Thomas Künneth

Java für Windows

Galileo Computing

Auf einen Blick

	Einleitung	13
1	Installation und Konfiguration	21
2	Entwicklungswerkzeuge	43
3	Feintuning der Benutzeroberfläche	65
4	Zusätzliche Komponenten für die Benutzeroberfläche	89
5	Kommunikation mit Microsoft Office	109
6	Datenbanken	135
7	Die JDesktop Integration Components	163
8	Zugriff auf die Registry	187
9	Java-COM-Brücken	213
10	Deployment	237
11	Multimedia	257
12	Kommunikation	277
A	Installation von MySQL	299
B	Die Begleit-CD	305
	Index	309

Inhalt

Einleitung 13

1 Installation und Konfiguration 23

1.1 SDK und JRE ... 23
1.1.1 Überlegungen vor der Installation ... 25
1.1.2 Die Installation des SDK ... 25
1.1.3 Ergänzende Installationsmaßnahmen ... 29
1.2 Zusätzliche Bibliotheken ... 34
1.2.1 Der Klassenpfad ... 34
1.2.2 Das Java-Erweiterungsverzeichnis ... 35
1.3 Parallele Nutzung mehrerer Java-Installationen ... 37
1.3.1 Wahl einer bestimmten Java-Version ... 37
1.3.2 Konsequenzen aus der Nutzung mehrerer Java-Versionen ... 39
1.3.3 Gemeinsame Verwendung von Bibliotheken ... 40

2 Entwicklungswerkzeuge 45

2.1 Text- und Programmeditoren ... 45
2.1.1 Notepad++ ... 46
2.1.2 jEdit ... 48
2.2 Integrierte Entwicklungsumgebungen ... 50
2.2.1 JCreator ... 50
2.2.2 NetBeans IDE ... 52
2.3 Sonstige Werkzeuge ... 57
2.3.1 Microsoft SyncToy ... 57
2.3.2 JAD ... 60
2.3.3 Die BeanShell ... 61
2.3.4 TKClassInspector ... 63

3 Feintuning der Benutzeroberfläche 67

3.1 Swing und das Konzept Pluggable Look and Feel ... 67
3.1.1 Unterschiede zwischen AWT und Swing ... 67
3.1.2 Das Pluggable Look and Feel-Konzept ... 68
3.1.3 Das Java Look and Feel ... 71
3.1.4 Die Klassen UIManager, LookAndFeel und LookAndFeelInfo ... 72

3.2 Die Datei swing.properties ... 75
3.2.1 Aufbau ... 75
3.2.2 TKPLAFUtility ... 77
3.2.3 Hinweise zum Umgang mit swing.properties ... 78
3.3 Das Windows Look and Feel ... 78
3.3.1 Unterschiede zum Vorbild ... 79
3.3.2 Das WinLAF-Projekt ... 81
3.4 Weitere interessante Look and Feels ... 82
3.4.1 Die OfficeLnFs Look and Feels ... 83
3.4.2 JGoodies Looks ... 85
3.5 Icon Sets ... 86
3.5.1 Java look and feel Graphics Repository ... 86
3.5.2 Icon Collection auf Sourceforge ... 87

4 Zusätzliche Komponenten für die Benutzeroberfläche 91

4.1 Nachrichtenfenster mit JToaster ... 91
4.1.1 Download und Installation ... 92
4.1.2 JToaster in der Praxis ... 92
4.2 L2FProd.com Common Components ... 94
4.2.1 Tipp des Tages-Dialoge ... 94
4.2.2 Verzeichnisse auswählen ... 97
4.2.3 Aufgaben mit JTaskPane ... 98
4.2.4 Outlook-Leisten ... 100
4.3 MDI-Anwendungen ... 103
4.3.1 MDI-Anwendungen in Swing ... 104
4.3.2 AceMDI in der Praxis ... 106

5 Kommunikation mit Microsoft Office 111

5.1 Excel ... 111
5.1.1 Jakarta POI – ein Überblick ... 112
5.1.2 Erzeugen von Excel-Arbeitsmappen ... 113
5.1.3 Lesen bestehender Excel-Dokumente ... 118
5.1.4 Anzeigen von Arbeitsmappen ... 122
5.2 Word ... 122
5.2.1 Lesen vorhandener Word-Dokumente mit HWPF ... 123
5.2.2 Java Bean Word Processing ... 124
5.3 Outlook ... 127
5.3.1 Anzeigen von Outlook-Kontakten ... 127
5.3.2 Schreiben von Notizen ... 131
5.3.3 Überblick über die JOC-Klassen ... 133

5.4 Weitere Office-Komponenten ... 133
5.4.1 Access ... 133
5.4.2 OneNote ... 133

6 Datenbanken 137

6.1 Grundlagen von Datenbanken ... 138
6.1.1 Structured Query Language (SQL) ... 138
6.1.2 ODBC ... 141
6.1.3 JDBC und die JDBC-ODBC-Brücke ... 143
6.2 JDBC im praktischen Einsatz ... 143
6.2.1 MySQL ... 144
6.2.2 Microsoft Access ... 149
6.2.3 Excel als Datenbank ... 154
6.3 Werkzeuge und Bibliotheken ... 156
6.3.1 SQirreL SQL Client ... 156
6.3.2 Jackcess ... 158

7 Die JDesktop Integration Components 165

7.1 Dateitypen ... 165
7.1.1 Ermitteln von Dateitypen und Aktionen ... 166
7.1.2 Registrieren von Dateitypen und Aktionen ... 170
7.2 Die Klassen Desktop und Message ... 174
7.2.1 Zugriff auf Standardanwendungen ... 174
7.2.2 E-Mails versenden ... 176
7.3 Den Browser in eigene Programme einbetten ... 177
7.3.1 Installation ... 178
7.3.2 Die Klasse WebBrowser ... 178
7.4 Symbole im Infobereich der Taskleiste ... 182
7.4.1 Die Klassen SystemTray und TrayIcon ... 182
7.4.2 Meldungen im Infobereich ... 184

8 Zugriff auf die Registry 189

8.1 Aufbau der Registrierung ... 189
8.1.1 Registry-Datentypen ... 192
8.2 Bearbeiten der Registrierung mit JNIRegistry ... 194
8.2.1 Das Paket com.ice.jni.registry ... 194
8.2.2 Schreibender Zugriff auf die Registry ... 197
8.3 Beispiele für den Zugriff auf die Registry ... 199
8.3.1 Installierte Java-Versionen ermitteln ... 199
8.3.2 Dokumentvorlagen ... 202
8.3.3 Einbinden von Java-Programmen in die Systemsteuerung ... 207

9 Java-COM-Brücken 215

9.1 Das Component Object Model (COM) ... 215
9.1.1 Grundlagen des COM ... 216
9.1.2 Type Libraries ... 217
9.2 JACOB ... 219
9.2.1 Ausblenden und Anzeigen von Fenstern ... 220
9.2.2 Sounds abspielen ... 221
9.2.3 iTunes fernsteuern ... 222
9.2.4 Rechtschreibprüfung mit Word ... 226
9.3 com4j ... 228
9.3.1 Sprachsynthese mit der Microsoft Speech API ... 229
9.3.2 Microsoft OneNote-Importer ... 231

10 Deployment 239

10.1 Installationswerkzeuge ... 239
10.1.1 WiX ... 240
10.1.2 IzPack ... 243
10.2 Java Web Start ... 247
10.2.1 Web Start aus der Sicht des Anwenders ... 247
10.2.2 Web Start-Anwendungen erstellen ... 249
10.3 Allgemeine Tools ... 252
10.3.1 Verknüpfungen mit JShortcut ... 252
10.3.2 launch4j ... 254

11 Multimedia 259

11.1 Java Image Management Interface (Jimi) ... 259
11.1.1 Laden von Bilddateien ... 260
11.1.2 Speichern von Bilddateien ... 261
11.2 Java Image I/O ... 263
11.2.1 Lesen von Bilddateien ... 263
11.2.2 Schreiben von Bilddaten ... 264
11.3 Java Media Framework ... 266
11.3.1 Installation des JMF ... 266
11.3.2 Audio-Wiedergabe mit dem JMF ... 268
11.4 Java 3D ... 270
11.4.1 Die Installation von Java 3D ... 270
11.4.2 Test der Java 3D-Installation ... 271
11.4.3 Virtuelle Welten mit Java 3D selbst erstellt ... 273

12 Kommunikation 279

12.1 Die Java Communications API ... 279
12.1.1 Installation ... 279
12.1.2 Ermitteln der verfügbaren Ports ... 280
12.1.3 Senden und Empfangen über die serielle Schnittstelle ... 281
12.2 Bluetooth ... 284
12.2.1 Harald ... 286
12.2.2 Java APIs for Bluetooth ... 288
12.2.3 Bluecove ... 288
12.2.4 Avetana Bluetooth-Stack ... 290
12.3 Instant Messaging ... 291
12.3.1 Yahoo Instant Messenger Protocol for Java ... 291
12.3.2 Windows Messenger ... 293
12.3.3 Skype ... 295

A Installation von MySQL 299

B Die Begleit-CD 305

Index 309

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.

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.

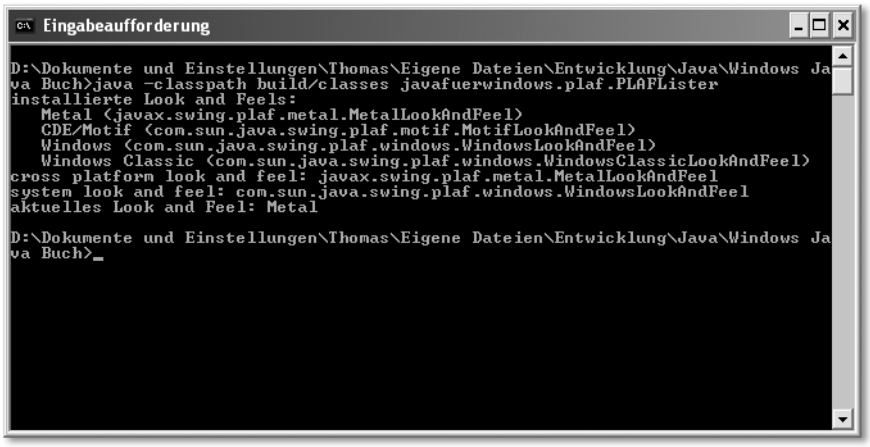

Abbildung 3.4 Ausgabe des Programms PLAFTester

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.

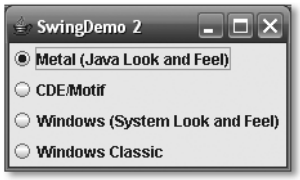

Abbildung 3.5 Das Programm SwingDemo2

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.

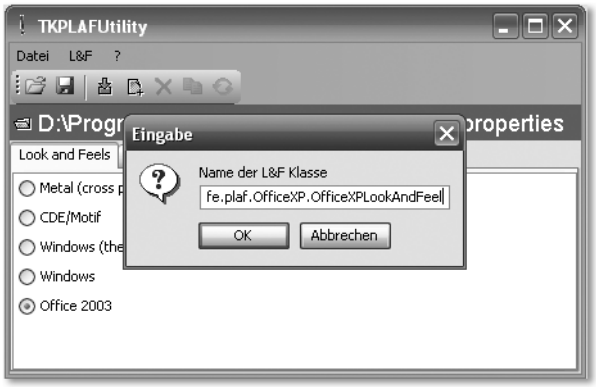

Abbildung 3.6 Hinzufügen des Office XP Look and Feels in TKPLAFUtility

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.

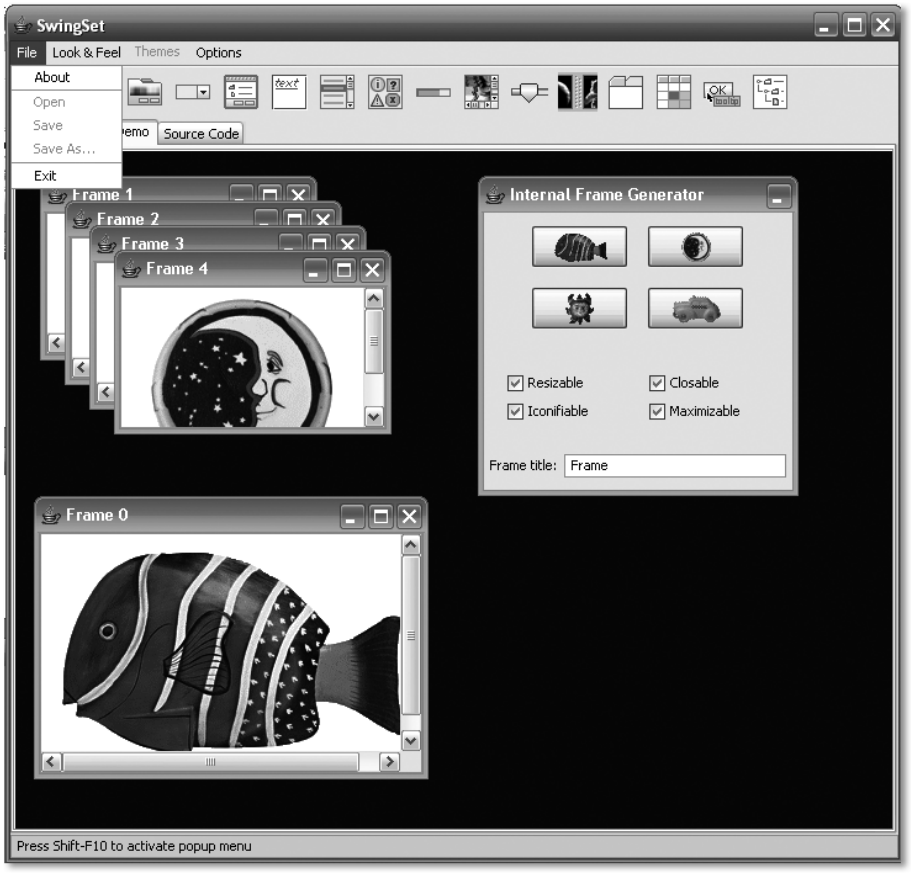

Abbildung 3.7 Die Beispiel-Anwendung SwingSet2

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.

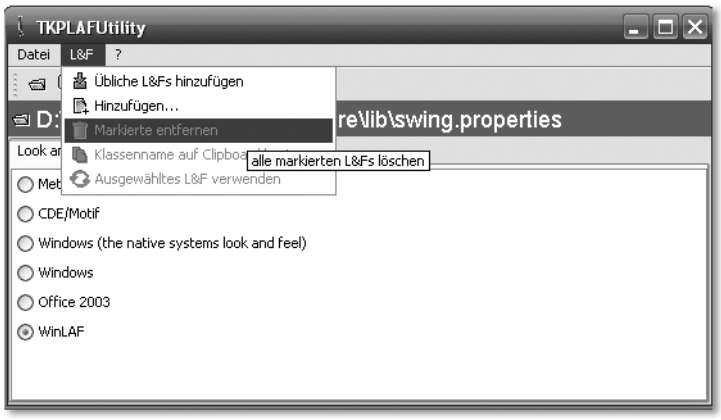

Abbildung 3.10 WinLAF stellt nicht anwählbare Menüeinträge richtig dar.

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/

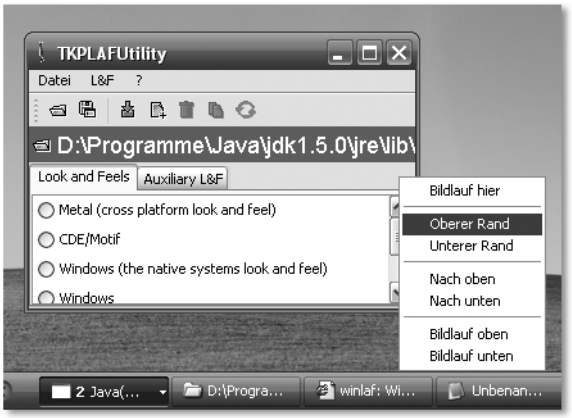

Abbildung 3.11 Fensterelemente von Swing-Anwendungen erhalten durch WinLAF ein Kontextmenü

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.

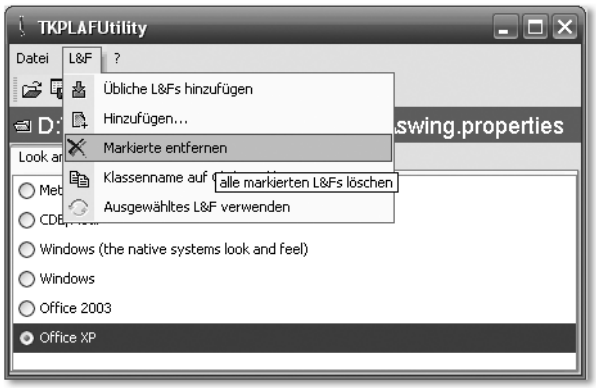

Abbildung 3.12 Das Office XP Look and Feel

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`.

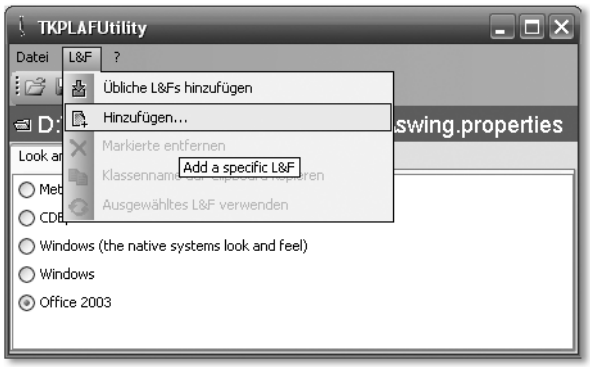

Abbildung 3.13 Das Office 2003 Look and Feel

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.

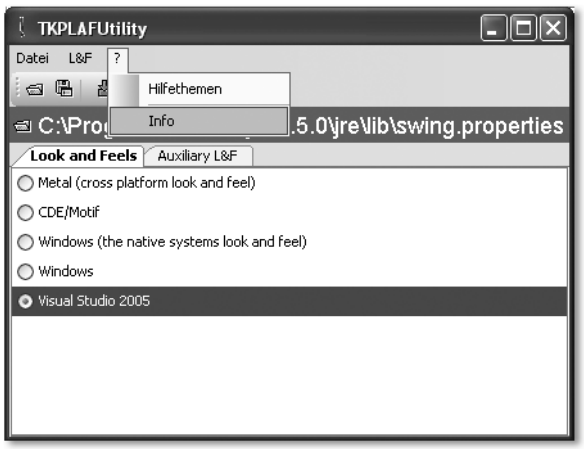

Abbildung 3.14 Das Visual Studio 2005 Look and Feel

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.

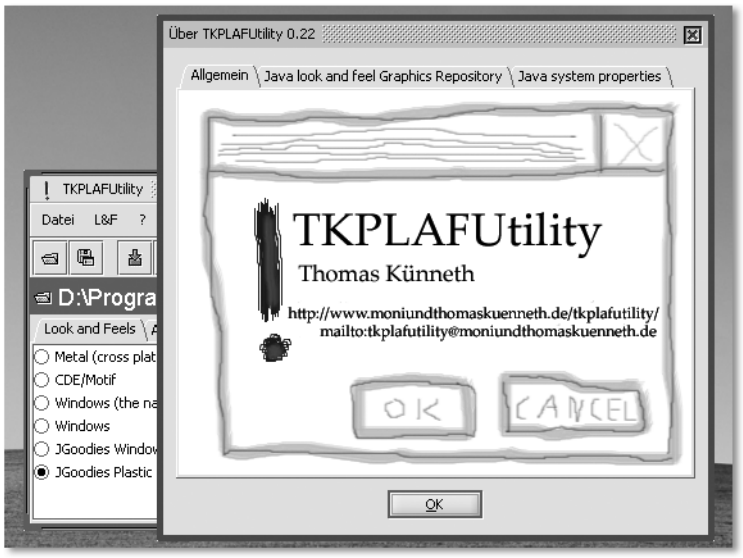

Abbildung 3.15 JGoodies Plastic 3D

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.

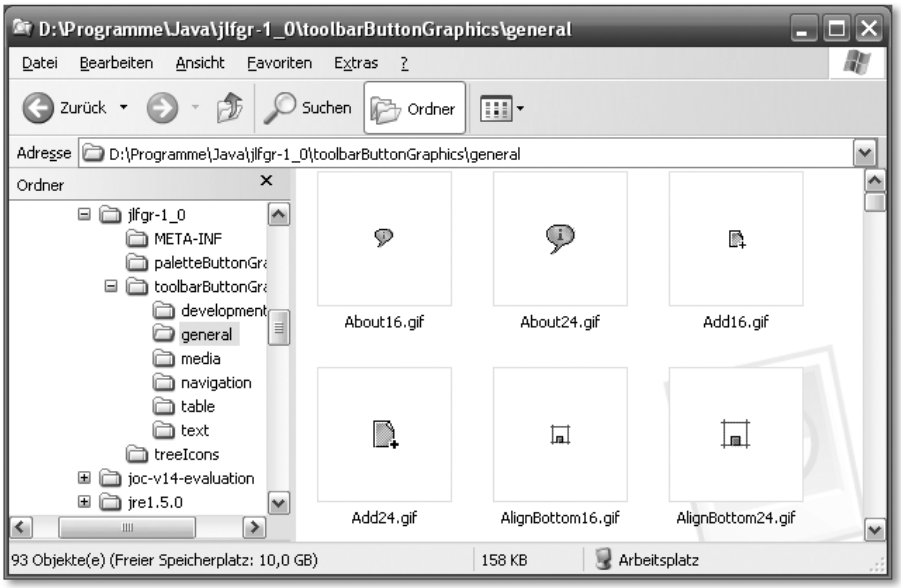

Abbildung 3.16 Das Java look and feel Graphics Repository

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.

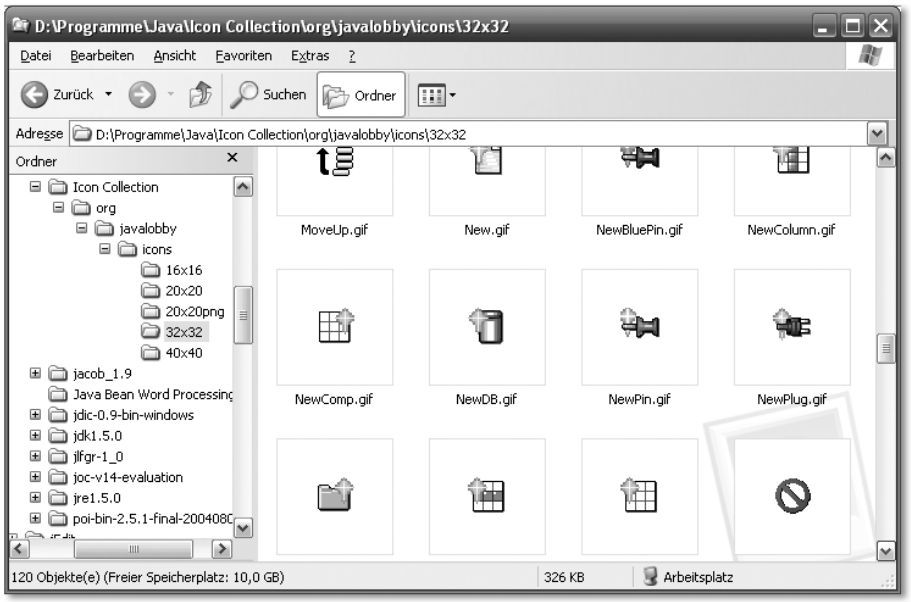

Abbildung 3.17 Dean S. Jones' Icon Collection

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

Index

.NET Framework 240
.properties 197
_import() 232

A
Abstract Wondow Toolkit 67
Access 133, 149
AccessDemo1 153, 159
AceMDI 106
Action 167–168
ActionEvent 182
Activated Intelligence 259
ActiveS 295
ActiveXComponent 221–222
add() 100, 107
addClassPath() 62
addEventListener() 283
AddRef() 217
AddRemoveDemo 207
addTab() 102
addTrayIcon() 183
Aktionen 166
Aktualisierung 30
Aktualisierungen 30
all-permissions 250
Andere Orte 98
Annotationen 228
Anwenderklassenpfad 34
Anzeigeprogramm für Java-Anwendungen 248
Apache Software Foundation 112
Application Managers 50
Arbeitsplatz 31
Association 167–168
AssociationService 167, 170
Aufgaben 98
auxiliary look and feel 77
avetana Bluetooth-Stack 290
avetanabt.stack 290

B
back() 180
BeanShell 36, 61
Befehl 168
Beschreibung 168
Bluecove 288
BluecoveDemo1 288
Bluetooth 284
Bluetooth-Stack 284, 286
BorderLayout 180
browse() 176, 178
BufferedImage 264

C
call() 222–223
candle.exe 242
CharArrayWriter 124
Class 145, 153
ClassFactory 232
ClassNotFoundException 145
CLASSPATH 34, 266
Class-Path 35, 255
Clip4Moni 182
closeKey() 196
code folding 47
com4j 228, 294
CommDemo1 280, 283
CommDemo2 281
Common Public Licence 240
CommPortIdentifier 281, 283
COM-Objekt 216
Component Object Model 216, 293, 295
COM-Port 281
Connection 146
ConnectionStatus() 297
Console 50
Count() 223
CREATE TABLE 139
createCell() 117
createCellStyle() 117
createNew() 131–132
createNewDocumentFromTemplate() 125
createRealizedPlayer() 270
createStatement() 146
createSubKey() 198
cross platform look and feel 71

D
Database 160
Datei- und Ordneraufgaben 98
Dateitypen 166, 191
Datenbank 138
Datenbankmanagementsystem 138
Datenquellen (ODBC) 141, 149, 154
Datentypen 192
Debug-Ausgaben 31
DefaultTip 96
deleteItem() 131

deleteSubKey() 199
Delphi 52, 217
Desktop 174, 176
DesktopDemo1 174
DesktopDemo2 177
Details 98
DirectionalLight 275
Discovery-Agent 290
DiscoveryAgent 290
Dispatch 222, 224
displayMessage() 184
documentCompleted() 181
Dokumentation 28
DriverManager 145, 153

E
edit() 176
Eingabeaufforderung 31, 33, 41, 45, 50, 60, 62, 240
Erweiterungsklassenpfad 34
Erweiterungsverzeichnis 280
Excel 111
ExcelDemo1 113, 155
ExcelDemo2 116
ExcelDemo3 118
exec() 127
executeQuery() 148
executeUpdate() 146, 148
exit() 130

F
Factory-Klasse 217
File 175
FiletypeDemo1 166
FiletypesDemo2 169
FiletypesDemo3 171
flushKey() 199
Fly Through 272
FolderType 131, 133
forName() 145, 153
Forté for Java 53
forward() 180

G
getActionByVerb() 168
getActionCommand() 74
getActionList() 167–168
getCellType() 122
getClassName() 73
getConnection() 145–146, 153
getCrossPlatformLookAndFeelClassName() 73
getCurrentOwner() 281
getDecoderTypes() 262
getDefaultFolder() 131, 133
getDefaultSystemTray() 183
getDescription() 167
getDirectory() 254
getDiscoveryAgent() 290
getEncoderTypes() 262
getEventType() 283
getFileExtensionAssociation() 167
getFirstCellNum() 121
getFirstRowNum() 121
getFriendList() 297
getIconFileName() 167
getImage() 261
getInstalledLookAndFeels() 73
getItems() 131–133
getLastCellNum() 121
getLastRowNum() 121
getLocalDevice() 290
getMimeTypeAssociation() 167
getName() 73
getNextRow() 161
getObject() 148
getPortIdentifier() 283
getPortIdentifiers() 281
getPortType() 281
getRuntime() 127
getSheetAt() 121
getSheetName() 117
getStringCellValue() 122
getStringValue() 196
getSystemLookAndFeelClassName() 73
getTable() 160
getTableNames() 160
getTipAt() 96
getTipCount() 96
getTopLevelKey() 196, 198
getValue() 196
Globally Unique Identifier 216
GNU Lesser General Public License 106

H
Harald 286
HaraldDemo1 287
Hauptzweige 190
heavyweight 68
HSSF 112–113
HSSFCell 117, 121
HSSFCellStyle 117
HSSFSheet 121

HSSFWorkbook 121–122
HWPF 112, 123

I
Icon Collection 87
IDispatch 217, 221–222
IITSourceCollection 223
Image 261
ImageIcon 183
ImageIO 264
Infobereich 182
Infobereich der Taskleiste 91
InputStream 283
Inquiry-Nachricht 284
INSERT INTO 139
Installation 25
Installationsverzeichnis 26–27
Installationswerkzeuge 239
Instant Messaging 291
InstantMessage() 295
integrierte Entwicklungsumgebung 50
Interface 216
Internet Explorer 27
Internet-Telefonie 291
invoke() 217, 221
isBackEnabled() 180
isCurrentlyOwned() 281
isEditable() 175
isForwardEnabled() 180
IShellDispatch4 218
ISimpleImporter 232
isPrintable() 175
Item() 225
ItemsCollection 131–133
ItemsIterator 131
ItemType 133
Iterator 168
iterator() 131
iTunes 222
ITunesDemo 222
IUnknown 217
IUserCollection 297
IVoicePlay 221
IzPack 156, 243

J
J2SE Development Kit 25
J2SE Runtime Environment 24
Jackcess 158
JACOB 219, 294
Jakarta 112
jarsigner.exe 250
Java 3D 270
Java Advanced Imaging 263
Java APIs for Bluetooth 288
Java Bean Word Processing 124
Java Communications API 279, 281, 286
Java Database Connectivity 143
Java Decompiler 60
Java Development Kit 24, 29, 45, 51
Java Foundation Classes 67
Java Image I/O 263
Java Look and Feel 71
Java look and feel Graphics Repository 86
Java Media Framework 266
Java Network Launching Protocol 249
Java Outlook Connector 127
Java Plug in 27, 30
Java Web Start 29, 247
java.ext.dirs 35, 40
java.home 35
JAVA_BIN 32, 39
JAVA_HOME 36
Java3DDemo1 273
Java-Basisverzeichnis 25
Java-Laufzeitumgebung 23
JavaVersionen 200
javaws.exe 248
JButtonBar 102
JBWPDemo1 124
JCreator 50
JDBC-ODBC-Bridge 143
JdbcOdbcDriver 153
JDesktopPane 105
JDirectoryChooser 97
JDK 24
jEdit 46, 48, 63
JFileChooser 97
JFrame 105
JGoodies 85
JGoodies Looks 85
JIIODemo1 263
JIIODemo2 264
Jimi 259, 261
JimiDemo1 260
JimiDemo2 262
JInternalFrame 105
JMFDemo1 268
JNIRegistry 194, 197
JNLP-Deskriptor 249
JOCDemo1 127
JOptionPane 222
JOutlookBar 101

JPanel 122
JPopupMenu 183
JRE 24
JShellLink 254
JShortcut 252
JShortcutDemo1 254
JSR-015 263
JSR-82 288
JTabbedPane 101
JTaskPane 99
JTaskPaneGroup 99
JTipOfTheDay 94, 96
JToaster 92
JToasterDemo1 92
Junction 41
jYMSG 291, 295
JYMSGDemo1 292

K
keyElements() 201
keystore 250
keytool.exe 250
Klassenpfad 34
Konsole 31

L
L2FProd.com Common Components 94
L2FProdDemo2 97
L2FProdDemo3 99
L2FProdDemo4 102
Laufzeitumgebung 23, 48
launch4j 254
light.exe 242
lightweight 68
List 168, 177
Local 287
LocalDevice 290
login() 292
logout() 292
Look and Feel 68

M
mail() 176–177
Main-Class 35, 254–255
Makrorecorder 47
Manager 270
Manifest.mf 29, 35
manifest.mf 255
Manifest-Datei 35
Map 161
Master 284
MDIFrame 107
MDIFrameListener 107
MDIView 107
MediaLocator 270
Message 176–177
MessengerDemo1 294–295
Metal 71
MetalTheme 73
Microsoft Speech API 231
MotifLookAndFeel 73
Mozilla 27
MSN Messenger 91, 291, 293
Multiple Document Interface 103
MySQL 140, 144
MySQL Administrator 145
MySQL Command Line Client 141
MySQL Connector/J 144
MySQL Query Browser 146
MySQLDemo1 144
MySQLDemo2 147
MyStatus() 295

N
navigateToPage() 232
NetBeans 52
NetBeans Developer X2 53
NetBeans IDE 53
NetBeans Mobility Pack 53
Netscape 27
NeueDatei2 205
next() 148
Notepad++ 46
notifyOnDataAvailable() 283

O
Object Exchange Protocol 291
Ocean 71
OfficeLnFs 83
OneNote 133, 231
OneNoteDemo 232
Open Database Connectivity 141
open() 160, 176, 283
OpenFile() 222
openSubKey() 196, 198, 201
optionale Pakete 35
Ordneroptionen 29, 37
os.arch 36
Outlook 127, 131
Outlook Express 226
OutlookAppointment 133
OutlookAttachment 133
OutlookFolder 133
OutputStream 283

P
Paint 80
Panel 180
Passport 291
PATH 31, 60, 178, 240
Piconet 284
Play() 222
Player 270
pluggable look and feel 68
Plugin Manager 48
POI 112
POIFS 112
POIFSFileSystem 121
PORT_PARALLEL 281
PORT_SERIAL 281
PowerBASIC 218
print() 176
Properties 96
Public JRE 25, 27, 29, 37
put() 227
putImage() 261

Q
QueryInterface() 217
Quit() 225

R
read() 264
Rechtschreibprüfung 226
RegBinaryValue 199, 205
RegDWordValue 198
Regedit 190
RegEmptyValue 204
Region 117
registerSystemAssociation() 170
registerUserAssociation() 170
Registrierung 189
Registry 189
RegistryDemo2 198
RegistryKey 196, 201
RegistryValue 196, 204
RegMultiStringValue 198
RegStringValue 198, 205
Release() 217
Remote 287
removeTrayIcon() 183
RenderedImage 264
Reparse Point 41
ResultSet 148
RS232Driver 287
Runtime 127
runtime environment 23

S
save() 132
saveDocumentAsAndClose() 127
Schlüssel 190
Schlüsselbund 250
Scriptsprache 61
SELECT 140
sendMessage() 293
serialEvent() 283
SerialPortEvent 283
SerialPortEventListener 283
Session 292
setBody() 132
setCellFormula() 118
setCellStyle() 117
setCellTyp() 118
setCellValue() 118
setCurrentTheme() 73
setDialogTitle() 98
setFolder() 254
setLookAndFeel() 74, 82
setName() 254
setPath() 254
setSerialPortParams() 283
Setup-Programm 26
setURL() 180
setValue() 198, 204–205
showDialog() 96
showOpenDialog() 98
showToaster() 93
Signin() 295
Signout() 295
SimpleUniverse 275
Single Document Interface 103
Skype 295
Skype-API 296
SkypeDemo1 296
Sources() 223
speak() 231
SpellCheck 226–227
Sprechblasen 184
SQirreL SQL Client 156
startInquiry() 290
Statement 146, 148
Steel 71
Structured Query Language 138
sun.java2d.d3d 272
sun.java2d.ddoffscreen 272
sun.java2d.noddraw 272
SViewerPanel 122
Swing 67
swing.defaultlaf 71

swing.metalTheme 71
swing.properties 75, 78
SwingMDIDemo 104, 106
SwingSet2 79
SyncToy 57
syntax highlighting 46
System 130
Systemklassenpfad 34
System-Property 35
Systemsteuerung 25, 27, 30–31, 39, 141, 149, 154, 207, 242, 249, 284
SystemTray 182

T
Tabbed Document Interface 104
Table 161
Taskleiste 182
TipLoader 96
TipModel 96
TipModel.Tip 96
Tipp des Tages 94
titleChange() 181
TKClassInspector 63
TKPLAFUtility 77, 244
Toaster 93
toDispatch() 223
ToggleDesktop() 218–221
ToggleDesktopDemo 220
ToolbarIcons 84, 87
toString() 124
Transform3D 275
TransformGroup 275
TrayIcon 182
Type Libraries 217
TypeLib Browser 217, 220
typeTextAtBookmark() 125

U
UIManager 72
UIManager.LookAndFeelInfo 73
Umgebungsvariablen 31
unregisterSystemAssociation() 170
unregisterUserAssociation() 170
updateComponentTreeUI() 74
URL 176
UUID 233

V
VALUES 139
Variant 222–224
Vector 177
Verb 168
Versionsverwaltung 50
Verzeichnisstruktur 25
Visual Basic 217
Visual Studio 2005 83
VoiceCtl 221
VTable 216

W
WebBrowser 178
WebBrowserDemo1 178, 181
WebBrowserListener 180
Widcomm-Stack 284, 290
Windows Explorer 29, 37, 46, 61, 64, 98, 169
Windows Installer 240
Windows Installer XML 240
Windows Messenger 91, 291, 293
Windows Performance Pack 266
Windows Task-Manager 254
Windows Update 240
WindowsLookAndFeel 73, 78
Windows-Registrierung 26
WindowsSecurity() 218
WinLAF 81
Word 122, 226
WordDemo1 123
WordDocument 124
WordProcessing 124, 127
WordStar 45
Works-Suite 226
writeAllText() 124
Writer 124

X
Xelfi 52

Y
Yahoo Messenger 291
Yahoo Messenger Protocol 291

Z
Zertifikat 248
Zertifizierungsstelle 250