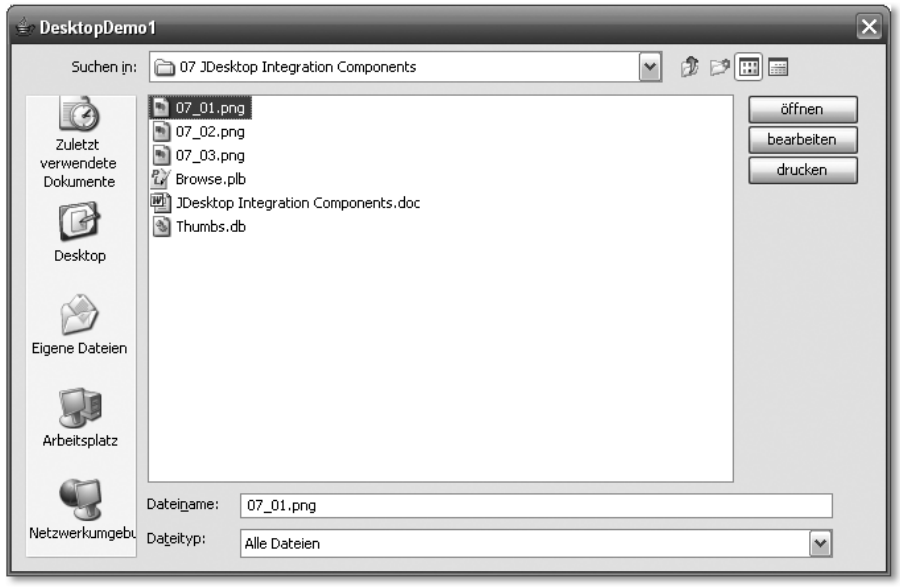

Abbildung 7.7 Das Programm DesktopDemo1

Die Anwendung zeigt eine Dateiauswahl auf den Bildschirm an. Sie enthält im rechten Dialogbereich die drei Knöpfe **Öffnen**, **Bearbeiten** und **Drucken**. Diese sind nur anwählbar, wenn bestimmte Bedingungen erfüllt sind. **Öffnen**

kann angeklickt werden, sobald eine Datei ausgewählt wird. **Bearbeiten** ist verfügbar, wenn für den Dateityp, zu dem die aktuell selektierte Datei gehört, eine Aktion mit dem Verb *edit* definiert wurde. Ähnliches gilt für **Drucken**. In diesem Fall muss eine Aktion mit dem Verb *print* vorhanden sein. Sie können dies testen, indem Sie vor dem Aufruf von *DesktopDemo1* das in Abschnitt 7.1.2 vorgestellte Programm *FiletypesDemo3* starten und anschließend in ein Verzeichnis wechseln, das eine *.abc*-Datei enthält.

Es folgen Teile des Quelltexts zu *DesktopDemo1*. Das komplette Listing finden Sie auf der Begleit-CD.


```
/*
 * macht die Knöpfe anwählbar oder sperrt sie
 */
private void knoepfeAktualisieren(File datei) {
    boolean bOeffnen = false;
    boolean bBearbeiten = false;
    boolean bDrucken = false;
    if (datei != null) {
        bOeffnen = true;
        bDrucken = Desktop.isPrintable(datei);
        bBearbeiten = Desktop.isEditable(datei);
        aktuelleDatei = datei;
    }
    oeffnen.setEnabled(bOeffnen);
    drucken.setEnabled(bDrucken);
    bearbeiten.setEnabled(bBearbeiten);
}
```

Die beiden Methoden `isPrintable()` und `isEditable()` können verwendet werden, um die Verfügbarkeit der Aktionen *print* und *edit* zu prüfen. In beiden Fällen verweist eine Instanz der Klasse `java.io.File` auf die zu testende Datei.

```
public void actionPerformed(ActionEvent actionEvent) {
    String cmd = actionEvent.getActionCommand();
    try {
        if (cmd.equals(STR_OEFFNEN)) {
            Desktop.open(aktuelleDatei);
        } else if (cmd.equals(STR_DRUCKEN)) {
            Desktop.print(aktuelleDatei);
        } else if (cmd.equals(STR_BEARBEITEN)) {
```

```
                Desktop.edit(aktuelleDatei);
            }
        } catch (DesktopException e) {
            System.err.println(e);
        }
    }
```

Um eine Aktion auszulösen, werden statische Methoden der Klasse `org.jdesktop.jdic.desktop.Desktop` verwendet, `open()`, `edit()` und `print()`. Alle drei erhalten als einzigen Parameter ein `File`-Objekt, das auf die zu öffnende, zu bearbeitende oder zu druckende Datei verweist.

`Desktop` bietet den Zugriff auf zwei weitere Standardanwendungen, den Browser sowie den E-Mail Client. Mit `browse()` können Sie eine beliebige HTML-Seite anzeigen lassen. Sie wird durch eine Instanz der Klasse `java.net.URL` festgelegt. Allerdings stellen die JDIC hierfür eine weitaus mächtigere Variante zur Verfügung, die ich Ihnen in Abschnitt 7.3 vorstelle. Mit `mail()` können Sie zudem ohne großen Aufwand den Versand von E-Mails vorbereiten. Wie dies funktioniert, zeigt der nächste Abschnitt.

7.2.2 E-Mails versenden

Die statische Methode `mail()` der Klasse `org.jdesktop.jdic.desktop.Desktop` bereitet das Versenden von Nachrichten durch den Standard-E-Mail-Client vor. Deren einfachste, parameterlose Variante öffnet ein Nachrichtenfenster ohne jeglichen Inhalt. Um die Empfänger, den Betreff sowie Nachrichtentext nebst Anhängen übergeben zu können, verwenden Sie eine Instanz der Klasse `org.jdesktop.jdic.desktop.Message`, die als Argument an `mail()` übergeben wird.

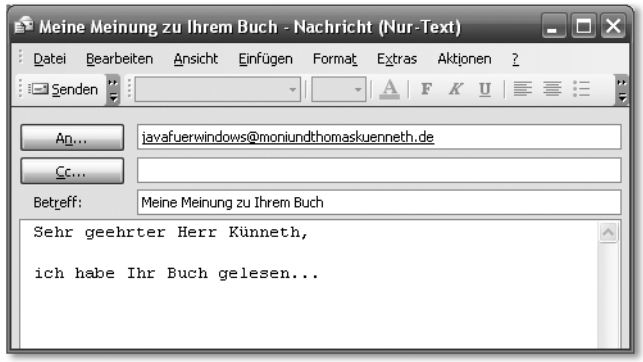

Abbildung 7.8 Ein von DesktopDemo2 geöffnetes Outlook-Fenster

Sehen Sie sich bitte die Methode `main()` des Programms *DesktopDemo2* an, die Ihnen das entsprechende Vorgehen demonstriert. Abbildung 7.8 zeigt das erzeugte *Outlook*-Nachrichtenfenster.

```
public static void main(String [] args) {
    /*
     * "Liste" der Empfänger zusammenstellen
     */
    Vector empfaenger = new Vector();
    empfaenger.add("
       javafuerwindows@moniundthomaskuenneth.de");
    /*
     * Nachrichtenobjekt anlegen und füllen
     */
    Message nachricht = new Message();
    nachricht.setToAddrs(empfaenger);
    nachricht.setSubject("Meine Meinung zu Ihrem Buch");
    nachricht.setBody("Sehr geehrter Herr Künneth,
         \n\nich habe Ihr Buch gelesen...");
    try {
        Desktop.mail(nachricht);
    } catch (DesktopException e) {
        System.err.println(e);
    }
}
```

Alle Methoden, die E-Mail-Adressen enthalten, erwarten als Argumente Klassen, die das Interface `java.util.List` implementieren. Dies trifft auf `java.util.Vector` zu. Da `Vector` zudem sehr leicht befüllt werden kann, bietet es sich hier als »Adressspeicher« an. Nach dem Vorbereiten des `Message`-Objektes wird es an `Desktop.mail()` übergeben.

Im Folgenden wenden wir uns einem weiteren, äußerst wichtigen Programm zu: dem Webbrowser. Ich zeige Ihnen, wie Sie den *Internet Explorer* sowie den ebenfalls äußerst weit verbreiteten *Mozilla* in Ihre eigenen Anwendungen einbetten.

7.3 Den Browser in eigene Programme einbetten

Die in Abschnitt 7.2.2 vorgestellte Methode `mail()` gestattet das einfache und bequeme Versenden von Nachrichten. Allerdings sieht der Anwender auf den ersten Blick, dass er den Kontext einer Anwendung verlässt, also mit einem

anderen Programm arbeitet. Ähnlich verhält es sich mit der Methode `browse()`, die die anzuzeigende Seite in einem eigenen Browserfenster öffnet. Auch hier ist der Programmwechsel deutlich sichtbar. In den folgenden Abschnitten wird das Paket `org.jdesktop.jdic.browser` erläutert, mit dem Sie die Rendering Engine des *Internet Explorers* oder *Mozillas* in Ihren Java Programmen nutzen können. Der Anwender arbeitet dabei weiter mit Ihrem Programm.

7.3.1 Installation

Um die Rendering Engine von Internet Explorer oder Mozilla in Ihren Programmen verwenden zu können, müssen Sie dafür sorgen, dass *JDIC* bestimmte ausführbare Dateien findet. Dies sind *IeEmbed.exe* und *nspr4.dll* bei Verwendung des Internet Explorers sowie *MozEmbed.exe*, wenn Sie Mozilla als Webbrowser bevorzugen. Hierzu bieten sich zwei Vorgehensweisen an. Die eine ist, die Umgebungsvariable `PATH` um das Verzeichnis zu erweitern, das die genannten Dateien enthält. Allerdings führt dieses Vorgehen sehr schnell zu langen, unhandlichen und unübersichtlichen Pfadketten. Deshalb schlage ich Ihnen folgende Variante vor.

> Erzeugen Sie ein Sammelverzeichnis für beliebige ausführbare Programme, die über `PATH` erreichbar sein müssen, beispielsweise *C:\Programme\bin*, und erweitern Sie `PATH` um dieses Verzeichnis. Im Fall von *MozEmbed.exe*, *IeEmbed.exe* und *nspr4.dll* ist es dann auch ausreichend, diese Dateien dorthin zu kopieren.

7.3.2 Die Klasse WebBrowser

Die wichtigste Klasse im Hinblick auf das Einbetten des Webbrowsers ist `org.jdesktop.jdic.browser.WebBrowser`. Sie wird von `java.awt.Canvas` abgeleitet, ist also eine schwergewichtige AWT-Komponente. Leider ist in allen bisher erschienenen Java-Versionen, die *Swing* enthalten, das Mischen von schwer- und leichtgewichtigen Klassen problematisch. Der englischsprachige Artikel *Mixing heavy and light components*[1] zeigt sehr detailliert auf, worauf Sie achten müssen, wenn Sie schwergewichtige Komponenten mit Swing mischen möchten.

Im Folgenden zeige ich Ihnen die Beispielanwendung *WebBrowserDemo1*, die einen äußerst rudimentären Webbrowser implementiert.

1 http://java.sun.com/products/jfc/tsc/articles/mixing/

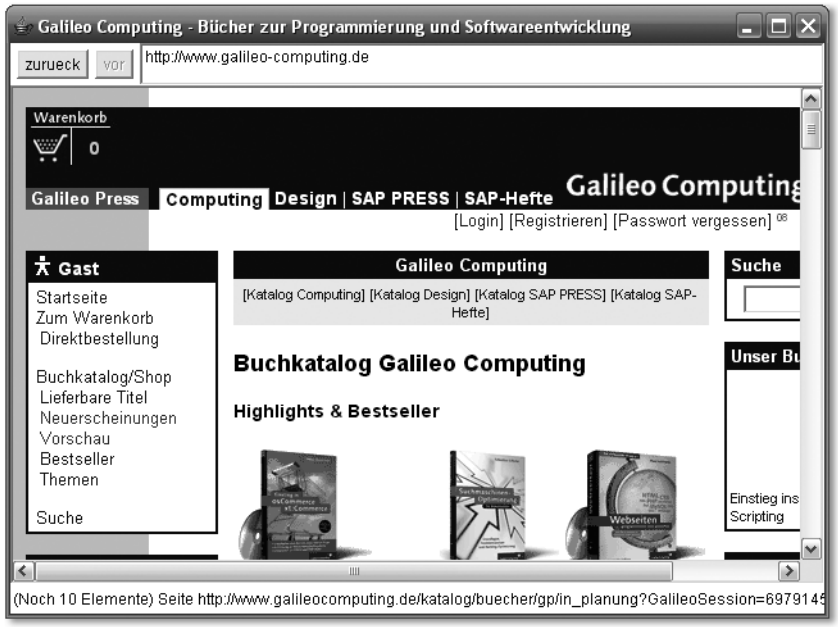

Abbildung 7.9 Die Homepage von Galileo Computing im Minimalbrowser WebBrowserDemo1

Das Programm besteht aus drei Bereichen, einer Leiste mit zwei Knöpfen und einer Eingabezeile im oberen Bereich, der eigentlichen Browserkomponente sowie einer Statuszeile. In der Eingabezeile können Sie URLs eintippen. Sobald Sie die **Enter**-Taste drücken, wird das angegebene Dokument geladen. Beachten Sie bitte, dass Sie das Protokoll angeben müssen, Ihre Eingabe muss also mit `http://` beginnen.

Um zu zeigen, wie `WebBrowser` eingesetzt wird, stelle ich Ausschnitte des Quelltextes zu *WebBrowserDemo1* vor. Das vollständige Listing findet sich auf der Begleit-CD.

```
/*
 * den Browser instantiieren
 */
browser = new WebBrowser();
browser.addWebBrowserListener(this);
Panel browserpanel = new Panel(new BorderLayout());
browserpanel.add(browser, BorderLayout.CENTER);
```

Um mit der Browserkomponente arbeiten zu können, müssen Sie zunächst ein entsprechendes Objekt instanziieren. Um Darstellungsproblemen vorzubeu-

gen, hat es sich als vorteilhaft erwiesen, `WebBrowser` zusätzlich in ein `java.awt.Panel` mit `java.awt.BorderLayout` einzubetten.

```
/*
 * diese Methode liest aus dem Textfeld die
 * anzuzeigende URL
 */
private void seiteAnzeigen() {
    try {
        URL seite = new URL(urleingabe.getText());
        browser.setURL(seite);
        statuszeile.setText(seite.toString());
    } catch (MalformedURLException e) {
    }
}
```

Sie können jederzeit mit der Methode `setURL()` das Anzeigen einer neuen Seite initiieren. Jedes aufgerufene Dokument und jeder angeklickte Link wird in einer Verlaufshistorie festgehalten.

```
/*
 * die Knöpfe in der Kopfzeile aktualisieren
 */
private void knoepfeAktualisieren() {
    vor.setEnabled(browser.isForwardEnabled());
    zurueck.setEnabled(browser.isBackEnabled());
}
```

In dieser Liste können Sie mit `forward()` und `back()` navigieren, wenn die korrespondierenden Methoden `isForwardEnabled()` und `isBackEnabled()` `true` liefern.

```
/*
 * springt zur vorherigen Seite in der History
 */
private void vorherigeSeite() {
    browser.back();
}
```

Schließlich noch die Methoden des `WebBrowserListener`-Interfaces: Sie sollten dieses Interface in jedem Fall implementieren, weil Sie auf diese Weise wichtige Statusmeldungen der Browserkomponente erhalten.

```
/*
 * WebBrowserListener interface
 */
public void documentCompleted(WebBrowserEvent
                              webBrowserEvent) {
    knoepfeAktualisieren();
}

public void downloadCompleted(WebBrowserEvent
                              webBrowserEvent) {
    statuszeile.setText(webBrowserEvent.getData());
}

public void downloadError(WebBrowserEvent webBrowserEvent) {
    statuszeile.setText(webBrowserEvent.getData());
}

public void downloadProgress(WebBrowserEvent
                             webBrowserEvent) {
    statuszeile.setText(webBrowserEvent.getData());
}

public void downloadStarted(WebBrowserEvent webBrowserEvent) {
    statuszeile.setText(webBrowserEvent.getData());
}

public void statusTextChange(WebBrowserEvent
                             webBrowserEvent) {
    statuszeile.setText(webBrowserEvent.getData());
}

public void titleChange(WebBrowserEvent webBrowserEvent) {
    setTitle(webBrowserEvent.getData());
}
```

Beispielsweise wird die Methode `documentCompleted()` aufgerufen, sobald ein Dokument vollständig geladen wurde. *WebBrowserDemo1* verwendet diesen Mechanismus, um die Navigationsknöpfe anwählbar zu machen oder sie zu sperren. `titleChange()` hat eine vergleichbare Aufgabe. Dieser Methode wird ein etwaiger neuer Seitentitel übergeben.

Das Einbetten einer ausgefeilten Rendering Engine für HTML-Seiten kann aus den unterschiedlichsten Gründen sinnvoll sein. Zum einen ist es sicher für den Anwender sehr angenehm, den Kontext einer Anwendung nicht verlassen zu müssen. Zudem bieten Internet Explorer und Mozilla weitaus bessere Resultate als die Java-eigene Komponente. Schließlich verlassen sich Websites häufig auf Plugins, die oft nur für Internet Explorer und Mozilla zur Verfügung stehen.

7.4 Symbole im Infobereich der Taskleiste

Der Infobereich der Taskleiste in der rechten unteren Bildschirmecke bietet Anwendungen die Möglichkeit, Icons abzulegen, beispielsweise um den Anwender über den Status einer Anwendung zu informieren oder um leichten Zugriff auf häufig verwendete Aktionen zu ermöglichen.

Abbildung 7.10 Der Infobereich der Taskleiste

Mit Hilfe der Icons lassen sich in der Regel unterschiedliche Aktionen auslösen. Häufig öffnet ein Klick mit der rechten Maustaste ein Kontextmenü, das eine Funktionsauswahl anbietet. Manchmal kann durch einfachen Klick mit der linken Maustaste eine Aktion ausgelöst werden, oft ist aber auch ein Doppelklick erforderlich.

Die Klassen des Paketes `org.jdesktop.jdic.tray` stellen diesen Mechanismen auch unter Java bereit. Das angezeigte Icon reagiert hierbei auf einen Klick mit der rechten Maustaste, indem es ein beliebiges Menü öffnet. Zudem kann ein Klick mit der linken Maustaste ein `ActionEvent` feuern. Die folgenden Abschnitte erläutern die Funktionsweise der zu dem Paket gehörenden Klassen.

7.4.1 Die Klassen SystemTray und TrayIcon

Um ein Icon im Infobereich anzuzeigen, sind nur die beiden Klassen `SystemTray` und `TrayIcon` erforderlich. Wie diese verwendet werden, möchte ich Ihnen an Ausschnitten des Quelltexts zu *Clip4Moni* zeigen, einer kleinen, aber recht praktischen Anwendung zur Verwaltung von Textschnipseln über das Systemklemmbrett. Die Idee ist, einem Textschnipsel einen kurzen, aussagekräftigen Namen zu geben, der in einem Menü angezeigt wird. Wie dies aussieht, zeigt Abbildung 7.11. Wenn der Anwender den entsprechenden Menüpunkt anklickt, wird der zugehörige Text auf das Klemmbrett gelegt. Von dort können Sie den Text in nahezu jede beliebige Anwendung übernehmen. Alle Klassen sowie das ausführbare Programm finden Sie auf der Begleit-CD.

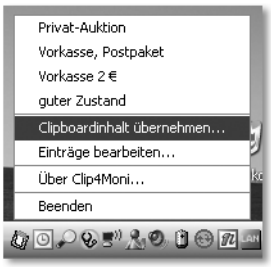

Abbildung 7.11 Das Menü von Clip4Moni

Die Klasse `org.jdesktop.jdic.tray.SystemTray` regelt den Zugriff auf den Infobereich. Mit den Methoden `addTrayIcon()` und `removeTrayIcon()` können Sie ein Icon hinzufügen und wieder entfernen. Es wird hierbei durch eine Instanz von `org.jdesktop.jdic.tray.TrayIcon` repräsentiert.

```
public static final SystemTray tray =
                    SystemTray.getDefaultSystemTray();
```

Im laufenden Betrieb gibt es stets nur eine Instanz der Klasse `SystemTray`, die mit der statischen Methode `getDefaultSystemTray()` ermittelt wird. Anschließend können Sie eine Instanz von `TrayIcon` erzeugen und mittels `addTrayIcon` im Infobereich anzeigen.

```
ImageIcon icon = getImageIcon(ICONFILENAME);
tray_icon = new TrayIcon(icon, PROGNAME, menu);
tray_icon.addActionListener(this);
tray.addTrayIcon(tray_icon);
```

Es bietet sich an, das anzuzeigende Icon in einer `ImageIcon`-Instanz zu kapseln. Diese lässt sich mit einer kurzen Hilfsmethode sehr leicht erzeugen.

```
private ImageIcon getImageIcon(String resource) {
    java.net.URL url =
                ClassLoader.getSystemResource(resource);
    return new ImageIcon(url);
}
```

Das Menü, das nach Anklicken des Icons mit der rechten Maustaste angezeigt wird, ist eine Instanz von `JPopupMenu`. Es kann nach Erzeugen des `TrayIcon`-Objektes weiterhin beliebig verändert werden, Sie können also jederzeit Elemente hinzufügen oder löschen. Außer auf Klicks mit der rechten Maustaste reagiert ein `TrayIcon`-Objekt auch auf Klicks mit der linken. In diesem Fall wird ein `ActionEvent` gefeuert, das sich derzeit am Kommando `PressAction` erkennen lässt. Im folgenden Abschnitt sehen Sie, wie Sie mit der Klasse

`TrayIcon` kleine Sprechblasen erzeugen können, die Sie von Windows-Statusmeldungen her kennen. In dem dazugehörigen Programmbeispiel wird auch erkennbar, wie Sie auf den Klick mit der linken Maustaste reagieren.

7.4.2 Meldungen im Infobereich

Sie können die Klasse `TrayIcon` auch dazu verwenden, Nachrichten im Infobereich anzuzeigen. Diese werden in Form kleiner Sprechblasen dargestellt, deren Spitze auf das auslösende Icon zeigt. Windows verwendet sie beispielsweise, um über den Status der Hardwareerkennung zu informieren oder den Anwender dazu aufzufordern, sein Notebook an das Stromnetz anzuschließen.

Abbildung 7.12 Eine von TrayIconDemo erzeugte Nachricht

Um eine Nachricht anzuzeigen, wie sie in Abbildung 7.12 zu sehen ist, müssen Sie nur die Methode `displayMessage()` einer `TrayIcon`-Instanz aufrufen. Das Icon muss zu diesem Zeitpunkt allerdings sichtbar sein. Das Programm *TrayIconDemo* zeigt eine kleine blaue Tasse im Infobereich an, die nach Klick mit der rechten Maustaste ein Menü einblendet. Klicken Sie hingegen mit der linken Maustaste auf das Icon, erscheint eine Nachrichtensprechblase.

```
package javafuerwindows.jdic;
import java.awt.event.ActionListener;
import java.net.URL;
import javax.swing.ImageIcon;
import javax.swing.JMenuItem;
import javax.swing.JPopupMenu;
import org.jdesktop.jdic.tray.SystemTray;
import org.jdesktop.jdic.tray.TrayIcon;

public class TrayIconDemo implements ActionListener {
    private static final String STR_BEENDEN = "Beenden";
    private TrayIcon trayicon;

    public TrayIconDemo() {
        SystemTray tray = SystemTray.getDefaultSystemTray();
        URL url =
       ClassLoader.getSystemResource("javafuerwindows.png");
```

```
        ImageIcon icon = new ImageIcon(url);
        JPopupMenu menu = new JPopupMenu();
        JMenuItem menueeintrag = new JMenuItem(STR_BEENDEN);
        menu.add(menueeintrag);
        menueeintrag.addActionListener(this);
        trayicon = new TrayIcon(icon, "TrayIconDemo", menu);
        trayicon.addActionListener(this);
        tray.addTrayIcon(trayicon);
    }

    public static void main(String [] args) {
        new TrayIconDemo();
    }

    /*
     * ActionListener interface
     */
    public void actionPerformed(java.awt.event.ActionEvent
                                actionEvent) {
        String kommando = actionEvent.getActionCommand();
        if (kommando.equals(STR_BEENDEN))
            System.exit(0);
        else if (kommando.equals("PressAction"))
            trayicon.displayMessage("eine Nachricht",
            "Dieses Beispiel gehört zum Buch Java für
             Windows", TrayIcon.INFO_MESSAGE_TYPE);
    }
}
```

Listing 7.4 TrayIconDemo.java

Wie Sie gesehen haben, sind die kleinen Nachrichtensprechblasen sehr schnell implementiert. Für den Anwender können sie allerdings lästig werden, wenn sie zu häufig oder aus nicht erkennbarem Grund erscheinen. Deshalb rate ich Ihnen, den Einsatz der Sprechblasen sehr genau abzuwägen.

8 Zugriff auf die Registry

8 Zugriff auf die Registry

Um alle Möglichkeiten von Windows auszuschöpfen, muss ein Java-Programm auf die Registry zugreifen. Dieses Kapitel zeigt interessante Einträge und wie Sie sie vornehmen und ändern.

Seit Windows 95 speichert die *Registrierung* oder *Registry* wichtige Informationen über die gegenwärtige Systemkonfiguration, über Anwendungen, Hardware und Benutzer. Die Inhalte dieser zentralen, hierarchischen Datenbank werden im laufenden Betrieb sowohl vom System selbst als auch von Anwendungen kontinuierlich abgefragt und verändert. Ob Sie also das gegenwärtig eingestellte Hintergrundbild austauschen, durch Doppelklick ein Dokument öffnen oder sich als anderer Benutzer anmelden, stets ist die Registrierung als systemweiter Speicher für Einstellungen beteiligt.

Der Zugriff auf die Registry ist für Java-Entwickler aus den unterschiedlichsten Gründen interessant. Zum einen speichern Anwendungen ihre Einstellungen (beispielsweise Lage und Größe von Fenstern, gewählte Farben und Zeichensätze) unter Windows gemeinhin in der Registrierung, nicht in Dateien. Zum anderen enthält die Registry interessante Daten, die Java selbst betreffen. Ich werde Ihnen beispielsweise zeigen, wie Sie die installierten Laufzeitumgebungen ermitteln können. Schließlich können Sie durch Setzen bestimmter Werte für eine nahtlose Integration Ihrer Anwendung in Windows sorgen. Dies betrifft unter anderem das Hinzufügen Ihres Programms zur Liste der installierten Software. Wie das geht, zeige ich Ihnen in Abschnitt 8.3.3.

Als Einstieg in dieses Kapitel möchte ich Ihnen zunächst den Aufbau und die Struktur der Registrierung erläutern. Eine grundlegende Kenntnis der Funktionsweise ist sehr wichtig, weil der falsche Umgang mit der Registrierungsdatenbank im ungünstigsten Fall zu irreparablen Schäden an Ihrer Windows-Installation führen kann. Wenn Sie mit dem Aufbau und der Struktur der Registry bereits vertraut sind, können Sie gleich mit Abschnitt 8.2 fortfahren. Dort erfahren Sie, wie Sie mit Java auf die Registry zugreifen.

8.1 Aufbau der Registrierung

Die Registry ist eine baumartig organisierte Datenstruktur. In diesem Abschnitt werden Sie die unterschiedlichen Bereiche der Registrierung kennen lernen und mit dem Dienstprogramm *Regedit* nach Inhalten suchen. Bitte beachten Sie allerdings, dass es sich bei der Registry um eine sehr komplexe Datenbank han-

delt. Wenn Sie planen, tiefer in deren Geheimnisse einzutauchen, rate ich dringend zu weiterführender Literatur.

Eine Tour mit dem Registrierungseditor

Microsoft stellt das Dienstprogramm *Regedit* zur Verfügung, mit dessen Hilfe Administratoren die Registrierung durchsuchen und gegebenenfalls auch verändern können. Wenn Sie als Benutzer mit Adminstrator-Rechten angemeldet sind, können Sie das Tool durch **Start · Ausführen** aufrufen. Als Name des zu öffnenden Programms geben Sie einfach `regedit` ein.

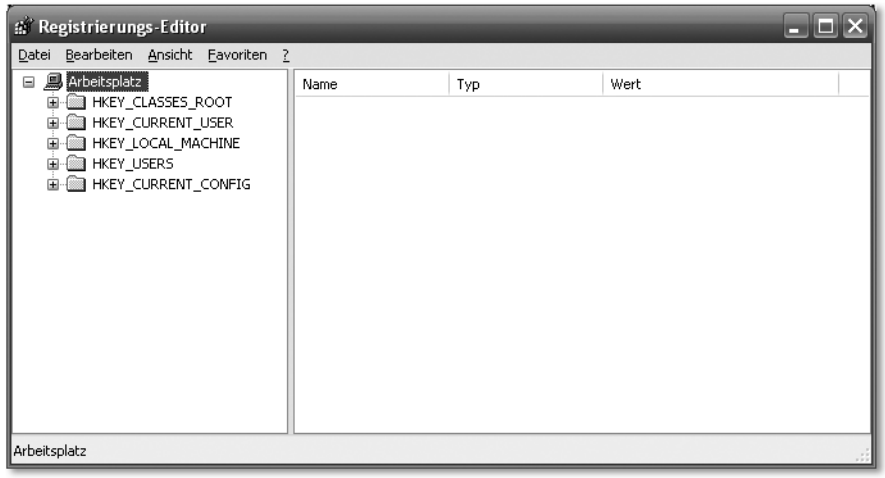

Abbildung 8.1 Das Regedit-Programmfenster

Die Registrierung ist in mehrere Hauptzweige unterteilt, die in *Regedit* unter der Wurzel **Arbeitsplatz** zusammengefasst werden. Die mit einem Ordner-Symbol versehenen Knoten werden *Schlüssel* genannt. Ein Schlüssel kann – wie wir noch sehen werden – eine ganze Reihe ineinander geschachtelter Kinder-Schlüssel besitzen. *HKEY_CURRENT_CONFIG* enthält Informationen zum gegenwärtig verwendeten Hardwareprofil des Computers. Dieser Teil der Registry ist aus der Sicht des Java-Entwicklers uninteressant. In *HKEY_USERS* werden Benutzerprofile gespeichert. Jedes Konto erhält einen eigenen Unter-Schlüssel. Das Profil des derzeit angemeldeten Anwenders ist über den Hauptzweig *HKEY_CURRENT_USER* erreichbar. Dieser verweist auf einen Unter-Schlüssel von *HKEY_USERS*. Der Schlüssel *HKEY_LOCAL_MACHINE* enthält Einstellungen, die für alle Benutzer des Rechners gelten. *HKEY_CLASSES_ROOT* ist wieder ein Verweis und zeigt auf den Schlüssel *HKEY_LOCAL_MACHINE\Software\Classes*. In diesem Zweig der Registrierung werden systemweit gültige Dateitypen und deren Verknüpfungen mit Programmen gespei-

chert. Es existiert noch ein vergleichbarer Zweig *HKEY_CURRENT_USER\Software\Classes*, in dem benutzerspezifische Zuordnungen und Dateitypen gespeichert werden und die Standardeinstellungen überschreiben. Die in Kapitel 7, *JDesktop Integration Components*, in Abschnitt 7.1.2 beschriebene Unterscheidung der Registrierung von Dateitypen auf Benutzer- oder Systemebene bezieht sich auf diese beiden Registry-Zweige.

Nun ein kurzer Blick auf die in der Registrierung abgelegten Daten. Rufen Sie hierzu bitte die Suchfunktion auf, die Sie durch **Bearbeiten · Suchen** erreichen.

Abbildung 8.2 Die Suchfunktion von Regedit

Bitte tragen Sie als Suchbegriff `Java Web Start` ein. Achten Sie darauf, dass vor **Schlüssel** ein Häkchen gesetzt ist. *Regedit* wird eine gewisse Zeit nach dem angegebenen Schlüssel suchen und ihn anschließend im linken Teil des Programmfensters anzeigen. Abbildung 8.3 zeigt den Inhalt des Schlüssels *Java Web Start* einschließlich seiner Unter-Schlüssel. Der rechte Teil des *Regedit*-Fensters enthält Daten, die sich auf den aktuell markierten Schlüssel beziehen.

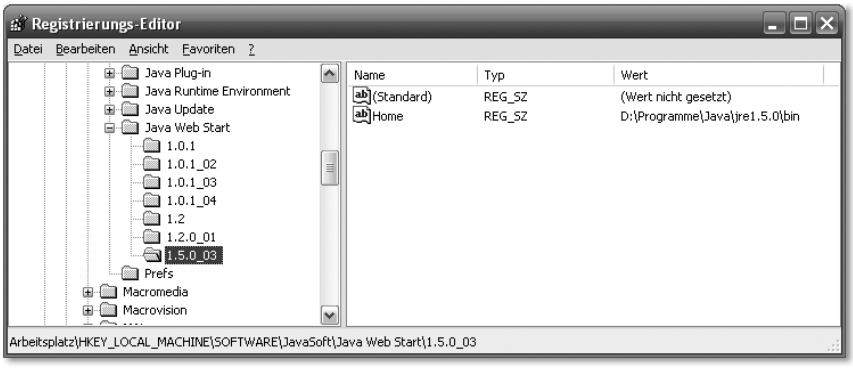

Abbildung 8.3 Ergebnis der Suche nach Java Web Start

Ein Schlüssel kann – wie in Abbildung 8.3 zu sehen ist – sowohl weitere Schlüssel als auch Werte enthalten. Ein Wert mit dem Namen *(Standard)* ist in jedem Schlüssel vorhanden. Um einen Schlüssel zu bearbeiten, verwenden Sie dessen Kontextmenü, das nach dem Anklicken mit der rechten Maustaste erscheint.

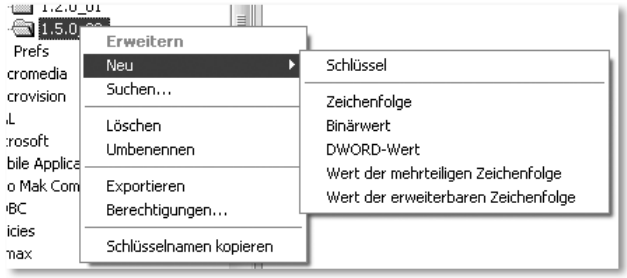

Abbildung 8.4 Kontextmenü zum Bearbeiten eines Schlüssels

Die Schlüssel geben der Registrierung ihre hierarchische Struktur. Die eigentlichen Daten, zum Beispiel Farbeinstellungen, Fensterpositionen, Dateitypen oder Pfade, werden in Name-Wert-Paaren abgelegt. In Abbildung 8.3 sehen Sie den Schlüssel *1.5.0_03*, der zwei Name-Wert-Paare enthält. Eines davon trägt den Namen *Home*. Dessen Wert ist *D:\Programme\Java\jre1.5.0\bin*. Die Werte eines Namens können nicht nur aus Zeichenketten bestehen. Microsoft hat zahlreiche Datentypen definiert. Einige davon sehen Sie in Abbildung 8.4. Diese stelle ich Ihnen im folgenden Abschnitt vor.

8.1.1 Registry-Datentypen

In der Registrierung werden unterschiedlichste Einstellungen abgelegt. Neben Pfadinformationen finden Datumsangaben, Formatierungsanweisungen, numerische Werte, maschinennah kodierte sowie gänzlich unstrukturierte Daten Platz. Microsoft trägt dieser Vielfalt Rechnung, indem die »Beschaffenheit« eines Name-Wert-Paares durch einen *Datentyp* festgelegt werden kann.

Ein äußerst häufig anzutreffender Datentyp ist die *Zeichenfolge*. *RegEdit* kennzeichnet Werte dieses Typs mit dem Kürzel *REG_SZ*. Zeichenfolgen können beispielsweise eingesetzt werden, um Pfade und Dateinamen zu speichern. Auch freie Texte sind mit diesem Datentyp möglich. Ein lustiges Beispiel hierfür hält der Schlüssel *HKEY_CURRENT_USER\Software\WinZip Computing* bereit, der zu dem Programm *WinZip* gehört. Er enthält nämlich folgenden Text: `Please look in the Nico Mak Computing section for WinZip keys, values, and settings.`

Binärwerte werden mit *REG_BINARY* gekennzeichnet. Hierbei handelt es sich häufig um Werte-Kolonnen aus Hardwarekomponenten, die der Registrierungseditor als Hex-Dumps anzeigt.

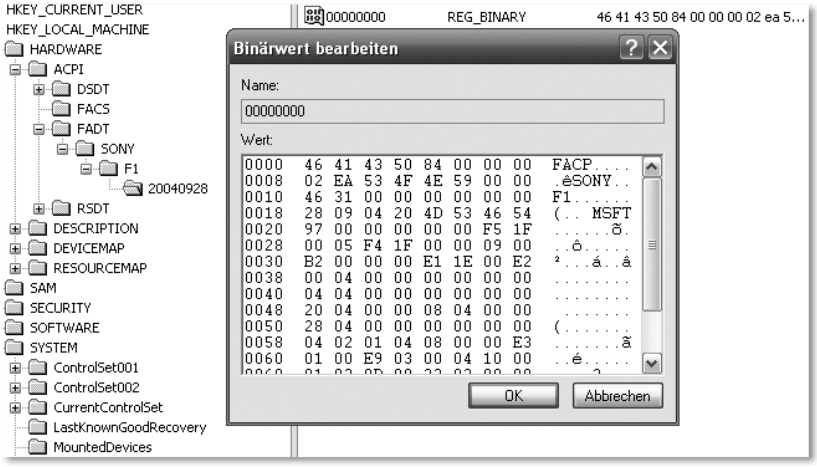

Abbildung 8.5 Darstellung von Binärwerten in Regedit

DWORD-Werte speichern Zahlen mit 32 Bit Länge. *Regedit* kennzeichnet sie mit dem Kürzel *REG_DWORD*. Dieser Datentyp findet oft in Verbindung mit Gerätetreibern Verwendung. Abbildung 8.6 zeigt, wie *Regedit DWORD-Werte* darstellen kann.

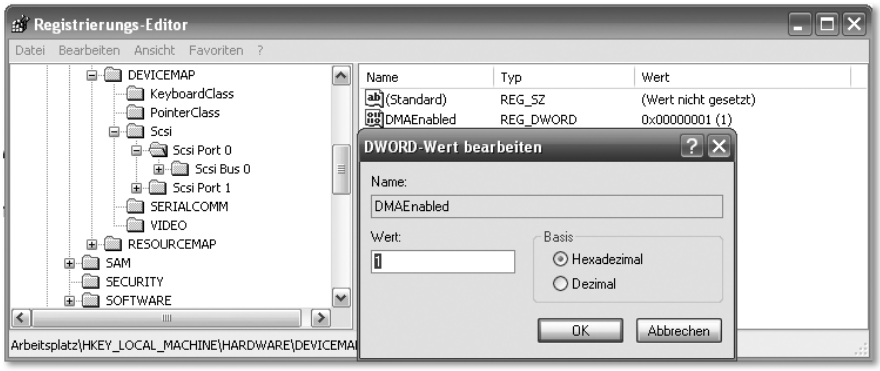

Abbildung 8.6 Darstellung von DWORD-Werten im Registrierungseditor

Der Datentyp *Wert der mehrteiligen Zeichenfolge* wird im Registrierungseditor mit *REG_MULTI_SZ* dargestellt. Er nimmt mehrere Zeichenfolgen oder Listen auf, die durch ein Zeichen – zum Beispiel Komma oder Leerzeichen – voneinander getrennt sind. Beispiele für die Verwendung finden Sie im Schlüssel *HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\Tcpip\Linkage*.

Der letzte hier vorgestellte Datentyp heißt *Wert der erweiterbaren Zeichenfolge*. Er wird mit *REG_EXPAND_SZ* gekennzeichnet und enthält Zeichenfolgen variabler Länge. Abbildung 8.7 zeigt den Wert *ImagePath* des Schlüssels *HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\Tcpip*.

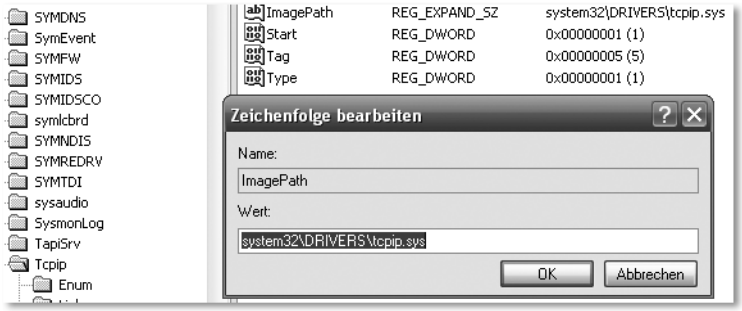

Abbildung 8.7 Ein Beispiel für erweiterbare Zeichenfolgen

Die folgenden Abschnitte behandeln die Klassenbibliothek *JNIRegistry* und zeigen, wie Sie mit ihr Registrierschlüssel anlegen, auslesen und verändern können.

8.2 Bearbeiten der Registrierung mit JNIRegistry

Die Public Domain-Klassenbibliothek *JNIRegistry* von Timothy Gerard Endres steht auf der Homepage des Autors zum Download bereit. Außerdem finden Sie die zum Zeitpunkt der Drucklegung aktuelle Version auf der Begleit-CD des Buches. Die Installation gestaltet sich äußerst einfach und verläuft analog zu den meisten anderen vorgestellten Klassenbibliotheken mit nativem Teil. Generell müssen Sie nur dafür sorgen, dass die beiden Dateien *registry.jar* und *ICE_JNIRegistry.dll* von der Java-Laufzeitumgebung gefunden werden. Nähere Informationen hierzu finden Sie in Kapitel 1, *Installation und Konfiguration*.

8.2.1 Das Paket com.ice.jni.registry

Alle Klassen, die Sie für den Zugriff auf die Windows-Registrierung benötigen, befinden sich im Paket `com.ice.jni.registry`. Dessen Klasse `Registry` bildet, wie Sie gleich sehen werden, den Einstiegspunkt in die Programmierung. Sie beinhaltet zudem einen kleinen interaktiven Modus. Dieser dient als Funktionstest für die native Schnittstelle und ist gleichzeitig ein Tutorial, das die programmiertechnische Anwendung der Klassenbibliothek zeigt. Wenn Sie die Datei *registry.jar* in das Verzeichnis für optionale Pakete kopiert haben, können Sie die in Abbildung 8.1 gezeigte Demonstration starten, indem Sie in der Eingabeaufforderung `java com.ice.jni.registry.Registry` eintippen. Nach

dem Start des Programms zeigt `help` eine kurze Funktionsübersicht. Möchten Sie alle Unter-Schlüssel zu einem bestimmten Schlüssel anzeigen, tippen Sie bitte `keys` gefolgt vom Schlüsselnamen ein, beispielsweise `keys HKEY_CURRENT_USER\\Software`. Das Schlüsselwort `values` zeigt alle Werte eines Schlüssels an. Durch Drücken von Strg + C beenden Sie den interaktiven Modus.

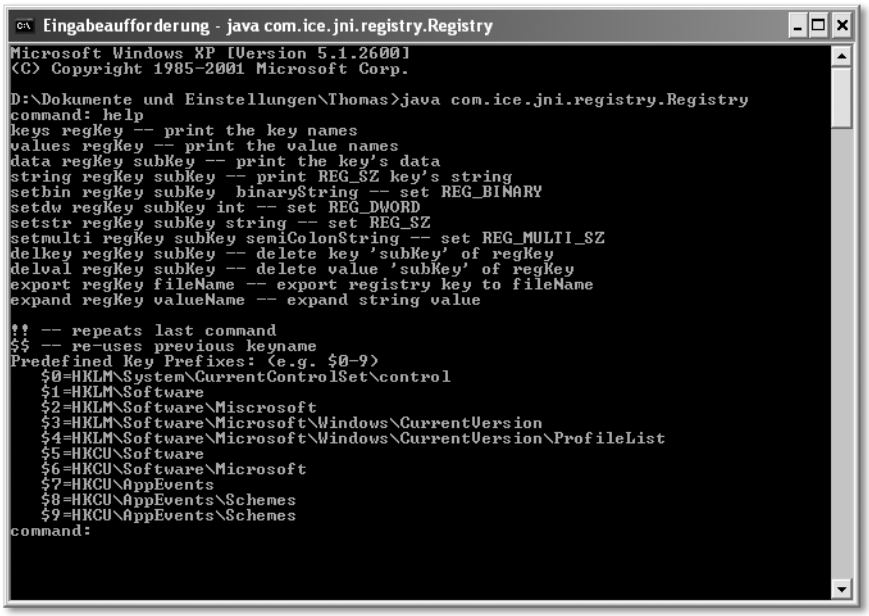

Abbildung 8.8 Der Demo-Modus von JNIRegistry

Im Folgenden möchte ich Ihnen zeigen, wie Sie mit Ihren Java-Programmen auf *JNIRegistry* zugreifen. Ich stelle Ihnen hierzu das kleine Programm *RegistryDemo1* vor, das den Namen des aktuellen Benutzers auf der Konsole ausgibt. Dieser Wert wird im Schlüssel *HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Explorer* unter dem Namen *Logon User Name* abgelegt.

```
package javafuerwindows.registry;
import com.ice.jni.registry.NoSuchKeyException;
import com.ice.jni.registry.Registry;
import com.ice.jni.registry.RegistryException;
import com.ice.jni.registry.RegistryKey;

public class RegistryDemo1 {
    public static void main(String [] args) {
```

```
        RegistryKey schluessel = Registry.getTopLevelKey(
                                        "HKEY_CURRENT_USER");
        try {
            RegistryKey explorer = schluessel.openSubKey(
                            "Software\\Microsoft\\Windows\\
                             CurrentVersion\\Explorer");
            System.out.println(explorer.getName());
            System.out.println(explorer.getStringValue(
                                        "Logon User Name"));
            explorer.closeKey();
            schluessel.closeKey();
        } catch (NoSuchKeyException e) {
            System.err.println(e);
        } catch (RegistryException e) {
            System.err.println(e);
        }
    }
}
```

Listing 8.1 RegistryDemo1.java

Die statische Methode `getTopLevelKey()` bildet den Einstieg in die Registry. Als Parameter übergeben Sie ihr einen der in Abschnitt 8.1 vorgestellten Hauptzweige, beispielsweise *HKEY_LOCAL_MACHINE* oder *HKEY_CLASSES_ROOT*. Die Methode liefert eine Instanz der Klasse `RegistryKey`, deren Methoden für den weiteren Zugriff auf Schlüssel und Werte verwendet werden. Um vom ausgewählten Hauptzweig aus einen bestimmten Schlüssel anzusprechen, verwenden Sie bitte die Methode `openSubKey()`. Sie erhält einen relativen Schlüsselnamen als Parameter. Relativ bedeutet in diesem Zusammenhang, dass der Teil fehlt, der den Hauptzweig spezifiziert. Dieser wurde ja bereits beim Aufruf von `getTopLevelKey()` angegeben, eine erneute Nennung wäre folglich unnötig. Auch `openSubKey()` liefert eine Instanz von `RegistryKey`.

Für den Zugriff auf die Werte eines Schlüssels stehen verschiedene Methoden zur Verfügung. Welche Sie verwenden, hängt vom Datentyp des zu ermittelnden Wertes ab. Der *Login Name* wird als Zeichenkette gespeichert, hat also den Datentyp *REG_SZ*. Aus diesem Grund verwendet *RegistryDemo1* die Methode `getStringValue()`. Um Werte mit anderen Typen auslesen zu können, steht die Methode `getValue()` zur Verfügung. Sie liefert eine Instanz der Klasse `RegistryValue`. Um nicht mehr benötigte Ressourcen freizugeben, sollten Sie nach Arbeiten an einem Schlüssel die Methode `closeKey()` aufrufen.

Nachdem Sie nun wissen, wie Sie mit *JNIRegistry* Werte aus der Windows-Registrierung auslesen, werden wir uns im nächsten Abschnitt mit dem Schreiben von Registry-Einträgen beschäftigen.

8.2.2 Schreibender Zugriff auf die Registry

Das Erzeugen neuer sowie das Verändern bestehender Schlüssel gehört zu den Routinearbeiten jeder Windows-Anwendung. Beispielsweise werden – wie bereits erwähnt – Programmeinstellungen nicht in Dateien, sondern in der Registrierung abgelegt. Aber selbst wenn Sie sich entscheiden, aus Gründen der Portabilität solche Daten in *.properties*-Dateien abzulegen, gibt es gute Gründe, schreibend auf die Registry zuzugreifen. Welche dies sind, zeigen Ihnen die Abschnitte 8.3.2 und 8.3.3. Zunächst möchte ich Sie aber mit den notwendigen Grundlagen vertraut machen. Sehen Sie sich hierzu bitte das Programm *RegistryDemo2* an.


```
package javafuerwindows.registry;
import com.ice.jni.registry.NoSuchKeyException;
import com.ice.jni.registry.RegStringValue;
import com.ice.jni.registry.Registry;
import com.ice.jni.registry.RegistryException;
import com.ice.jni.registry.RegistryKey;
import javax.swing.JOptionPane;

public class RegistryDemo2 {

    public static void main(String [] args) {
        RegistryKey schluessel = Registry.getTopLevelKey(
                                    "HKEY_LOCAL_MACHINE");
        String schluesselName = "Thomas Kuenneth";
        try {
            RegistryKey software = schluessel.openSubKey(
                                        "Software");
            RegistryKey meiner = software.createSubKey(
                                    schluesselName, null);
            RegStringValue wert = new RegStringValue(
                                       meiner,
                                       "Info",
                                       "ein Testeintrag");
            meiner.setValue(wert);
            meiner.flushKey();
```

```
            meiner.closeKey();
            /*
             * eine kleine Pause einlegen
             */
            JOptionPane.showMessageDialog(null,
                                  "Schlüssel wird gelöscht");
            software.deleteSubKey(schluesselName);
            software.closeKey();
            schluessel.closeKey();
        } catch (NoSuchKeyException e) {
            System.err.println(e);
        } catch (RegistryException e) {
            System.err.println(e);
        }
    }
}
```

Listing 8.2 RegistryDemo2.java

Das Programm erzeugt unter *HKEY_LOCAL_MACHINE\SOFTWARE* einen Schlüssel mit Namen *Thomas Kuenneth*. Dieser enthält neben dem obligatorischen Wert *(Standard)* die Zeichenkette *Info*. Sie beinhaltet den Text `ein Testeintrag`. Abbildung 8.9 zeigt den erzeugten Schlüssel.

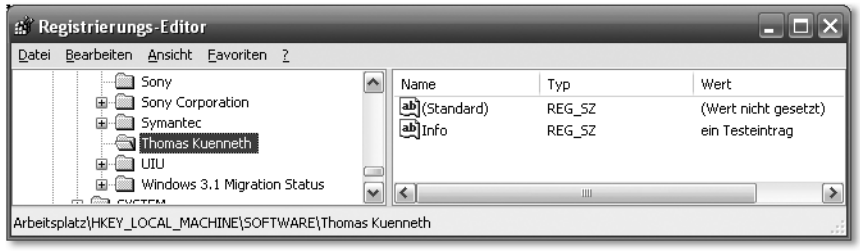

Abbildung 8.9 Der durch RegistryDemo2 erzeugte Schlüssel

Nachdem *RegistryDemo2* den Schlüssel in der Registrierung eingetragen hat, wird ein kleiner Dialog geöffnet. Sobald Sie diesen schließen, wird der Schlüssel gelöscht.

Um einen neuen Schlüssel anzulegen, rufen Sie die Methode `createSubKey()` des Eltern-Schlüssels auf. Diesen können Sie beispielsweise mit `getTopLevelKey()` und `openSubKey()` instantiieren. Mit `setValue()` fügen Sie dem neuen Schlüssel Werte hinzu. Es handelt sich hierbei um Instanzen der Klassen `RegStringValue`, `RegMultiStringValue`, `RegDWordValue` oder `RegBinary-`

`Value`. Die genannten Klassen repräsentieren Datentypen, die ich Ihnen in Abschnitt 8.1.1 vorgestellt habe. Beispielsweise wird die Zeichenkette (*REG_SZ*) auf die Klasse `RegStringValue` abgebildet. Um sicherzustellen, dass Änderungen in die Registry übernommen werden, sollten Sie vor dem Schließen des Schlüssels die Methode `flushKey()` aufrufen. Für das Löschen eines Schlüssels steht Ihnen `deleteSubKey()` zur Verfügung.

8.3 Beispiele für den Zugriff auf die Registry

Die Windows-Registrierung ist als zentrale System-Datenbank für den Java-Entwickler äußerst interessant. Zum einen enthält sie Informationen, die sich mit »Bordmitteln« nicht ohne weiteres abfragen lassen. Zum anderen wird die Integration von Anwendungen in Windows traditionell über die Registry abgewickelt. Hier lässt Java zahlreiche Möglichkeiten ungenutzt. Der Zugriff auf die Registrierung öffnet Ihren Programmen folglich viele Türen. Was sich hinter diesen Türen verbirgt, zeigt dieser Abschnitt.

8.3.1 Installierte Java-Versionen ermitteln

Wie ermitteln Sie, welche Versionen der Java-Laufzeitumgebung und des Development Kits auf einem Rechner installiert sind? Dies ist natürlich besonders im Hinblick auf Installationsroutinen interessant, die dem Anwender die Möglichkeit geben sollen, ein Java-Programm in einer ganz bestimmten Laufzeitumgebung auszuführen. Wenn Sie nun einwenden, dass für den Start eines Java-gestützten Setup-Programms mindestens eine geeignete Laufzeitumgebung installiert sein muss, haben Sie selbstverständlich Recht. Glücklicherweise ist dies auf den meisten aktuell zu erwerbenden Rechnern der Fall, da die Hersteller dazu übergehen, zumindest eine Java-Laufzeitumgebung aufzuspielen.

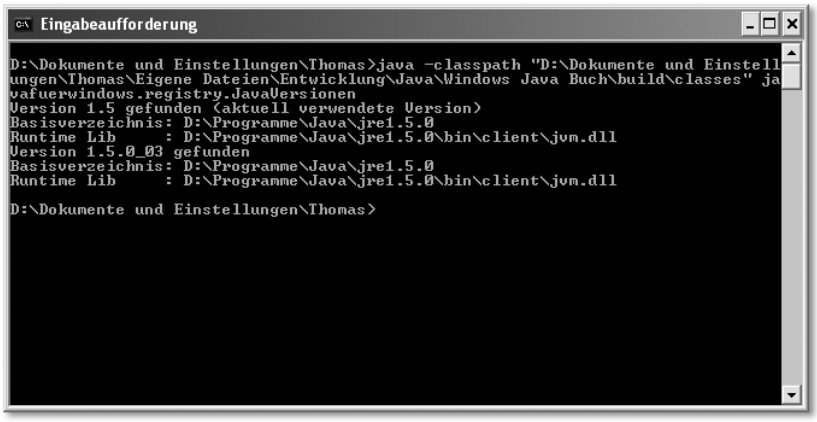

Abbildung 8.10 Ausgabe des Programms JavaVersionen

Das in Abbildung 8.10 gezeigte Programm *JavaVersionen* veranschaulicht die Vorgehensweise beim Ermitteln der installierten Java-Versionen. Sehen Sie sich bitte zunächst den Quelltext an.

```
package javafuerwindows.registry;
import com.ice.jni.registry.NoSuchKeyException;
import com.ice.jni.registry.Registry;
import com.ice.jni.registry.RegistryException;
import com.ice.jni.registry.RegistryKey;
import java.util.Enumeration;
public class VavaVersionen {
    public static void main(String [] args) {
        RegistryKey schluessel = Registry.getTopLevelKey(
                                    "HKEY_LOCAL_MACHINE");
        try {
            RegistryKey javaSoft = schluessel.openSubKey(
                                     "Software\\JavaSoft");
            RegistryKey jre = javaSoft.openSubKey(
                                "Java Runtime Environment");
            Enumeration elemente = jre.keyElements();
            String aktuell = jre.getStringValue(
                                        "CurrentVersion");
            while (elemente.hasMoreElements()) {
                String strVersion =
                          elemente.nextElement().toString();
                System.out.print("Version " + strVersion
                                        + " gefunden");
                if (strVersion.equals(aktuell))
                    System.out.print( " (aktuell verwendete
                                     Version)");
                System.out.println();
                RegistryKey version = jre.openSubKey(
                                        strVersion);
                System.out.println("Basisverzeichnis: " +
                      version.getStringValue("JavaHome"));
                System.out.println("Runtime Lib     : " +
                      version.getStringValue("RuntimeLib"));
                version.closeKey();
            }
            jre.closeKey();
            javaSoft.closeKey();
```

```
            schluessel.closeKey();
        } catch (NoSuchKeyException e) {
            System.err.println(e);
        } catch (RegistryException e) {
            System.err.println(e);
        }
    }
}
```

Listing 8.3 JavaVersionen.java

Ein wichtiger Schlüssel im Hinblick auf installierte Laufzeitumgebungen und Java Development Kits ist *HKEY_LOCAL_MACHINE\SOFTWARE\JavaSoft*. Er beinhaltet unter anderem die Unter-Schlüssel *Java Development Kit* und *Java Runtime Environment*. Die beiden sind sehr ähnlich aufgebaut und enthalten Schlüssel, die Versionsnummern als Namen tragen.

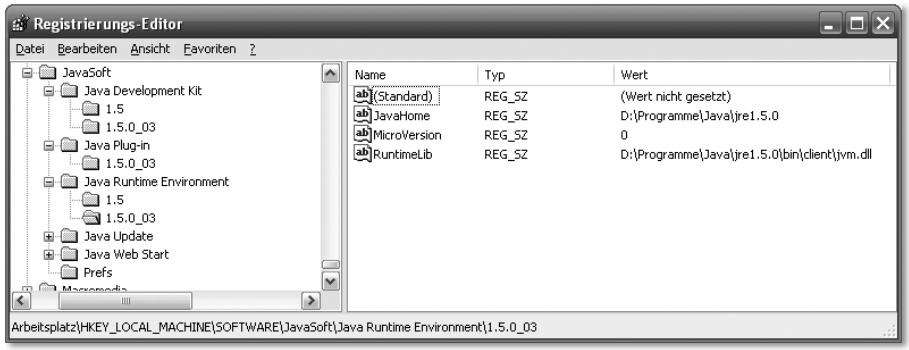

Abbildung 8.11 Anzeige der installierten Java-Versionen durch Regedit

Bitte beachten Sie, dass der in Abbildung 8.11 zu sehende Wert *RuntimeLib* nur im Zweig von *Java Runtime Environment* gespeichert wird. Die Unter-Schlüssel von *Java Development Kit* beinhalten ihn nicht. Von Bedeutung ist *RuntimeLib*, wenn Sie beim Aufruf von Java-Anwendungen nicht den Standard-Launcher verwenden möchten, sondern eine eigene, native Anwendung. Um die installierten Java-Versionen ermitteln zu können, verwendet *JavaVersionen* die Methode `keyElements()` der `RegistryKey`-Instanz, die auf *Java Runtime Environment* verweist. Sie liefert eine Aufzählung aller Namen der Kinder-Schlüssel. Deren Elemente können als Argument für `openSubKey()` genutzt werden.

Wie Sie sehen, ist es mit sehr geringem Aufwand möglich, die auf einem Rechner zur Verfügung stehenden Versionen der Java-Laufzeitumgebung zu ermitteln. Dies kann übrigens auch dann von Interesse sein, wenn Ihr Programm

nicht auf eine Version festgelegt ist. Beispielsweise ist es auf diese Weise möglich, ein optionales Paket automatisch in das Erweiterungsverzeichnis aller Laufzeitumgebungen zu kopieren.

8.3.2 Dokumentvorlagen

In Kapitel 7 zeige ich Ihnen, wie Sie mit der Klassenbibliothek *JDesktop Integration Components* neue Dateitypen registrieren können. Kurz gesagt ordnet ein Dateityp allen Dateien mit einer bestimmten Endung ein Icon und eine Reihe von Aktionen zu. Beispielsweise öffnet sich nach einem Doppelklick auf eine solche Datei die ihr zugeordnete Anwendung. Dies ist für Programme mit eigenem Dokumentenformat natürlich äußerst interessant. Das in Abbildung 8.12 gezeigte Kontextmenü von Dateiordnern bietet für einige Dateitypen das Anlegen von neuen Dokumenten an. Es liegt eigentlich auf der Hand, eigene Dokumenttypen in dieses Menü einzubinden. Leider bieten die *JDesktop Integration Components* einen vergleichbaren Mechanismus bisher nicht an. Im Folgenden wird deshalb beschrieben, welche Einträge Sie in der Registry vornehmen müssen, um die gewünschte Funktionalität selbstständig zu realisieren.

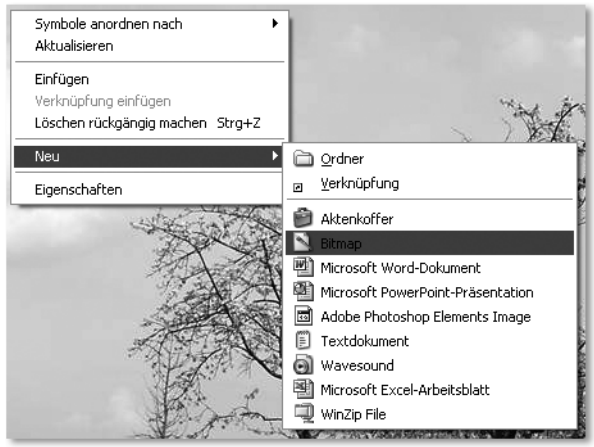

Abbildung 8.12 Anlegen neuer Dateien

Viele Informationen rund um Dateitypen werden im Registry-Zweig *HKEY_CLASSES_ROOT* abgelegt. Beispielsweise existiert für jede Dateiendung ein Unter-Schlüssel, dessen Name der Dateiendung entspricht. Abbildung 8.13 zeigt den Dateityp *.abc*, der durch das Programm *FiletypesDemo3* aus Kapitel 7, *JDesktop Integration Components*, angelegt wird.

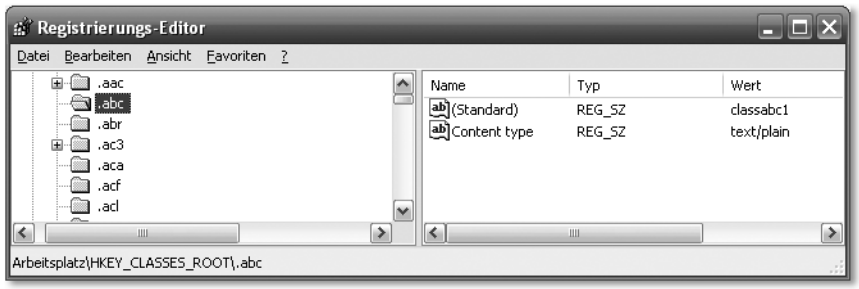

Abbildung 8.13 Der durch FiletypesDemo3 erzeugte Dateityp

Um dem Menü **Neu** weitere Einträge hinzuzufügen, müssen Sie einen Schlüssel unterhalb des Dateitypschlüssels erzeugen, der den Namen *ShellNew* trägt. Dessen Wert legt den Inhalt der anzulegenden Datei fest. Die einfachste Variante weist Windows an, eine 0 Byte große Datei zu erzeugen. Sehen Sie sich hierzu bitte das Programm *NeueDatei1* an.


```
package javafuerwindows.registry;
import com.ice.jni.registry.NoSuchKeyException;
import com.ice.jni.registry.Registry;
import com.ice.jni.registry.RegistryException;
import com.ice.jni.registry.RegistryKey;
import com.ice.jni.registry.RegistryValue;

public class NeueDatei1 {
    public static void main(String [] args) {
        RegistryKey schluessel = Registry.getTopLevelKey(
                                    "HKEY_CLASSES_ROOT");
        try {
            RegistryKey dateiendung = schluessel.openSubKey(
                                                ".abc");
            RegistryKey shellNew = dateiendung.createSubKey(
                                            "ShellNew", "");
            RegistryValue wert = new RegEmptyValue(shellNew,
                                                "NullFile");
            shellNew.setValue(wert);
            shellNew.flushKey();
            shellNew.closeKey();
            dateiendung.closeKey();
        } catch (NoSuchKeyException e) {
            System.err.println(e);
        } catch (RegistryException e) {
```

```
            System.err.println(e);
        }
    }
}
```

Listing 8.4 NeueDatei1.java

Der Wert des Schlüssels *ShellNew* wird mit der Methode `setValue()` gesetzt. Als Parameter übergebe ich eine Instanz der Klasse `RegEmptyValue`. Diese ist nicht Teil der *JNIRegistry*-Klassenbibliothek, sondern wurde von mir programmiert. Mit dieser Klasse soll demonstriert werden, wie Sie fehlende `RegistryValue`-Implementierungen nachrüsten können. Den Quelltext finden Sie auf der Begleit-CD.

Abbildung 8.14 Eine neue Dokumentvorlage

Abbildung 8.14 zeigt das Menü **Neu** nach dem Start von *NeueDatei1*. Bitte beachten Sie, dass der Dateityp *.abc* zu diesem Zeitpunkt registriert sein muss. Andernfalls würde das Programm den benötigten Registry-Schlüssel nicht finden. Aus diesem Grund sollten Sie vorher *FiletypesDemo3* aufrufen. Wenn Sie **der Dateityp .abc** anklicken, erzeugt Windows eine leere Datei. Verantwortlich hierfür ist der Name *NullFile*, der im Programm *NeueDatei1* für den Wert des Schlüssels *ShellNew* vergeben wurde. Neben diesem sieht Microsoft drei weitere Namen vor.

Mit *FileName* können Sie eine Datei festlegen, die als Vorlage für die neu zu erstellende Datei dient. *FileName* ist in diesem Fall vom Typ *REG_SZ* und erhält den Vorlagennamen.

```
/*
 * Achtung: bitte Pfad anpassen
 */
RegistryValue wert = new RegStringValue(shellNew,
       "FileName",
```

```
        "D:\\Dokumente und Einstellungen\\Thomas\\Eigene
         Dateien\\Entwicklung\\Java\\Windows Java Buch\\src\\
         Vorlage fuer Dateityp abc.abc");
shellNew.setValue(wert);
```

Das Programm *NeueDatei2* weicht nur in wenigen Zeilen von *NeueDatei1* ab. Es erzeugt eine Instanz der Klasse `RegStringValue`, die auf die Vorlagendatei verweist. Der Name *FileName* weist Windows an, die durch den Dateinamen referenzierte Datei zu kopieren. Sie können einen absoluten Pfadnamen angeben oder nur den Dateinamen eintragen. In diesem Fall wird die Vorlage im Unter-Ordner *ShellNew* des Windows-Verzeichnisses gesucht.

Als dritte Variante können Sie den Inhalt der neu zu erstellenden Datei direkt in der Registry ablegen. Im Vergleich zu *FileName* hat dies den Vorteil, dass die Vorlagendatei nicht verloren gehen kann. Dies könnte passieren, wenn der Anwender Ihres Programms die Vorlage löscht oder an einen anderen Ort verschiebt. Auf der anderen Seite rät Microsoft davon ab, große Datenmengen in der Registrierung abzulegen. Als zentrale Konfigurations-Datenbank wird sie ständig von Anwendungen und vom System selbst abgefragt, sodass aus Performancegründen unnötiger Daten-Ballast vermieden werden sollte. Das Programm *NeueDatei3* zeigt das Vorgehen beim Ablegen von Vorlagendaten direkt in der Registry. Es weicht nur in wenigen Zeilen von seinen Vorgängern ab. Sie sehen deshalb nur einen kurzen Ausschnitt. Die vollständige Fassung finden Sie aber selbstverständlich auf der Begleit-CD.

```
byte [] daten = new String("NeueDatei3: Data") .getBytes();
RegistryValue wert = new RegBinaryValue(shellNew,
                                        "Data", daten);
shellNew.setValue(wert);
```

Wie schon in *NeueDatei1* und *NeueDatei2* wird ein Wert erzeugt, der mit `setValue()` an den Schlüssel *ShellNew* gebunden wird. Er hat den Namen *Data* und ist vom Typ *REG_BINARY*. Der Konstruktor der *JNIRegistry*-Klasse `RegBinaryValue` erwartet ein `byte`-Feld, das wir mit den Daten der Vorlage füllen.

Die letzte und mächtigste Variante, Dateivorlagen zu erzeugen, ist das Ausführen eines Programms, nachdem der Benutzer den Dokumenttyp im Menü **Neu** ausgewählt hat. Auch hier will ich Ihnen anhand eines Beispiels das Vorgehen verdeutlichen. Sehen Sie sich bitte zunächst folgende Zeilen aus dem Programm *NeueDatei4* an.

```
/*
 * Achtung: bitte Pfad anpassen
 */
RegistryValue wert = new RegStringValue(shellNew,
                                        "Command",
        "javaw.exe -classpath \"D:\\Dokumente und
         Einstellungen\\Thomas\\Eigene
         Dateien\\Entwicklung\\Java\\Windows Java
         Buch\\build\\classes\"
         javafuerwindows.registry.NeueDatei4Helper \"%1\"");
shellNew.setValue(wert);
```

Es wird ein neuer *REG_SZ*-Wert mit Namen *Command* angelegt, der einen vollständigen Programmaufruf beinhaltet. Das zu startende Programm heißt *NeueDatei4Helper*. Bei der eigentlichen Ausführung wird ihm ein Parameter übergeben. Hierzu wird `%1` durch den vollständigen Namen der anzulegenden Datei ersetzt.

```
package javafuerwindows.registry;
import java.io.FileWriter;
import java.io.IOException;

public class NeueDatei4Helper {
    public static void main(String [] args) {
        String name = args[0];
        FileWriter fw = null;
        try {
            fw = new FileWriter(name);
            fw.write(Integer.toString(args.length) +
                                ", " + args[0]);
            fw.flush();
        } catch (IOException e) {
        } finally {
            if (fw != null) {
                try {
                    fw.close();
                } catch (IOException e) {
                }
            }
```

```
            }
        }
}
```

Listing 8.5 NeueDatei4Helper.java

Das Programm *NeueDatei4Helper* wird aufgerufen, wenn Sie im Menü **Neu** den Eintrag **der Dateityp .abc** anklicken. Vereinfacht gesagt: Die in diesem Zusammenhang gestarteten Programme müssen die ihnen übergebene Datei anlegen. Ihren Inhalt kann die Anwendung festlegen.

Für Java-Programme mit eigenem Dokumentenformat ist die Möglichkeit, Dateien über das systemweite Menü **Neu** zu erzeugen, zweifelsohne eine willkommene Ergänzung des Funktionsumfangs. In Verbindung mit den in Kapitel 7 beschriebenen *JDesktop Integration Components* können Sie Ihre Anwendung nahtlos in das System einbinden, sodass sich Benutzer Ihrer Programme nicht umstellen müssen. Vielmehr können sie gewohnte Arbeitsabläufe beibehalten, beispielsweise das Öffnen von Dokumenten durch Doppelklick oder das schnelle Erzeugen neuer Dokumente.

Auch im nächsten Abschnitt geht es um eine möglichst nahtlose Integration von Java-Programmen in Windows. Ich werde Ihnen zeigen, wie Sie Ihre Programme in das Modul *Software* der *Systemsteuerung* einbetten können.

8.3.3 Einbinden von Java-Programmen in die Systemsteuerung

Windows-Anwender sind gewohnt, Programme über das Start-Menü oder durch Doppelklick auf ein Desktop-Icon ausführen zu können. Tipps hierzu finden Sie in Kapitel 10, *Deployment*. Ferner gehört es zum guten Ton, dass eine Anwendung im Modul *Software* der *Systemsteuerung* erscheint und von dort aus auch entfernt werden kann. Wenn Sie Ihre Programme mit einer eigenen Installationsroutine versehen, sollten Sie diesen Mechanismus im Hinblick auf eine optimale Integration in das System ebenfalls unterstützen. Wie dies funktioniert, sehen Sie anhand des Programms *AddRemoveDemo*.

Um eine Anwendung in der Liste der installierten Software erscheinen zu lassen, sind nur einige wenige Einträge in der Registry erforderlich. Der Schlüssel *HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Uninstall* enthält für jedes installierte Programm einen eigenen Unter-Schlüssel.

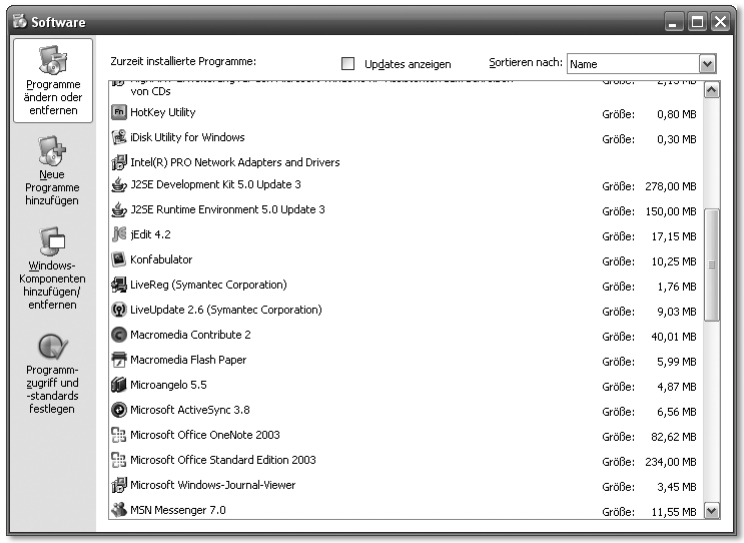

Abbildung 8.15 Das Modul »Software« der Systemsteuerung

```
    private static final String strUninstall =
        "Software\\Microsoft\\Windows\\CurrentVersion\\Uninstall";
    private static final String strAddRemoveDemo =
        "AddRemoveDemo";

    private static void installieren() {
        RegistryKey schluessel =
                  Registry.HKEY_LOCAL_MACHINE;
        try {
            RegistryKey uninstall =
               schluessel.openSubKey(strUninstall);
            RegistryKey anwendung =
               uninstall.createSubKey(strAddRemoveDemo, "");
            /*
             * angezeigter Name
             */
            anwendung.setValue(erzeugeWert(anwendung,
            "DisplayName", "Add/Remove Demonstration"));
            /*
             * wird ausgeführt, um das Programm zu entfernen
             */
            anwendung.setValue(erzeugeWert(anwendung,
"UninstallString",
```

```
"\"D:\\Programme\\Java\\jdk1.5.0\\bin\\javaw.exe\"
-classpath \"D:\\Dokumente und
Einstellungen\\Thomas\\Eigene
Dateien\\Entwicklung\\Java\\Windows Java
Buch\\build\\classes\"
javafuerwindows.registry.AddRemoveDemo \"-entfernen\""));
         /*
          * wird ausgeführt, um das
          * Programm zu modifizieren
          */
         anwendung.setValue(erzeugeWert(anwendung,
       "ModifyPath",
       "\"D:\\Programme\\Java\\jdk1.5.0\\bin\\javaw.exe\"
              -classpath \"D:\\Dokumente und
               Einstellungen\\Thomas\\Eigene
               Dateien\\Entwicklung\\Java\\Windows Java
               Buch\\build\\classes\"
               javafuerwindows.registry.AddRemoveDemo
               \"-modifizieren\""));
         /*
          * ein neben dem Programmnamen angezeigtes Icon
          */
         anwendung.setValue(erzeugeWert(anwendung,
         "DisplayIcon", "D:\\Dokumente und
          Einstellungen\\Thomas\\Eigene
          Dateien\\Entwicklung\\Java\\Windows Java
          Buch\\src\\Buchtasse.ico"));
         /*
          * diese Angaben erscheinen
          * unter “Supportinformationen”
          */
         anwendung.setValue(erzeugeWert(anwendung,
               "DisplayVersion", "1.0 build 123"));
         anwendung.setValue(erzeugeWert(anwendung,
               "Publisher", "Thomas Kuenneth"));
         anwendung.setValue(erzeugeWert(anwendung,
     "HelpLink", "http://www.addremovedemo.de/hilfe/"));
         anwendung.setValue(erzeugeWert(anwendung,
      "UrlInfoAbout", "http://www.addremovedemo.de/"));
         anwendung.setValue(erzeugeWert(anwendung,
```

```
                "UrlUpdateInfo",
                "http://www.addremovedemo.de//update/"));
            anwendung.flushKey();
            anwendung.closeKey();
            uninstall.closeKey();
        } catch (NoSuchKeyException e) {
            System.err.println(e);
        } catch (RegistryException e) {
            System.err.println(e);
        }
    }
```

Sie müssen für Ihr Java-Programm also zunächst einen neuen Schlüssel mit beliebigem Namen anlegen und zumindest die Werte *DisplayName* und *UninstallString* setzen. Der Erste legt den Namen der Anwendung fest, wie er vom Modul *Software* angezeigt wird. *UninstallString* bestimmt das Programm, das gestartet wird, wenn der Anwender die Software entfernen möchte. Mit *DisplayIcon* können Sie ein beliebiges Icon neben dem Programmnamen anzeigen lassen.

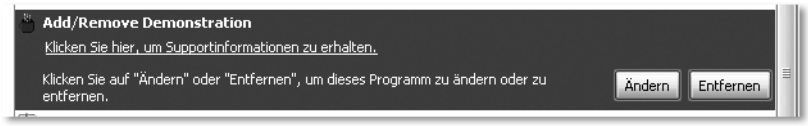

Abbildung 8.16 AddRemoveDemo im Modul »Software« der Systemsteuerung

Die in Abbildung 8.16 zu sehende Schaltfläche **Ändern** erscheint nur, wenn Sie den Wert *ModifyPath* hinzufügen. Auch er enthält den Pfad eines Programms einschließlich etwaiger Parameter, das nach Anklicken der Schaltfläche aufgerufen wird. Es bietet sich an, Änderungen an der Installation sowie das Entfernen der Anwendung durch ein gemeinsames Programm durchführen zu lassen. Welche Aktion ausgeführt werden soll, können Sie dem Programm als Parameter übermitteln. Eine mögliche Vorgehensweise sehen Sie im weiter oben abgedruckten Programmfragment, beim Setzen der beiden Werte *UninstallString* und *ModifyPath*. Die `main`-Methode wertet dann die übergebenen Argumente aus und vollführt die gewünschte Aktion.

```
public static void main(String [] args) {
    if (args.length == 0) {
        installieren();
    } else {
        String cmd = args[0];
```

```
        if (cmd.equals("-entfernen")) {
            entfernen();
        } else if (cmd.equals("-modifizieren")) {
            JOptionPane.showMessageDialog(null,
                      "modifizieren", "Info",
                      JOptionPane.INFORMATION_MESSAGE);
        } else {
            System.out.println(cmd + " ist kein gültiges
                               Argument");
        }
    }
    System.exit(0);
}
```


Das Modul *Software* zeigt weitergehende Informationen zu einer installierten Anwendung an, wenn der Benutzer den Link **Klicken Sie hier, um Supportinformationen zu erhalten** anklickt. Sie können den Inhalt des Dialogs durch das Hinzufügen weiterer Werte beeinflussen.

Abbildung 8.17 Der Dialog Supportinformationen

Die Werte *Publisher* und *UrlInfoAbout* werden als **Herausgeber** angezeigt, *HelpLink* und *UrlUpdateInfo* erscheinen neben **Supportinformationen** bzw. **Produktupdates**.

9 Java-COM-Brücken

9 Java-COM-Brücken

Das Component Object Model (COM) erlaubt Windows-Programmen, auf Funktionen anderer Anwendungen zuzugreifen, beispielsweise auf Dienste des Windows Media Players, des Windows-Adressbuchs oder der Microsoft Office-Suite. Dieses Kapitel gibt Aufschluss darüber, wie Sie Ihre Java-Programme um diese Fähigkeit erweitern.

In Kapitel 5, *Kommunikation mit Microsoft Office*, zeige ich Ihnen, wie Sie mit Hilfe der Klassenbibliothek *POI* Excel-Dateien lesen und schreiben können. Das Bemerkenswerte an *POI* ist, dass Sie hierzu nicht erst *Excel* installieren müssen. *POI* ist plattformunabhängig und setzt keine nativen Ressourcen voraus. Anders verhält es sich mit dem *Java Outlook Connector*, der im selben Kapitel vorgestellt wird. Er basiert auf Teilen von *Outlook* und funktioniert demzufolge nur auf Windows-Rechnern, auf denen diese Anwendung installiert wurde. Auch Teile der *JDesktop Integration Components* – Thema in Kapitel 7 – setzen native Programme voraus, beispielsweise die Klassen des Pakets `org.jdesktop.jdic.browser`, die Ihren Java-Programmen Zugriff auf den Internet Explorer und Mozilla gewähren. Wie aber realisieren die genannten Java-Klassenbibliotheken die Kommunikation? Oder anders gefragt, muss der Datenaustausch zwischen Java und einer Windows-Anwendung jedes Mal aufs Neue realisiert werden, oder gibt es einen gemeinsamen Nenner, eine Basis, die als Bindeglied zwischen Windows und Java fungiert?

Bei Java-COM-Brücken handelt es sich um Klassenbibliotheken, die Java-Programmen den Datenaustausch mit vielen Windows-Anwendungen gestatten. Datenaustausch meint in diesem Zusammenhang allerdings keineswegs nur das Lesen oder Schreiben von Werten. Vielmehr können Sie Windows-Programme »fernsteuern«, also Funktionen der betreffenden Anwendung nutzen. Möglich wird dies durch das *Component Object Model*, das unter Windows seit vielen Jahren die Grundlage für die Kommunikation zwischen Anwendungen bildet.

9.1 Das Component Object Model (COM)

Java-Programme bestehen im Allgemeinen aus einer Vielzahl von Klassen. Während der Abarbeitung eines Programms werden Objekte instantiiert, die wiederum Methoden anderer Objekte aufrufen, Ihnen Werte in Form von Parametern übergeben und Ergebnisse (Rückgabewerte) erhalten. Wie die aufgerufenen Objekte intern arbeiten, ist in diesem Zusammenhang völlig unerheb-

lich. Wichtig ist nur, dass ein Objekt eine bestimmte Anzahl von Methoden bereithält, die von anderen verwendet werden können. Das COM funktioniert auf ganz ähnliche Weise. Eine Komponente stellt Funktionen in Form von *Interfaces* zur Verfügung. Um die Dienste eines COM-Objekts in Anspruch zu nehmen, ruft eine andere Komponente eine Funktion aus einem der bereitgestellten Interfaces auf. Wie dies im Einzelnen funktioniert, zeige ich im folgenden Abschnitt. Er vermittelt die technischen Grundlagen des COM, soweit sie für das Verständnis und die Anwendung von Java-COM-Brücken relevant sind. Wenn Sie sich ausführlicher mit dessen Architektur befassen möchten, finden Sie interessante Hintergrundinformationen im englischsprachigen Artikel »The Component Object Model: Technical Overview«.[1]

9.1.1 Grundlagen des COM

Das COM definiert die technischen Voraussetzungen, damit Programme ihre Funktionen anderen Anwendungen zur Verfügung stellen können. Diese Funktionen werden zu Interfaces zusammengefasst, die letztlich Zeiger auf die angebotenen Dienste enthalten. Interfaces legen also den nach außen sichtbaren Funktionsumfang eines COM-Objekts fest. Allerdings kann eine Komponente im Prinzip beliebig viele Interfaces zur Verfügung stellen. Wie aber werden Komponenten und Interfaces eigentlich identifiziert? In Java referenzieren wir eine Klasse durch Nennung des Pakets, zu dem sie gehört, sowie ihren Namen. Es gibt eine Konvention, derzufolge sich Paketnamen an Internetdomains orientieren sollen. Da die meisten Java-Entwickler diesen Vorgaben folgen, sind Namenskonflikte zum Glück selten, allerdings keineswegs ausgeschlossen. Microsoft hat für COM eine andere Strategie gewählt. Um Komponenten und Interfaces weltweit eindeutig identifizieren zu können, werden sie mit so genannten *Globally Unique Identifiers* (GUIDs) versehen. Hierbei handelt es sich um Pseudo-Zufallszahlen, die mit Hilfe eines Algorithmus berechnet werden, der nur extrem selten Zahlen mehrfach generiert. Selbst wenn also Entwickler Interfaces mit denselben Namen erfinden, kann praktisch ausgeschlossen werden, dass ein Programm die falsche Komponente anspricht. GUIDs, die sich auf Komponenten beziehen, werden Klassen-IDs (CLSIDs) genannt. Interfaces hingegen werden durch Interface-IDs (IIDs) eindeutig bestimmt. Interfaces sind übrigens keineswegs Instanzen von COM-Objekten. Ähnlich den Interfaces in Java legen sie den Funktionsumfang fest, den eine Komponente nach außen zur Verfügung stellt. Aus technischer Sicht ist ein Interface ein Zeiger auf eine *VTable* genannte Datenstruktur, eine Tabelle, die die Pointer auf die eigentlichen Funktionen enthält. Um eine Instanz eines COM-Objektes zu erhalten,

1 http://www.cs.umd.edu/~pugh/com/

wird zunächst deren Klassen-ID an eine COM-Laufzeitumgebung übermittelt. Diese liefert eine Factory-Klasse, mit deren Hilfe in einem zweiten Schritt die eigentliche Instanz der Komponente erzeugt wird. Hierzu wird die ID des gewünschten Interfaces übergeben. Jede Komponente muss mindestens das Interface `IUnknown` implementieren, das drei grundlegende Funktionen beinhaltet. Dies sind `AddRef()`, `QueryInterface()` und `Release()`. Sie werden für die Kontrolle des Lebenszyklus einer Instanz sowie für den Zugriff auf ihre Interfaces benötigt. Alle anderen Interfaces leiten von `IUnknown` ab. Sie enthalten die für den Programmierer eigentlich interessanten Funktionen einer Komponente. Neben dem Aufruf von Funktionen eines COM-Objekts über dessen *VTable* hat Microsoft noch eine zweite Kommunikationsmöglichkeit vorgesehen. Das Interface `IDispatch`, das ebenfalls von `IUnknown` ableitet, definiert unter anderem die Funktion `invoke()`, die ebenfalls den Zugriff auf Funktionen und Eigenschaften einer Komponente ermöglicht. Es wurde eingeführt, um auch Skriptsprachen die Kommunikation mit COM-Komponenten zu ermöglichen.

Vielleicht haben Sie sich schon gefragt, wie Sie herausfinden können, welche Funktionen eine Komponente anbietet. Im folgenden Abschnitt möchte ich diese Fragen klären und Ihnen die so genannten *Type Libraries* vorstellen. Sie beinhalten den »Funktionsumfang« eines COM-Objekts. Traditionell werden diese Bibliotheken von RAD-Tools wie *Visual Basic* oder *Delphi* verwendet, können aber auch von anderen Programmen ausgewertet werden. Eine solche Anwendung, den *TypeLib Browser* werden Sie gleich kennen lernen.

9.1.2 Type Libraries

Die Java *Reflection*-API gibt Ihnen die Möglichkeit, viele interessante Informationen über Objekte und Klassen zu erhalten, beispielsweise die Klasse eines Objekts, ihre Methoden und deren Parameter, den Namen einer Klasse und das Paket, zu dem sie gehört. *Type Libraries* haben im COM eine vergleichbare Aufgabe. Sie beschreiben eine Komponente, indem sie deren Klassen-ID sowie alle Interface-IDs nennen. Zudem enthalten sie Beschreibungen der Funktionen aller Interfaces. Das Programm *TypeLib Browser*[2] von José Roca stellt unter anderem die in den Type Libraries enthaltenen Informationen dar. Auf diese Weise erhalten Sie einen Einblick in die Funktionen, die ein COM-Objekt zur Verfügung stellt. Das Programm ist kostenlos und steht auf der Homepage des Autors zum Download bereit. Nach dem Herunterladen und dem Entpacken können Sie es ohne weitere Installation direkt starten. Beachten Sie bitte, dass es keineswegs im Hinblick auf Java entwickelt wurde, sondern in erster Linie

2 http://com.it-berater.org/typelib_browser.htm

für Anwender der Programmiersprache *PowerBASIC* gedacht ist. Trotzdem werden Ihnen seine Analyse- und Anzeigefunktionen sehr gute Dienste leisten. Abbildung 9.1 zeigt das Hauptfenster des *TypeLib Browsers* unmittelbar nach dem Programmstart.

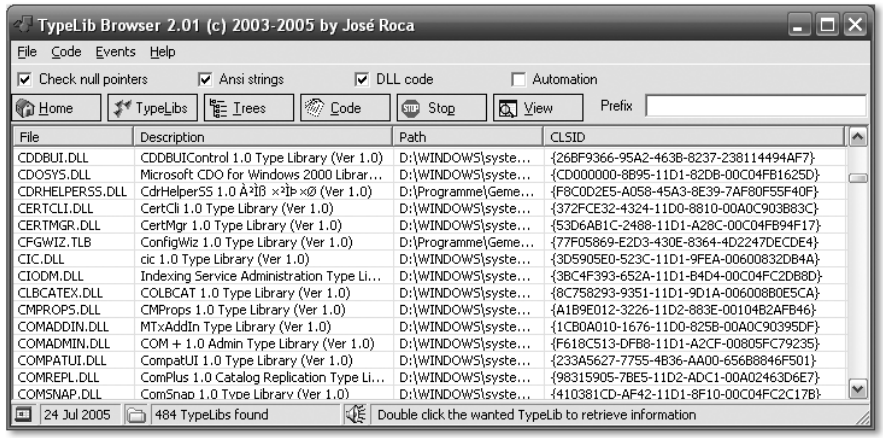

Abbildung 9.1 Der TypeLib Browser

Die Hauptanzeige, die Sie übrigens jederzeit durch Anklicken der Schaltfläche **TypeLibs** erreichen, enthält alle auf Ihrem Rechner verfügbaren Type Libraries. Naturgemäß variiert der Inhalt der Liste, je nachdem welche Anwendungen installiert wurden. Detaillierte Informationen zu einer bestimmten Komponente erhalten Sie nach einem Doppelklick auf den entsprechenden Eintrag. Abbildung 9.2 zeigt beispielsweise das Interface `IShellDispatch4` von *shell32.dll* und zwei seiner Funktionen: `ToggleDesktop()` und `WindowsSecurity()`.

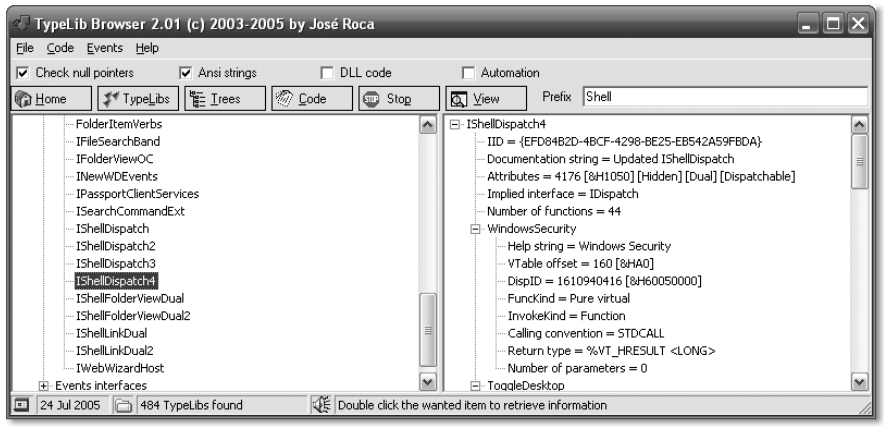

Abbildung 9.2 Detaillierte Informationen zu shell32.dll

Welche Informationen lassen sich der in Abbildung 9.2 gezeigten Detailansicht entnehmen? Der Knoten **Dual interfaces** enthält eine Liste der für uns relevanten Interfaces, die eine Komponente implementiert. Nach einem Doppelklick auf einen Interface-Namen erscheinen im rechten Fensterbereich alle Funktionen des Interfaces. Ich werde noch ausführlicher darauf eingehen. Der Knoten **Documentation** enthält allgemeine Informationen, beispielsweise die Versionsnummer sowie eine kurze Beschreibung, zum Beispiel *Microsoft Shell Controls And Automation*. **Class identifiers** und **Interface identifiers** enthalten die bereits angesprochen *Globally Unique Identifiers* (GUIDs), die Komponenten und Interfaces eindeutig kennzeichnen. Um die Funktionen eines Interfaces in eigenen Programmen nutzen zu können, müssen Sie natürlich wissen, welche Parameter eine Funktion erwartet und welches Ergebnis sie liefert. In Abbildung 9.3 sehen Sie, wie der *TypeLib Browser* die Funktion `ToggleDesktop()` darstellt.

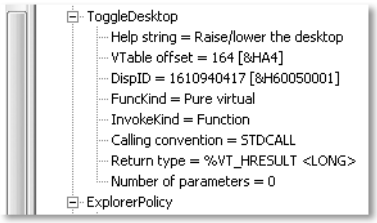

Abbildung 9.3 Die Funktion ToggleDesktop

Die Angabe `Number of parameters = 0` zeigt uns an, dass die Funktion keine Parameter erwartet. In Verbindung mit der Beschreibung `Raise/lower the desktop` lässt sich die Funktionsweise gut abschätzen: Ein Aufruf der Funktion blendet alle Fenster aus, ein erneuter Aufruf stellt den ursprünglichen Zustand wieder her. In Abschnitt 9.2.1 werde ich Ihnen das Programm *ToggleDesktop-Demo* vorstellen, das `ToggleDesktop()` nutzt.

Ziel dieser Einführung war, Ihnen ein grundlegendes Verständnis des Component Object Models zu vermitteln. In den folgenden Abschnitten werden wir auf diesem Wissen aufbauen und uns mit Java-COM-Brücken beschäftigen. Sie bilden das Java-Objektmodell auf COM ab und machen COM-Objekte für Java-Programme nutzbar.

9.2 JACOB

JACOB ist eine Java-COM-Brücke, die seit 1999 von Dan Adler entwickelt wird. Die Klassenbibliothek orientiert sich in ihrem Aufbau stark am Paket `com.ms.com`, das ein wichtiger Teil von Microsofts Java-Implementierung ist.

Die derzeit aktuelle Version 1.9 der kostenlosen Klassenbibliothek kann von der Projekt-Homepage[3] heruntergeladen werden. Unter **http://danadler.com/jacob/** findet sich noch eine ältere Fassung der JACOB-Website, auf der Dan Adler zahlreiche Tipps zur Verwendung gibt. Allerdings bezieht sich diese Seite auf zurückliegende Programmversionen. Beispielsweise muss die dort gezeigte Excel-Demo vor der Verwendung mit der aktuellen Version von JACOB leicht angepasst werden. Die Installation der Klassenbibliothek gestaltet sich äußerst einfach. Nach dem Download und dem Entpacken des Archivs müssen Sie dafür sorgen, dass die Java-Laufzeitumgebung die beiden Dateien *jacob.jar* und *jacob.dll* findet. Wie Sie hierzu vorgehen können, zeige ich Ihnen in Kapitel 1, *Installation und Konfiguration*.

9.2.1 Ausblenden und Anzeigen von Fenstern

In Abschnitt 9.1.2 wurde gezeigt, wie Sie mit Hilfe des *TypeLib Browsers* Informationen zu den Funktionen von COM-Interfaces erhalten können. Jetzt werden wir die in Abbildung 9.3 zu sehende Funktion `ToggleDesktop()` in ein kleines Java-Programm einbetten. Sehen Sie sich bitte zunächst den Quelltext des Programms *ToggleDesktopDemo* an.

```
package javafuerwindows.jacob;
import com.jacob.activeX.ActiveXComponent;
import javax.swing.JOptionPane;
public class ToggleDesktopDemo {
    public ToggleDesktopDemo() {
        ActiveXComponent shellAXC = new ActiveXComponent("
                                        Shell.Application");
        shellAXC.invoke("ToggleDesktop");
        JOptionPane.showMessageDialog(null,
                    "gleich erscheinen die Fenster wieder",
                    "ToggleDesktop",
                    JOptionPane.INFORMATION_MESSAGE);
        shellAXC.invoke("ToggleDesktop");
    }

    public static void main(String [] args) {
        new ToggleDesktopDemo();
    }
}
```

Listing 9.1 ToggleDesktopDemo.java

3 http://sourceforge.net/projects/jacob-project/

Wie Sie sehen, ist das Programm sehr kurz. Zunächst wird eine Instanz der Klasse `com.jacob.activeX.ActiveXComponent` erzeugt. Als Parameter wird dem Konstruktor ein so genannter *Program identifier* übergeben, in unserem Beispiel `Shell.Application`. Der *TypeLib Browser* zeigt eine Liste dieser *ProgIDs* in seiner Detailansicht im Knoten **Version independent ProgIDs**. Der Aufruf der Funktion `ToggleDesktop()` erfolgt indirekt durch den Aufruf der Methode `invoke()`, einer Instanz von `ActiveXComponent`, der als Parameter der Name der aufzurufenden Interface-Funktion übergeben wird. *JACOB* basiert also auf der in Abschnitt 9.1.1 vorgestellten Kommunikationsvariante via `IDispatch`-Interface.

Selbstverständlich können auf diese Weise nicht nur parameterlose Funktionen aufgerufen werden. Wie Sie Werte an andere COM-Objekte übergeben und die Ergebnisse der Funktionsaufrufe auswerten, erläutern die Abschnitte 9.2.2 und 9.2.3.

9.2.2 Sounds abspielen

Anhand des Beispiels *VoiceCtl* möchte ich Ihnen zeigen, wie Sie Sounds abspielen können, die als *.wav* Dateien vorliegen. Wir verwenden hierfür das Interface `IVoicePlay`, das in der Datei *vcomctl.dll* definiert wird. Sie befindet sich im Verzeichnis *C:\Programme\Common Files\Microsoft Shared\NoteSync Forms* und wird zusammen mit *Microsoft ActiveSync*[4] installiert.

Sehen Sie sich zunächst bitte den Quelltext des Programms an.

```
package javafuerwindows.jacob;
import com.jacob.activeX.ActiveXComponent;
import com.jacob.com.Dispatch;
import javax.swing.JOptionPane;
public class VoiceCtl {
    public static void main(String [] args) {
        ActiveXComponent voiceplay = new ActiveXComponent(
                                        "VoiceCtl.VoicePlay");
        Dispatch.call(voiceplay,
                      "OpenFile",
                      "C:\\windows\\media\\chimes.wav");
        Dispatch.call(voiceplay, "Play");
        JOptionPane.showMessageDialog(null,
                                  "Abspielvorgang läuft",
```

4 http://www.microsoft.com/downloads/details.aspx?FamilyID=d2645c21–8a85–45a2–8d13–653beb6cdddc&DisplayLang=en

```
                                        "Bitte warten",
                                        JOptionPane.PLAIN_MESSAGE);
    }
}
```

Listing 9.2 VoiceCtl.java

Nach der Instantiierung eines Objekts der Klasse `ActiveXComponent` wird der Funktion `OpenFile()` der Dateiname *C:\windows\media\chimes.wav* übergeben. Die Ausführung erfolgt wie gewohnt indirekt, nämlich durch den Aufruf der Klassenethode `call()` der Klasse `Dispatch`. Der eigentliche Abspielvorgang wird durch die Funktion `Play()` gestartet.

> Das Anzeigen eines Dialogs mittels `JOptionPane` verhindert übrigens die vorzeitige Beendigung des Programms. Dies ist erforderlich, da sonst der Sound nicht vollständig abgespielt würde.

Bitte beachten Sie, dass der Dateiname als `String` übergeben wird. Andere Datentypen müssen vor der Übergabe an eine Funktion in eine Art »Container« verpackt werden. Hierbei handelt es sich um die Klasse `com.jacob.com.Variant`, die wir im folgenden Beispiel sehr häufig verwenden werden.

9.2.3 iTunes fernsteuern

Die Anwendung *iTunes* bildet nicht nur das Portal zu Apples Online-Musikladen, sondern realisiert auch die Anbindung der portablen Abspielgeräte *iPod* an den PC und beinhaltet eine ausgereifte Medienverwaltung sowie Dienste zum Aufnehmen und Abspielen von Musikstücken. *iTunes* stellt eine sehr umfangreiche COM-Schnittstelle zur Verfügung, die nahezu alle Bereiche der Anwendung von außen zugänglich macht. Die *Apple Developer Connection* bietet das *iTunes COM for Windows SDK*[5] zum kostenlosen Download an. Das Archiv beinhaltet unter anderem einige Beispiele in *JScript* sowie eine ausführliche Dokumentation der Interfaces und Funktionen.

Wie Sie diese über das `IDispatch`-Interface ansprechen können, zeigt mein Beispielprogramm *ITunesDemo*, dessen Bildschirmausgabe in Abbildung 9.4 zu sehen ist. Es ermittelt alle *Quellen*, die *Abspiellisten* enthalten können und gibt den Inhalt der ersten *Playlist* aus. Quellen sind beispielsweise CDs, Geräte sowie die ITunes-Bibliothek.

5 http://developer.apple.com/sdk/itunescomsdk.html

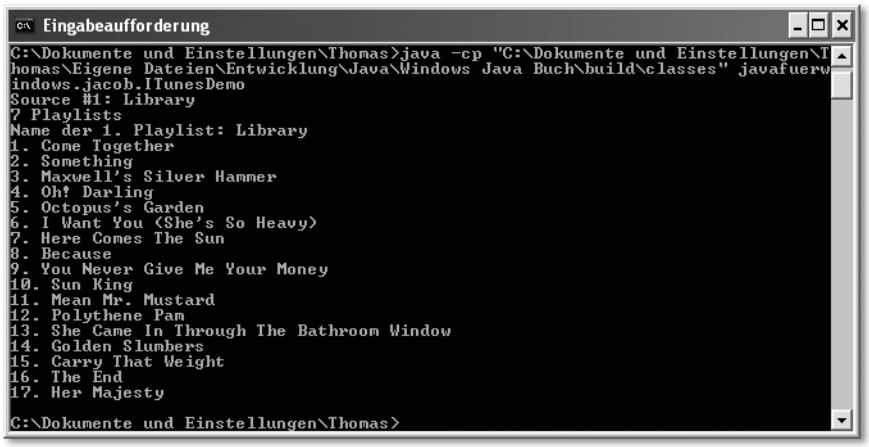

Abbildung 9.4 Ausgabe des Programms ITunesDemo

Unten sehen Sie interessante Auszüge aus dem Programm. Der vollständige Quelltext befindet sich auf der Begleit-CD.

```
ActiveXComponent iTunes = new ActiveXComponent(
                                "iTunes.Application");
/*
 * liefert das Interface IITSourceCollection
 */
Dispatch sources = Dispatch.call(iTunes,
                                 "Sources").toDispatch();
Variant count = Dispatch.call(sources, "Count");
int anzahlSources = count.getInt();
```

Die Anzahl der *iTunes* bekannten Quellen wird mit Hilfe der Funktion `Count()` des Interfaces `IITSourceCollection` ermittelt. Einen Verweis auf dieses Interface liefert `Sources()` des iTunes-Basisinterfaces. Sie erhalten den Wert durch Aufruf von `Dispatch.call()`. Allerdings erzeugt `call()` Objekte des Typs `com.jacob.com.Variant`. Um eine Funktion des gelieferten `IITSourceCollection`-Interfaces benutzen zu können, muss es als `Dispatch`-Objekt vorliegen. Dies lässt sich mit Hilfe der Methode `toDispatch()` der Klasse `Variant` erzeugen.

```
Variant name;
for (int i = 1; i <= anzahlSources; i++) {
/*
 * IITSource
 */
```

```
 Dispatch source = Dispatch.call(sources,
                                 "Item",
                                 new Variant(i)).toDispatch();
name = Dispatch.call(source, "Name");
System.out.println("Source #" + i + ": " +
                   name.getString());
```

Die Vorgehensweise beim Aufruf einer Funktion ist stets die gleiche. `Dispatch.call()` erhält als ersten Parameter ein `Dispatch`-Objekt, das auf das Interface mit der anzusprechenden Funktion verweist. Der zweite Wert enthält den Funktionsnamen als `String`. Alle weiteren Parameter werden als Argumente an die aufzurufende Funktion übergeben. Wenn es sich bei dem zu verwendenden Interface um das Basisinterface handelt, übergeben Sie an `call` die Instanz Ihrer `com.jacob.activeX.ActiveXComponent`-Klasse. Andernfalls muss das Interface das Ergebnis eines Funktionsaufrufs gewesen sein. Da `call com.jacob.com.Variant-Objekte` liefert, müssen Sie die Methode `toDispatch` aufrufen.

```
/*
 * IITPlaylistCollection
 */
Dispatch playlists = Dispatch.call(source,
                                   "Playlists").toDispatch();
count = Dispatch.call(playlists, "Count");
anzahlPlaylists = count.getInt();
```

`com.jacob.com.Variant` bietet nicht nur die Möglichkeit, `Dispatch`-Objekte zu erzeugen, sondern enthält auch einige Methoden, die den gespeicherten Wert auf Java-Datentypen abbilden.

```
System.out.println(anzahlPlaylists + " Playlists");
/*
 * IITPlaylist
 */
if (anzahlPlaylists >= 1) {
    Dispatch playlist = Dispatch.call(playlists,
                                  "Item",
                                  new Variant(1)).toDispatch();
```

Wenn an Funktionen Werte übergeben werden, geschieht dies entweder als `String` oder in Form einer Instanz von `Variant`. In diesem Beispiel erwartet

`Item()` eine Zahl, die eine Playlist referenziert. Da `int` nicht direkt übergeben werden kann, wird eine Instanz von `Variant` erzeugt, die die Zahl enthält.

```
name = Dispatch.call(playlist, "Name");
System.out.println("Name der 1. Playlist: " +
                   name.getString());
/*
 * IITTrackCollection
 */
Dispatch tracks = Dispatch.call(playlist,
                                "Tracks").toDispatch();
count = Dispatch.call(tracks, "Count");
anzahlTracks = count.getInt();
for (int j = 1; j <= anzahlTracks; j++) {
    /*
     * IITTrack
     */
     Dispatch track = Dispatch.call(tracks,
                                    "Item",
                                new Variant(j)).toDispatch();
    name = Dispatch.call(track, "Name");
    System.out.println(j + ". " + name);
}
}
}
/*
 * iTunes beenden
 */
Dispatch.call(iTunes, "Quit");
}
```

Vor dem Programmende sollten Sie die Funktion `Quit()` des *iTunes*-Basisinterfaces aufrufen. Andernfalls bleibt *iTunes* weiterhin geöffnet. Versucht der Anwender, das Programm zu schließen, erhält er unter Umständen die in Abbildung 9.5 gezeigte Meldung. Mit dieser Rückfrage stellt *iTunes* sicher, dass der Benutzer nicht unbewusst die Kommunikation mit einem anderen Programm unterbricht.

Abbildung 9.5 iTunes-Sicherheitsabfrage

9.2.4 Rechtschreibprüfung mit Word

Microsoft hat in seinen aktuellen Windows-Versionen keine systemweite Rechtschreibprüfung integriert, wie dies unter Apples Betriebssystem Mac OS X der Fall ist. Einige der mit Windows gelieferten Anwendungen, beispielsweise *Outlook Express*, bieten aber eine Korrektur der Rechtschreibung an. Sie greifen hierzu auf Prüfroutinen zu, die Word und aktuelle Versionen der Works-Suite bereitstellen.

Auch Ihre Java-Anwendungen können die Rechtschreibprüfung von Word nutzen. Die grundsätzliche Vorgehensweise demonstriert Microsoft im Artikel *How To Automate Word From Visual Basic or Visual Basic for Applications For Spell Checking*[6]. Grundidee ist, in ein leeres Word-Dokument den zu prüfenden Text zu übernehmen, beispielsweise vom Klemmbrett. Nach dem Aufruf der eigentlichen Rechtschreibprüfung wird der korrigierte Text auf dem Klemmbrett abgelegt und kann von dort aus weiterverarbeitet werden. Betrachten Sie zunächst bitte den Quelltext meines Programms *SpellCheck*. Er zeigt, wie Sie die hierfür notwendigen Schritte in Java realisieren.

```
package javafuerwindows.jacob;
import com.jacob.activeX.ActiveXComponent;
import com.jacob.com.Dispatch;
import com.jacob.com.Variant;

public class SpellCheck {
    public static void main(String [] args) {
        ActiveXComponent word = new ActiveXComponent(
                                    "Word.Application");
        Dispatch.put(word, "Visible", new Variant(true));
        Dispatch.put(word, "WindowState", new Variant(0));
        Dispatch.put(word, "Top", new Variant(-3000));
        Dispatch documents = Dispatch.call(word,
```

6 http://support.microsoft.com/kb/243844/en-us

```
                                    "Documents").toDispatch();
        Dispatch doc = Dispatch.call(documents,
                                        "Add").toDispatch();
        Dispatch content = Dispatch.call(doc,
                                     "Content").toDispatch();
        Dispatch.call(content, "Paste");
        Dispatch.call(doc, "Activate");
        Dispatch.call(doc, "CheckSpelling");
        Dispatch.call(content, "Copy");
        Dispatch.put(doc, "Saved", new Variant(true));
        Dispatch.call(doc, "Close");
        Dispatch.call(word, "Quit");
    }
}
```

Listing 9.3 SpellCheck.java

Die Funktionen `put()` manipulieren einige Eigenschaften des Word-COM-Objekts. Es wird sichergestellt, dass das Hauptfenster zwar geöffnet, aber außerhalb des sichtbaren Bildschirmbereichs positioniert wird. Anschließend legt *SpellCheck* ein neues, leeres Dokument an und fügt den Inhalt des Klemmbretts ein. Kern des Programms ist der Aufruf der Funktion `CheckSpelling()`. Im Anschluss daran wird der ggf. veränderte Dokumentinhalt auf dem Klemmbrett abgelegt und das Dokument geschlossen.

Um SpellCheck zu testen, müssen Sie zunächst einen zu korrigierenden Text erfassen und auf das Klemmbrett packen. Anschließend starten Sie bitte *SpellCheck*.

Abbildung 9.6 Der zu korrigierende Text

In Abbildung 9.7 sehen Sie ein von Word moniertes Wort sowie die Korrekturvorschläge. Nachdem Sie alle falsch geschriebenen Wörter korrigiert haben, können Sie die fehlerbereinigte Fassung Ihres Testtextes vom Klemmbrett nehmen, um ihn weiter zu bearbeiten.

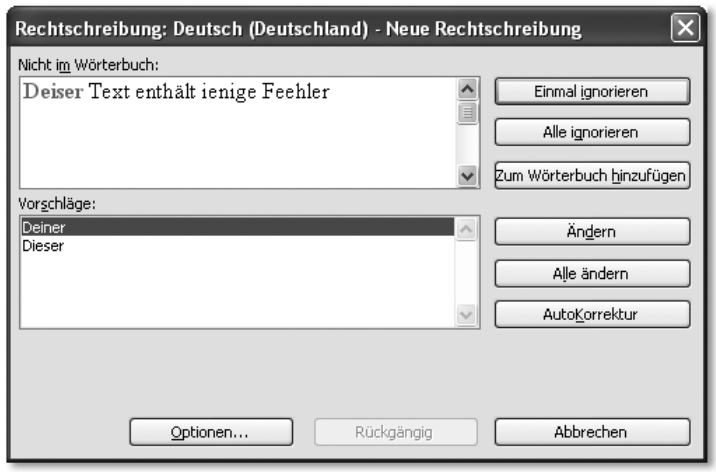

Abbildung 9.7 Word erfragt das gemeinte Wort

Sie haben mit *JACOB* eine Java-COM-Brücke kennen gelernt, die auf recht einfache Weise die Kommunikation mit COM-Objekten ermöglicht. Mit Hilfe des *TypeLib Browsers* als eigenständige und von *JACOB* völlig unabhängige Anwendung können Sie Typbibliotheken auslesen, die den Funktionsumfang einer Komponente beschreiben. Dies ist vor allem dann äußerst nützlich, wenn keine Dokumentation der implementierten Funktionen verfügbar ist. Allerdings führt die Verwendung des `IDispatch`-Interfaces zu umfangreichen und vergleichsweise schwer verständlichen Quelltexten. Zudem müssen häufig Container-Objekte erzeugt werden, die die Argumente der aufzurufenden Funktionen enthalten.

Im folgenden Abschnitt möchte ich Ihnen deshalb eine weitere Java-COM-Brücke als Alternative vorstellen, die einen gänzlich anderen Ansatz mit weniger »Ballast« verfolgt.

9.3 com4j

Die Klassenbibliothek *com4j* von Kohsuke Kawaguchi realisiert wie *JACOB* eine Schnittstelle zu Microsofts Component Object Model. Sie setzt, wie Sie im Folgenden noch sehen werden, konsequent auf *Annotationen*, die Sun mit Java SE 5 eingeführt hat. *com4j* ist kostenlos und kann von der Projekt-Homepage[7] bezogen werden. Die zum Zeitpunkt der Drucklegung aktuelle Version finden Sie auch auf der Begleit-CD. Die Installation ist sehr einfach. Damit die gelieferten Klassen sowie DLLs von der Java-Laufzeitumgebung gefunden werden,

7 https://com4j.dev.java.net/

sollten Sie alle *.jar*- und *.dll*-Dateien wie in Kapitel 1, *Installation und Konfiguration*, beschrieben in das Erweiterungsverzeichnis der von Ihnen verwendeten Laufzeitumgebung kopieren. Bitte denken Sie auch daran, den Pfad der Dokumentation in Ihrer Entwicklungsumgebung einzutragen, damit Sie bei Bedarf Parameter nachschlagen können.

9.3.1 Sprachsynthese mit der Microsoft Speech API

Kohsuke Kawaguchi liefert zu *com4j* einige Beispiele, um die Funktionsweise seiner Klassenbibliothek zu demonstrieren. Im Verzeichnis *samples\speech\src\speech* findet sich die Klasse `Main`, die die *Microsoft Speech API* verwendet, um die Präambel der US-amerikanischen Verfassung durch den Computer sprechen zu lassen. Wenn Sie einen Blick auf den Quelltext werfen, werden Sie feststellen, dass hierzu auf Klassen des Pakets `speech` zugegriffen wird. Es enthält die Interfaces des durch die DLL bereitgestellten COM-Objektes. Bevor Sie also *Main.java* übersetzen können, müssen Sie das Paket generieren. *com4j* enthält das hierfür benötigte Tool. Sie finden es in der Datei *tlbimp.jar*.

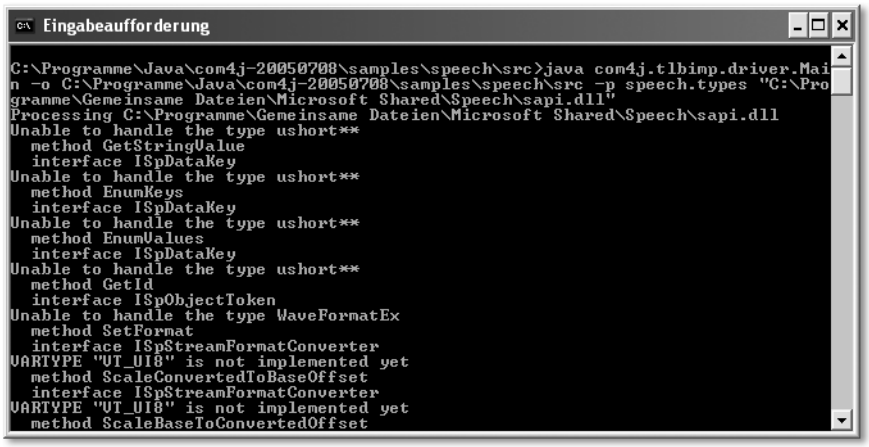

Abbildung 9.8 Erzeugen der Klassendefinitionen von sapi.dll mit tlbimp

Wenn Sie diese in das Java-Erweiterungsverzeichnis kopiert haben, können Sie das Programm wie in Abbildung 9.8 zu sehen ist aufrufen. Alternativ ist natürlich auch der Start mittels `java -jar tlbimp.jar ...` möglich. An *tlbimp* werden drei Parametern übergeben. `-o` spezifiziert das Basisverzeichnis für Quelltexte. Eine Klasse `Xyz`, die keinem Paket zugeordnet wurde, wird hier abgelegt. Der zweite Parameter `-p` gibt den Namen des Pakets an, dem die generierten Klassen zugeordnet werden sollen. In unserem Fall ist dies `speech.types`. Die Unterverzeichnisse für Pakete werden relativ zum Quell-

text-Basisverzeichnis angelegt. Schließlich wird noch der Name der Typbibliothek, des ausführbaren Programms oder der DLL erwartet. Die während des Generierens erzeugten Meldungen, dass bestimmte Typen nicht behandelt werden können, ignorieren Sie bitte. Anschließend können Sie `speech.Main` sowie die Klassen des Pakets `speech.types` durch die Eingabe von `javac speech/Main.java` übersetzen. Der Programmstart erfolgt mittels `java speech.Main`. Denken Sie bitte daran, die Lautstärke auf einen passenden Wert einzustellen, damit Sie die durch den Computer synthetisierten Worte gut verstehen. Nun werfen wir einen Blick auf den Quelltext von *Main.java*, um zu sehen, wie die Sprachausgabe realisiert wird.

```
package speech;
import speech.types.ClassFactory;
import speech.types.ISpeechVoice;
import speech.types.SpeechVoiceSpeakFlags;

/**
 * An example that uses the Microsoft Speech API.
 *
 * @author Kohsuke Kawaguchi
 */
public class Main {
    public static void main(String[] args)
                        throws Exception {
        ISpeechVoice v = ClassFactory.createSpVoice();
        System.out.println("Make sure your
                            speaker is not off...");
        v.speak("We the People of the
                 United States, in Order to " +
                "form a more perfect Union,
                 establish Justice, " +
                "insure domestic Tranquility,
                 provide for the " +
                "common defence, promote
                 the general Welfare, " +
                "and secure the Blessings
                 of Liberty to ourselves " +
                "and our Posterity, do
                 ordain and establish this " +
                "Constitution for
                 the United States of America.",
```

```
                SpeechVoiceSpeakFlags.SVSFDefault);
        }
}
```

Listing 9.4 speech/Main.java von Kohsuke Kawaguchi

Die Sprachsynthese und anschließende Ausgabe lässt sich in zwei Schritten bewerkstelligen. Zunächst wird eine Instanz der Klasse `ISpeechVoice` erzeugt. Danach wird deren Methode `speak()` aufgerufen.

Selbstverständlich bietet die *Microsoft Speech API* weitaus mehr Möglichkeiten. Ausführliche Informationen hierzu finden Sie auf den Seiten des *Microsoft Developer Networks*.[8]

9.3.2 Microsoft OneNote-Importer

Das Microsoft-Programm *OneNote* hilft unter anderem beim Sammeln und Verwalten von Notizen, Ideen und Skizzen. Der Office-Bestandteil macht aber nicht nur als Vorzeige-Anwendung auf dem Tablet-PC eine recht gute Figur, sondern auch in Verbindung mit herkömmlichen PCs oder Notebooks. *OneNote* ist gerade für den Software-Entwickler ein geeignetes Werkzeug, wenn es um das Ordnen von URLs, Tipps und Tricks sowie Quelltext-Schnipseln geht. Microsoft nennt denn auch das Bündeln von Informationen aus verschiedenen Quellen als eine der Stärken der Anwendung. Konsequenterweise müsste es programmtechnisch sehr einfach sein, einem OneNote-Dokument neue Seiten nebst Inhalt hinzuzufügen. In der Tat gibt es mittlerweile einige »Power Toys«[9], die die Übernahme von Informationen aus Outlook und dem Internet Explorer ermöglichen. Allerdings bietet OneNote nicht wie seine Office-Kollegen eine umfangreiche Sammlung von COM-Objekten. Ein Fernsteuern der Anwendung, wie dies mit Outlook, Excel oder Word möglich ist, lässt sich mit OneNote bisher nicht realisieren. Microsoft bietet erst mit dem Service Pack 1 rudimentäre COM-Unterstützung. Aus diesem Grund funktionieren die genannten Power Toys auch nur mit der nachgebesserten Version der Anwendung. Immerhin lassen sich nun programmtechnisch Seiten und Inhalte in ein bestehendes Dokument übernehmen. Wie dies realisiert wird, zeige ich Ihnen gleich. Der erste Schritt ist wie in Abschnitt 9.3.1 das Erzeugen der Java-Klassen. Der Aufruf des Tools *tlbimp* könnte folgendermaßen aussehen:

```
java com4j.tlbimp.driver.Main -o "C:\Dokumente und Einstellungen\Thomas\Eigene Dateien\Entwicklung\Java\Windows Java
```

8 http://msdn.microsoft.com/library/default.asp?url=/library/en-us/SAPI51sr/html/Getting_Started.asp

9 http://office.microsoft.com/en-us/assistance/HA011408961033.aspx

```
Buch\src" -p javafuerwindows.com4j.onenote "C:\Programme\Micro-
soft Office\OFFICE11\ONENOTE.EXE"
```

Als Paketnamen habe ich `javafuerwindows.com4j.onenote` vergeben. Die generierten Klassen finden sich somit in einem eigenen Unterverzeichnis, was die Trennung von eigenhändig erstelltem Quelltext erleichtert. Dies ist vor allem dann ratsam, wenn eine Anwendung viele COM-Objekte und Interfaces anbietet. Im Falle von *OneNote* werden nur zwei Klassen generiert: `ClassFactory` und `ISimpleImporter`. Erstere ist Ihnen schon bekannt. Wir werden sie verwenden, um eine Instanz der eigentlich interessanten Klasse `ISimpleImporter` zu erzeugen. Wenn Sie einen Blick auf ihren Quelltext werfen, werden Sie feststellen, dass lediglich zwei Methoden deklariert werden, und zwar `navigateToPage()` und `_import()`. Wie sollte sich auf diese Weise beliebiger Inhalt zu OneNote-Dokumenten hinzufügen lassen? Die Antwort hierauf mag verblüffen. Die zu importierenden Daten werden als XML-Datenstrom kodiert und an `_import()` übergeben. Sehen Sie sich hierzu bitte den Quelltext des Programms *OneNoteDemo* an.

```
package javafuerwindows.com4j;
import java.util.UUID;
import javafuerwindows.com4j.onenote.ClassFactory;
import javafuerwindows.com4j.onenote.ISimpleImporter;

public class OneNoteDemo {
    public static void main(String [] args) {
        ISimpleImporter importer =
                        ClassFactory.createCSimpleImporter();
        String seite = erzeugeGUID();
        String xml = "<?xml version=\"1.0\"?>";
        xml += "<Import
                xmlns=\"http://schemas.microsoft.com/office/
                onenote/2004/import\">";
        xml += "<EnsurePage path=\"Test.one\"
                 guid=\"" +
                 seite + "\"
                 title=\"Testseite\"/>";
        xml += "<PlaceObjects
                 pagePath=\"Test.one\"
                 pageGuid=\"" +
                 seite + "\">";
        xml += "<Object guid=\"" + erzeugeGUID() + "\">";
```

```
        xml += "<Position x=\"50\" y=\"100\"/>";
        xml += "<Outline width=\"400\">";
        xml += "<Html>";
        xml += "<Data>";
        xml += "<![CDATA[";
        xml += "<html><body>
                <p>Hier steht ein beliebiger Text
                </p></body></html>";
        xml += "]]>";
        xml += "</Data>";
        xml += "</Html>";
        xml += "</Outline>";
        xml += "</Object>";
        xml += "</PlaceObjects>";
        xml += "</Import>";
        importer._import(xml);
    }

    public static String erzeugeGUID() {
        UUID guid = UUID.randomUUID();
        return "{" + guid.toString() + "}";
    }
}
```

Listing 9.5 OneNoteDemo.java

Der größte Teil des Programms besteht aus dem Zusammenstellen eines `String`-Objektes, das den vollständigen an *OneNote* zu übergebenden XML-Strom enthält. In Abbildung 9.9 sind die verwendeten Tags und Attribute zu sehen. Microsoft hat den englischsprachigen Artikel *Importing Content into OneNote 2003 SP1* veröffentlicht,[10] der sich ausführlich mit dem Datenimport beschäftigt. Grundsätzlich werden alle Import-Anweisungen mit `<Import>` und `</Import>` geklammert. Unmittelbar unterhalb der Wurzel sind nur die beiden Elemente `<EnsurePage>` und `<PlaceObjects>` definiert. Ersteres stellt sicher, dass eine Kombination aus Ordner, Abschnitt und Seite verfügbar ist. Mittels `<PlaceObjects>` werden Seiten dann Inhalte hinzugefügt oder gelöscht. Um Seiten eindeutig zu kennzeichnen, verwendet Microsoft in beiden Elementen die Ihnen bereits bekannten Globally Unique Identifiers. Da Sun seit Java 5 die Klasse `java.util.UUID` zur Verfügung stellt, ist das Erzeugen einer solchen ID

10 http://msdn.microsoft.com/library/default.asp?url=/library/en-us/odc_on2003_ta/html/odc_on_importapi.asp

sehr einfach. Die eigentlichen Inhalte werden in `<Object>`-Elementen abgelegt, die unterhalb von `<PlaceObjects>` angesiedelt sind. Auch sie werden durch einen GUID gekennzeichnet. Position, Größe und Inhalt eines Objekts werden in eigenen Elementen abgelegt, beispielsweise `<Position>` und `<Outline>`. Eine vollständige Aufstellung aller angebotenen Elemente liefert Microsoft auf der MSDN-Seite *OneNote SimpleImport Schema Structure*[11]. Beim Generieren des XML-Stroms müssen Sie sehr sorgfältig vorgehen. Der OneNote-Importer ist nicht fehlertolerant, sondern bricht den Importvorgang unmittelbar nach Entdecken eines Fehlers ab. Dies betrifft insbesondere nicht vorhandene GUIDs, aber auch Fehler in den Elementen. Eine Ausnahme bildet hier nur `<EnsurePage>`. Dieses Element stellt sicher, dass eine Seite zur Verfügung steht. Falls Sie also eine unbekannte ID angeben, wird die Seite neu erstellt. Da *OneNoteDemo* für die Erzeugung eines Globally Unique Identifiers die Methode `UUID.randomUUID` aufruft, wird bei jedem Programmstart eine neue Seite erzeugt.

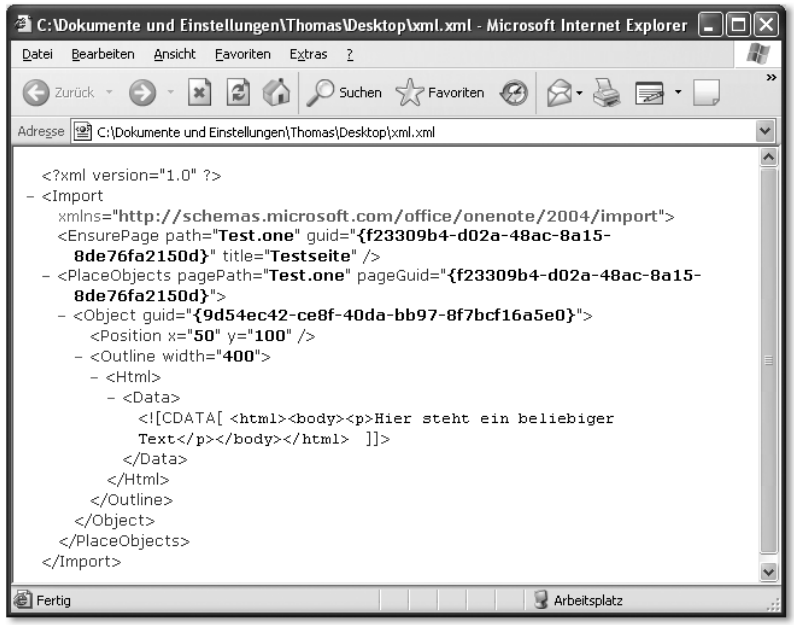

Abbildung 9.9 Der durch OneNoteDemo generierte XML-Datenstrom

In Abbildung 9.10 sehen Sie *OneNote* nach dem Start von *OneNoteDemo*.

11 http://msdn.microsoft.com/library/?url=/library/enus/ONXML/html/oconSchemaStructure_HV01140478.asp

Abbildung 9.10 OneNote nach dem Start von OneNoteDemo

Im Vergleich zu *JACOB* wirkt der Quelltext von Programmen, die *com4j* einsetzen, aufgeräumter und leichter verständlich. Allerdings ist hierbei zu berücksichtigen, dass *com4j* intensiv Gebrauch von Annotationen macht und deshalb nur in Verbindung mit Java 5 eingesetzt werden kann. Zudem kann *com4j* in der aktuellen Version nicht alle Typbibliotheken vollständig in Java-Klassen umwandeln.

10 Deployment

10 Deployment

Selbst die schönste Anwendung sorgt für Frust, wenn der Weg zum ersten Start allzu steinig ist. Deshalb zeigt dieses Kapitel, welche Möglichkeiten der Softwareverteilung es gibt und wie Sie Installationswerkzeuge effizient einsetzen

Solange Sie ein Programm ausschließlich für den eigenen Gebrauch schreiben, mag es unnötig erscheinen, sich Gedanken über seine Installation oder die Art seiner Verteilung zu machen. Schließlich wissen Sie, in welchem Verzeichnis Sie das Programm abgelegt haben und welche Aufrufparameter es erwartet. Sobald aber Ihre Anwendung auch von anderen Personen genutzt wird, stellt sich die Frage, in welcher Form Sie die Parameter weitergeben, welche Schritte vor dem ersten Start nötig sind und wie das Programm aufgerufen wird. Beispielsweise sind Windows-Benutzer gewohnt, eine Anwendung durch Start von *setup.exe* oder Doppelklick auf eine *.msi*-Installationsdatei einzurichten. Ferner gehört es zum guten Ton, dass ein Programm über das Start-Menü aufgerufen werden kann. Zudem sollte es über einen Eintrag im Modul *Software* der *Systemsteuerung* verfügen, über den sich beispielsweise Änderungen an der Installation durchführen lassen. Bei solchen Aufgaben helfen Installationswerkzeuge, die ich Ihnen in Abschnitt 10.1 vorstelle. Java bietet mit *Web Start* eine eigene Methode der Softwareverteilung, die ohne Installationsroutinen auskommt. Der Anwender ruft einmalig einen Link auf einer Webseite auf. Um alle weiteren Schritte kümmert sich dann die Laufzeitumgebung. Ausführliche Informationen hierzu sind in Abschnitt 10.2 zusammengestellt. In Abschnitt 10.3 schließlich präsentiere ich Tools, die unabhängig von der gewählten Form der Verteilung für eine nahtlose Einbindung in Windows sorgen, indem sie beispielsweise Java-Programme in *.exe*-Dateien kapseln und Ihnen helfen, *Verknüpfungen* anzulegen.

10.1 Installationswerkzeuge

Zu einer Anwendung gehören meistens viele Dateien. Neben dem eigentlichen Programm zählen hierzu Hilfetexte, Programmdokumentationen, Lizenzbestimmungen und Beispiele oder Vorlagen. Um diese Dateien auf unkomplizierte Weise verteilen zu können, haben sich *Installationswerkzeuge* etabliert. Sie fassen die Komponenten einer Anwendung in einem Archiv zusammen, das sich leicht handhaben und kopieren lässt. Um das Entpacken und Durchführen der eigentlichen Installation kümmert sich später eine Routine, die entweder Teil der weitergegebenen Datei oder in das Betriebssystem integriert ist. Diese

Komponente erledigt zudem Aufgaben, die über das bloße Kopieren der Dateien in ein Zielverzeichnis hinausgehen, beispielsweise das Registrieren von dynamisch gelinkten Bibliotheken oder das Schreiben von Schlüsseln in die Windows-Registrierung. Fester Bestandteil aller modernen Windows-Versionen ist der *Windows Installer*[1], der mit *.msi*-Dateien arbeitet. Hierbei handelt es sich vereinfacht ausgedrückt um Datenbanken, die Informationen über zu installierende Komponenten (sowie natürlich die Dateien selbst) enthalten. Eine weitere Programmgattung zur Erzeugung von Installationsdateien basiert auf Java und fügt die zu installierenden Dateien zu *.jar*-Archiven zusammen. Da diese durch Doppelklick gestartet werden können, verhalten sich die Installationsdateien so wie ihre nativen Pendants. Im Folgenden stelle ich Ihnen Vertreter beider Varianten vor und mache Sie mit Grundlagen ihrer Bedienung vertraut.

10.1.1 WiX

Windows Installer XML (WiX)[2] ist eine Sammlung von Tools, die das Erzeugen von *.msi*-Installationsdateien aus XML-Quelltexten erlauben. *WiX* ist Open Source und wird von Microsoft unter der *Common Public Licence* zur Verfügung gestellt. Das Produkt kann also kostenlos von der Projekt-Homepage heruntergeladen[3] und benutzt werden. Sie können das Archiv *binaries-2.0.3309.0.zip* in einem beliebigen Verzeichnis entpacken, eine gesonderte Installation ist nicht erforderlich. Da die Bestandteile von WiX zumeist aus der *Eingabeaufforderung* heraus gestartet werden, sollten Sie dieses Basisverzeichnis allerdings in die Umgebungsvariable `PATH` eintragen, damit Windows WiX auch ohne die Angabe von absoluten Pfaden findet. Bitte beachten Sie, dass WiX auf dem *.NET Framework* basiert. Deshalb müssen Sie dessen Version 1.1 einschließlich Service Packs installiert haben, beispielsweise über *Windows Update*.

Im Folgenden wird die Funktionsweise von WiX in Grundzügen erläutert. Ausführliche Informationen finden Sie in der englischsprachigen Programmdokumentation *WiX.chm* im Verzeichnis *doc*.

WiX dient dazu, Installationsdateien zu erzeugen, die mit einem Doppelklick gestartet werden können. Der spätere Inhalt einer *.msi*-Datenbank wird hierzu in XML formuliert und in *.wxs*-Dateien gespeichert. Diese Quelldateien werden zunächst gegen das Schema *wix.xsd* validiert. Anschließend erzeugt WiX Objektdateien, die auf *.wixobj* enden. Aus ihnen wiederum werden die

1 http://msdn.microsoft.com/library/default.asp?url=/library/en-us/msi/setup/windows_installer_start_page.asp
2 http://wix.sourceforge.net/
3 http://wix.sourceforge.net/latestrelease.html

gewünschten Installationsdateien generiert. Für das Übersetzen der XML-Dateien ist *candle.exe* zuständig, die Generierung von *.msi*-Datenbankdateien übernimmt *light.exe*. Wie aber sind *.wxs*-Dateien aufgebaut, welche Tags und Attribute enthalten sie? Sehen Sie sich hierzu bitte das folgende Beispiel an, das sich in ähnlicher Form auch in der WiX-Online-Dokumentation findet. Es beschreibt eine minimale Installationsdatei ohne grafische Benutzerführung.

```
<?xml version='1.0'?>
<Wix xmlns='http://schemas.microsoft.com/wix/2003/01/wi'>
<Product Id='12345678-1234-1234-1234-123456789012'
         Name='WiX Test'
    Language='1033'
    Version='1.0.0.0'
    Manufacturer='Thomas Kuenneth'
>
<Package Id='12345678-1234-1234-1234-123456789012'
         Description='Dies ist ein WiX-Test'
    Comments='kopiert die Datei
                  hallo.txt in das Verzeichnis WiX-Test'
    Manufacturer='Thomas Kuenneth'
    InstallerVersion='200'
    Compressed='yes'
/>
<Media Id='1'
       Cabinet='product.cab'
       EmbedCab='yes'
/>
<Directory Id='TARGETDIR'
           Name='SourceDir'
>
<Directory Id='ProgramFilesFolder'
           Name='PFiles'
>
<Directory Id='MyDir'
           Name='WiX-Test'
>
<Component Id='MyComponent'
           Guid='12345678-1234-1234-1234-123456789012'
>
<File Id='hello'
      Name='hallo.txt'
```

```
      DiskId='1'
      src='hallo.txt'
/>
</Component>
</Directory>
</Directory>
</Directory>
<Feature Id='MyFeature'
         Title='My 1st Feature'
    Level='1'
>
<ComponentRef Id='MyComponent' />
</Feature>
</Product>
</Wix>
```

Um eine Installationsdatenbank zu erzeugen, müssen Sie die Quelldatei zunächst mit *candle.exe* übersetzen und anschließend mit *light.exe* verlinken. Das entstandene *.msi*-Archiv wird im Verzeichnis der XML-Ausgangsdatei angelegt. Da im Beispiel versucht wird, *hallo.txt* zu installieren, müssen Sie vor dem Aufruf der beiden Tools sicherstellen, dass sich eine solche Datei im selben Verzeichnis befindet. In Abbildung 10.1 sehen Sie alle Stationen bis zum fertigen Installationsarchiv.

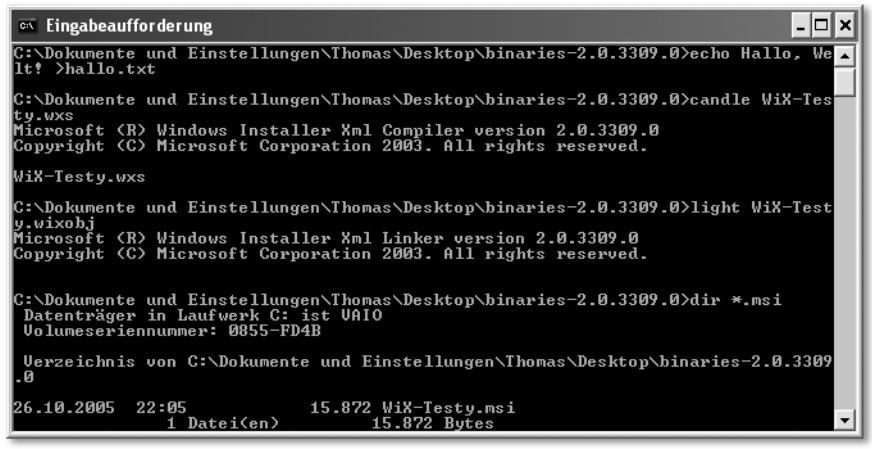

Abbildung 10.1 Aufruf von candle.exe und light.exe

Nachdem Sie die Installationsdatei durch einen Doppelklick geöffnet und damit das Setup gestartet haben, finden Sie im Modul *Software* der *Systemsteuerung* einen neuen Eintrag.

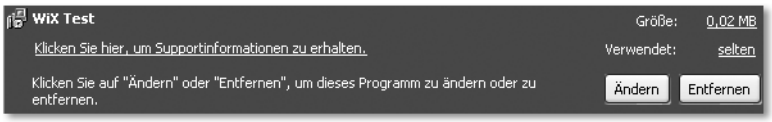

Abbildung 10.2 Der durch die Installationsdatei erzeugte Eintrag im Modul »Software«

Nach Anklicken des Links **Klicken Sie hier, um Supportinformationen zu erhalten** öffnet sich der in Abbildung 10.3 gezeigte Dialog **Supportinformationen**, der einige Informationen der *.wxs*-Quelldatei ausgibt, beispielsweise die vierstellige Versionsnummer sowie den Hersteller der Software. Bevor Sie *WiX-Test* löschen, sollten Sie einen Blick in den Standard-Ordner für Programme werfen, also normalerweise *C:\Programme*. Dort wurde durch die Installationsroutine das Verzeichnis *WiX-Test* angelegt, das die Datei *hallo.txt* enthält.

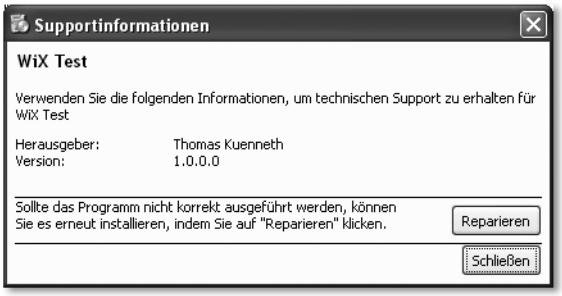

Abbildung 10.3 Der Dialog »Supportinformationen«

Mit *WiX* lassen sich keineswegs nur automatisch ablaufende Installationsprozeduren realisieren. Wie Sie die gängigen Dialoge, die dem Anwender im Verlauf einer Installation angezeigt werden, implementieren können, zeigt auf sehr anschauliche Weise das englischsprachige *WiX tutorial*[4].

10.1.2 IzPack

Auch das von Julien Ponge entwickelte *IzPack* ist ein Generator für Installationsdateien. Im Gegensatz zu *WiX* erzeugt diese Anwendung allerdings keine plattformspezifischen Installationspakete (also beispielsweise *.msi*-Datenbanken), sondern *.jar*-Archive. Diese können logischerweise auf jedem System mit vorhandener Java-Laufzeitumgebung gestartet werden. *IzPack* selbst wurde übrigens in Java geschrieben und ist damit ebenfalls auf allen Systemen mit aktueller Java-Version lauffähig. Das Projekt ist Open Source und kann kostenlos heruntergeladen[5] und verwendet werden. Die zum Zeitpunkt der Druckle-

4 http://www.tramontana.co.hu/wix/
5 http://www.izforge.com/izpack/

gung aktuelle Version finden Sie zudem auf der Begleit-CD zum Buch. Die Installation basiert selbst auf *IzPack* und ist nach wenigen Benutzereingaben abgeschlossen. In Abbildung 10.4 sehen Sie einen von *IzPack* erzeugten Dialog, der während der Installation der Anwendung erscheint.

IzPack verwendet wie schon *WiX* XML für die Definition aller Installationsschritte. Die zur Verfügung stehenden Tags und Attribute sind in der ausführlichen Dokumentation beschrieben, die Sie im Verzeichnis *doc* des *IzPack*-Basisverzeichnisses finden. Allgemein werden in einer XML-Datei so genannte *Panels* definiert und es wird die Reihenfolge festgelegt, in der diese während der Installation erscheinen.

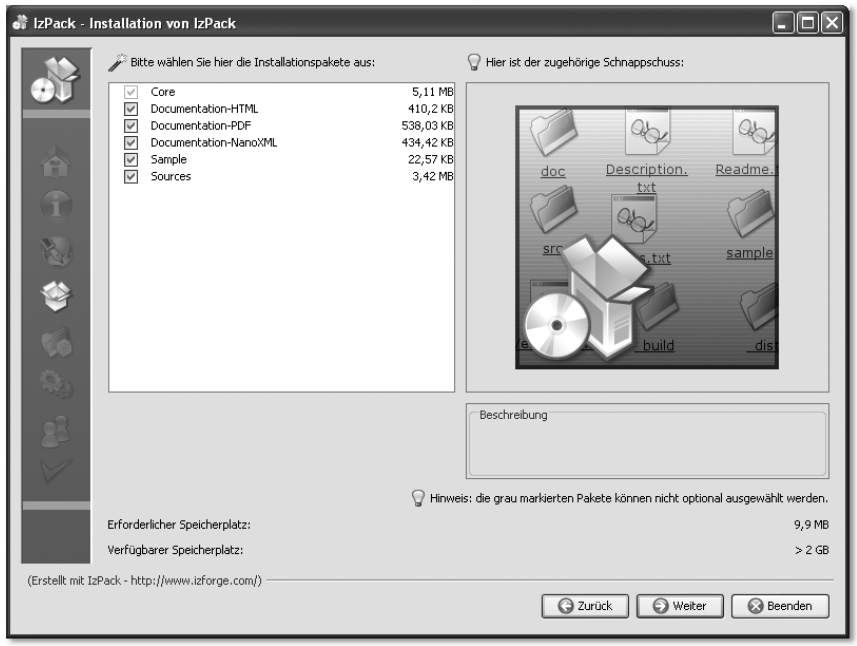

Abbildung 10.4 Die Installation von IzPack

Im Folgenden sehen Sie Teile der XML-Datei, die ich für das Erzeugen des *TKPLAFUtility*-Installationsarchivs verwende.

```
<info>
<appname>
TKPLAFUtility
</appname>
<appversion>
0.21
</appversion>
```

```
<authors>
<author name="Thomas Künneth"
        email=tkplafutility@moniundthomaskuenneth.de
/>
</authors>
<url>
http://www.moniundthomaskuenneth.de/tkplafutility/
</url>
</info>
```

Das Tag `info` kapselt allgemeine Informationen über die zu installierende Anwendung, beispielsweise den Namen und die E-Mail-Adresse des Programmautors.

```
<panels>
<panel classname="HelloPanel"/>
<panel classname="HTMLInfoPanel"/>
<panel classname="HTMLLicencePanel"/>
<panel classname="PacksPanel"/>
<panel classname="TargetPanel"/>
<panel classname="InstallPanel"/>
<panel classname="ShortcutPanel"/>
<panel classname="SimpleFinishPanel"/>
</panels>
```

Die anzuzeigenden *Panels* sowie deren Reihenfolge werden mit dem Tag `panels` festgelegt.

```
<packs>
<pack name="Base"
      required="yes"
>
<description>
The base files
</description>
<file src="readme.html"
      targetdir="$INSTALL_PATH"
/>
<file src="licence.html"
      targetdir="$INSTALL_PATH"
/>
<file src="important.html"
```

```
      targetdir="$INSTALL_PATH"
/>
<file src="history.html"
      targetdir="$INSTALL_PATH"
/>
<file src="TKPLAFUtility.ico"
      targetdir="$INSTALL_PATH"
/>
<file src="Uninstaller.ico"
      targetdir="$INSTALL_PATH"
/>
<file src="../dist/TKPLAFUtility.jar"
      targetdir="$INSTALL_PATH"
/>
<file src="../../TJCL2/dist/TJCL2.jar"
      targetdir="$INSTALL_PATH"
/>
</pack>
</packs>
```

Das Tag `packs` schließlich legt fest, aus welchen Komponenten eine Anwendung besteht, welche Teile installiert werden müssen und welche Bestandteile optional sind.

Ein großer Vorteil von *IzPack* ist ganz sicher seine Plattformunabhängigkeit. Sie können also Installationsarchive erzeugen, die nicht nur unter Windows lauffähig sind, sondern auf jedem System, das eine Java-Laufzeitumgebung zur Verfügung stellt. Auf der anderen Seite kann *IzPack* strukturbedingt keine so weit reichende Integration in das System bieten wie beispielsweise der Windows-Installer. So können Sie zwar Einträge im Start-Menü anlegen, derzeit werden mit *IzPack* installierte Anwendungen aber nicht im Modul *Software* der *Systemsteuerung* verankert. Wie Sie diese Funktionalität unabhängig von *IzPack* zur Verfügung stellen können, erläutert Kapitel 8, *Zugriff auf die Registry*.

Abgesehen vom Ausgabeformat – also *.jar*- oder *.msi*-Dateien – verfolgen die meisten Installationswerkzeuge den gleichen Ansatz: Sie erzeugen Setup-Dateien, die auf allen Rechnern, die mit einem Programm versorgt werden sollen, ausgeführt werden müssen. Die Installationspakete werden hierzu entweder auf einem Trägermedium (USB-Stick, Diskette oder CD/DVD) zur Verfügung gestellt oder aber über das Intranet/Internet heruntergeladen. Im nächsten Abschnitt stelle ich Ihnen mit *Web Start* eine Java-eigene Technologie vor, die eine Softwareverteilung ohne die sonst üblichen Installationsschritte ermöglicht.

10.2 Java Web Start

Hinter *Java Web Start* steckt die attraktive Idee, Programme durch einmaliges Anklicken eines Links auf einer Webseite verfügbar zu machen. Die sonst erforderlichen Schritte (Herunterladen, Entpacken und Installieren) entfallen. Nach dem ersten Programmstart steht die Anwendung in einer Art Programm-Manager jederzeit wieder bereit. Zudem bietet *Web Start* die Möglichkeit, Verknüpfungen auf dem Desktop und im Start-Menü anzulegen. Der Aufruf der Anwendung erfolgt dann wie gewohnt. Umfangreiche Sicherheitsvorkehrungen sorgen übrigens dafür, dass der Benutzer vor Gefahren von mittels Web Start bezogenen Programmen weitestgehend geschützt wird. Beispielsweise laufen nicht vertrauenswürdige Programme in einer abgeschotteten Umgebung, die wir auch von Applets kennen. Fordert eine Web Start-Anwendung Rechte an, die denen lokaler Programme entsprechen, muss der Anwender dem ausdrücklich zustimmen. Ein weiterer, für Benutzer und Entwickler gleichermaßen interessanter Aspekt ist, dass sich Web Start automatisch um Updates kümmert. Steht eine neue Programmversion zur Verfügung, wird diese geladen und ersetzt die alte lokale Kopie. Wie dies alles in der Praxis abläuft, zeige ich Ihnen im folgenden Abschnitt. Dort sehen Sie den Beginn einer Web Start-Anwendung aus der Sicht des Benutzers.

10.2.1 Web Start aus der Sicht des Anwenders

Der erstmalige Aufruf einer Web Start-Anwendung erfolgt durch Anklicken eines Links auf einer Webseite. Dieser verweist auf eine *.jnlp*-Datei, die zahlreiche Informationen über das zu ladende Programm enthält. Ein Beispiel für eine solche Datei finden Sie in Abschnitt 10.2.2. Da der Dateityp *.jnlp* mit *javaws.exe* verknüpft ist, wird diese Komponente der Laufzeitumgebung nach dem herunterladen der Datei gestartet und übernimmt deren Auswertung und damit das Starten der Anwendung. In Abbildung 10.5 sehen Sie den Ladevorgang der Web Start-Version von *TKPLAFUtility*.

Abbildung 10.5 Start einer Web Start-Anwendung

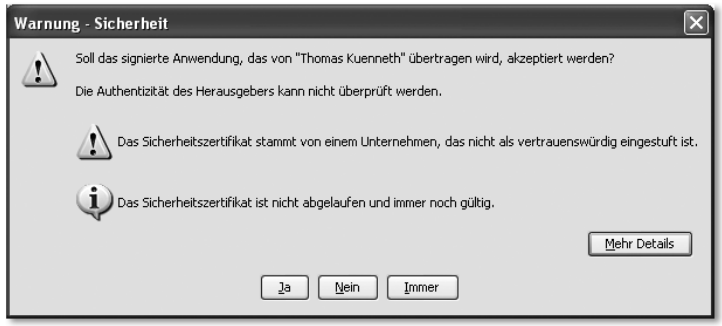

Abbildung 10.6 Sicherheitshinweis beim Laden einer Web Start-Anwendung

Hat die Anwendung erweiterte Rechte angefordert, greifen verschärfte Sicherheitsprüfungen auf der Basis so genannter Zertifikate. Diese werden vom Entwickler mit der Anwendung verknüpft. Wie dieser *Signieren* genannte Vorgang funktioniert, zeige ich Ihnen in Abschnitt 10.2.2. Das Ergebnis der Zertifikatsprüfung zeigt Web Start in einem Dialog an, den Sie in Abbildung 10.6 sehen. Die Anwendung wird nur geladen und gestartet, wenn der Benutzer dies durch Anklicken der Schaltflächen **Ja** oder **Immer** bestätigt.

Für die Verwaltung von Web Start-Anwendungen gibt es das *Cache-Anzeigeprogramm für Java-Anwendungen*. Hinter diesem langen Namen verbirgt sich übrigens *javaws.exe*. Wenn Sie es aufrufen, sehen Sie ein Fenster ähnlich dem der Abbildung 10.7.

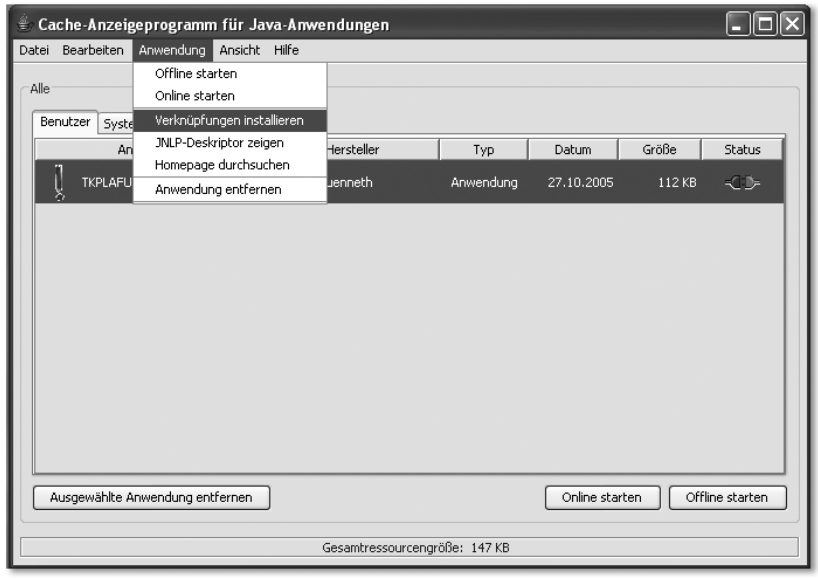

Abbildung 10.7 Web Start-Verwaltungszentrum javaws.exe

Das Programm beinhaltet einige äußerst nützliche Funktionen. Beispielsweise können Sie jederzeit für eine Web Start-Anwendung Verknüpfungen anlegen oder entfernen. Zudem ist es sehr einfach möglich, die Homepage der Anwendung zu besuchen und natürlich das Programm ggf. auch zu löschen.

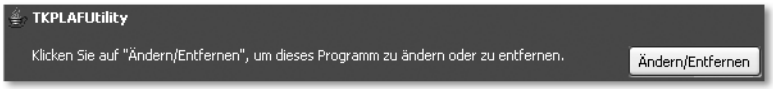

Abbildung 10.8 Eine Web Start-Anwendung im Modul »Software« der Systemsteuerung

Dies ist aber auch über das Modul *Software* der *Systemsteuerung* möglich. Für jede Web Start-Anwendung wird dort nämlich ein eigener Eintrag angelegt. Ein Beispiel hierfür sehen Sie in Abbildung 10.8.

Wie Sie als Entwickler Ihre Programme zu Web Start-Anwendungen machen können, zeigt nun der folgende Abschnitt.

10.2.2 Web Start-Anwendungen erstellen

Web Start implementiert das so genannte *Java Network Launching Protocol*. Wie Sie gesehen haben, initiieren Sie den erstmaligen Programmstart durch Klick auf einen Weblink, der auf eine *.jnlp*-Datei verweist. Um eine Web Start-Anwendung bereitzustellen, legen Sie eine solche Datei, das eigentliche Programm sowie die HTML-Datei, die den Link auf Ihre *.jnlp*-Datei enthält, auf einem Webserver ab.

Wie aber sind die *JNLP-Deskriptoren* aufgebaut? Sehen Sie sich hierzu bitte die Datei *TKPLAFUtility.jnlp* an.

```
<?xml version="1.0" encoding="UTF-8"?>
<jnlp codebase=
      "http://www.moniundthomaskuenneth.de/tkplafutility/"
      href="TKPLAFUtility.jnlp">
<information>
<title>TKPLAFUtility</title>
<vendor>Thomas Kuenneth</vendor>
<offline-allowed />
<homepage href="http://www.moniundthomaskuenneth.de/tkplafutility
/" />
<description>
Anwendung zum Setzen des Look and Feels von Swing-Anwendungen
</description>
<icon href="TKPLAFUtility.gif" />
```

```
<icon href="TKPLAFUtility.jpg"
      kind="splash" />
</information>
<resources>
<j2se version="1.5+"/>
<jar href="TKPLAFUtility.jar"/>
<jar href="TJCL2.jar"/>
</resources>
<security>
<all-permissions />
</security>
<application-desc main-class="tkplafutility.TKPLAFUtility"/>
</jnlp>
```

Listing 10.1 TKPLAFUtility.jnlp

JNLP-Deskriptoren beschreiben also umfassend, was für Start und Betrieb einer Web Start-Anwendung notwendig ist. Beispielsweise können Sie festlegen, welche Java-Version eine Anwendung benötigt, welche *.jar*-Archive geladen werden müssen und ob die Anwendung lokal betrieben werden kann.

Interessant ist das Tag `all-permissions`, das für *TKPLAFUtility* weit reichende Rechte einfordert. Damit solche Anwendungen überhaupt geladen werden, müssen alle beteiligten *.jar*-Archive *signiert* werden. Hierfür benötigen Sie die beiden Programme *jarsigner.exe* und *keytool.exe* des Java Development Kits. In einem ersten Schritt müssen Sie einen so genannten Schlüsselbund (engl. keystore) anlegen und einen Schlüssel an diesem befestigen. In Abbildung 10.9 sehen Sie den Aufruf und die Bildschirmausgaben von *keytool.exe*. Sowohl der Schlüsselbund selbst als auch seine Schlüssel werden mit Passwörtern gesichert. Bitte beachten Sie, dass die getippten Zeichen dieser Worte als Klartext in der Eingabeaufforderung zu sehen sind. Der zweite Schritt ist das Anlegen eines selbst signierten Zertifikats.

```
keytool -selfcert -alias ich -keystore MeineSchluessel
```

Diese gelten als nicht vertrauenswürdig, da der Anwender die Angaben im Zertifikat nicht überprüfen kann. Denn die Daten wurden ja nicht durch eine vertrauenswürdige Stelle oder Institution bestätigt. Aus diesem Grund sollten solche Zertifikate nur zu Testzwecken eingesetzt werden. Wenn Sie planen, Web Start-Anwendungen allgemein zugänglich zu machen, ist es ratsam, ein Zertifikat bei einer vertrauenswürdigen Zertifizierungsstelle zu beantragen.

Im letzten Schritt werden alle *.jar*-Archive einer Anwendung signiert. Sie verwenden hierfür das Tool *jarsigner.exe*.

```
jarsigner -keystore MeineSchluessel
          TKPLAFUtility\dist\TKPLAFUtility.jar
          Ich
```

> Bitte beachten Sie, dass jede Änderung an einer *.jar*-Datei ein erneutes Signieren erfordert. Es ist deshalb ratsam, den Aufruf von *jarsigner.exe* in den Build-Prozess zu integrieren.

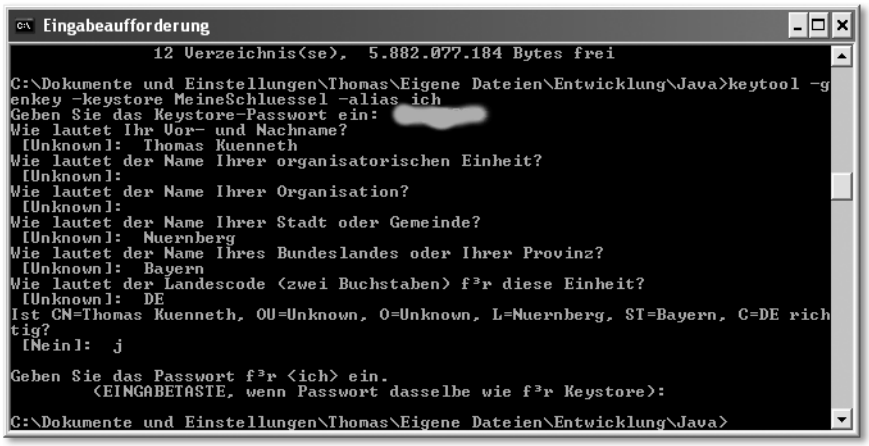

Abbildung 10.9 Ausgaben von keytool.exe

Der Aufwand beim Signieren von Web Start-Anwendungen mag viele Entwickler davon abhalten, ihre Anwendung außerhalb der sicheren Sandbox zu betreiben. Auf der anderen Seite schränkt sie die Möglichkeiten eines Programms jedoch stark ein, beispielsweise ist eine Netzwerkkommunikation nur mit dem Server möglich, von dem die Anwendung bezogen wurde. Außerdem sind keinerlei lokalen Dateizugriffe gestattet. Aus diesem Grund hat Sun eine Reihe von Klassen definiert, die auch nicht vertrauenswürdigen Anwendungen den Zugriff auf lokale Ressourcen erlauben. Der Vorteil hierbei ist, dass die Laufzeitumgebung den Anwender vor dem Ausführen potenziell gefährlicher Aktionen informieren und seine Zustimmung zur Durchführung einholen kann. Ausführliche Informationen hierzu sowie zahlreiche andere Aspekte rund um Web Start finden Sie im englischsprachigen Artikel *Java Web Start Developer's Guide*[6].

6 http://java.sun.com/products/javawebstart/1.2/docs/developersguide.html

Im nächsten Abschnitt stelle ich Ihnen eine Reihe von Tools und Klassenbibliotheken vor, die sich unabhängig von der gewählten Form der Softwareverteilung einsetzen lassen.

10.3 Allgemeine Tools

In Kapitel 2, *Entwicklungswerkzeuge*, habe ich sinngemäß geschrieben, dass erst die richtige Auswahl von Tools den Prozess des Programmierens wirklich angenehm macht. Dies trifft auch auf die Softwareverteilung zu. Hier werden deshalb einige Projekte vorgestellt, die Ihnen als Entwickler bei den Vorbereitungen helfen oder es Ihnen ermöglichen, Ihre Programme noch enger mit Windows zu verzahnen.

10.3.1 Verknüpfungen mit JShortcut

Die von Jim McBeath entwickelte Klassenbibliothek *JShortcut* erlaubt das Anlegen und Auslesen von *Verknüpfungen*. Sie können beispielsweise Einträge im Start-Menü oder der Schnellstartleiste erzeugen. Dies bietet sich an, wenn Sie Ihre Anwendung mit einer selbst entwickelten Installationsroutine ausstatten möchten. Das Projekt[7] ist Open Source und wird unter der *GNU Lesser General Public License* vertrieben. Sie können *JShortcut* also kostenlos herunterladen, verwenden und weitergeben. Die zum Zeitpunkt der Drucklegung aktuelle Version finden Sie auf der Begleit-CD zum Buch.

Abbildung 10.10 Das Installationsprogramm von JShortcut

Bevor Sie die Klassenbibliothek in Ihren Programmen verwenden können, müssen Sie das Archiv *jshortcut-0_4.jar* einmalig starten, woraufhin der in Abbildung 10.10 gezeigte Dialog erscheint. Klicken Sie auf **Nein**, um ein beliebiges anderes Installationsverzeichnis auszuwählen. Die Dateien werden in das Verzeichnis *jshortcut-0_4* innerhalb des von Ihnen angegebenen Ordners kopiert. Damit ist der eigentliche Installationsvorgang beendet. Um *JShortcut*

7 http://alumnus.caltech.edu/~jimmc/jshortcut/index.html

in Ihren Anwendungen nutzen zu können, müssen Sie noch dafür sorgen, dass die Java-Laufzeitumgebung die beiden Dateien *jshortcut.jar* und *jshortcut.dll* findet. Ausführliche Hinweise hierzu finden Sie in Kapitel 1, *Installation und Konfiguration*.

Wie Sie *JShortcut* verwenden, möchte ich Ihnen anhand des Programms *JShortcutDemo1* zeigen. Hier zunächst dessen Quelltext.

```
package javafuerwindows.jshortcut;
import java.io.File;
import java.io.FileWriter;
import net.jimmc.jshortcut.JShellLink;

public class JShortcutDemo1 {
    private static final String [] verz = {
        "common_desktopdirectory",
                "common_programs",
                "desktop",
                "personal",
                "programs",
                "program_files"
    };
    public static void main(String [] args) {
        try {
            for (int i = 0; i < verz.length; i++) {
                System.out.println(verz[i] + ": " +
                          JShellLink.getDirectory(verz[i]));
            }
            File f = new File(JShellLink.getDirectory(
                                  "desktop"), "hallo.txt");
            FileWriter fw = new FileWriter(f);
            fw.write("Hallo, Welt!");
            fw.close();
            JShellLink link = new JShellLink();
            link.setFolder(f.getParent());
            link.setName("Verknüpfung zu " + f.getName());
            link.setPath(f.getAbsolutePath());
            link.setDescription("Hallo, ich bin
                                eine Verknüpfung");
            link.save();
        } catch (Exception e) {
```

```
            System.err.println(e);
        }
    }
}
```

Listing 10.2 JShortcutDemo1.java

Die komplette Funktionalität wird durch die Klasse `net.jimmc.jshortcut.JShellLink` zur Verfügung gestellt. *JShortcut* bietet mit der statischen Methode `getDirectory()` eine sehr bequeme Möglichkeit, den Speicherort zahlreicher spezieller Verzeichnisse zu erfragen. Beispielsweise sehen Sie im Quelltext zu *JShortcutDemo1*, wie Sie Dateien auf dem Desktop ablegen können – im konkreten Fall die Datei *hallo.txt*. Um eine Verknüpfung anzulegen, instantiieren Sie zunächst ein Objekt von `JShellLink`. Die beiden Methoden `setFolder()` und `setName()` legen den Speicherort sowie den Namen dieser Verknüpfung fest. `setPath()` setzt das Ziel der Verknüpfung, im Fall von *JShortcutDemo1* ist dies die Datei *hallo.txt*. *JShortcut* ist ideal, um eigene Installationsroutinen zu entwickeln. Die Möglichkeit, Verknüpfungen anzulegen, ist aber auch für viele andere Programme interessant.

10.3.2 launch4j

Java-Programme lassen sich problemlos mit einem Doppelklick starten, wenn sie in *.jar*-Archiven vorliegen und diese eine Manifest-Datei mit richtig gesetztem `Main-Class:`-Attribut enthalten. Außerdem muss der Dateityp *.jar* mit *java.exe* oder *javaw.exe* verknüpft sein. Was aber passiert, wenn auf dem Zielsystem keine geeignete Java-Laufzeitumgebung installiert wurde oder die Dateiverknüpfung verloren ging? Ein anderes Problem, das die beiden Standard-Launcher *java.exe* und *javaw.exe* aufwerfen, ist, dass man im *Windows Task-Manager* nicht erkennen kann, für welche Java-Anwendung sie zuständig sind, sobald mehr als eine Instanz ausgeführt wird.

Abhilfe schaffen hier so genannte *.exe*-Wrapper. Die ihnen zugrunde liegende Idee ist, dem Anwender ein kleines Win32-Programm zur Verfügung zu stellen, das die Java-Anwendung repräsentiert. Dieses Programm kümmert sich dann um den Start der eigentlichen Anwendung. Wie dieser technisch abläuft, ist vom verwendeten Tool abhängig. Mittlerweile existieren zahlreiche kommerzielle und kostenlose Lösungen. Eine davon möchte ich Ihnen ausführlicher vorstellen.

Das von Grzegorz Kowal entwickelte *launch4j* wird unter der *GNU General Public Licence* vertrieben, kann also kostenlos von der Projekt-Homepage[8] heruntergeladen, verwendet und weitergegeben werden.

Die zum Zeitpunkt der Drucklegung aktuelle Version finden Sie auf der Begleit-CD. Eine Installation ist nicht erforderlich. Sie können das Archiv *launch4j-1.4.2.zip* in ein beliebiges Verzeichnis entpacken. Das Unterverzeichnis *bin* enthält das Programm *launch4j.exe*, der Ordner *web* enthält unter anderem die Datei *docs.html*, in der die Bedienung von *launch4j* beschrieben wird. *launch4j.exe* arbeitet mit Konfigurationsdateien, die die zu erzeugenden *.exe*-Dateien beschreiben.

```
jar TKPLAFUtility.jar
header guihead.bin
outfile out\TKPLAFUtility.exe
javamin 1.5.0
icon TKPLAFUtility.ico
errTitle TKPLAFUtility Fehler
splash TKPLAFUtility.bmp
waitfor TKPLAFUtility
splashTimeout 45
chdir true
setProcName true
```

Listing 10.3 TKPLAFUtility.cfg

In der Konfigurationsdatei werden alle Daten eingetragen, die *launch4j* für das Erzeugen der *.exe*-Datei benötigt. Die Datei ist zeilenweise aufgebaut und enthält Name-Wert-Paare, die durch ein Leerzeichen getrennt sind. Beispielsweise folgt auf das Schlüsselwort `jar` das *.jar*-Archiv der Java-Anwendung. Es muss eine Manifest-Datei mit gültigem `Main-Class:`-Attribut enthalten. Falls die Anwendung auf externe Bibliotheken angewiesen ist, müssen diese im `Class-Path:`-Attribut von *manifest.mf* aufgeführt werden. Diese zusätzlichen Bibliotheken werden übrigens nicht in die *.exe*-Datei übernommen. Sie müssen ggf. von einem Installationsprogramm in das gleiche Verzeichnis kopiert werden, in dem auch die *.exe*-Datei liegt. Die Schlüssel `splash`, `waitfor` und `splashTimeout` definieren einen (mittlerweile fast obligatorischen) Startdialog, der während des Ladevorgangs angezeigt wird. Sie legen fest, welche Grafik Verwendung findet und wie lange diese zu sehen ist.

Abbildung 10.11 zeigt den Aufruf und die Bildschirmausgaben von *launch4j*.

8 http://launch4j.sourceforge.net/

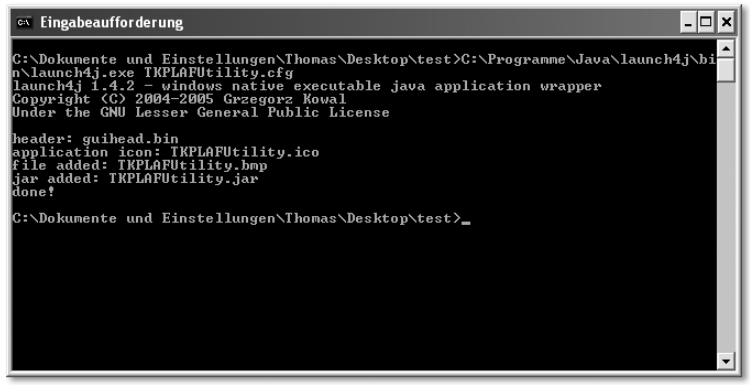

Abbildung 10.11 Aufruf und Ausgaben von launch4j

Mit *launch4j* haben Sie eine Möglichkeit kennen gelernt, Java-Anwendungen ohne die Verknüpfung mit dem Dateityp *.jar* zu starten. Neben ästhetischen Verbesserungen (anstelle eines generischen Java-Icons können Sie Ihr Programm beispielsweise mit einer individuellen Grafik versehen) spricht für die Verwendung solcher Werkzeuge, dass Java-Anwendungen im *Windows Task-Manager* eindeutig erkennbar sind (siehe Abbildung 10.12). Außerdem können Sie gezielt angeben, welche Version der Laufzeitumgebung Ihr Programm nutzen soll. Ist keine passende vorhanden, ruft *launch4j* eine Webseite mit der Möglichkeit zum Download auf. Schließlich können Sie Parameter an Ihre Anwendung übergeben und Änderungen an der Konfiguration der virtuellen Maschine vornehmen, beispielsweise die Heap-Größe verändern.

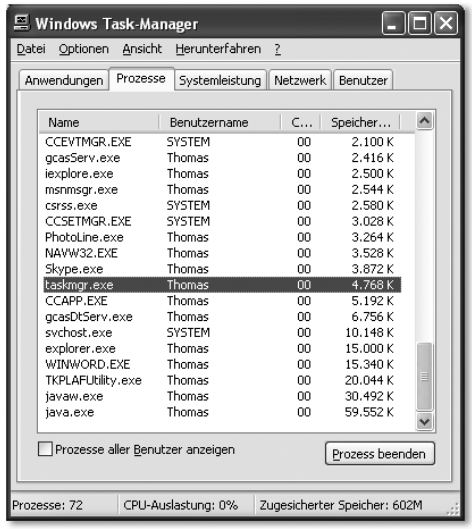

Abbildung 10.12 Windows Task-Manager

11 Multimedia

11 Multimedia

Egal ob MP3, Bildbearbeitung oder virtuelle Welten, in jedem der genannten Bereiche ist Rechengeschwindigkeit Trumpf. In diesem Kapitel lernen Sie, wie Sie die multimedialen Fähigkeiten Ihres Computers auch unter Java ausreizen.

Gerade in den ersten Jahren seines Bestehens hatte Java den Ruf, langsam zu sein, und damit nicht gerade die erste Wahl, um Multimedia-Anwendungen zu realisieren. In der Tat konnte Java in vielen Bereichen nicht mit nativen Programmen Schritt halten. Dies lag aber nicht nur an einer zu geringen Ausführungsgeschwindigkeit, sondern auch an der Tatsache, dass für viele Bereiche keine leistungsfähigen Klassenbibliotheken existierten. Dies hat sich glücklicherweise geändert. So hat Sun kontinuierlich die Leistungsfähigkeit der virtuellen Maschine, aber auch der Standard-Klassenbibliothek steigern können. Und gerade im Hinblick auf multimediale Anwendungen kann der Entwickler auf einige äußerst anspruchsvolle APIs zurückgreifen, wobei einige davon in diesem Kapitel vorgestellt werden. In den ersten beiden Abschnitten zeige ich Ihnen, wie Sie in Java auf Bilddateien zugreifen können. Sie lernen hierzu die Klassenbibliothek *Jimi* kennen, die unter anderem das Lesen und Schreiben einiger älterer Dateiformate wie *PCX*, *PICT* oder *Targa*, nicht ganz so gebräuchlicher Formate wie beispielsweise *XBM*, *XPM* oder *Sunraster*, aber auch die Windows-Standards *.ico* und *.bmp* ermöglicht. Der zweite Abschnitt beschäftigt sich mit *Java Image I/O*, einer Implementierung des JSR-015 und seit Java 1.4 Teil der Standard-Installation. Wie *Jimi* dient auch *Java Image I/O* dem Laden und Speichern von Bilddaten. Deshalb werde ich Ihnen die Funktionsweise erläutern sowie Vor- und Nachteile gegenüber Jimi aufzeigen. Der dritte Abschnitt beschäftigt sich mit dem *Java Media Framework*, das die Verarbeitung zeitbasierter Audio- und Videoströme gestattet. In Abschnitt 11.4 schließlich lernen Sie mit *Java 3D* die faszinierenden Möglichkeiten der 3D-Grafikprogrammierung kennen.

11.1 Java Image Management Interface (Jimi)

Die Klassenbibliothek *Jimi*[1] dient in erster Linie dem Lesen und Schreiben von Bilddateien in zahlreichen Formaten. Sie wurde ursprünglich von der Firma *Activated Intelligence* entwickelt, aber später von Sun gekauft. Da *Jimi* schon seit geraumer Zeit auf dem Markt ist, lässt sich die Klassenbibliothek sogar

1 http://java.sun.com/products/jimi/

noch unter Java 1.1.x betreiben. Um sie in eigenen Programmen verwenden zu können, laden Sie bitte zunächst die Datei *jimi1_0.zip* herunter und entpacken sie in einem beliebigen Verzeichnis. *JimiProClasses.zip* enthält alle für den Betrieb notwendigen Klassen. Am besten kopieren Sie deshalb dieses Archiv in das Java-Erweiterungsverzeichnis. Alternativ können Sie sie auch dem Klassenpfad Ihrer Anwendung hinzufügen. Ausführliche Hinweise hierzu finden Sie in Kapitel 1, *Installation und Konfiguration*. Die Unterverzeichnisse *docs* und *examples* enthalten eine ausführliche Dokumentation der Bibliothek sowie eine Reihe von Beispielen.

11.1.1 Laden von Bilddateien

Um Ihnen einen ersten Eindruck zu vermitteln, wie die Klassen von *Jimi* eingesetzt werden, folgt hier das Programm *JimiDemo1*, das einen einfachen Bild-Betrachter implementiert.

```
package javafuerwindows.jimi;
import com.sun.jimi.core.Jimi;
import java.awt.Image;
import javax.swing.ImageIcon;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JTabbedPane;

public class JimiDemo1 {
    public static void main(String [] args) {
        if (args.length > 0) {
            JFrame f = new JFrame("JimiDemo1");
            f.setDefaultCloseOperation(
                                    JFrame.EXIT_ON_CLOSE);
            JTabbedPane p = new JTabbedPane();
            f.setContentPane(p);
            int pos;
            for (int i = 0; i < args.length; i++) {
                String fn = args[i];
                Image image = Jimi.getImage(fn);
                ImageIcon ii = new ImageIcon(image);
                JLabel l = new JLabel(ii);
                pos = fn.lastIndexOf(
                                java.io.File.separatorChar);
                p.add(l, fn.substring(pos + 1));
```

```
            }
            f.pack();
            f.setVisible(true);
        }
    }
}
```

Listing 11.1 JimiDemo1.java

Der größte Teil des Programms beschäftigt sich mit dem Aufbau der Benutzeroberfläche. Die Hauptaufgabe, das Laden einer Bilddatei, wird durch Aufruf der statischen Methode `getImage()` der Klasse `com.sun.jimi.core.Jimi` bewältigt. Diese liefert eine Instanz von `java.awt.Image`, die beliebig weiterverarbeitet werden kann.

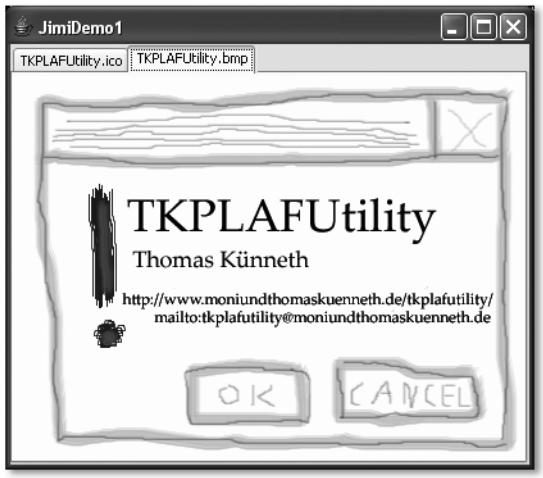

Abbildung 11.1 Der Bild-Betrachter JimiDemo1

JimiDemo1 erwartet die anzuzeigenden Grafiken als Argumente der Kommandozeile. Abbildung 11.1 zeigt zwei übergebene Dateien, *TKPLAFUtility.ico* sowie *TKPLAFUtility.bmp*.

Das Speichern einer Datei ist ebenfalls auf sehr einfache Weise möglich. Wie es funktioniert, zeige ich Ihnen im folgenden Abschnitt.

11.1.2 Speichern von Bilddateien

Das Speichern von Grafiken wird ebenfalls mit der Klasse `Jimi` realisiert. Sie müssen die Methode `putImage()` nur mit einigen zusätzlichen Parametern aufrufen, unter anderem den Mime-Typ des zu schreibenden Formats. Mit den

beiden Methoden `getEncoderTypes()` und `getDecoderTypes()` können Sie ermitteln, welche Formate für das Schreiben und Lesen von Bilddaten zur Verfügung stehen. Wie Sie diese Methoden in der Praxis einsetzen, zeigt Ihnen das Programm *JimiDemo2*.

```
package javafuerwindows.jimi;
import com.sun.jimi.core.Jimi;
import com.sun.jimi.core.JimiException;
import java.awt.Color;
import java.awt.Graphics;
import java.awt.image.BufferedImage;

public class JimiDemo2 {
    public static void main(String [] args) {
        String [] enc = Jimi.getEncoderTypes();
        System.out.println("folgende
                        Formate können geschrieben werden:");
        for (int i = 0; i < enc.length; i++) {
            System.out.println(enc[i]);
        }
        String [] dec = Jimi.getDecoderTypes();
        System.out.println("folgende
                        Formate können gelesen werden:");
        for (int i = 0; i < dec.length; i++) {
            System.out.println(dec[i]);
        }
        BufferedImage bi = new BufferedImage(200, 200,
                                BufferedImage.TYPE_INT_RGB);
        Graphics g = bi.getGraphics();
        g.setColor(Color.red);
        g.fillRect(0, 0, 200, 200);
        g.setColor(Color.yellow);
        g.drawString("Hallo, Welt!", 50, 50);
        try {
            Jimi.putImage("image/jpeg", bi, "bild.jpg");
        } catch (JimiException e) {
            System.err.println(e);
        }
    }
}
```

Listing 11.2 JimiDemo2.java

Das Programm gibt zunächst die Liste der Mime-Typen aus, die gelesen und geschrieben werden können. Anschließend wird eine Grafik erzeugt, die auf rotem Hintergrund den Text *Hallo, Welt!* in gelber Schrift zeigt. Sie wird unter dem Namen *bild.jpg* im JPEG-Format gespeichert. *Jimi* eignet sich aufgrund der Vielzahl an unterstützten Formaten und der unkomplizierten Handhabung ideal für das Laden und Speichern von Bilddaten.

Im nächsten Abschnitt stelle ich Ihnen Java Image I/O vor, seit Java 1.4 Teil der Standard-Klassenbibliothek. Auch hier geht es um das Lesen und Schreiben von Bilddaten.

11.2 Java Image I/O

Java Image I/O dient dazu, Pixeldaten aus Dateien und über das Netzwerk zu lesen und zu schreiben. Die Klassenbibliothek wurde in JSR-015 definiert und ist seit Java 1.4 fester Bestandteil von Java SE. Ziel bei der Entwicklung war eine leicht einsetzbare, durch Plugins erweiterbare Architektur. Die Lese- und Schreibklassen sollten nicht nur den Zugriff auf die eigentlichen Bildinformationen, sondern auch auf etwaige Metadaten und Vorschaubilder erlauben. Einige bekannte Formate werden von Haus aus unterstützt, weitere lassen sich durch eigene Klassen nachrüsten. *Java Image I/O* gehört formal übrigens zu der Klassenbibliothek *Java Advanced Imaging*, ist aber unabhängig von deren übrigen Klassen einsetzbar. Seit Februar 2005 ist Java Image I/O ein Open Source-Projekt auf **www.java.net.**[2]

11.2.1 Lesen von Bilddateien

In diesem Abschnitt sehen Sie, wie leicht Sie mit *Java Image I/O* Bilddateien lesen können. Hier zunächst der Quelltext des Programms *JIIODemo1*, das einen einfachen Bildbetrachter implementiert.

```
package javafuerwindows.jiio;
import java.awt.Dimension;
import java.awt.image.BufferedImage;
import java.io.File;
import java.io.IOException;
import javax.imageio.ImageIO;
import javax.swing.ImageIcon;
import javax.swing.JFileChooser;
import javax.swing.JLabel;
import javax.swing.JOptionPane;
```

2 https://jai-imageio.dev.java.net/

```
import javax.swing.JScrollPane;
public class JIIODemo1 {
    public static void main(String [] args) {
        JFileChooser fc = new JFileChooser();
        fc.setDialogTitle("zu ladendes Bild auswählen");
        if (fc.showOpenDialog(null) ==
                        JFileChooser.APPROVE_OPTION) {
            File f = fc.getSelectedFile();
            try {
                BufferedImage bi = ImageIO.read(f);
                ImageIcon ii = new ImageIcon(bi);
                JLabel l = new JLabel(ii);
                JScrollPane sp = new JScrollPane(l);
                sp.setPreferredSize(new Dimension(640,
                                                  480));
                JOptionPane.showMessageDialog(null,
                                              sp,
                                f.getAbsolutePath(),
                                JOptionPane.PLAIN_MESSAGE);
            } catch (IOException e) {
                System.err.println(e);
            }
        }
    }
}
```

Listing 11.3 JIIODemo1.java

Das eigentliche Laden eines Bildes lässt sich durch eine einzige Methode erreichen. Die Klasse `javax.imageio.ImageIO` stellt hierfür `read()` zur Verfügung. Ihr wird der vollständige Name der zu lesenden Datei übergeben. Wenn *Java Image I/O* ihr Format bestimmen konnte, liefert `read()` eine Instanz von `java.awt.image.BufferedImage`.

11.2.2 Schreiben von Bilddaten

Um Bilddaten in einem bestimmten Format zu speichern, verwenden Sie bitte die Methode `write()`. Neben einem Objekt, das das Interface `RenderedImage` implementiert (beispielsweise `BufferedImage`) erwartet sie einen Formatnamen. Wie Sie diesen ermitteln können, zeigt das Programm *JIIODemo2*.

```
package javafuerwindows.jiio;
import javax.imageio.ImageIO;
public class JIIODemo2 {
    public static void main(String [] args) {
        ausgabe("folgende Formate können gelesen werden:",
                ImageIO.getReaderFormatNames());
        ausgabe("folgende Formate können
                 geschrieben werden:",
                ImageIO.getWriterFormatNames());
    }

    private static void ausgabe(String titel,
                                String [] namen) {
        System.out.println(titel);
        for (int i = 0; i < namen.length; i++) {
            System.out.println(namen[i]);
        }
    }
}
```

Listing 11.4 JIIODemo2.java

Java Image I/O ist eine sehr leicht einsetzbare Klassenbibliothek, die zudem in jeder aktuellen Java-Version enthalten ist. Allerdings ist derzeit die Zahl der unterstützten Bildformate im Vergleich zu *Jimi* äußerst bescheiden. Glücklicherweise kann die Formatliste durch zusätzlich installierte Plugins im Prinzip beliebig erweitert werden. Ausführliche englischsprachige Informationen zur Verwendung der API und Programmierung eigner Plugins finden Sie im *Java Image I/O API Guide.*[3]

Im nächsten Abschnitt stelle ich Ihnen das *Java Media Framework* vor. Es erweitert Java um die Fähigkeit, komplexe Audio- und Videoströme wiederzugeben, aufzuzeichnen und zu verarbeiten.

3 http://java.sun.com/j2se/1.5.0/docs/guide/imageio/spec/title.fm.html

11.3 Java Media Framework

Das *Java Media Framework*[4] (JMF) ist eine Erweiterung der Standard-Klassenbibliothek, die die Verarbeitung von Audio, Video und anderen zeitbasierten Medien in Java-Anwendungen und Applets ermöglicht. Neben einer plattformunabhängigen Version stellt Sun eine für Windows optimierte Ausgabe des JMF zur Verfügung.

11.3.1 Installation des JMF

Um das Java Media Framework in Ihren Programmen verwenden zu können, müssen Sie zunächst das *Windows Performance Pack* (*jmf-2_1_1e-windows-i586.exe*) herunterladen.[5] Anschließend starten Sie bitte dieses Programm, das Sie durch den Installationsvorgang führt. Nach dem Bestätigen der Lizenzvereinbarung und Festlegen des Zielverzeichnisses sehen Sie den in Abbildung 11.2 gezeigten Dialog **Select Components**.

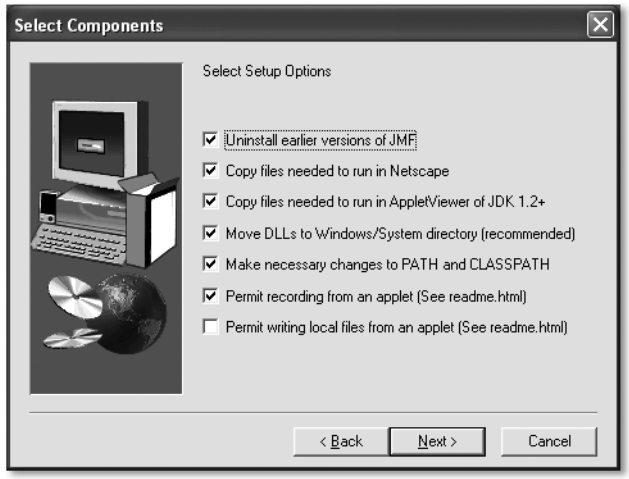

Abbildung 11.2 Die Installation des Java Media Frameworks

Bitte achten Sie darauf, dass sowohl **Move DLLs to Windows/System directory** als auch **Make neccessary changes to PATH and CLASSPATH** mit einem Häkchen versehen sind. Zwar erwähnte ich in Kapitel 1, *Installation und Konfiguration*, dass Sun selbst von der Verwendung der Umgebungsvariable `CLASSPATH` abrät, allerdings stellen Sie auf diese Weise einen reibungslosen Betrieb des JMF sicher. Klicken Sie auf die Schaltfläche **Next**, um den Installationsvorgang abzuschließen.

4 http://java.sun.com/products/java-media/jmf/index.jsp
5 http://java.sun.com/products/java-media/jmf/2.1.1/download.html

Falls Sie das Setup-Programm auffordert, Ihren Rechner neu zu starten, rate ich Ihnen, diesen Neustart vor weiteren Arbeiten mit dem Java Media Framework durchzuführen.

Sun stellt ein Applet[6] zur Verfügung, mit dem Sie Ihre JMF-Installation testen können. In Abbildung 11.3 sehen Sie seine Bildschirmausgaben nach einer erfolgreich durchgeführten Installation. Um mit dem Java Media Framework auch MP3-Dateien wiedergeben zu können, müssen Sie ein entsprechendes Plugin herunterladen und installieren.

Abbildung 11.3 JMF – Diagnostics Applet

Laden Sie bitte zunächst die Datei *javamp3–1_0.exe* herunter[7] und starten Sie sie. Wie schon beim JMF selbst führt Sie das Setup-Programm durch die Installationsschritte. Nach Bestätigung der Lizenzvereinbarung werden Sie aufgefordert, das Zielverzeichnis anzugeben, in welches das Plugin kopiert werden soll. Bitte achten Sie bei Development Kits darauf, nicht dessen Basisverzeichnis anzugeben, sondern das Basisverzeichnis seiner Laufzeitumgebung. In Abbildung 11.4 ist unter **Destination Folder** ein solches angezeigtes Verzeichnis zu sehen. Hinweise zu der Verzeichnisstruktur der Java-Laufzeitumgebung und des Development Kits finden Sie in Kapitel 1, *Installation und Konfiguration*.

Das *Java Media Framework* lässt sich mit dem in Abbildung 11.5 gezeigten Programm *JMF Registry* administrieren. Beispielsweise können Plugins hinzugefügt und gelöscht sowie verschiedene Benutzereinstellungen vorgenommen werden.

Nachdem die Installation des JMF aus Benutzersicht abgeschlossen ist, zeige ich im Folgenden, wie Sie sich als Programmierer dieser Klassenbibliothek nähern können.

6 http://java.sun.com/products/java-media/jmf/2.1.1/jmfdiagnostics.html
7 http://java.sun.com/products/java-media/jmf/mp3/download.html

Abbildung 11.4 Installation des MP3-Plugins

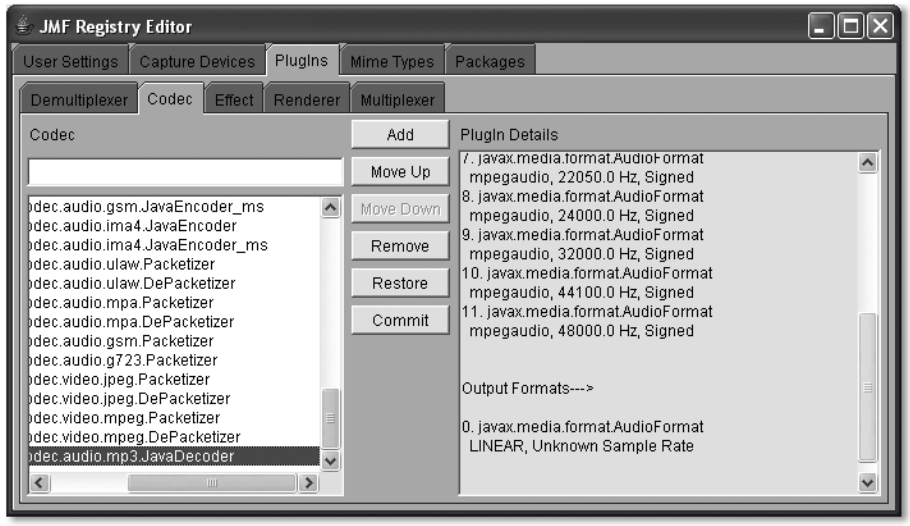

Abbildung 11.5 JMF Registry

11.3.2 Audio-Wiedergabe mit dem JMF

Obwohl das *Java Media Framework* eine sehr umfangreiche und komplexe Klassenbibliothek ist, lassen sich einfache Aufgaben mit äußerst geringem Aufwand realisieren. Sehen Sie sich hierzu bitte das Programm *JMFDemo1* an, das ein minimales Abspielprogramm für Audiodateien implementiert.

```
package javafuerwindows.jmf;
import java.io.File;
```

```
import javax.media.ControllerEvent;
import javax.media.ControllerListener;
import javax.media.EndOfMediaEvent;
import javax.media.Manager;
import javax.media.MediaLocator;
import javax.media.Player;
import javax.swing.JFileChooser;

public class JMFDemo1 implements ControllerListener {
    private File [] songs;
    private int pos;

    public JMFDemo1(File [] songs) {
        this.songs = songs;
        pos = 0;
        naechsterSong();
    }

    private void naechsterSong() {
        if (pos == songs.length) {
            System.exit(0);
        }
        try {
            MediaLocator ml = new MediaLocator(
                                songs[pos].toURL());
            Player player = Manager.createRealizedPlayer(
                                                    ml);
            player.addControllerListener(this);
            player.start();
            pos++;
        } catch (Exception e) {
            System.err.println(e);
        }
    }

    public void controllerUpdate(ControllerEvent e) {
        if (e instanceof EndOfMediaEvent) {
            naechsterSong();
        }
    }
```

```
    public static void main(String [] args) {
        JFileChooser fc = new JFileChooser();
        fc.setMultiSelectionEnabled(true);
        if (fc.showOpenDialog(null) ==
                              JFileChooser.APPROVE_OPTION) {
            new JMFDemo1(fc.getSelectedFiles());
        }
    }
}
```

Listing 11.5 JMFDemo1.java

Die Abspiellogik wird vollständig in der Methode `naechsterSong()` realisiert. Zunächst wird für die Datei, die wiedergegeben werden soll, eine Instanz der Klasse `javax.media.MediaLocator` erzeugt. Diese wird an die statische Methode `createRealizedPlayer()` der Klasse `javax.media.Manager` übergeben. Sie liefert ein Objekt des Typs `javax.media.Player`.

Wenn Ihnen *JMFDemo1* Appetit auf mehr gemacht hat, sollten Sie einen Blick auf die Beispiele werfen, die Sun auf der Seite *Java Media Framework 2.1 – Sample Code*[8] zur Verfügung stellt. Sehr lesenswert ist das Buch *Medienverarbeitung in Java*[9], es bietet eine gute Einführung in das Java Media Framework und die Medienverarbeitung in Java.

Auch im nächsten Abschnitt stelle ich Ihnen eine umfangreiche Klassenbibliothek vor: Java 3D. Sie erweitert die Standard-Klassenbibliothek um die Möglichkeit, attraktive, realistisch wirkende virtuelle Welten zu erschaffen.

11.4 Java 3D

Die Klassenbibliothek *Java 3D* wurde ursprünglich von Sun entwickelt. Mittlerweile wird sie aber im Rahmen eines Community Source-Projekts auf **java3d.dev.java.net** gepflegt.

11.4.1 Die Installation von Java 3D

Die derzeit aktuelle Release-Version der Klassenbibliothek ist 1.3.2. Sie können die für den Einsatz erforderlichen Komponenten *java3d-1_3_2-doc.zip* und *java3d-1_3_2-windows-i586.zip* von der Projekt-Seite *Java 3D Binary Builds* herunterladen.[10] Entpacken Sie zunächst das Archiv *java3d-1_3_2-windows-*

8 http://java.sun.com/products/java-media/jmf/2.1.1/samples/samplecode.html
9 Horst Eidenberger, Roman Divotkey: Medienverarbeitung in Java, dpunkt Verlag

i586.zip in ein beliebiges Verzeichnis. Neben einigen Textdateien, beispielsweise *BINARY-CODE-LICENSE.txt* und *README.txt*, wurde ein weiteres Archiv erstellt: *j3d-132-win.zip*. Bitte entpacken Sie dieses Archiv. Es enthält zwei Verzeichnisse: *bin* und *lib*. Diese beiden Ordner müssen einschließlich aller in ihnen enthaltenen Dateien und Unterverzeichnisse in das Basisverzeichnis der Laufzeitumgebung kopiert werden, mit der Sie Java 3D verwenden möchten. Wenn Sie Java 3D in Verbindung mit dem Development Kit nutzen möchten, beachten Sie bitte, dass nicht das Installationsverzeichnis des JDK gemeint ist, sondern das Basisverzeichnis seiner Laufzeitumgebung, also das Verzeichnis *jre*. Nach der Kopieraktion sollten sich die drei Dateien *j3dcore-d3d.dll*, *j3dcore-ogl.dll* und *j3dutils.dll* im Verzeichnis *bin* des Basisverzeichnisses der Java-Laufzeitumgebung befinden. Die drei Dateien *j3dcore.jar*, *j3dutils.jar* und *vecmath.jar* gehören in das Java-Erweiterungsverzeichnis, also *lib\ext* unterhalb des Basisverzeichnisses. Ausführliche Informationen zur Verzeichnisstruktur einer Java-Laufzeitumgebung und zum Konzept des Erweiterungsverzeichnisses finden Sie in Kapitel 1, *Installation und Konfiguration*. Das Archiv *java3d-1_3_2-doc.zip* enthält die vollständige Klassen-Dokumentation von Java 3D. Sie können es in einem beliebigen Verzeichnis entpacken. Um beim Programmieren Zugriff auf Beschreibungen zu haben, sollten Sie das Javadoc-Verzeichnis in Ihrer Entwicklungsumgebung registrieren.

11.4.2 Test der Java 3D-Installation

Bevor Sie mit der Entwicklung eigener Java 3D-Anwendungen beginnen, wollen wir die nun abgeschlossene Installation testen. Leider enthält *java3d-1_3_2-windows-i586.zip* keine Demo-Anwendungen. Glücklicherweise finden sich im *World Wide Web* viele Java 3D-gestützte Applets. Daniel Hirsch bietet auf seiner Homepage[11] beispielsweise eine sehr hübsche Implementierung des berühmten Rubic's Cube, die Sie in Abbildung 11.6 sehen.

Sollten Sie Probleme haben, Java 3D-Programme oder Applets auszuführen, prüfen Sie bitte, ob sich die in Abschnitt 11.3.1 genannten Dateien in den richtigen Verzeichnissen befinden. Startet das Applet, erzeugt aber Darstellungsfehler oder reagiert nicht mehr, liegt das Problem möglicherweise im Zusammenspiel zwischen AWT, Swing, Java 3D und DirectShow. AWT und Swing nutzen diese Windows-Komponente, um 2D-Operationen zu beschleunigen. Leider klappt das Zusammenspiel mit Java 3D und einigen (fehlerhaften) Grafikkarten-Treibern nicht richtig. Dies zeigt sich durch Programmabstürze und »Hänger«. In diesem Fall müssen Sie die Verwendung von DirectX unterbin-

10 https://java3d.dev.java.net/binary-builds.html
11 http://www.daniel-hirsch.de/

den. Hierzu stehen Ihnen einige System-Properties zur Verfügung, die Sie in verschiedenen Kombinationen testen müssen.

```
sun.java2d.d3d=false
sun.java2d.ddoffscreen=false
sun.java2d.noddraw=true
```

Sie können System-Properties beim Start der Java-Laufzeitumgebung mit der `-D`-Option setzen, beispielsweise:

```
java.exe -Dsun.java2d.d3d=false -jar Programm.jar
```

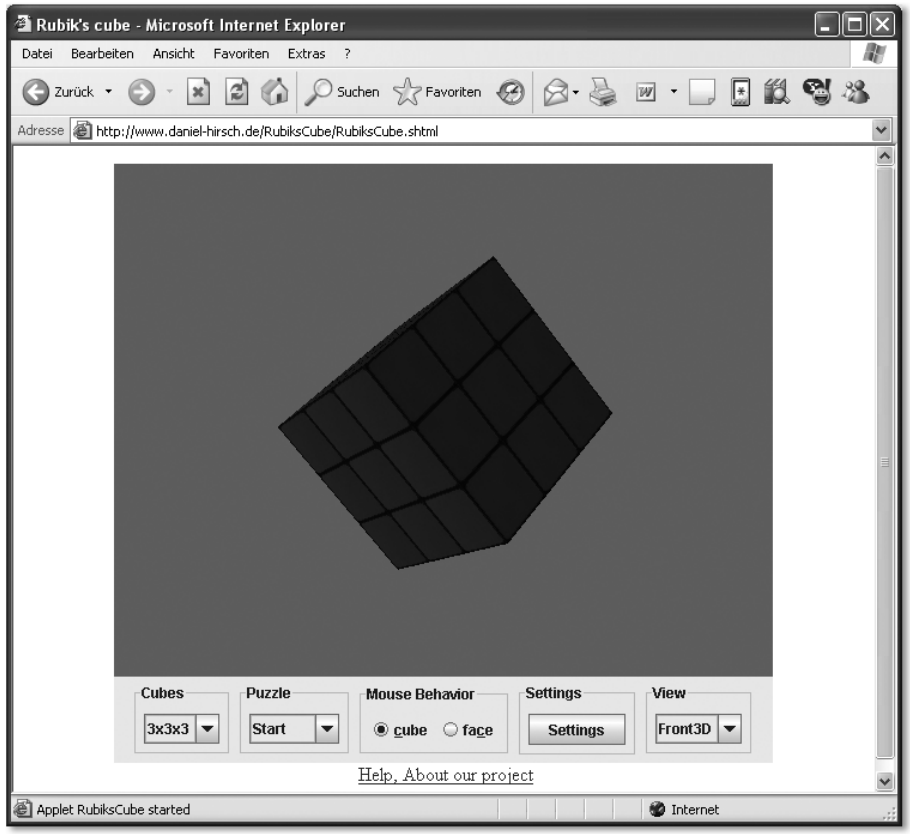

Abbildung 11.6 Der berühmte Rubic´s Cube als Java 3D-Applet

Eine andere, sehr beeindruckende Demonstration der Fähigkeiten von Java 3D ist *Fly Through*,[12] für die Sie auch die Quelltexte herunterladen können. Durch Ausführen von *j3dfly-1_0-win.exe* starten Sie einen Setup-Assistenten, der Sie

12 http://java.sun.com/products/java-media/3D/flythrough.html

durch die Installation der Demoanwendung führt. Um *Fly Through* zu starten, öffnen Sie bitte *j3dfly.bat* im Installationsverzeichnis. Anschließend müssen Sie durch Anklicken der Schaltfläche **Load** Geometriedaten laden.

Danach können Sie per Klicken und gleichzeitigem Schieben der Maus durch die Landschaft wandern.

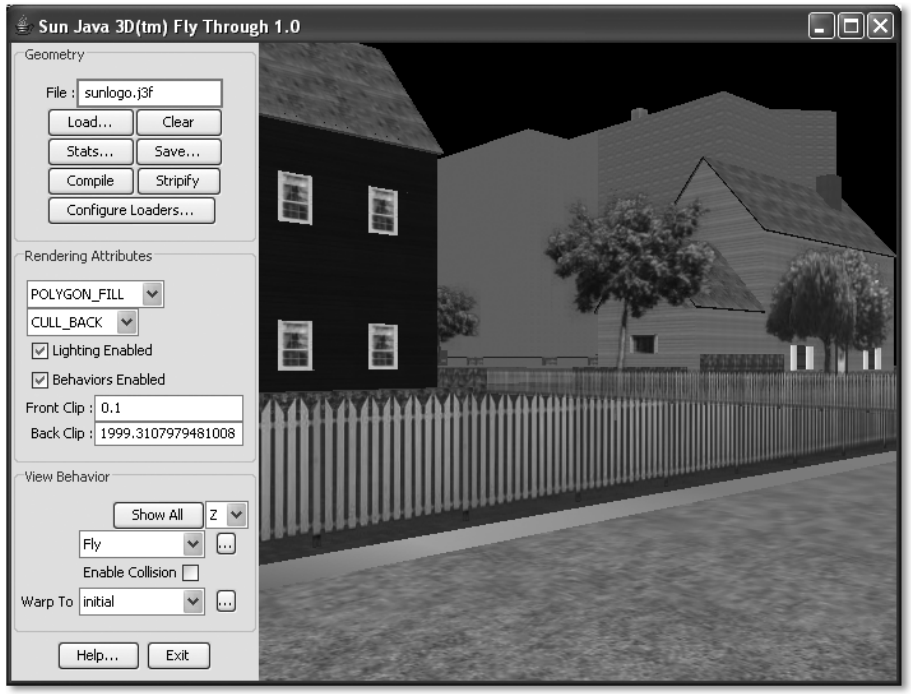

Abbildung 11.7 Die Demo-Anwendung Fly Through

11.4.3 Virtuelle Welten mit Java 3D selbst erstellt

Java 3D ist eine sehr mächtige Klassenbibliothek: Dennoch gelingt der Einstieg überraschend schnell, wie Sie anhand des folgenden Beispiels *Java3DDemo1* sehen können. Bevor wir uns seinen Quelltext ansehen, werfen Sie bitte einen Blick auf die Bildschirmausgabe in Abbildung 11.8, die das Programm zeigt.

Das Programm erzeugt zwei geometrische Körper, eine Kugel und einen Zylinder, die von einer Lichtquelle angestrahlt werden.

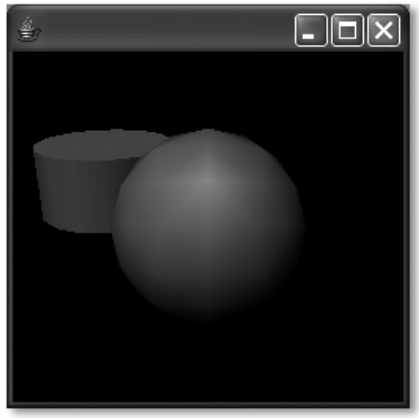

Abbildung 11.8 Java3DDemo1

```
package javafuerwindows.java3d;
import com.sun.j3d.utils.geometry.Cylinder;
import com.sun.j3d.utils.geometry.Sphere;
import com.sun.j3d.utils.universe.SimpleUniverse;
import javax.media.j3d.BoundingSphere;
import javax.media.j3d.BranchGroup;
import javax.media.j3d.DirectionalLight;
import javax.media.j3d.Transform3D;
import javax.media.j3d.TransformGroup;
import javax.vecmath.Color3f;
import javax.vecmath.Point3d;
import javax.vecmath.Vector3f;

public class Java3DDemo1 {
    public Java3DDemo1() {
        SimpleUniverse universe = new SimpleUniverse();
        BranchGroup group = new BranchGroup();
        Transform3D t = new Transform3D();
        TransformGroup tg = new TransformGroup();
        Cylinder cylinder = new Cylinder(0.4f, 0.4f);
        Vector3f vector = new Vector3f(-.6f, .3f , -.4f);
        t.rotX(0.4f);
        t.setTranslation(vector);
        tg.setTransform(t);
        tg.addChild(cylinder);
        group.addChild(tg);
```

```
        Sphere sphere = new Sphere(0.5f);
        group.addChild(sphere);
        Color3f light1Color = new Color3f(0f, 1.0f, 1.0f);
        BoundingSphere bounds = new BoundingSphere(
                                    new Point3d(0.0,0.0,0.0),
                                    100.0);
        Vector3f light1Direction = new Vector3f(2.0f,
                                              -4.0f, -8.0f);
        DirectionalLight light1 = new DirectionalLight(
                                light1Color, light1Direction);
        light1.setInfluencingBounds(bounds);
        group.addChild(light1);
        universe.getViewingPlatform().
                              setNominalViewingTransform();
        universe.addBranchGraph(group);
    }

    public static void main(String[] args) {
        new Java3DDemo1();
    }
}
```

Listing 11.6 Java3DDemo1.java

Einstieg in das Programm ist das Erzeugen eines so genannten Universums, das durch eine Instanz der Klasse `com.sun.j3d.utils.universe.SimpleUniverse` repräsentiert wird. Diesem Universum werden zwei Objekte, eine Kugel sowie ein Zylinder hinzugefügt. Ohne weiteres Zutun werden sie im Zentrum des Universums positioniert. Um ein Objekt, beispielsweise den Zylinder, an einer anderen Stelle erscheinen zu lassen, werden die Klassen `javax.media.j3d.Transform3D` und `javax.media.j3d.TransformGroup` verwendet. Erstere beschreibt die Art der auszuführenden Positionsänderung, `TransformGroup` kombiniert das zu bewegende Objekt mit `Transform3D`. Unser Universum enthält auch eine Lichtquelle, die durch die Klasse `javax.media.j3d.DirectionalLight` definiert wird. Für die Lichtquelle wird angegeben, welche Farbe sie hat, in welche Richtung sie strahlt sowie ihre Reichweite.

Natürlich kann diese knappe Einführung nur ein erster Einstieg sein. Wenn Sie sich ausführlicher mit den faszinierenden Möglichkeiten von Java 3D beschäftigen möchten, finden Sie beispielsweise auf **java3d.j3d.org** zahlreiche Tutorials, Tipps und Tricks.

Auch Suns Webseiten[13] zu Java 3D sind eine riesige Fundgrube für technische Artikel, Quelltext-Schnipsel, Hilfestellungen bei technischen Problemen und FAQs. Besonders lehrreich ist das englischsprachige *Java 3D API Tutorial*[14], das Sun auch als Komplettpaket[15] mit allen Kapiteln zum Download anbietet.

13 http://java.sun.com/products/java-media/3D/index.jsp
14 http://java.sun.com/developer/onlineTraining/java3d/
15 http://java.sun.com/developer/onlineTraining/java3d/javaa3d.zip

12 Kommunikation

12 Kommunikation

Datenaustausch über Computergrenzen hinweg wird immer wichtiger. Dieses Kapitel zeigt Ihnen deshalb, wie Sie in Java sowohl kabelgebunden als auch drahtlos Kontakt zu Computern, Mobiltelefonen und Organizern herstellen können.

In Zeiten von Bluetooth, WLAN und USB treten ältere kabelgebundene Übertragungsmethoden, beispielsweise über die serielle RS232-Schnittstelle oder den parallelen IEEE1284-Port zunehmend in den Hintergrund. Längst werden die meisten Drucker über den USB-Anschluss mit dem Computer verbunden oder sie sind gar Teil eines Netzwerks. Und immer mehr Anwender verabschieden sich von Ihren Modems, weil die mit diesen Geräten möglichen Übertragungsgeschwindigkeiten modernen Ansprüchen nicht mehr gerecht werden. Dennoch kann es nötig sein, auf die beiden betagten Anschlüsse zuzugreifen. So werden beispielsweise unzählige Messgeräte über diese Schnittstellen mit dem Computer verbunden. Außerdem basieren moderne Kommunikationsprotokolle oftmals auf der (kabelgebundenen) seriellen Übertragung. Deshalb stelle ich Ihnen zunächst die *Java Communications API* vor, die Ihnen den plattformunabhängigen Zugriff auf RS232- und IEEE1284-Schnittstellen ermöglicht. Im weiteren Verlauf dieses Kapitels erfahren Sie, wie Sie in Java mittels Bluetooth Verbindungen zu drahtlosen Endgeräten herstellen können. Schließlich wird erläutert, wie Ihre Programme mit der Internet-Telefonie-Software *Skype* kommunizieren und die Funktionen so genannter Instant Messaging-Anwendungen nutzen.

12.1 Die Java Communications API

Die *Java Communications API* wurde erstmals 1997 veröffentlicht. Derzeit existieren zwei Versionen der Spezifikation: Die diesem Kapitel zugrunde liegende Version 2.0 sowie eine um einige Funktionen erweiterte Version 3.0, die allerdings nur für Solaris und Linux, nicht aber für Windows verfügbar ist.

12.1.1 Installation

Nachdem Sie die *Java Communications API Specification* für Windows[1] heruntergeladen haben, können Sie das Archiv *javacomm20-win32.zip* in ein beliebiges Verzeichnis entpacken. Anschließend kopieren Sie bitte *comm.jar* in das Erweiterungsverzeichnis Ihres Java Development Kits, beispielsweise *C:\Pro-*

1 http://java.sun.com/products/javacomm/downloads/index.html

gramme\Java\jdk1.5.0\jre\lib\ext. Die Datei *win32com.dll* kopieren Sie bitte in das Verzeichnis *x86* unterhalb des Erweiterungsverzeichnisses. Die Datei *javax.comm.properties* schließlich gehört in das *lib*-Verzeichnis, also eine Ebene über dem Erweiterungsverzeichnis.

> Achten Sie bitte genau darauf, das »richtige« *lib*-Verzeichnis zu verwenden. Andersfalls findet die *Java Communications API* keine Schnittstellen.

Ausführliche Informationen zum Java-Erweiterungsmechanismus und der Installation von Klassenbibliotheken finden Sie in Kapitel 1, *Installation und Konfiguration*.

12.1.2 Ermitteln der verfügbaren Ports

Um Ihnen die grundsätzliche Verwendung der *Java Communications API* vor Augen zu führen, folgt hier das Programm *CommDemo1*, das alle verfügbaren Schnittstellen auf dem Bildschirm ausgibt. Sehen Sie sich zunächst bitte den vergleichsweise kurzen Quelltext an.

```
package javafuerwindows.com;
import java.util.Enumeration;
import javax.comm.CommPortIdentifier;

public class CommDemo1 {
    public static void main(String [] args) {
        Enumeration portList =
               CommPortIdentifier.getPortIdentifiers();
        while (portList.hasMoreElements()) {
            CommPortIdentifier  portId =
                (CommPortIdentifier) portList.nextElement();
            String strPortType = "unbekannt";
            if (portId.getPortType() ==
                CommPortIdentifier.PORT_SERIAL) {
                strPortType = "seriell";
            } else if (portId.getPortType() ==
                       CommPortIdentifier.PORT_PARALLEL) {
                strPortType = "parallel";
            }
            String strOwner = "verfügbar";
            if (portId.isCurrentlyOwned() == true) {
                strOwner = portId. getCurrentOwner();
```

```
            }
            System.out.println(portId.getName() + ":" +
                               strPortType + "," + strOwner);
        }
    }
}
```

Listing 12.1 CommDemo1.java

In Abbildung 12.1 sehen Sie den Aufruf und die Bildschirmausgabe des Programms. Es meldet die beiden seriellen Anschlüsse *COM3* und *COM4* sowie zwei parallele Schnittstellen *LPT1* und *LPT2*. Der Anschluss *COM3* ist durch das Programm *CommDemo2* belegt.

Abbildung 12.1 Ausgabe des Programms CommDemo1

Die statische Methode `getPortIdentifiers()` der Klasse `javax.comm.CommPortIdentifier` liefert eine Aufzählung von `CommPortIdentifier`-Objekten. `getPortType()` ermittelt den Anschlusstyp, der durch die Konstanten `PORT_SERIAL` und `PORT_PARALLEL` repräsentiert wird. Mittels `isCurrentlyOwned()` können Sie feststellen, ob die Schnittstelle durch eine andere Anwendung belegt ist. Deren Name lässt sich ggf. mit `getCurrentOwner()` herausfinden.

12.1.3 Senden und Empfangen über die serielle Schnittstelle

Beispiel *CommDemo2* fußt auf der Tatsache, dass die meisten der derzeit verkauften Rechner über ein eingebautes Modem verfügen. Da es über einen so genannten COM-Port angesprochen wird, lässt es sich sehr leicht mit der *Java Communications API* abfragen. Bitte sehen Sie sich auch hier zunächst den Quelltext an.

```
package javafuerwindows.com;
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
```

```
import javax.comm.CommPortIdentifier;
import javax.comm.SerialPort;
import javax.comm.SerialPortEvent;
import javax.comm.SerialPortEventListener;

public class CommDemo2 implements SerialPortEventListener {
    private InputStream inputStream;

    public CommDemo2() {
        String strIN = "COM3";
        String strMessage = "ati1\r\n";
        try {
            CommPortIdentifier comport =
                CommPortIdentifier.getPortIdentifier(strIN);
            if ((comport != null) &&
                    (comport.getPortType() ==
                    CommPortIdentifier.PORT_SERIAL)) {
                System.out.println("öffne " +
                                   comport.getName());
                SerialPort serialPort =  (SerialPort)
                            comport.open("CommDemo2", 2000);
                serialPort.addEventListener(this);
                serialPort.notifyOnDataAvailable(true);
                serialPort.setSerialPortParams(9600,
                                    SerialPort.DATABITS_8,
                                    SerialPort.STOPBITS_1,
                                    SerialPort.PARITY_NONE);
                OutputStream outputStream =
                               serialPort.getOutputStream();
                outputStream.write(strMessage.getBytes());
                inputStream = serialPort.getInputStream();
            }
        } catch (Exception e) {
            System.err.println(e);
        }
    }

    public void serialEvent(SerialPortEvent serialPortEvent)
    {
        if (serialPortEvent.getEventType() ==
```

```
                SerialPortEvent.DATA_AVAILABLE) {
            try {
                int data;
                int len = inputStream.available();
                if (len > 0) {
                    byte [] buf = new byte [len];
                    inputStream.read(buf);
                    System.out.println(new String(buf));
                }
            } catch (IOException e) {
            }
        }
    }

    public static void main(String [] args) {
        new CommDemo2();
    }
}
```

Listing 12.2 CommDemo2.java

Das Programm ermittelt nicht wie *CommDemo1* eine Aufzählung der verfügbaren Schnittstellen, sondern versucht durch Aufruf von `CommPortIdentifier.getPortIdentifier(strIN)` gezielt einen Port, der durch `strIN` festgelegt wird, zu belegen und mittels `open()` zu öffnen. Die Methode `setSerialPortParams()` konfiguriert die Schnittstelle. Das Senden und Empfangen von Daten erfolgt über Klassen der Typen `java.io.InputStream` bzw. `java.io.OutputStream`. Um Daten, die von der Gegenstelle gesendet wurden, lesen zu können, ist es ratsam, mit `addEventListener()` einen `SerialPortEventListener` zu registrieren. Dieses Interface besteht aus der Methode `public void serialEvent(SerialPortEvent serialPortEvent)`, innerhalb der sich die Art des aufgetretenen Ereignisses durch `getEventType()` erfragen lässt. Wichtig ist, das Informieren über ankommende Daten durch den Aufruf von `notifyOnDataAvailable(true)` zu aktivieren.

Die *Java Communications API* erlaubt die einfache Realisierung serieller und paralleler Datenübertragungen. Aufgrund ihrer grundsätzlichen Plattformunabhängigkeit (neben der Windows-Version stehen Portierungen für Solaris und Linux zur Verfügung) empfiehlt sie sich nicht nur für die Ansteuerung von spezieller Hardware (Messgeräte, Robotersteuerungen oder Arbeitshilfen für Sehbehinderte), sondern kann auch als Basis für die Implementierung moderner Übertragungstechniken dienen.

Der nächste Abschnitt erläutert, wie Sie in Ihrem Java mittels Bluetooth Daten austauschen können. Um diese Technologie verwenden zu können, benötigen Sie einen *Bluetooth Stack*. Zwei solche Stacks sind vollständig in Java implementiert und setzen auf der *Java Communications API* auf.

12.2 Bluetooth

Der Industriestandard *Bluetooth*[2] gestattet die drahtlose Vernetzung von mobilen Kleingeräten wie Handys und Organizern, aber auch von Computern und deren Peripherie, beispielsweise Drucker, Tastaturen und Mäuse. Maximal acht Geräte schließen sich dabei zu einem so genannten *Piconet* zusammen, wobei ein *Master* die Kommunikation koordiniert. Grundlage des Datenaustauschs ist zum einen eine weltweit eindeutige 48 Bit lange Seriennummer oder *Bluetooth-Adresse*, die ein Gerät eindeutig identifiziert, sowie zum anderen der Versand und Empfang von Nachrichten. Beispielsweise initiiert der Master die Kommunikation, indem er eine *Inquiry*-Nachricht aussendet und alle Geräte in Reichweite ermittelt.

Die Kommunikation mit Bluetooth-fähiger Hardware, das Einrichten von sicheren Verbindungen und die Anzeige von Geräten in Reichweite werden von einem *Bluetooth-Stack* übernommen. Bevor Microsoft das Service-Pack 2 für Windows XP veröffentlicht hat, wurde der in Abbildung 12.2 gezeigte *Widcomm-Stack* am häufigsten eingesetzt, da er den meisten Bluetooth-Adaptern beiliegt. Mittlerweile verwenden aber viele Anwender den Windows-eigenen Stack, der allerdings weniger Funktionen bietet als das Widcomm-Pendant. Beispielsweise lassen sich derzeit bestimmte Geräteklassen nicht ansteuern. Abbildung 12.3 zeigt den Einstellungsdialog von Microsofts Bluetooth-Stack, der über das Modul *Bluetooth-Geräte* der *Systemsteuerung* erreichbar ist.

Um mit Java Kontakt zu Bluetooth-Geräten aufzunehmen, bieten sich zwei Möglichkeiten an. Einerseits kann eine Klassenbibliothek auf einem bereits vorhandenen Bluetooth-Stack aufsetzen. Vertreter dieser Gattung stelle ich Ihnen in den Abschnitten 12.2.3 und 12.2.4 vor. Andererseits ist es möglich, selbst einen kleinen Stack zu realisieren. Diesen Weg geht das Projekt *Harald*, das Sie in Abschnitt 12.2.1 kennen lernen.

2 http://www.bluetooth.com/

Abbildung 12.2 Die Bluetooth-Umgebung des Widcomm-Stacks

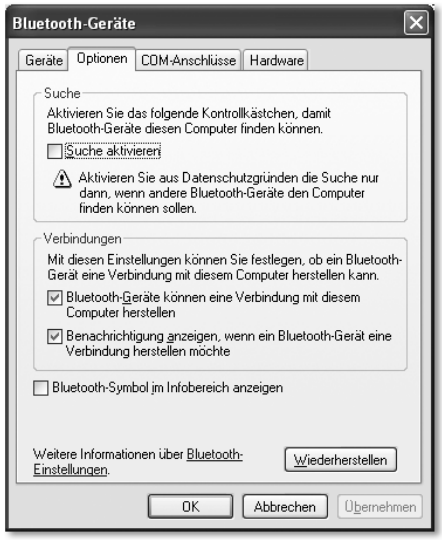

Abbildung 12.3 Der Einstellungsdialog des Microsoft Bluetooth-Stacks

Natürlich kann im Rahmen dieses Buches nur ein erster Einstieg in die Kommunikation mit Bluetooth vermittelt werden. Wenn Sie sich ausführlicher mit der Materie befassen möchten, finden Sie im englischsprachigen Werk *Bluetooth Application Programming with the JAVA APIs*[3] umfangreiches Informationsmaterial. Ebenfalls sehr interessant ist die unter **www.javabluetooth.com** erreichbare Website, die zahlreiche Informationen rund um Java und Bluetooth bereithält.

12.2.1 Harald

Harald ist ein kleiner, vollständig in Java realisierter Bluetooth-Stack, der auf der *Java Communications API* aufsetzt. Die Bibliothek lässt sich nicht nur unter Windows nutzen, sondern auf jedem System, für das eine Implementierung der Klassen des `javax.comm`-Pakets existiert. *Harald* wurde am *Department of Automatic Control*[4] der Universität Lund entwickelt und kann kostenlos eingesetzt und weitergegeben werden. Falls Sie sich fragen, wie wohl der Name zustande kommt: Es handelt sich um eine Anspielung auf den dänischen König Harald I. Blauzahn Gormson[5], der schon für Bluetooth selbst Namensgeber war. Um *Harald* in eigenen Programmen verwenden zu können, müssen Sie zunächst die Quelltexte der Bibliothek übersetzen. Die aktuellste Version finden Sie auf der Begleit-CD zum Buch. Eine etwas ältere Fassung können Sie von der Projekt-Homepage[6] herunterladen. Entpacken Sie die Datei *harald_1.1.zip* bitte in ein beliebiges Verzeichnis. Das mitgelieferte *Makefile* sorgt dafür, dass Klassendateien und Hilfstexte in eigenen Unterverzeichnissen generiert werden. Da *make* ein klassisches UNIX-Tool ist, über das Sie wahrscheinlich nicht auf Ihrem Rechner verfügen, stelle ich Ihnen im Folgenden eine kleine Batch-Datei vor, die im Wesentlichen die gleichen Befehle ausführt wie das *Makefile*. Die Quelltexte der Beispiele werden allerdings nicht übersetzt.

```
mkdir .\classes
mkdir .\doc
javac se\lth\control\harald\*.java -d .\classes
cd .\classes
jar -cf harald.jar se\lth\control\harald\*.class
cd ..
javadoc -breakiterator
        -sourcepath . se.lth.control.harald -d .\doc
```

Listing 12.3 make.bat

Kopieren Sie *make.bat* bitte in das Verzeichnis, in das Sie *Harald* entpackt haben und das auch das *Makefile* enthält, und starten Sie es durch Doppelklick. Das Skript wird daraufhin zwei Verzeichnisse anlegen, *classes* und *doc*. In *classes* wird anschließend *Harald.jar* erzeugt. Um die Klassenbibliothek in Ihren Programmen verwenden zu können, müssen Sie den Klassenpfad entsprechend erweitern oder das *.jar*-Archiv im Erweiterungsverzeichnis ablegen. Nähere

3 C.Bala Kumar, Paul Kline, Tim Thompson: Bluetooth Application Programming with the JAVA APIs, Morgan Kaufmann Publishers
4 http://www.control.lth.se/
5 http://de.wikipedia.org/wiki/Harald_Blauzahn
6 http://www.control.lth.se/~johane/harald/

Informationen hierzu finden Sie in Kapitel 1, *Installation und Konfiguration*. Die Klassen-Dokumentation finden Sie im Verzeichnis *doc*.

Im Folgenden zeige ich Ihnen das Programm *HaraldDemo1*, das einen kurzen Einblick in die bereitgestellten Klassen und ihre Verwendung gibt.

```
package javafuerwindows.com;
import java.util.ArrayList;
import se.lth.control.harald.HCI;
import se.lth.control.harald.Local;
import se.lth.control.harald.RS232Driver;

public class HaraldDemo1 {
    public static void main(String [] args) {
        String strCOM = "COM6";
        try {
            RS232Driver driver = new RS232Driver(strCOM);
            Local local = new Local(driver);
            local.setDebug(true);
            local.init();
            ArrayList discoveredUnits = local.inquiry(5);
            for (int i = 0; i < discoveredUnits.size();
                            i++) {
                System.out.println(discoveredUnits.get(i));
            }
            System.exit(0);
        } catch (Exception e) {
            System.err.println(e);
        }
    }
}
```

Listing 12.4 HaraldDemo1.java

Zunächst wird eine Instanz der Klasse `se.lth.control.harald.RS232Driver` erzeugt. Der Konstruktor erhält den Namen des zu verwendenden Anschlusses. Sie wird für die Kommunikation mit der Bluetooth-Hardware verwendet, die eine Instanz der Klasse `se.lth.control.harald.Local` kapselt. Der eigentliche Datenaustausch findet dann zwischen diesem `Local`-Objekt und einer Instanz von `se.lth.control.harald.Remote` statt.

Weitergehende Informationen zur Nutzung ihrer Methoden lassen sich der Klassendokumentation sowie den mitgelieferten Beispielen entnehmen. *Harald*

setzt also auf eigens entwickelte Klassen. Falls Sie bereits mit Java auf Ihrem Handy oder Organizer Kontakt hatten, fragen Sie sich möglicherweise, ob es nicht standardisierte Klassen für die Nutzung von Bluetooth unter Java gibt.

12.2.2 Java APIs for Bluetooth

JSR-82[7] definiert APIs, die Java-Anwendungen den Zugriff auf Bluetooth-fähige Geräte ermöglichen. Allerdings spricht die Spezifikation stets von der *Java 2 Platform, Micro Edition*. Bei der Definition der beteiligten Pakete, Interfaces und Klassen standen also mobile Kleingeräte im Vordergrund, nicht aber Anwendungen für Desktop-basierte Systeme, die im Kontext der *Java 2 Platform, Standard Edition* betrieben werden. Somit existiert auch keine Referenz-Implementierung im klassischen Sinne des Java Community Process. Selbstverständlich können aber Klassenbibliotheken die in der Spezifikation beschriebenen Funktionalitäten bereitstellen. Zwei Vertreter dieser Gattung stelle ich Ihnen in den Abschnitten 12.2.3 und 12.2.4 vor. Anwendungen, die Bluetooth JSR-82-konform ansprechen, sind damit sehr wahrscheinlich mit beliebigen Bluetooth-APIs ablauffähig, sofern diese den in JSR-82 festgelegten Funktionsumfang realisieren.

12.2.3 Bluecove

Die *Bluecove*[8] API setzt auf dem Microsoft Bluetooth-Stack auf und implementiert derzeit Teile des JSR-82. Er ist Open Source und kann von der Projekthomepage[9] kostenlos heruntergeladen werden. Die zum Zeitpunkt der Drucklegung aktuelle Version finden Sie auch auf der Begleit-CD zum Buch. Um *Bluecove* in Ihren eigenen Programmen nutzen zu können, müssen Sie dafür sorgen, dass die Java-Laufzeitumgebung die benötigten Dateien findet. Nach dem Entpacken des Archivs kopieren Sie daher bitte die Datei *intelbth.dll* in das Verzeichnis x86 des Erweiterungsverzeichnisses Ihres Development Kits, die Datei *BlueCove-20050514.jar* in das Erweiterungsverzeichnis. Ausführliche Informationen über die Installation von Klassenbibliotheken finden Sie in Kapitel 1, *Installation und Konfiguration*.

Anhand eines kleinen Beispiels folgend nun einige Klassen der *Java APIs for Bluetooth*. Sehen Sie sich hierzu bitte den Quelltext des Programms *BluecoveDemo1* an.

7 http://www.jcp.org/en/jsr/detail?id=82
8 http://bluecove.sourceforge.net/
9 http://sourceforge.net/projects/bluecove/

```
package javafuerwindows.com;
import java.io.IOException;
import javax.bluetooth.BluetoothStateException;
import javax.bluetooth.DeviceClass;
import javax.bluetooth.DiscoveryAgent;
import javax.bluetooth.DiscoveryListener;
import javax.bluetooth.LocalDevice;
import javax.bluetooth.RemoteDevice;
import javax.bluetooth.ServiceRecord;

public class BluecoveDemo1 implements DiscoveryListener {
    public BluecoveDemo1() {
        try {
            LocalDevice ld = LocalDevice.getLocalDevice();
            // System.out.println(ld.getBluetoothAddress());
            DiscoveryAgent agent = ld.getDiscoveryAgent();
            agent.startInquiry(DiscoveryAgent.GIAC, this);
        } catch (BluetoothStateException e) {
            System.err.println(e);
        }
    }

    public static void main(String [] args) {
        new BluecoveDemo1();
    }

    /*
     * DiscoveryListener interface
     */

    public void deviceDiscovered(RemoteDevice remoteDevice,
                                 DeviceClass deviceClass) {
        try {
            System.out.println(
                        remoteDevice.getFriendlyName(true));
        } catch (IOException e) {
        }
    }

    public void inquiryCompleted(int param) {
```

```
    }

    public void serviceSearchCompleted(int param,
                                       int param1) {
    }

    public void servicesDiscovered(int param,
                           ServiceRecord[] serviceRecord) {
    }
}
```

Listing 12.5 BluecoveDemo1.java

Die meisten Klassen gehören zu dem Paket `javax.bluetooth`, das neben `javax.obex` Teil der *Java APIs for Bluetooth* ist. Im ersten Schritt wird eine Instanz der Klasse `LocalDevice` ermittelt. Dies geschieht durch Aufruf von `LocalDevice.getLocalDevice()`. Sie erlaubt die Steuerung des lokalen Bluetooth-Gerätes. So lassen sich mit Hilfe eines *Discovery-Agenten* andere Bluetooth-Geräte in Reichweite ausfindig machen. Eine Instanz der Klasse `DiscoveryAgent` kann durch die Methode `getDiscoveryAgent()` ermittelt werden. Mit `startInquiry()` initiieren Sie den Suchvorgang.

Bluecove funktioniert derzeit nur mit dem Windows-eigenen Bluetooth-Stack. Falls Sie auf Ihrem Rechner die Widcomm-Software einsetzen, können Sie eine Implementierung der Firma Avetana GmbH verwenden, die Thema des folgenden Abschnitts ist.

12.2.4 Avetana Bluetooth-Stack

Der Bluetooth-Stack der Firma Avetana GmbH[10] steht in Versionen für Windows, Linux und Mac OS X zur Verfügung. Die Software ist kostenpflichtig, allerdings bietet der Hersteller eine 14 Tage gültige Demolizenz an, mit der Sie den Stack vor einem dauerhaften Einsatz erproben können. Um die Testversion zu erhalten, müssen Sie sich bei Avetana registrieren. Sie erhalten daraufhin eine E-Mail, die den Link auf die herunterladbare Demoversion enthält. Die Windows-Version unterstützt sowohl den Windows-eigenen als auch den Widcomm-Stack. Die Software erkennt automatisch, welcher Stack verwendet werden soll. Gegebenenfalls können Sie aber mit der System-Property `avetanabt.stack` gezielt vorgeben, mit welchem Sie arbeiten möchten. Um den Avetana-Stack in Ihren Programmen verwenden zu können, ist keine geson-

10 http://www.avetana-gmbh.de/avetana-gmbh/produkte/jsr82.xml

derte Installation erforderlich. Es genügt, den Klassenpfad um das *.jar*-Archiv zu erweitern oder die Datei in das Java-Erweiterungsverzeichnis zu kopieren.

Der Stack stellt auch das in den *Java APIs for Bluetooth* enthaltene Paket `javax.obex` zur Verfügung und realisiert damit das so genannte *Object Exchange Protocol*, das in Verbindung mit Handys und Organizern oft für den Versand und Empfang von Kalenderdaten, Bildern und Audiodateien verwendet wird.

12.3 Instant Messaging

Instant Messaging-Programme erfreuen sich zunehmender Beliebtheit. Die ihnen zugrunde liegende Idee ist, während der Arbeit am PC mit Freunden, Familienmitgliedern oder Geschäftspartnern kommunizieren zu können. Dabei hat die Zahl der von den Instant Messaging-Anwendungen angebotenen Formen der Kommunikation stetig zugenommen. Waren zunächst nur reine Textnachrichten möglich, kam schnell die Übertragung von Bildern und Videosequenzen hinzu. Mittlerweile sind sie zum Teil zu vollständigen Internet-Telefonie-Lösungen gereift. Den Trend frühzeitig erkannt hat auch Microsoft und liefert den *Windows Messenger* als Bestandteil von Windows XP aus. Die Parallel-Entwicklung *MSN Messenger* ist optisch ansprechender und mit weitaus mehr Funktionen ausgestattet. Beide Programme setzen allerdings für die Nutzung eine Mitgliedschaft in Microsofts *Passport*-Netzwerk voraus. Das Einrichten eines Benutzerkontos ist allerdings für praktisch alle IM-Programme obligatorisch. Dies gilt auch für den *Yahoo Messenger*. Aus der Sicht des Java-Entwicklers ist diese Anwendung besonders interessant, weil sich das ihr zugrunde liegende Kommunikationsprotokoll aus Java heraus nutzen lässt.

12.3.1 Yahoo Instant Messenger Protocol for Java

Die unter der GNU General Public Licence stehende Klassenbibliothek *jYMSG* implementiert das *Yahoo Messenger Protocol*, das die technische Grundlage für den *Yahoo Messenger* bildet. Sie kann kostenlos von der Projekt-Homepage[11] heruntergeladen werden. Die zum Zeitpunkt der Drucklegung aktuelle Version finden Sie auch auf der Begleit-CD zum Buch. Um die Bibliothek in eigenen Programmen nutzen zu können, müssen Sie die *.jar*-Dateien dem Klassenpfad Ihrer Anwendung hinzufügen. Alternativ können Sie die Dateien auch in das Java-Erweiterungsverzeichnis kopieren. In diesem Fall sind keine Änderungen am Klassenpfad erforderlich.

11 http://jymsg9.sourceforge.net/index.html

Der Zugriff auf grundlegende Messaging-Funktionen ist sehr einfach zu realisieren wird. Werfen Sie hierzu bitte einen Blick auf das unten abgedruckte Programmfragment, das den An- und Abmeldevorgang beim Yahoo Messenger Service sowie den Versand einer kurzen Textnachricht zeigt. Den vollständigen Quelltext des Programms *JYMSGDemo1* finden Sie auf der Begleit-CD.

```
public class JYMSGDemo1 implements SessionListener {
    public JYMSGDemo1() {
        Session session = new Session();
        session.addSessionListener(this);
        try {
            session.login("mein_user_name",
                          "mein_passwort");
            session.sendMessage("empfaenger",
                                "Hallo, ein Test");
            JOptionPane.showMessageDialog(null,
                                          "Bitte warten",
                                          "INFO",
                                JOptionPane.PLAIN_MESSAGE);
            session.logout();
        } catch (Exception e) {
            System.err.println(e);
        }
        System.exit(0);
    }

    public static void main(String [] args) {
        new JYMSGDemo1();
    }

    /*
     * SessionListener interface
     */
    public void buzzReceived(SessionEvent sessionEvent) {
    }
```

Praktisch die gesamte Kommunikation wird über eine Instanz der Klasse `ymsg.network.Session` abgewickelt. Die Methoden `login()` und `logout()` sorgen für die An- und Abmeldung beim Yahoo Messenger Service.

Abbildung 12.4 Hinweis, wenn sich ein Kontakt anmeldet

In diesem Fall erhalten Ihre Online-Kontakte den in Abbildung 12.4 abgebildeten Hinweis. Mit der Methode `sendMessage()` können Sie Textnachrichten versenden. Wie diese beim Empfänger angezeigt werden, sehen Sie in Abbildung 12.5.

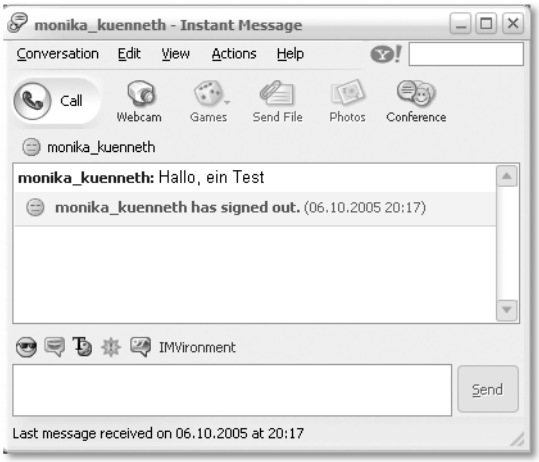

Abbildung 12.5 Eingang einer Textnachricht

Das Nachrichtenfenster zeigt dem Empfänger einer Mitteilung auch an, wenn sich der Versender der Nachricht nach dem Senden der Nachricht bereits wieder abgemeldet hat. Die Klassenbibliothek *jYMSG* erlaubt auf sehr komfortable Weise den Versand sowie das Empfangen von Textnachrichten. Sie bietet unzählige interessante Einsatzmöglichkeiten, beispielsweise das im englischsprachigen Artikel *Receive Application Errors via Yahoo Messenger* beschriebene Melden von Anwendungsfehlern.[12]

12.3.2 Windows Messenger

Für die beiden Programme *Windows Messenger* und *MSN Messenger* gibt es meiner Kenntnis nach derzeit keine Java-Klassenbibliothek, um auf ihre Funktionen zuzugreifen. Das *Component Object Model*, das ausführlich in Kapitel 9,

12 http://today.java.net/pub/a/today/2005/09/16/errors-via-yahoo-im.html

Java-COM-Brücken, vorgestellt wurde, erlaubt aber eine im Prinzip sprachunabhängige Nutzung der beiden Instant Messaging-Anwendungen.

Ein kleines Demo-Programm veranschaulicht Ihnen die grundsätzliche Vorgehensweise für Java. Microsoft hat die durch den *Windows Messenger* bereitgestellten Interfaces und ihre Funktionen auf **msdn.microsoft.com** dokumentiert[13]. Um in Java mit Anwendungen zu kommunizieren, die das Component Object Model nutzen, benötigen Sie eine geeignete Klassenbibliothek, beispielsweise *com4j* oder *JACOB*. Beide Pakete haben Sie in Kapitel 9 kennen gelernt. Für *MessengerDemo1* verwende ich *JACOB*, da diese Bibliothek ohne die Generierung von Wrapper-Klassen aus Typbibliotheken auskommt.

```
package javafuerwindows.com;
import com.jacob.activeX.ActiveXComponent;
import com.jacob.com.Dispatch;
import com.jacob.com.Variant;

public class MessengerDemo1 {
    public static final int MISTATUS_OFFLINE = 0x0001;
    public static final int MISTATUS_NLINE = 0x0002;
    private ActiveXComponent messenger;

    public MessengerDemo1() {
        messenger = new ActiveXComponent(
                            "Messenger.UIAutomation");
        if (myStatus() == MISTATUS_OFFLINE) {
            Dispatch.call(messenger, "Signin",
                          null,
                          new Variant("name@email.adr"),
                          new Variant("geheim"));
        }
        // Warten, bis wir online sind
        try {
            while (true) {
                Thread.currentThread().sleep(1000);
                if (myStatus() == MISTATUS_NLINE) {
                    break;
                }
            }
```

13 http://msdn.microsoft.com/library/default.asp?url=/library/en-us/winmessenger/winmessenger/messenger_entry.asp

```
            Dispatch.call(messenger,
                          "InstantMessage",
                          "email@domain.com");
        } catch (InterruptedException e) {
        }
        if (myStatus() == MISTATUS_NLINE) {
            Dispatch.call(messenger, "Signout");
        }
    }

    private int myStatus() {
        Variant status = Dispatch.get(messenger,
                                      "MyStatus");
        return status.toInt();
    }

    public static void main(String [] args) {
        new MessengerDemo1();
    }
}
```

Listing 12.6 MessengerDemo1.java

MessengerDemo1 zeigt die Verwendung der Funktionen `Signin()`, `Signout()`, `MyStatus()` und `InstantMessage()`. Im Prinzip ist ihre Nutzung nicht aufwändiger als die vergleichbarer Methoden aus *jYMSG*. Falls Sie eine Anwendung auf dieser Basis implementieren möchten, bietet es sich aber an, API-Klassen zu definieren, die den Aufruf der Funktionen mittels *JACOB* kapseln, um die Les- und Wartbarkeit der eigentlichen Anwendung zu erhöhen.

12.3.3 Skype

Die Internet-Telefonie-Anwendung *Skype*[14] hat aufgrund ihrer Verfügbarkeit auf allen wichtigen Plattformen zahlreiche Anhänger. Die Funktionen des Skype-Clients, beispielsweise das Tätigen eines Anrufs oder das Versenden einer Textnachricht, lassen sich über eine definierte Programmierschnittstelle[15] ansprechen. *ActiveS*[16] der Firma KHAOSLABS legt eine ActiveX-Schicht über diese Schnittstelle und macht *Skype* damit über das *Component Object Model* steuerbar. Die Software erlaubt damit indirekt auch Java-Entwicklern den

14 http://www.skype.com/
15 http://share.skype.com/developer_zone/documentation/documentation/
16 http://www.khaoslabs.com/actives.php

Zugriff auf die Skype-API. Die Verwendung der Bibliothek ist kostenlos. Die Installationsdatei kann von der Homepage des Herstellers heruntergeladen werden.

Im Folgenden zeige ich Ihnen anhand eines kleinen Beispiels, wie Sie in Java die Namen Ihrer befreundeten Kontakte anzeigen können. Ausführliche Informationen über das Component Object Model und die Klassenbibliothek JACOB finden Sie in Kapitel 9, *Java-COM-Brücken*.

```
package javafuerwindows.jacob;
import com.jacob.activeX.ActiveXComponent;
import com.jacob.com.Dispatch;
import com.jacob.com.Variant;

public class SkypeDemo1 {
    public SkypeDemo1() {
        ActiveXComponent skype = new ActiveXComponent(
                                    "SkypeAPI.Access");
        System.out.println(Dispatch.call(skype,
                                        "ConnectionStatus"));
        Dispatch iUserCollection = Dispatch.call(skype,
                               "GetFriendList").toDispatch();
        int anz = Dispatch.get(iUserCollection,
                                "Count").toInt();
        for (int i = 1; i <= anz; i++) {
            Dispatch iUser = Dispatch.call(iUserCollection,
                                "Item",
                                new Variant(i)).toDispatch();
            System.out.println(Dispatch.call(iUser,
                                              "FullName"));
        }
    }

    public static void main(String [] args) {
        new SkypeDemo1();
    }
}
```

Listing 12.7 SkypeDemo1.java

Wenn Sie *SkypeDemo1* das erste Mal starten, sehen Sie den in Abbildung 12.6 gezeigten Dialog **Anwendungsfreigabe**, mit dem Sie einer Anwendung

(*java.exe*) den Zugriff auf *Skype* gestatten müssen. Das Programm zeigt unter anderem die Verwendung der Funktion `ConnectionStatus()`. Die Online-Dokumentation zu *ActiveS* gibt Auskunft über die möglichen Rückgabewerte. `GetFriendList()` liefert einen Zeiger auf das Interface `IUserCollection`, dessen Funktionen Ihnen den Zugriff auf Ihre befreundeten Kontakte erlauben.

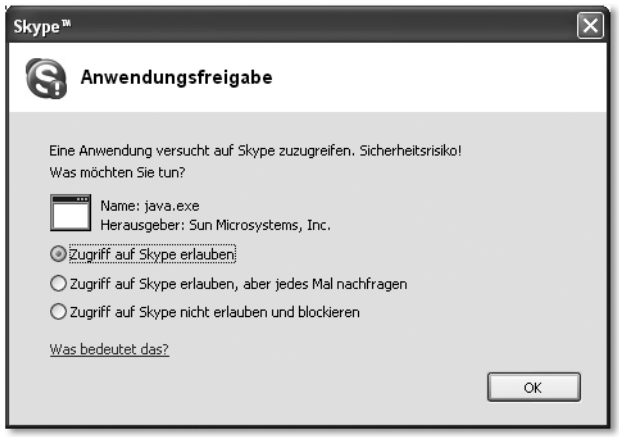

Abbildung 12.6 Die Anwendungsfreigabe in Skype

Auch *Skype* lässt sich mit geringem Aufwand durch Java fernsteuern. Wenn Sie hierfür die Klassenbibliothek *JACOB* verwenden, bietet es sich an, die Funktionsaufrufe in eigenen Klassen zu kapseln, um die Les- und Wartbarkeit Ihres Quelltextes zu verbessern.

A Installation von MySQL

MySQL ist ein äußerst mächtiges und leistungsfähiges Datenbanksystem. In diesem Anhang begleite ich Sie durch seine Installation.

Das als Open Source entwickelte relationale Datenbanksystem *MySQL* hat in den letzten Jahren zu Recht sehr viele Anhänger gefunden. Neben der Leistungsfähigkeit der eigentlichen Datenbank-Engine liegt dies sicher auch an der Verfügbarkeit von mächtigen und dennoch leicht bedienbaren Werkzeugen, beispielsweise dem *MySQL Query Browser* sowie dem *MySQL Administrator*. Wir werden diese – sowie natürlich das eigentliche Datenbanksystem – im Folgenden Schritt für Schritt installieren.

MySQL Server 4.1

Der *MySQL Server 4.1* kann von der Download-Seite[1] des Herstellers heruntergeladen werden. Sie benötigen das Archiv *Windows Essentials (x86)*. Zum Zeitpunkt der Drucklegung ist die Version 4.1.15 aktuell. Um die Installation zu starten, öffnen Sie bitte die Datei *mysql-essential-4.1.15-win32.msi* durch einen Doppelklick. Sie sehen den Willkommensdialog des Setup-Assistenten. Wechseln Sie bitte durch Anklicken der Schaltfläche **Next** auf die zweite Seite. Dort können Sie die Art der Installation bestimmen. Wählen Sie bitte **Typical** und klicken auf **Next**. Die folgende Seite *MySQL.com Sign up* bietet Ihnen die Möglichkeit, sich beim Hersteller zu registrieren. Wenn Sie dies wünschen, können Sie beispielsweise durch Anklicken von **Create a new free MySQL.com account** eine Anmeldung vornehmen. Alternativ können Sie den Installationsvorgang aber auch ohne Anmeldung fortsetzen. Wählen Sie hierzu bitte **Skip Sign-Up** und klicken anschließend auf **Next**. Nach dem Installationsvorgang meldet der Setup-Assistent **Wizard Completed**. Bevor Sie die Schaltfläche **Finish** anklicken, prüfen Sie bitte, ob **Configure the MySQL Server now** mit einem Häkchen versehen ist. Falls nicht, holen Sie dies bitte nach.

Durch das Setzen des Häkchens starten Sie den *MySQL Server Instance Configuration Wizard*, der Ihnen beim Einrichten Ihres frisch installierten MySQL Servers hilft. Sobald Sie die erste Seite dieses Assistenten sehen, klicken Sie bitte auf **Next**. Auf der nun folgenden Seite können Sie zwischen **Detailed Configuration** und **Standard Configuration** wählen. Markieren Sie bitte **Standard Configuration** und klicken auf **Next**.

1 http://dev.mysql.com/downloads/mysql/4.1.html

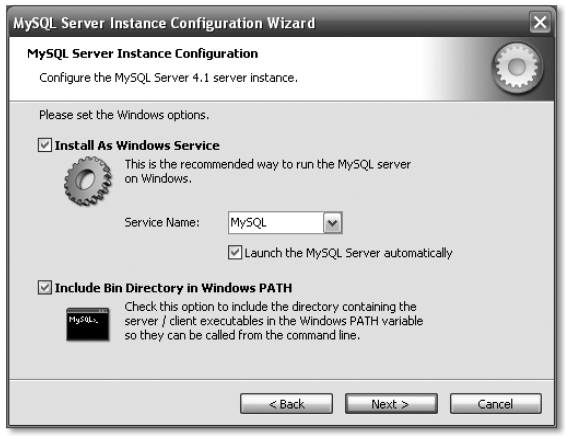

Abbildung A.1 Einrichten Ihres MySQL Servers

Stellen Sie auf der in Abbildung A.1 gezeigten Seite bitte sicher, dass sowohl **Install As Windows Service** als auch **Include Bin Directory in Windows PATH** mit einem Häkchen versehen sind. Klicken Sie anschließend auf **Next**. Sie werden nun aufgefordert, Sicherheitseinstellungen vorzunehmen. Setzen Sie bitte ein Häkchen vor **Modify Security Settings** und vergeben Sie ein Passwort. Die Optionen **Enable root access from remote machines** und **Create An Anonymous Account** sollten kein Häkchen haben.

Die folgende Seite des Konfigurationsassistenten, die Sie in Abbildung A.2 sehen, zeigt den Verlauf der Server-Konfiguration. Klicken Sie bitte auf **Finish**, um den Assistenten zu beenden. Sie haben damit Ihren MySQL Server erfolgreich installiert und eingerichtet. Um seine Funktion zu testen, können Sie beispielsweise den *MySQL Command Line Client* aufrufen, den Sie im Start-Menü finden.

Abbildung A.2 Verlauf der Server-Konfiguration

MySQL Administrator

Der *MySQL Administrator* ist ein grafisches Werkzeug zum Warten und Konfigurieren von MySQL Server-Instanzen. Sie können ihn von der Download-Seite[2] des Herstellers herunterladen. Die zum Zeitpunkt der Drucklegung des Buches aktuelle Version für Windows ist 1.1.4. Bitte starten Sie die Installation durch Doppelklick auf *mysql-administrator-1.1.4-win.msi*. Sie sehen die Willkommensseite des MySQL Administrator-Setup-Assistenten, auf der Sie bitte die Schaltfläche **Next** anklicken. Auf der folgenden Seite müssen Sie den Lizenzbestimmungen durch Anklicken von **I accept the terms in the licence agreement** zustimmen. Klicken Sie bitte auf **Next** und legen anschließend das Installationsverzeichnis fest. Bestätigen Sie auch diese Seite bitte mit **Next**. Sie haben nun die Möglichkeit, die Art der Installation zu bestimmen. Wählen Sie bitte **Complete** und klicken dann auf **Next**. Vor dem Start der Installation sehen Sie eine Zusammenfassung, die Sie durch Klick auf **Install** bestätigen. Nach dem Beenden des Installationsvorgangs meldet der Assistent **Wizard Complete**. Sie können ihn mit **Finish** schließen.

Sie können den *MySQL Administrator* über das Start-Menü aufrufen. Sie sehen dann den in Abbildung A.3 gezeigten Anmeldedialog, in dem Sie als Benutzername `root` und als Passwort das von Ihnen während der Installation des Servers gewählte Passwort eintragen.

Abbildung A.3 MySQL Administrator-Anmeldedialog

Nach dem Anmeldevorgang sehen Sie das Hauptfenster des MySQL Administrators. Die Icon-Leiste am linken Fensterrand gestattet den Zugriff auf die verschiedenen Bereiche der Server-Konfiguration. Sie können beispielsweise neue

2 http://dev.mysql.com/downloads/administrator/1.1.html

Datenbankbenutzer anlegen, Sicherungen von Datenbanken erstellen oder zurückspielen, Einsicht in so genannte Server-Logs nehmen sowie Änderungen an der Konfiguration durchführen. In Abbildung A.4 sehen Sie beispielsweise, wie Sie einen neuen Benutzer anlegen können.

Im nächsten Abschnitt zeige ich Ihnen ein weiteres wichtiges Werkzeug, den *MySQL Query Browser*. Mit ihm können Sie auf sehr komfortable Art SQL-Anfragen formulieren und an einen MySQL Server senden.

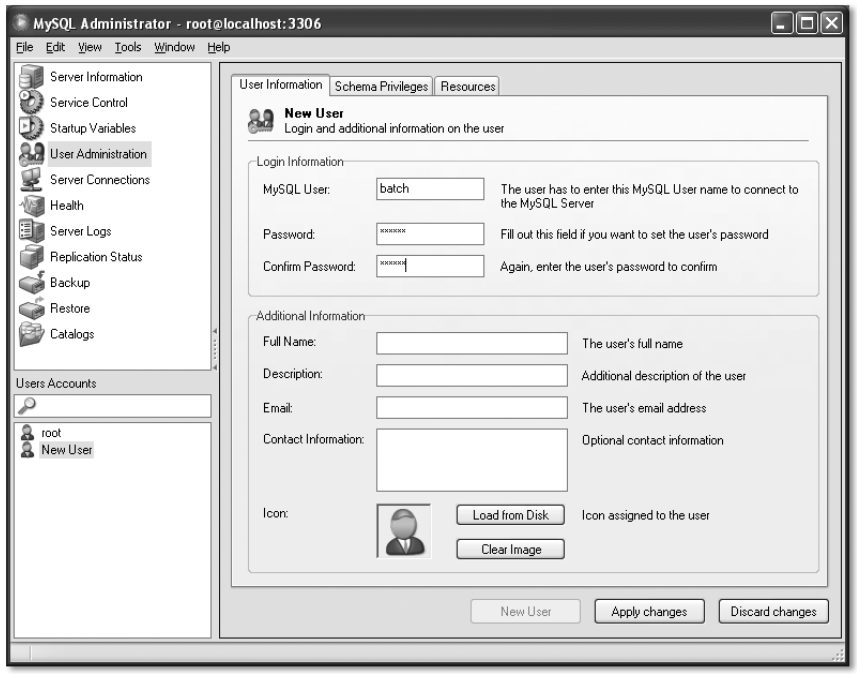

Abbildung A.4 Der MySQL Administrator

MySQL Query Browser

Der Query Browser steht auf der Download-Seite[3] des Herstellers bereit. Die zum Zeitpunkt der Drucklegung aktuelle Version für Windows ist 1.1.17. Nach dem Herunterladen der Datei *mysql-query-browser-1.1.17-win.msi* können Sie diese durch Doppelklick öffnen. Die Installation verläuft analog zu der des MySQL Administrators. Nach dem Bestätigen der Lizenzvereinbarung können Sie ein Zielverzeichnis auswählen. Achten Sie bitte darauf, auch hier als Art der Installation **Complete** zu wählen. Nachdem Sie eine Zusammenfassung der auszuführenden Schritte bestätigt haben, beginnt die eigentliche Installation.

3 http://dev.mysql.com/downloads/query-browser/1.1.html

Wenn Sie den *MySQL Query Browser* über das Start-Menü aufrufen, sehen Sie den in Abbildung A.5 gezeigten Anmeldedialog. Solange Sie noch keine zusätzlichen Datenbankbenutzer angelegt haben, tragen Sie auch hier wie schon beim *MySQL Administrator* als Benutzername `root` sowie dessen Passwort ein.

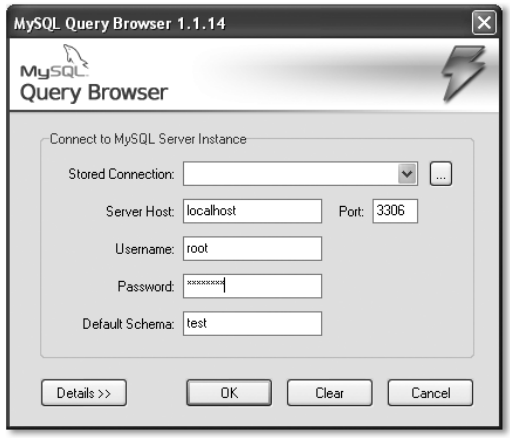

Abbildung A.5 Der Anmeldedialog des MySQL Query Browsers

In Abbildung A.6 sehen Sie das Hauptfenster des Query Browsers. Es zeigt die vorhandenen Datenbank-Schemata, eine SQL-Abfrage (`select * from personen`) sowie deren Ergebnis.

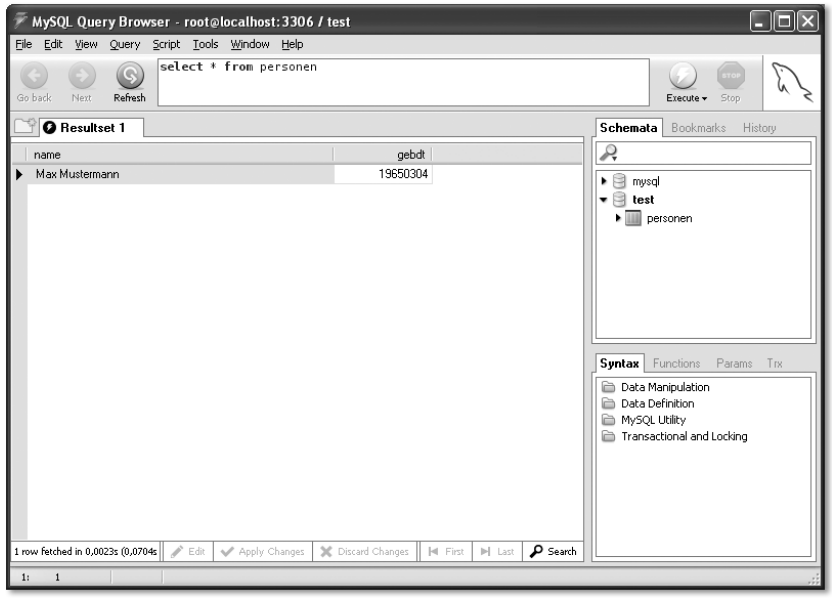

Abbildung A.6 MySQL Query Browser

B Die Begleit-CD

Einlegen und loslegen. Die Begleit-CD enthält neben den vollständigen Beispielen viele Programme und Tools, die Ihnen beim Schreiben Ihrer Java-Anwendungen unschätzbare Dienste leisten werden.

In diesem Anhang möchte ich Ihnen den Inhalt der Begleit-CD vorstellen sowie einige Tipps zum effizienten Umgang mit ihr geben. Zunächst richte ich aber nochmals meinen tief empfundenen Dank an alle, die mir die Erlaubnis gegeben haben, ihre Programme auf der CD zu veröffentlichen.

Die CD enthält auf der obersten Ebene zwei Verzeichnisse, *Software* und *Quelltexte*. *Software* enthält die Installationsdateien vieler Anwendungen, die ich Ihnen im Verlauf des Buches vorstelle. *Quelltexte* enthält die von mir verfassten Beispiele, die Ihnen zum Teil vollständig, zum Teil in Auszügen in gedruckter Form vorliegen.

Das Verzeichnis Software

Das *Software*-Verzeichnis übernimmt die Struktur des Buches. Sie finden deshalb die einzelnen Kapitel als Unterordner. Jedes Programm erscheint wiederum als Unterverzeichnis des entsprechenden Kapitel-Ordners. Auf diese Weise finden Sie mit wenigen Mausklicks zur gewünschten Anwendung. In Abbildung B.1 sehen Sie diese Verzeichnisstruktur beispielhaft für Kapitel 2, *Entwicklungswerkzeuge*.

Die Datei *index.html*, die Sie in *Software* finden, listet nochmals alle Programme nach Kapiteln sortiert auf und enthält Links auf die Installationsarchive sowie – soweit vorhanden – auf die Hersteller- bzw. Projekt-Homepages. Sie können die Anwendungen auf diese Weise sehr leicht installieren und haben zudem die Möglichkeit, sich über etwaige Updates zu informieren.

Das Verzeichnis Quelltexte

Das Verzeichnis *Quelltexte* enthält zwei Unterverzeichnisse, *java* und *class*. Der Aufbau dieser beiden Ordner ist identisch, wobei *class* Klassendateien (*.class*) enthält und *java* die dazu gehörenden Quelltexte (*.java*). Deren Unterordner orientieren sich nicht direkt an den Kapitelnamen des Buches, sondern entsprechen den Paketnamen, die Sie aus den im Buch abgedruckten Listings kennen.

Auf diese Weise können Sie die fertig übersetzten Programme problemlos aus der *Eingabeaufforderung* heraus starten. Sie müssen hierzu *java.exe* nur die

Option `-cp` übergeben, wobei Sie den vollständigen Pfad des Verzeichnisses *class* übergeben, also beispielsweise

```
java -cp H:\Quelltexte\class
     javafuerwindows.java3d.Java3DDemo1
```

Falls eine Demoanwendung eine Klassenbibliothek benötigt, muss diese im Erweiterungsverzeichnis der zu verwendenden Laufzeitumgebung vorliegen. Alternativ können Sie aber auch die benötigten *.jar*-Archive bei `-cp` angeben.

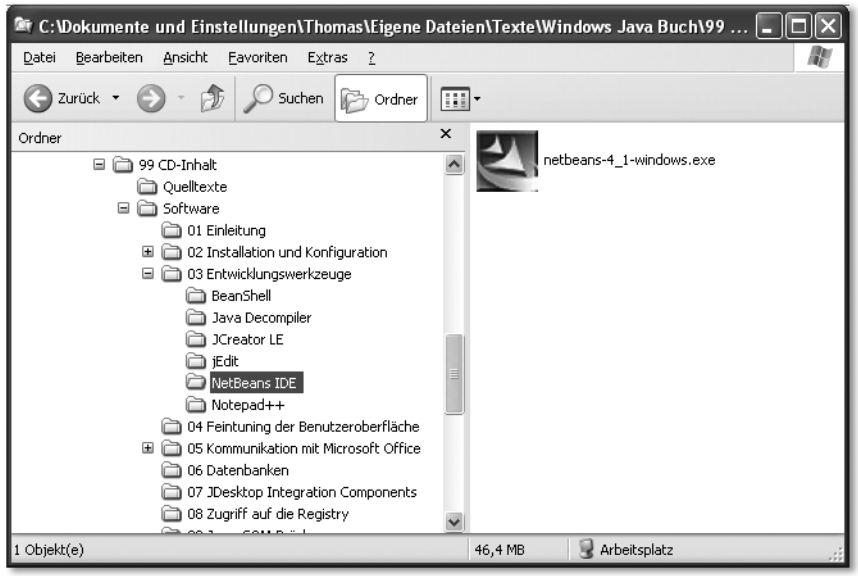

Abbildung B.1 Die Verzeichnisstruktur der Begleit-CD

Möchten Sie eines meiner Beispiele selbst übersetzen, können Sie die betreffende Quelltext-Datei aus dem Verzeichnis *java* auf Ihre Festplatte kopieren. Allerdings schlage ich Ihnen vor, stattdessen in Ihrer Entwicklungsumgebung ein zunächst leeres Projekt anzulegen. Kopieren Sie nun den kompletten Inhalt des Verzeichnisses *java* auf Ihre Festplatte und importieren dann die Quelltexte in Ihr neues Projekt. *NetBeans* bietet hierzu unter *New Project* die Option **Java Project with Existing Sources**. Abbildung B.2 zeigt Ihnen die dritte Seite des entsprechenden NetBeans-Assistenten. Um ein Verzeichnis auszuwählen, klicken Sie bitte auf **Add Folder**.

Um Ihnen die Zuordnung der einzelnen Pakete meiner Beispiele zu den entsprechenden Kapiteln zu erleichtern, finden Sie im Verzeichnis *Quelltexte* die Datei *inhalt.html*, die die einzelnen Beispiele kapitelweise auflistet.

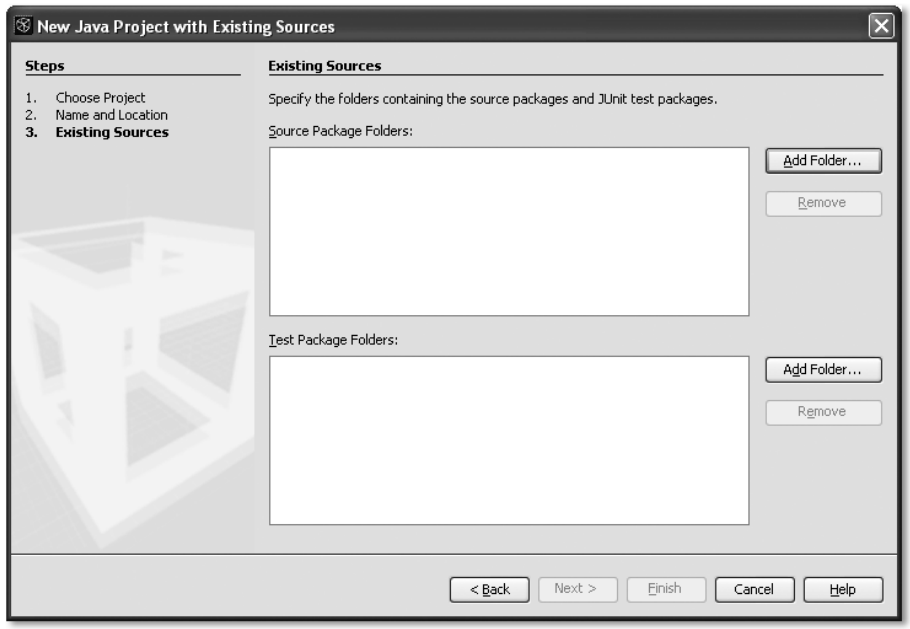

Abbildung B.2 Anlegen eines neuen Projekts mit vorhandenen Quellen

Index