7 Grundlagen der Applet-Erstellung

Bestandteile der Definition für Leben sind Wachstum, Bewegung und Reaktion auf äußere Ereignisse. Seit einiger Zeit scheint mit dieser Definition das World Wide Web anzufangen, eine Art Leben zu entwickeln, lebendig zu werden. Immer mehr neue Entwicklungen hauchen den bisher statischen Hypertext-Seiten Leben in Form von Animation und Bewegung bzw. komfortabler Interaktion mit den Anwendern ein.

Das WWW ist die wichtigste Säule des Internets. Erst durch das auf das HTTP-Protokoll aufsetzende WWW (neben der E-Mail-Funktion als die zweite tragende Säule des Internets) ist das Internet wirklich anwenderfreundlich, für die breite Masse interessant und damit richtig populär geworden.

Trotz der Vielzahl neuer Sprachentwicklungen ist das WWW nach wie vor im Wesentlichen durch so genannte Hypertexte aufgebaut, die mit der Internetsprache HTML entwickelt wurden (und werden). Ein Hypertext führt durch markierte Wörter (so genannte Hyperlinks) zu weiteren Seiten im WWW oder einem anderen Internetdienst. Hypertext ist also von der Bedeutung her erstmals eigentlich nur ein Text mit Verweisen auf andere Texte. Dieser Verweis kann mit der Maus angeklickt werden und es wird sofort zu dem gewünschten Dokument verzweigt.

HTML-Dateien bestehen aus reinem ASCII-Text, das heißt Klartext. Dadurch bleiben HTML-Dokumente plattformunabhängig. Plattformabhängig ist immer nur die Software zum Interpretieren der HTML-Dateien (der Browser).

Ein in HTML geschriebenes Dokument kann außer Text ebenso Grafiken sowie multimediale Elemente (Sound, Video usw.) enthalten. Solche Elemente werden als Referenz auf eine entsprechende Grafik- oder Multimedia-Datei notiert. Natürlich muss die Präsentations-Software entsprechende Softwaremodule und die Hardware die zugehörigen Komponenten (beispielsweise eine Soundkarte für akustische Daten) besitzen oder aufrufen, mit deren Hilfe solche Effekte dargestellt werden können.

Daneben ist es mit HTML – mit gewissen Einschränkungen – möglich weitergehende Funktionalität wie Datenbankabfragen zu formulieren, die

Resultate optisch aufzubereiten und Menüstrukturen aufzubauen. Das Einbinden des E-Mail-Dienstes in HTML-Seiten ist eine weitere sehr wichtige Eigenschaft und auch Interaktionen mit Anwendern sind zu realisieren.

Die größten Nachteile von HTML sind die unbefriedigende Interaktion mit den Anwendern, fehlende eigenständige Aktion (sprich ein echtes Programm auf Clientseite) und mangelnde echte Animation. Hier kommen nun die Java-Applets ins Spiel, die in HTML-Seiten eingebaut vollkommen neue Möglichkeiten schaffen. Wir müssen uns wohl nicht weiter über die Vorteile und Möglichkeiten von Java-Applets in HTML-Seiten unterhalten – Sie werden schon wissen, warum Sie in Java programmieren wollen (und warum Sie sich das Buch gekauft haben ;-)).

Wenn Sie sich gute Java-Applets als Beispiele zum Lernen oder auch nur zum Anwenden laden wollen, ist Gamelan die Adresse im Internet. Jede Menge Applets gibt es unter `http://www.jars.com`. Dort finden Sie ebenfalls diverse Informationen zu JavaScript und ActiveX. Auch ist Jars.com die richtige Adresse, wenn Sie bestimmte Java-Tools wie beispielsweise Applet-Editoren benötigen.

7.1 Die Vorbereitung

Bevor wir mit der konkreten Programmierung anfangen, wollen wir zunächst einmal für die Applets eine sinnvolle Entwicklungsumgebung erstellen. Dies betrifft im Wesentlichen Verzeichnisstrukturen, in welchen für jedes Applet einzeln oder für alle Applets gemeinsam (hat beides Vor- und Nachteile) der Applet-Code und die zugehörigen HTML-Seiten, sowie eventuelle Tondateien und Bilder platziert werden. Die folgende Verzeichnis-Strukturen erscheinen dafür ganz angebracht:

Tabelle 7.1: Verzeichnis-Strukturen

Verzeichnis	Beschreibung
`/<NamederKlasse>`	Für .HTML/.HTM-Dateien
`/<NamederKlasse>`	Für `.java-` und `.classes`-Dateien
`/<NamederKlasse>/images`	Für `.gif-` oder `.jpeg`-Bilddateien bzw. alle weiteren Bild-Dateien

Wenn Sie alle Applets in einer gemeinsamen Struktur verwalten, können Sie vor allem eventuell vorhandene Tondateien und Bilder leichter gemeinsam nutzen (wir werden das auf der Buch-CD so machen). Andererseits geht in der Praxis bei mehreren Projekten natürlich die Übersichtlichkeit verloren, weshalb eine solche Verzeichnisstruktur innerhalb eines eigenen Applet-Verzeichnisses eigentlich sinnvoller ist.

7.2 Grundlagen der Einbindung von Java-Applets in HTML-Seiten

Kommen wir nun zur konkreten Einbindung der Java-Applets in HTML-Seiten. Es gab ursprünglich zwei konkurrierende Syntaxmethoden, um Java-Applets in einer HTML-Datei zu referenzieren. Die eine Syntax basiert auf einer Netscape-Entwicklung und wird unter anderem vom Navigator ab der Version 2.0 oder dem Internet Explorer interpretiert, die andere Syntax stammt von Sun selbst. Deren Browser HotJava in den frühen Versionen interpretierte diese Syntax. Die HotJava-Syntax gilt für Applets, die mit dem Java-1.0-alpha-Compiler erzeugt wurden, und wird mittlerweile nicht mehr unterstützt. Die Netscape-Syntax für Java-Applets gilt für alle Applets, die mit dem Java-1.0-beta-Compiler aufwärts erzeugt wurden, und der vom aktuellen HTML 4.0 unterstützte Standard bezieht sich auf die Netscape-Syntax. Auch der HotJava-Browser nutzt mittlerweile diese Syntax. Seit HTML 4.0 gibt es zwei zusätzliche Möglichkeiten, Java-Applets in eine Webseite zu integrieren.

Java-Applets werden innerhalb von HTML-Seiten in Form von Referenzen eingebunden. HTML gibt als ein Dokumentenformat in Form von Klartext (ASCII-Text) unverbindliche Empfehlungen an eine Darstellungssoftware (den Browser), wie eine bestimmte Dokumentenstruktur darzustellen und welche Funktionalität wie auszuführen ist, damit sie dem geplanten Layout und der vorgesehenen Funktionalität entspricht. Es gibt in reinem HTML[1] keine absolute Darstellungsvorschrift, weswegen sich Aufführungen von HTML-Seiten in verschiedenen Browsern oft erheblich unterscheiden können (eine Folge der geplanten Plattformunabhängigkeit).

Um nun ein Applet in eine HTML-Seite integrieren zu können, müssen wir uns ein wenig mit dem Aufbau einer solchen Webseite beschäftigen.

7.2.1 HTML-Grundlagen

HTML ist die Abkürzung für Hypertext Markup Language und eine Dokumentbeschreibungssprache aus Klartext, mit der die logischen Strukturen eines Dokuments plattformunabhängig beschrieben werden. Das World Wide Web besteht in seiner Grundstruktur aus HTML. Hinter der Normung der Sprache HTML stand und steht auch heute noch das World Wide Web-Consortium (W3C – ursprünglich W3O – O für Organisation – genannt – `http://www.w3.org`). HTML liegt derzeit in der Version 4.0 vor. HTML ist viel einfacher als Programmiersprachen und auch als Scriptsprachen. Es gibt beispielsweise keinerlei Kontrollstrukturen in Form von Bedin-

[1] von Stylesheets abgesehen, aber die gehören nicht direkt zu HTML

gungen, Sprüngen oder Schleifen. Es gibt auch keine Befehle im Sinne von Befehlswörtern, die eine Aktion auslösen. HTML beinhaltet in seiner aktuellen Version nur einige Erweiterungen für den Aufruf von Scripten.

Die Sprache HTML ist eine äußerst fehlertolerante Beschreibungssprache. Selbst in Situationen, wo in anderen Sprachen geschriebene Dokumente oder Programmstrukturen einen Fehler oder einen Programmabbruch auslösen würden, werden HTML-Dokumente trotzdem oft noch brauchbare Resultate liefern. Der Grund dafür ist das so genannte Prinzip der Fehlertoleranz von HTML. Dies bedeutet im Wesentlichen, dass die Programme zur Auswertung von HTML-Dokumenten so fehlertolerant wie irgend möglich programmiert werden, um syntaktisch unkorrekte Dokumente so weit wie möglich auswerten zu können. Soweit die Browser dann korrekte HTML-Anweisungen vorfinden, werden diese Anweisungen ausgeführt und angezeigt. Falsche oder unvollständige Anweisungen werden ganz einfach ignoriert. Dies kann so weit gehen, das sogar HTML-Dokumente ohne das – eigentlich zwingende – Grundgerüst einer Webseite weitgehend dargestellt werden. Jeder Text, den der Browser vorfindet und der nicht als fehlerhafte Syntax ignoriert wird, wird von ihm angezeigt.

Reine HTML-Seiten haben in der Regel die Endung `HTM`, `HTML` oder `SHTML`. Alleine diese Endungen lassen das Dokument zu einem HTML-Dokument werden. Für eine HTML-Seite gibt es jedoch dennoch wie für jedes andere Dokument oder andere Programmiersprache immer ein Grundgerüst und gewisse Grundregeln. Dabei sind insbesondere zwei Begriffe wichtig: Steueranweisungen und Tags.

HTML-Steueranweisungen

Die HTML-Befehle werden Steueranweisungen genannt. Die HTML-Sprache hat vom Aufbau her eine gewisse Ähnlichkeit zur einfachen Batch-Programmierung unter DOS. Die Steueranweisungen werden in Form einer Stapeldatei in Klartext erstellt. Sprünge und Schleifen im Source kommen nicht vor, dafür sind Layout-Möglichkeiten in HTML sehr ausgeprägt.

Alle HTML-Steueranweisungen werden in so genannte Tags geschrieben, die von spitzen Klammern – dem »kleiner« und dem »größer«-Zeichen – begrenzt werden. Ein HTML-Tag sieht also von der Struktur her immer so aus: `<xyz>`

Dabei unterscheidet man zwischen Einleitungs- und Abschluss-Tags. Abschluss-Tags sind bis auf einen den öffnenden Klammer folgenden Slash (`/`) identisch zum Einleitungs-Tag. Das Abschluss-Tag zum obigen Einleitungs-Tag würde wie folgt aussehen: `</xyz>`

Muster:

```
<i>...</i>
<h6>...</h6>
```

Bei den HTML-Steueranweisung spielt es im Gegensatz zu Java-Token keine Rolle, ob sie groß oder klein geschrieben werden. Die Anweisung `<h4>` *bewirkt das Gleiche wie* `<H4>`*. Auch eine unterschiedliche Groß- und Kleinschreibung im Einleitungs- und Abschluss-Tag hat keine negativen Auswirkungen, erhöht jedoch nicht gerade die Lesbarkeit. Für die Zukunft steht aber zu erwarten, dass auch in HTML Groß- und Kleinschreibung relevant wird. Dann werden (von XML beeinflusst) HTML-Anweisungen wahrscheinlich klein geschrieben.*

Es gibt Tags, die zwingend ein Einleitungs- und einen Abschluss-Tag benötigen. Beide zusammen bilden einen so genannten Container.

Beispiel (Zentrieren von Elementen): `<CENTER> ... </CENTER>`

Andere Tags kommen in der strengen HTML-Syntax hingegen nur als Einleitungs-Tag vor.

Beispiel (ein Zeilenumbruch): `<BR>`

Das Grundgerüst einer HTML-Seite

Eine HTML-Seite wird immer in die Anweisung `<HTML>` am Anfang und `</HTML>` am Ende eingeschlossen. Die beiden Anweisungen bilden das äußere Gerüst einer HTML-Seite.

Beispiel:

```
<HTML>
   ...weitere HTML-Anweisungen...
</HTML>
```

Davor dürfen höchstens Kommentarzeilen stehen, die natürlich auch überall im Inneren einer HTML-Seite verwendet werden können. Die Steuerzeichen für Kommentarzeilen sind ein Ausrufezeichen, zwei Striche am Anfang und zwei Striche am Ende des Tags. Jeder in diesem Tag stehende Text wird vom interpretierenden Browser als Kommentar betrachtet. Ein Kommentar kann über mehrere Zeilen gehen und natürlich ebenfalls im Inneren einer HTML-Seite verwendet werden. Auch ein Kommentar-Tag muss von spitzen Klammern eingeschlossen werden, also sehen die Zeichenfolgen so aus:

```
<!-- ... -->
```

Beispiel:

```
<!-- Generated by RJS - Last Updated: Feb 28, 2001 -->
```

Ein wichtiger Spezialfall für die Verwendung der Kommentare ist die Einbindung von Script-Elementen in Webseiten. Sie werden damit vor dem HTML-Interpreter des Browsers versteckt, stehen hingegen dem Script-Interpreter des Browsers unverändert zur Verfügung (ein HTML-Kommentar ist kein Script-Kommentar).

Das weitere Grundgerüst einer HTML-Datei besteht grundsätzlich aus folgenden zwei Teilen:

- dem (optionalen) Header (Kopf)
- dem Body (Körper)

Nach der einleitenden HTML-Anweisung kommt auf einer Webseite normalerweise ein Header-Teil, das heißt ein Kopfteil, in dem die allgemeinen Informationen über die Seite zusammengefasst werden. Beispielsweise der Titel, der in die Zeichenketten <TITLE> und </TITLE> eingeschlossen wird.

Ein Header wird mit dem Steuerzeichen <HEAD> begonnen und mit </HEAD> entsprechend wieder geschlossen. Ein einfaches Headergerüst ist wie folgt aus:

Beispiel:

```
<HEAD>
   <TITLE>Titel</TITLE>
</HEAD>
```

Die eigentlichen Daten, die der WWW-Browser einem Anwender auf dem Bildschirm anzeigen soll, werden in den Body geschrieben. Die Tags <BODY> ... </BODY> definieren den Body. In ihm ist der eigentliche Text mit Überschriften, Verweisen, Grafikreferenzen und auch die Referenz für Java-Applets zu notieren.

Vor Beginn des HTML-Tags kann mit einem so genannten <DOCTYPE>-Tag ein Verweis auf eine DTD-Datei integriert werden. Diese legt die HTML-Version fest und im Prinzip muss jedem SGML-Dokument (so auch einer HTML-Seite) eine solche DTD-Datei zugeordnet sein. Dieses Tag ist jedoch optional, da das WWW so funktioniert, dass alleine die Dateiendung HTML oder HTM bereits ausreicht, um ein Dokument als HTML-Datei zu identifizieren. Der jeweilige Browser benötigt also nicht explizit die Document Type Definition. Es ist sogar so, dass die entsprechende Angabe von den meisten Browsern ignoriert wird bzw. keine weiteren Konsequenzen für die Darstellung der HTML-Datei hat.

Das vollständige Grundgerüst einer normalen HTML-Datei sieht also schematisch immer so aus.

Listing 7.1: Schematisches Grundgerüst einer Webseite

```
<!DOCTYPE HTML PUBLIC "  ... ">
<!-- Kommentar -->
<HTML>
  <HEAD>
    <TITLE>Titel</TITLE>
  </HEAD>
  <BODY>
    Überschriften, Text,
    Verweise, Grafiken, Java-Applet-Referenz  usw.
  </BODY>
</HTML>
```

HTML-Strukturen können verschachtelt werden, das heißt, zwischen einem einleitenden und einem abschließenden Tag können beliebige – sinnvolle – andere Steueranweisungen stehen. Dies ist sogar zwingend, da insbesondere die vollständige HTML-Seite ein einleitendes und ein abschließendes Tag benötigt werden und alle weiteren Steueranweisungen darin verschachtelt werden müssen. Bei der Verschachtelung von Tags sollte jedoch unbedingt die Reihenfolge eingehalten werden! Das bedeutet, wenn ein Container (Einleitungs- und Abschluss-Tag) weitere Tags enthalten soll, sollten diese vollständig darin enthalten sein und in umgekehrter Reihenfolge der Einleitungs-Tags die Abschluss-Tags aufweisen. Zwar kann in vielen Fällen auch eine sich überschneidende Verschachtelung funktionieren, sie ist aber schlecht lesbar und kann in einigen Fällen zu Fehlern führen.

Wichtige HTML-Elemente

HTML bietet inzwischen eine Menge Gestaltungselemente. Das fängt bei Farben an und geht über Textformatierung, Überschriften, Listen, Tabellen, Eingabefeldern, Buttons bis hin zu Multimedia-Effekten. Diese gehören jedoch nicht in ein Java-Buch, sondern in eine Abhandlung über HTML. Sie sollten jedoch den in unserem Zusammenhang wichtigsten Bestandteil kennen, die Referenz auf andere Objekte innerhalb einer Webseite.

Der wichtigste Fall einer Referenz auf andere Objekte ist der Verweis oder Hyperlink. Das WWW lebt nur von diesen Hyperlinks, den lokalen Sprüngen und vor allem den Verweisen auf die verschiedensten anderen Internet-Inhalte. Mit einem Verweis verzweigen Sie durch Anklicken sofort zu einer angegebenen Adresse, laden ein Programm auf dem lokalen Rechner oder senden eine E-Mail an die im Verweis angegebenen Adresse. Die Grundstruktur eines Verweises in einer Datei setzt man mit den folgenden Steueranweisungen:

```
<A "Typ"="Bezeichner" >Verweistext</A>
```

Hierbei müssen die einzelnen Bestandteile dieser Grundstruktur natürlich noch genauer bezeichnet werden, was wir uns hier schenken wollen. Nur so viel:

- Der Typ gibt die Art des Verweises an.
- Der Bezeichner ist die konkrete Adresse (eine URL).

Bei einem Verweis unterscheidet man zwischen verschiedenen Typen von Verweisen und daraus ergibt sich die genaue Syntax des Verweises. Wir wollen uns nur exemplarisch ein paar Beispiele mit verschiedenen Verweistypen ansehen und diese dann kurz beschreiben.

Tabelle 7.2: Einige Beispiele für Hyperlinks

Referenz	Beschreibung.
`<A href="datei.htm">Zur Datei</A>`	Beispiel für einen Verweis auf eine andere Datei innerhalb des gleichen Verzeichnisses.
`<A href = "../weg/datei.htm">Zur Datei </A>`	Beispiel für einen Verweis auf eine andere Datei in einem relativen Verzeichnis.
`<A href= "file://localhost/c:/UV/datei.htm"> Dies ist ein Verweis mit absoluter Pfadangabe auf einem lokalen PC</A>`	Beispiel für einen Verweis auf eine lokale, absolut adressierte Datei.
`<A href="http://www.rjs.de">Der Autor</A>`	Beispiel für einen nicht-lokalen Verweis.
`<A href="mailto:hansdampf@aol.com">Ab ins Mehl</A>`	Aufruf des E-Mail-Clients aus der Webseite heraus.

Ein anderer Typ von Referenz sind Grafikbezüge.

Beispiel:

Tabelle 7.3: Eine Grafikreferenz

Referenz	Beschreibung
`<IMG src="rjsfoto.gif" height=100>`	Referenz auf eine Grafik.

Kommen wir nun zu dem nächsten Referenztyp: der Java-Applet-Referenz (und da wollen wir ja eigentlich hin).

7.3 Konkrete Referenzierung eines Java-Applets

Es gibt derzeit drei aktuelle Varianten, wie Sie ein Applet in eine Webseite integrieren können. Mit dem <APPLET>-Tag, dem <EMBED>-Tag und dem <OBJECT>-Tag. Wie bereits angesprochen, ist die ab dem HTML-Standard 4.0 erlaubte Referenzierung eines Applets über das <OBJECT>-Tag mit Vorsicht zu genießen, weil dieser Standard noch nicht von vielen Browsern unterstützt wird und das <APPLET>-Tag sicher noch einige Jahre verwendet werden kann. Das <EMBED>-Tag sollte ebenso mit Vorsicht eingesetzt werden.

7.3.1 Die <APPLET>-Syntax

Schauen wir uns zuerst die konkrete Referenzierung eines Java-Applets mit dem <APPLET>-Tag an. Zwischen dem einleitenden Tag und dem abschließenden Tag </APPLET> können Sie gegebenenfalls benötigte Parameter, die dem Applet beim Aufruf übergeben werden, oder auch beliebigen Text eingeben.

Die einfachste Form der Applet-Referenz

Ein Java-Applet in der einfachsten Form (das heißt ohne irgendwelche Parameter und optionale Attribute) wird mit folgender HTML-Syntax in eine HTML-Seite eingebunden:

```
<APPLET CODE="[classelement]" WIDTH=[Wert] HEIGHT=[Wert]></APPLET>
```

Dabei bezeichnet

- `[classelement]` die Applet-Klasse, also das Applet selbst (mit Namenserweiterung .`class` oder auch ohne – es geht beides)
- `WIDTH=[Wert]` die Breite des Applets in Pixel.
- `HEIGHT=[Wert]` die Höhe des Applets in Pixel.

`WIDTH` und `HEIGHT` bestimmen also die anfängliche Größe der Darstellung des Applets innerhalb der HTML-Seite. Diese Größenangaben haben nichts direkt mit der »echten« Größe des Applets zu tun. Es kann also durchaus zu Verzerrungen (auch gewollten) des Applets kommen oder unter Umständen wird bei zu kleiner Wahl der Breite und Höhe nicht das vollständige Applet angezeigt.

Die Angabe der Breite und Höhe ist bei der Verwendung des Appletviewers zwingend, während beim Fehlen einer dieser Angaben in vielen Browsern das Applet dennoch angezeigt wird. Setzen Sie aber zur Sicherheit immer beide.

Abbildung 7.1: Da fehlt was

Es folgt ein Beispiel für die Syntax zur Einbindung eines einfachen Java-Applets ohne weitere Angaben (nur die Breite und Höhe). Wir verwenden am Anfang ein Applet, das wir ja sowieso noch haben sollten.

Beispiel:

Listing 7.2: Einfachste Form der Applet-Referenz

```
<APPLET CODE="HelloJavaApplet.class"
    WIDTH=150 HEIGHT=75></APPLET>
```

Die Angabe der Applet-Klasse erfolgt in der Regel in Hochkommata und mit Dateierweiterung. Beides ist aber – zumindest in den meisten Browsern – nicht zwingend, aber sinnvoll. Der Anzeigebereich des Applets wäre in obigem Beispiel 150 Pixel breit und 75 Pixel hoch.

Obwohl es für unser Applet noch nicht viel Sinn macht (der Text bleibt einfach so klein wie er ist), wollen wir die Größenangaben einmal auf `WIDTH=150` und `HEIGHT=275` verändern.

Über die optionale Angabe `SRC` kann man innerhalb der Applet-Referenzierung zusätzlich die Source-Datei (die Datei mit der Kennung `.java`) eines Java-Applets angegeben. Diese wird jedoch im Prinzip nicht benötigt, um ein Applet in eine Webseite einzubauen oder zu starten, nur zur Dokumentation der Funktionalität und einer eventuellen Modifizierung des Applets.

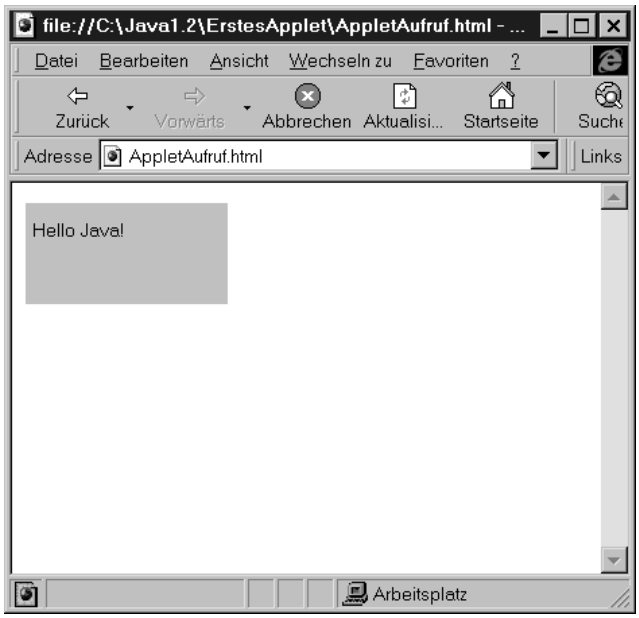


Abbildung 7.2: Unser Applet 150 Pixel ist breit und 75 Pixel hoch

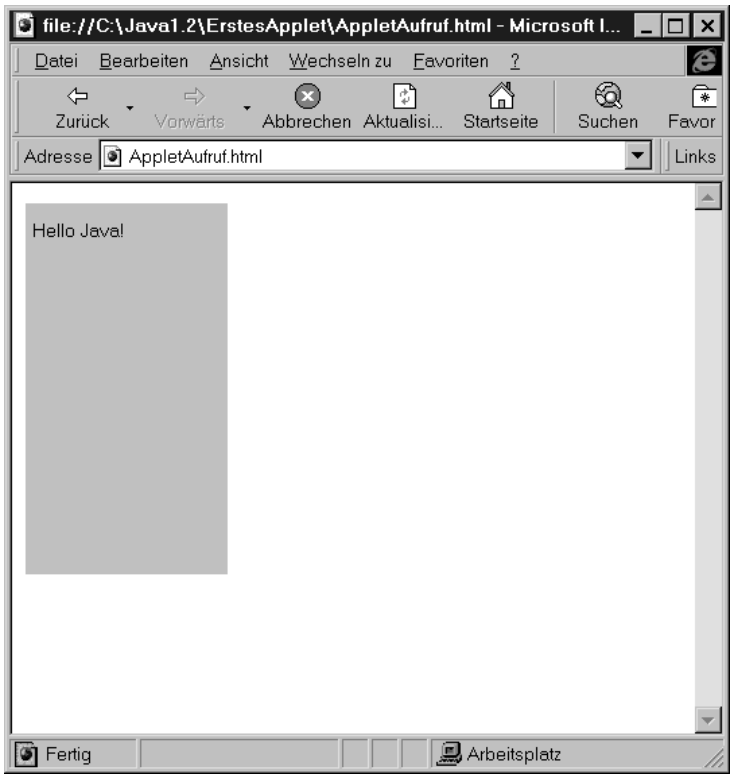


Abbildung 7.3: Andere Größenangaben

Beispiel:

Listing 7.3: Applet-Referenz mit Source-Angabe

```
<APPLET CODE="HelloJavaApplet.Class"
    SRC="HelloJavaApplet.Java"
    WIDTH=100 HEIGHT=50></APPLET>
```

Applet-Referenz mit Pfaden

Die bisherigen Beispiele funktionieren nur, wenn sich das Applet im gleichen Verzeichnis wie die aufrufende HTML-Datei befindet. Wenn sich das Applet in einem anderen Verzeichnis befindet, wird dieses über das Attribut `CODEBASE` bestimmt. Dies ist die Suchpfadangabe für das Applet – eine übliche URL. Hier ein Beispiel für die Einbindung eines Java-Applets mit zusätzlich Pfadangaben (relativ) auf das Unterverzeichnis `classes`.

Beispiel:

Listing 7.4: Applet-Referenz mit relativer Angabe eines Suchpfades für die .class-Datei

```
<APPLET CODEBASE="classes"
    CODE="HelloJavaApplet.class"
    WIDTH=100 HEIGHT=50></APPLET>
```

Das Verzeichnis in der `CODEBASE`-Anweisung wird in der Regel ebenfalls in Hochkommata stehen und kann vor oder hinter der Codeanweisung stehen. In diesem Beispiel wird die Position des Applets relativ zur Position des HTML-Dokuments angegeben.

Ein Applet kann genauso gut mit einer absoluten Pfadangabe referenziert werden. Dabei spielt es keine Rolle, ob sich das Applet auf dem gleichen Rechner oder einem anderen Server befindet, wie das folgende Beispiel für die Einbindung eines Java-Applets mit absoluten Pfadangaben zeigt (mit Kommentar innerhalb des Containers, der in nicht-Java-fähigen Browsern angezeigt wird).

Beispiel:

Listing 7.5: Applet-Referenz mit absoluter URL-Angabe eines Suchpfades für die .class-Datei

```
<APPLET CODEBASE="http://www.rjs.de"
    CODE="HelloJavaApplet.class"
    WIDTH=100 HEIGHT=50>
Das Applet wird über http://www.rjs.de adressiert
</APPLET>
```

Für `CODEBASE` kann also eine beliebige URL gesetzt werden. Es können also darüber ganz leicht Netzwerkzugriffe realisiert werden (wenngleich auf diese Art von außerhalb des Applets).

Die genaue Position des Applets in der Webseite

Die Position eines Applets auf der Seite ergibt sich aus der Stelle, wo das <APPLET>-Tag in das HTML-Dokument eingefügt ist. Dabei kann ein Applet genau wie andere Objekte (etwa Überschriften oder Grafiken) innerhalb der Seite mit allen üblichen HTML-Spezifikationen ausgerichtet werden. Ein Applet, das beispielsweise in die HTML-Zentrierungs-Tags <CENTER> und </CENTER> eingeschlossen ist, wird beim Anzeigen zentriert ausgegeben.

Beispiel:

```
<CENTER><APPLET code="HelloJavaApplet.class"
   width="500" height="600"></APPLET></CENTER>
```

Listing 7.6: Zentrierung eines Applets

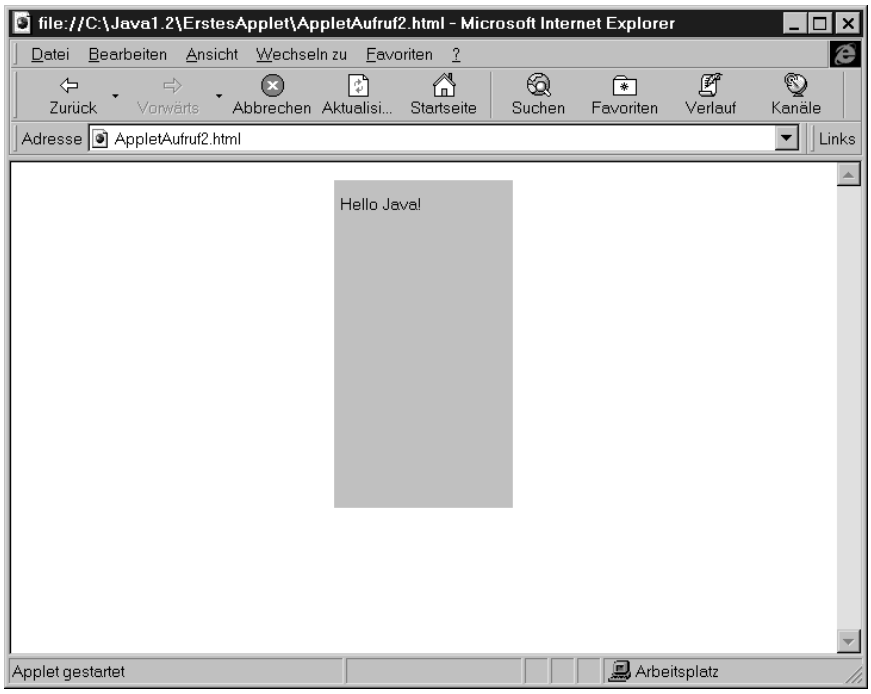


Abbildung 7.4: Das Applet zentriert

Zusätzlich lassen sich bei einem Applet noch Angaben über den Textfluss um das Applet herum machen, wie das nachfolgende Beispiel für die Einbindung eines Java-Applets mit Angaben zum Textfluss zeigt.

Beispiel:

```
<APPLET CODE="HelloJavaApplet.class"
   WIDTH=100 HEIGHT=50
   ALIGN=middle VSPACE=25 HSPACE=30></APPLET>
```

Listing 7.7: Angaben zum Textfluss um das Applet herum

Dabei gibt es für die Angabe ALIGN, die die Ausrichtung von einem dem Applet folgenden Text angibt, folgende Werte:

- LEFT: links ausgerichtet
- RIGHT: rechts ausgerichtet
- MIDDLE: zentriert in der horizontalen Ausrichtung am Applet

Die Angabe VSPACE definiert den vertikalen Abstand zwischen Applet und Text oberhalb und unterhalb des Applets. HSPACE definiert den horizontalen Abstand links und rechts vom Applet.

Der Container-Inhalt

Zwischen dem einleitenden Tag <APPLET> und dem abschließenden </APPLET>-Tag kann beliebiger Text stehen. Dies nutzt man hauptsächlich dafür, in einem nicht Java-fähigen Browser deutlich zu machen, dass an dieser Stelle ein Java-Applet steht. Sie sollten sich nicht täuschen – so was gibt es wirklich noch. Und nicht zuletzt: Die Java-Option lässt sich auch in modernen Browsern deaktivieren. Es ist kein guter Stil, wenn man sich einfach auf den Standpunkt stellt, dass jeder Besucher einer Webseite gefälligst Java-fähige Browser zu nutzen und die entsprechende Option zu aktivieren hat. Denken Sie daran, dass in der Regel nicht der Surfer etwas von Ihnen will, sondern Sie ihm etwas anbieten wollen. Machen Sie doch einfach Surfer mit fehlender Java-Funktionalität im Browser neugierig, indem Sie mit einem netten Sätzchen locken.

Man macht sich übrigens bei dem Text zwischen den <APPLET>*-Tags wie auch beispielsweise bei Grafiken oder der Frame-Technik das Prinzip der Fehlertoleranz zunutze. Wenn Browser unbekannte Tags – in diesem Fall das* <APPLET>*-Tag – finden, werden sie den Tag ignorieren und den nachstehenden Text als reinen ASCII-Text auf dem Bildschirm anzeigen. Java-fähige Browser (mit aktivierter Java-Option) sind so konzipiert, dass sie den Text zwischen ein- und abschließenden* </APPLET>*-Tags nicht anzeigen, sondern eben das Applet selbst.*

Sie können zum Beispiel angeben, dass dort ein Java-Applet steht, das nicht Java-fähige Browser eben nicht anzeigen.

Beispiel:

```
<APPLET code="HelloJavaApplet.class" width="500"  height="600">Wenn Sie hier
diesen Text sehen, ist Ihr Browser nicht in der Lage, Java-Applets
darzustellen oder Sie haben die entsprechende Option deaktiviert, denn hier
sollte ein Applet zu sehen sein! Ich kann nur sagen - Sie verpassen etwas.
Wie wäre es - aktivieren Sie Ihre Java-Option und laden Sie dann die Seite
neu. Falls Sie keinen Java-fähigen Browser haben - etwas weiter unten
```

```
befinden sich ein paar Hyperlinks, von denen Sie sich solche Browser laden
können. Und dann: Come back and enjoy Java!</APPLET>
```

Der alte Mosaic-Browser von CompuServe ist ein Exempel für einen nicht-Java-fähigen Browser. Hier wird statt des Applets der freundliche Hinweis auf ein Applet zu sehen sein.

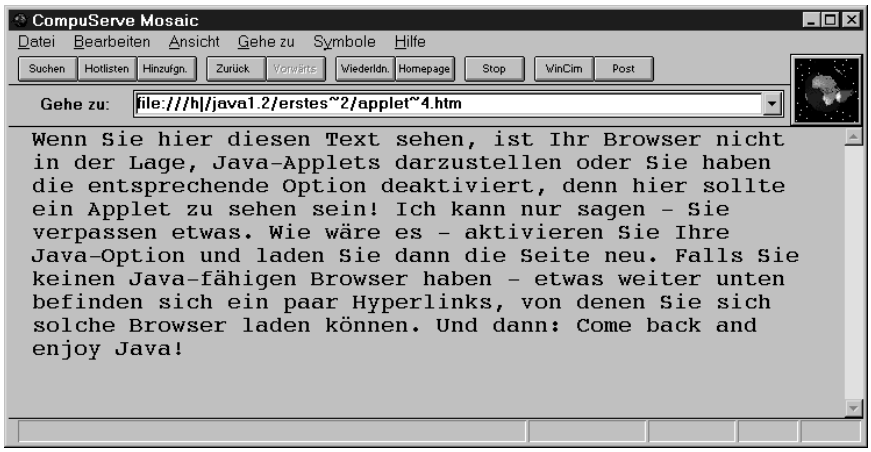


Abbildung 7.5: Langsam wird es Zeit für ein Update.

Die gleiche HTML-Datei sieht in einem Java-fähigen Browser wie wir oben schon gesehen haben oder vergleichbar aus.

Bei Text, der das Applet umfließt, kann es zu Missverständnissen kommen, da vor und nach dem Applet stehender Text den Hinweis auf das Applet in nicht-Java-fähigen Browsern unter Umständen nicht auffällig genug macht. Verwenden Sie am besten irgendwelche Unterschiede zwischen normalem Text und dem Hinweis auf das Applet (etwa Farbe oder Schriftgröße).

7.3.2 Die Parameter des <APPLET>-Tags

Eine ganz wichtige Art von Angaben kann zwischen dem einleitenden Tag `<APPLET>` und dem abschließenden Tag stehen – die Parameter, die über eigene Tags innerhalb des Containers notiert werden. Unter solchen Parametern versteht man Werte, die ein Programm – oder in diesem Fall das Applet – benötigt, um laufen zu können und die den Ablauf des Programms/Applets beeinflussen.

Die Situation ist uns ja nicht unbekannt. Bei vielen DOS-Befehlen muss man Parameterangaben machen, damit der Befehl erst einen Sinn bekommt. Denken Sie an den Kopierbefehl `COPY`, der erst durch Angabe von zwei Parametern – der Quelle und dem Ziel – vom Betriebssystem ausgeführt werden kann und auch dann erst Sinn macht.

Abbildung 7.6: Schlecht: Der Applethinweis ist kaum zu erkennen.

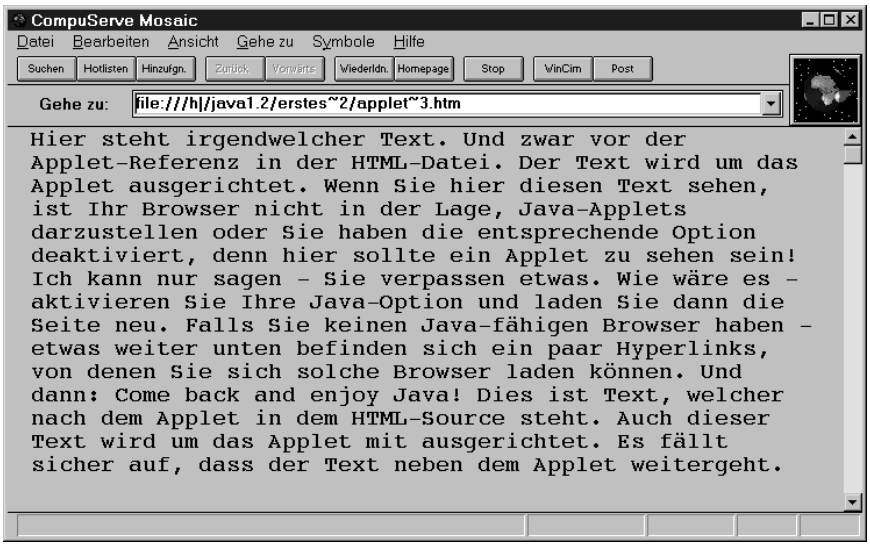
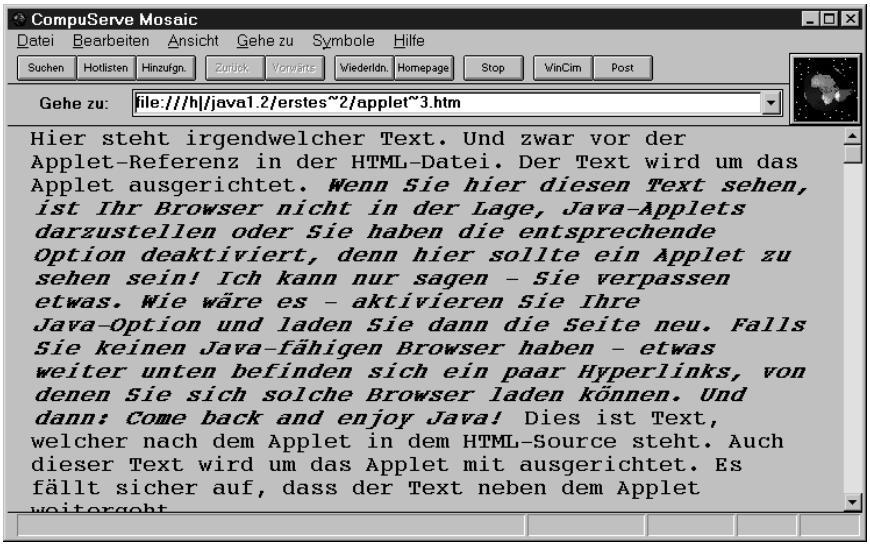


Abbildung 7.7: So ist es zumindest zu erkennen.



Wir kennen auch bereits einen Java-Fall, wo wir Parameter von außen an ein Java-Programm übergeben haben. Um Argumente auf Befehlszeilenebene an ein Java-Programm weiterzugeben, müssen diese einfach an den Programmaufruf – mit Leerzeichen abgetrennt – angehängt werden.

Die Parameter in dem <APPLET>-Container der HTML-Seite sind nun bei einem Applet das Analogon zu den Argumenten auf Befehlszeile bei einer eigenständigen Applikation.

Wenn Sie ein Applet nur in eine Webseite einbinden wollen, haben Sie keinen Einfluss auf die verwendeten Parameter. Dies ist natürlich eine andere Situation, als wenn Sie den Source des Applets erstellen. Welche Parameter ein Applet konkret benötigt, ist fest in dem Java-Applet einprogrammiert und kann von einem reinen Anwender beim Aufruf nicht verändert werden. Viele Applets benötigen keine Parameter, andere wiederum mehrere. Wenn Sie ein bestehendes Applet einbinden wollen, müssen Sie diese Information aus der Dokumentation des Applets entnehmen. Wenn Sie keinen Hinweis auf Parameter finden, ist es sehr wahrscheinlich, dass das Applet auch keine braucht. Auf jeden Fall gilt dann »Try and Error«. Wenn es funktioniert, dann benötigt es wirklich keine Parameter.

Wir kommen in einem nachfolgenden Abschnitt zur Auswertung und Verwendung von Parametern im Source des Applets. Lassen Sie uns zuerst noch die HTML-Syntax für die Parameter beenden. Die Angabe von Parametern erfolgt einfach über Parameter-Tags – gegebenenfalls mehrere hintereinander. In diesem Fall spezifiziert jedes Parameter-Tag einen anderen Applet-Parameter. Sie sehen wie folgt aus:

```
<PARAM name=[name] value=[wert]>.
```

In den Parameterangaben werden zwei Bezeichner verwendet:

- `name` – der Name des Parameters
- `value` – der Wert des angegebenen Parameters

Ein Beispiel für die Einbindung eines Java-Applets mit Parameter-Angaben:

```
<APPLET code="Ataxxlet.class"
    width=450 height=550>
  <PARAM name=Dim value=15>
  <PARAM name=ssize value=30>
  <PARAM name=Player1pos value="0,0:14,14">
</APPLET>
```

Listing 7.8: Beispiel einer Applet-Referenz mit Parametern

Im gesamten `<PARAM>`*-Tag kann man zwar Groß- und Kleinschreibung verwenden (es ist ja HTML), aber beim name- und value-Wert kann Groß- und Kleinschreibung eine Rolle spielen. Diese Werte werden ja an das Java-Applet übergeben und da kommt es darauf an, wie dieses die Werte verwendet.*

Abschließend soll besprochen werden, was für und was gegen die Applet-Referenzierung mit dem `<APPLET>`-Tag spricht.

Kontra:

1. Der Tag ist vom W3C als deprecated erklärt worden. Das will aber nicht viel heißen, denn viele als deprecated erklärte Tags werden unverändert weiter verwendet. Oft ist es sogar so, dass deren Verwendung statt der offiziellen Befehle von Profis gerade ausdrücklich empfohlen wird. So beruhen Optimierungsmaßnahmen in einer HTML-Seite beispielsweise darauf, bei einer Zentrierung statt des `align`-Attributs für Absatz-bezogene HTML-Tags (offizielle Empfehlung) das als deprecated gekennzeichnete `<CENTER>`-Tag zu verwenden, da damit Zeichen gespart werden können und die Seite kleiner und damit schneller zu laden ist.
2. Die Unterstützung der Java-Plug-Ins funktioniert mit diesem Tag nicht. Wenn in einem Applet also Java-APIs verwendet werden, die von der virtuellen Maschine des Browsers nicht unterstützt werden, wird das Applet nicht laufen. Um es drastisch zu formulieren – ein mit diesem Tag referenziertes Applet sollte nur Techniken verwenden, die maximal auf Java 1.0.2 beruhen oder zumindest in allen potenziellen Zielbrowsern getestet werden.

Pro:

1. Der Tag ist einfach anzuwenden.
2. Der Tag wird von allen Java-fähigen Browsern verstanden – auch den älteren.
3. Es wird kein zusätzliches Plug-In benötigt.
4. Der Tag ist populär. Das soll Verschiedenes bedeuten. Einmal kennen viele Anwender dieses Tag. Das führt dazu, dass jeder, der den Quelltext liest, leicht erkennen kann, dass hier ein Java-Applet referenziert wird. Die Konkurrenz-Tags machen das dem Laien nicht so deutlich. Diese suggerieren etwa im Fall des `<OBJECT>`-Tags eine Verbindung zu der umstrittenen ActiveX-Technologie und das kann zur Ablehnung durch den Anwender führen. Die Popularität bedeutet aber auch, dass viele HTML-Tools dieses Tag automatisch verwenden, wenn ein Anwender ein Java-Applet einbindet.

7.3.3 Die <EMBED>-Syntax

Das `<EMBED>`-Tag ist eine Netscape-Erweiterung von HTML 3.2 zur allgemeinen Einbindung von Multimedia-Elementen und auch Applets in eine Webseite. Er wird gelegentlich auch Java-Plug-In-Tag genannt. Die Verwendung sollte mit Vorsicht geschehen, da die Technik im Wesentlichen auf Netscape Browser ab der Version 4.x beschränkt ist (einige Ausprägungen

des Tags werden aber auch in neueren Versionen des Internet Explorers verstanden). Java-Applets werden damit mit dem angegebenen Java-Plug-In von Sun gestartet.

Die allgemeine Syntax sieht so aus:

Listing 7.9: Schema für die Einbindung eines Applets mit <EMBED>

```
<EMBED type="[PlugIn]"
 pluginspage ="[URL]"
 code="[NameApplet].class"
 name="[NameApplet]"
 object="[serializedObjectOderJavaBean]"
 codebase="[VerzeichnisclassFile]"
 width="[Breite]"
 height="[Höhe]"
 align = [Ausrichtung]
 vspace = [Textfluss vertikal]
 hspace = [Textfluss vertikal] >
 ...
</EMBED>
```

Über das Attribut `type` geben Sie das verwendete Plug-In an. Diese Angabe kann das Plug-In noch genauer spezifizieren (etwa `"application/x-java-applet;version=1.2"`). Über die optionale Angabe `pluginspage` kann man eine Downloadmöglichkeit in Form einer URL angegeben, wo der ein lokal nicht vorhandenes Plug-In bei Bedarf automatisch geladen werden kann (etwa `pluginspage="http://java.sun.com/products/plugin/1.3/plugin-install.html"`).

Die Angaben `code` und `object` dürfen nicht gleichzeitig verwendet werden, die restlichen Attribute sind wie beim <APPLET>-Tag zu verstehen.

Die Einbindung von `HelloJavaApplet.class` funktioniert beispielsweise so:

Listing 7.10: Die Einbindung des Applets HelloJavaApplet.class mit <EMBED>

```
<HTML>
<BODY>
<EMBED
   type="application/x-java-applet;version=1.3"
   CODE = "HelloJavaApplet.class"
   CODEBASE = "."
   NAME = "HelloJavaApplet"
   WIDTH = 400
   HEIGHT = 300
   ALIGN = middle
   VSPACE = 0
   HSPACE = 0
   pluginspage=
   "http://java.sun.com/products/plugin/1.3/plugin-install.html">
</EMBED>
</BODY>
</HTML>
```

Abbildung 7.8: Die Darstellung im Navigator funktioniert mit <EMBED>

Mit dieser Form der Einbindung gibt es leider diverse Probleme.

Kontra:

1. Der Tag ist vom W3C als deprecated erklärt worden. Das ist aber wie beim <APPLET>-Tag nur ein »Papier-Tiger«-Argument.
2. Der Tag ist mit seinen Parametern relativ komplex. Bei vielen »Hobby-Homepage-Bauern« wird das ein Problem.
3. Der Tag ist in dieser vollständigen Version auf neuere Navigator-Versionen und kompatible Browser beschränkt. Man muss also für Internet-Explorer-Anwender zusätzlich ein weiteres Tag verwenden.
4. Ein Applet kann nicht dargestellt werden, wenn das angegebene Plug-In nicht zur Verfügung steht und gerade keine Onlineverbindung besteht, der Anwender kein extra Plug-In installieren möchte oder der Browser mit dem Plug-In nicht zurechtkommt.
5. Wenn das Java-Plug-In verwendet werden soll, muss beim Client eine passende, zusätzliche virtuelle Maschine vorhanden sein.

Pro:

1. Das Java-Plug-In wird unterstützt. Das bedeutet, ein Java-Applet kann theoretisch die gleichen APIs nutzen wie eine vollständige Java-Applikation. Das setzt aber voraus, dass auf der Client-Plattform eine passende virtuelle Maschine vorhanden ist.

7.3.4 Die <OBJECT>-Syntax

Das <OBJECT>-Tag ist *das* neue Standard-Tag unter HTML 4.0 zum Einbinden von Applets und beliebigen Multimedia-Objekten in eine Webseite. Eingeführt wurde der Tag in der Grundform ursprünglich als Erweiterung des Internet Explorer 4.x für HTML 3.2. Die Syntax sieht schematisch so aus:

```
<OBJECT
   classid="[ActiveX-ControlId des Java-Plug-In]"
   codebase="[URL Java-Plug-In]"
   WIDTH = [Breite]
   HEIGHT = [Höhe]
   NAME = "[Name Applet]"
   ALIGN = [Ausrichtung]
   VSPACE = [vertikal Abstand]
   HSPACE = [horizontal Abstand] >
   <PARAM
   NAME = CODE VALUE = "[Applet-Name].class">
   <PARAM NAME = CODEBASE
   VALUE = "[Verzeichnis-Class-File]" >
   <PARAM NAME = NAME VALUE = "[Applet-Name]" >
   <PARAM NAME="type"
   VALUE="application/x-java-applet; version=1.3">
</OBJECT>
```

Der Wert des `classID`-Attributs ist die weltweit eindeutige ActiveX-ID für das Java-Plug-In (immer `classid="clsid:8AD9C840-044E-11D1-B3E9-00805F499D93"`), das Sie setzen müssen, damit das `<OBJECT>`-Tag es im Browser aktiviert. Das `codebase`-Attribut wird dafür verwendet, das Java-Plug-In aus dem Netz zu laden, wenn es fehlt (etwa `codebase="http://java.sun.com/products/plugin/1.3/jinstall-13-win32.cab#Version=1,3,0,0"`).

Verwechseln Sie nicht das `codebase`-Attribut des `<OBJECT>`-Tags mit dem Parameter `codebase` (`<param name="codebase" value="[Verzeichnis-Class-File]">`), der optional eine relative URL zur Angabe des Ortes von der Applet-Klasse spezifizieren kann.

Die Angabe `<param name="JAVA_CODE"...` ist eine Spezifizierung für den Fall, dass es Konflikte zwischen anderen Parametern der Form `<param name="CODE"...` gibt.

Das `type`-Parameter-Tag wird zum Laden des Plug-Ins verwendet. In der Regel ist das wieder eine Angabe wie `<PARAM NAME="type" VALUE="application/x-java-applet; version=1.3">`

Für ein serialisiertes Objekt oder JavaBean wird das `type`-Parameter-Tag in der Regel so aussehen:

```
<param name="type" value="application/x-java-bean;version=1.1">
```

oder

```
<param name="type" value="application/x-java-bean">
```

Was spricht nun für und gegen den Einsatz dieses Tags?

Kontra:

1. Der Tag ist mit seinen Parametern relativ komplex. Mehr noch als beim <EMBED>-Tag werden viele Webseiten-Ersteller von der Syntax abgeschreckt. Allerdings können viele HTML-Tools die manuelle Erstellungen abnehmen.
2. Der Tag ist auf neuere Internet-Explorer-Versionen und kompatible Browser beschränkt. Man muss also für Navigator-Anwender zusätzlich einen weiteren Tag verwenden.
3. Ein Applet kann nicht dargestellt werden, wenn das angegebene Plug-In nicht zur Verfügung steht und gerade keine Onlineverbindung besteht, der Anwender kein extra Plug-In installieren möchte oder der Browser mit dem Plug-In nicht zurechtkommt.
4. Wenn das Java-Plug-In verwendet werden soll, muss beim Client eine passende, zusätzliche virtuelle Maschine vorhanden sein.
5. Der Tag arbeitet mit einer ActiveX-ID. ActiveX-bezogene Technik wird weder von allen Browsern im Internet unterstützt, noch erlauben halbwegs sicherheitsbewusste Surfer deren Verwendung. Unnötigerweise wird eine absolut sichere Technologie wie Java mit dem wohl größten Sicherheitsrisiko im Web in Verbindung gebracht.

Pro:

1. Der Tag ist vom W3C zum Standard-Tag erklärt worden. Aber genau wie bei den als deprecated gekennzeichneten Tags ist das kein echtes Argument. Weder Anwender noch die meisten Browser-Hersteller kümmert das in irgendeiner Weise.
2. Das Java-Plug-In wird unterstützt. Das bedeutet, ein Java-Applet kann theoretisch die gleichen APIs nutzen wie eine vollständige Java-Applikation. Das setzt aber voraus, dass auf der Client-Plattform eine passende virtuelle Maschine vorhanden ist.

7.3.5 Welches Tag ist sinnvoll?

Da es für jede der drei beschriebenen Referenzmodell Pro- und Kontra-Argumente gibt, muss jeder Anwender abwägen, was ihm am wichtigsten ist. Einige Aussagen können dabei aber als Leitschnur dienen:

- Wenn das Applet möglichst große Verbreitung erhalten soll und keine Java-Technologie jenseits von Java 1.0.2 verwendet wird, ist das <APPLET>-Tag die einzig sinnvolle Wahl. Offizielle Quellen hin oder her. Diese argumentieren politisch.

- Wird Java-Technologie verwendet, deren Unterstützung in den virtuellen Maschinen der Browser man sich nicht sicher ist, sollte man mit dem <APPLET>-Tag die potenziellen Zielbrowser testen. Läuft das Applet mit der <APPLET>-Referenz, sollte diese auf jeden Fall genommen werden.
- Verwendet man Technologie, die jenseits der virtuellen Maschinen der Zielbrowser angesiedelt ist, kommt man um die Tags zum Aufruf des Java-Plug-Ins nicht herum. Mit allen Konsequenzen. In einem Intranet sind die negativen Auswirkungen zu minimieren. Im Internet sollte man sehr gut abwägen. Die bessere Alternative ist es sicher, wenn es irgend geht, bei Applets auf Java-APIs zu verzichten, die in Browsern Probleme machen können.

Nutzen Sie den in Kapitel 1 näher beschriebenen HTML-Konverter (zu laden von der Sun-Plug-In-Homepage `http://java.sun.com/products/plugin`*), der aus einer Webseite mit dem* <APPLET>*-Tag eine solche mit* <OBJECT>*-Tag (für den Internet Explorer) bzw.* <EMBED>*-Tag (für den Netscape Navigator) macht.*

7.3.6 Die ursprüngliche HotJava-Syntax

Nur der Vollständigkeit halber sei die HotJava-Syntax zum Referenzieren eines Applets innerhalb einer HTML-Seite noch kurz angerissen. Sie definieren die Referenz auf ein Applet mit dem Befehl <APP>. Dieser Befehl hat offiziell kein abschließendes Tag. Parameterangaben werden in Form von Zusatzangaben innerhalb des <APP>-Tags notiert. Beachten muss man, dass die Klasse ohne `.class` angegeben werden muss.

7.4 Die interne Arbeitsweise eines Applets

Was passiert nun mit den Parametern, die beispielsweise mit dem <PARAM>-Tag an das Applet weitergereicht werden? Und wie wird eigentlich ein Applet in seinem Inneren funktionieren? Es geht also ab jetzt darum, HTML wieder zu verlassen und zu Java zurückzukehren. Um diese Fragen zu beantworten, sollten wir uns an den Aufbau eines Applets erinnern, wie wir ihn schon vor unserem ersten Applet angesprochen haben. Eine eigenständig lauffähige Java-Applikation benötigt – wie wir jetzt schon mehrfach gesehen haben – immer nur eine `main()`-Methode. Ein Java-Applet besitzt im Gegensatz zu einem Java-Programm im Allgemeinen keine `main()`-Methode[2], die beim Laden gestartet wird und das Programm solange am Leben erhält, bis der Benutzer es beendet. Statt dessen sorgen bei voller Funktions-

[2] oder benutzt zumindest keine

fähigkeit mindestens vier andere Methoden – die `init()`-Methode, die `start()`-Methode, die `stop()`-Methode und die `destroy()`-Methode – im Programm dafür, dass sich das Applet in seine Umgebung korrekt verhält.

Ein Applet befindet sich immer innerhalb eines Containers und muss sich kooperativ zu ihm verhalten. Es muss also die Möglichkeiten nutzen, die ihm der Container zur Verfügung stellt. Um nun erklären zu können, was beispielsweise konkret mit den Parametern passiert, müssen wir noch ein wenig weiter ausholen.

7.4.1 Erstellung und Grundmethoden eines Applets

Um ein Applet zu erstellen, müssen Sie immer eine Subklasse der Klasse `java.applet.Applet` (fundamentaler Bestandteil von Java) erzeugen. Diese Klasse beinhaltet bereits sämtliche Eigenschaften, die die Kooperation mit dem Container sicherstellen, die Fähigkeiten des AWT für Oberflächenprogrammierung (Menüs, Button, Mausereignisse usw.) nutzbar machen und auch die lebenswichtigen Grundmethoden für ein Applet. Letzteres war auch der Grund, warum unser erstes Applet funktioniert hat, obwohl wir auf diese Methoden verzichtet haben. Sie wurden einfach aus der Superklasse gezogen. Normalerweise werden Sie diese jedoch überschrieben.

Jede Applet-Klasse wird mit folgender Syntax als Unterschrift erstellt:

```
public class <AppletName> extends java.applet.Applet
```

Java setzt übrigens für die sinnvolle Verwendung voraus, dass eine Applet-Klasse als public deklariert wird.

Applets sollten `java.applet.*` *importieren, um sich im Laufe des Quelltexts Schreibarbeit zu sparen. Applets mit grafischen Schnittstellen benötigen meist auch java.awt.*.*

Im Inneren des Applet-Codes werden dann die notwendigen Methoden für die Funktionalität des Applets platziert. Darunter fallen natürlich ebenfalls die Grundmethoden, die das Leben eines Applets festlegen.

Schauen wir uns nun diese vier Stationen im Leben eines Applets im Detail an. Sie werden normalerweise in einem Applet sie teilweise oder gar allesamt überschreiben.

Die Initialisierung

Der Einstieg in ein Applet-Leben ist immer die Initialisierung. Diese erfolgt immer mit der Methode `public void init()`, die automatisch aufgerufen

wird, sobald das Applet in den Browser (oder einen anderen Container wie den Appletviewer) geladen wird. Hier wird das System zur Ausführung des Applets vorbereitet. An dieser Stelle werden beispielsweise die Bildschirmanzeige initialisiert, Netzwerkverbindungen oder Datenbankzugriffe aufgebaut oder Schriften eingestellt.

Und hier ist auch die Stelle, wo die Parameter aus dem HTML-Code der Seite eingelesen werden können. Sie können hier mit der Methode `public String getParameter(String name)` auf diese Parameter zugreifen.

Mehr zur Auswertung von Übergabewerten an ein Applet finden Sie etwas weiter unten auf Seite 368.

Der Start

Die Methode `public void start()` wird beim Start des Applets direkt nach der Initialisierung aufgerufen. Während die Initialisierung nur einmal erfolgt, kann (und wird normalerweise) der Start beliebig oft wiederholt werden. Beispielsweise, wenn das Applet gestoppt wurde oder das Applet durch scrollen des HTML-Dokuments erneut in den sichtbaren Bereich des Browsers geholt wird. Um das Startverhalten Ihres Applets zu spezifizieren, überschreiben Sie die `start()`-Methode. Dort können diverse Funktionalitäten stehen, beispielsweise das Starten eines Threads oder der Aufruf einer anderen Methode.

Das Stoppen eines Applets

Das Stoppen eines Applets hört sich schlimmer an, als es wirklich ist. Die Methode `public void stop()` wird automatisch aufgerufen, sobald Sie die HTML-Seite, auf der das Applet läuft, verlassen. Ein gestopptes Applet ist jedoch weiter voll funktionsfähig und wird bei der Rückkehr auf die Seite automatisch durch die Methode `start()` wieder zu neuem Leben erweckt.

Erinnern Sie sich an den Appletviewer? In dem Menü finden Sie die Start- und Stop-Funktionalitäten wieder.

Das Zerstören eines Applets

Wenn ein Applet-Container beendet wird, ermöglicht die Methode `public void destroy()` dem Applet, vernünftig hinter sich aufzuräumen und die letzten Systemressourcen, die es belegt hatte, freizugeben. Unter solche Aufräumarbeiten fallen die Beendigung von laufenden Threads oder die Freigabe von anderen laufenden Objekten. Normalerweise müssen Sie die `destroy()`-Methoden nicht überschreiben, sofern Sie nicht explizit Ressourcen freigeben müssen.

Noch einmal zur Erinnerung: Die allgemeinere `finalize()`*-Methode ist hier nicht zu verwenden, da diese eher für ein einzelnes Objekt irgendeines Typs zum Aufräumen verwendet wird und vor allem, weil im Gegensatz zu der* `finalize()`*-Methode die* `destroy()`*-Methode immer beim Beenden eines Browsers oder beim Neuladen eines Applets automatisch ausgeführt wird.*

7.4.2 Ein Musterapplet als Schablone

Nachfolgend sehen Sie ein Musterapplet, das als Gerüst für sämtliche Applets verwendet werden kann. Es besteht im Wesentlichen aus dem Import einiger wichtiger Java-Pakete, die für das Applet von Bedeutung sein können. Diese sind in der Schablone auskommentiert und müssen bei Bedarf aktiviert werden. Wenn Sie weitere Pakete benötigen, können Sie diese selbstverständlich hinzufügen. Sie können sogar alle denkbaren Pakete importieren (ob Sie diese dann brauchen oder nicht), denn das Java-Importieren ist ja kein Einbeziehen der Pakete in den Source wie mit `include` in C. Daher werden Sie keinen größeren Sourcecode bekommen und auch sonst keine Verluste erleiden. Es könnte nur sein, dass Ihr Quellcode an Übersichtlichkeit verliert.

Der Klassenname in der Top-level-Klassen-Deklaration des Applets muss natürlich durch die korrekte Bezeichnung ersetzt werden.

Listing 7.11: Eine Schablone für ein Applet

```
// Name der Klasse
// Beschreibung:
// Import der Pakete
// import java.applet.*;
// import java.awt.*;
// import java.awt.image.*;
// import java.awt.peer.*;
// import java.io.*;
// import java.lang.*;
// import java.net.*;
// import java.util.*
// Top-level-Klassen-Deklaration  des Applets.
public [Klassenname] extends java.applet.Applet {
// Variablen-Deklaration
// Bauen Sie hier Ihre eigenen Methoden ein
// Die Methoden, die überschrieben werden
 public void init() {
 }
public void start() {
}
public void stop() {
}
public void destroy() {
}
// Optional, aber sehr wahrscheinlich -
```

```
// die Ausgabemethode
// public void paint(Graphics g) {
// }
// Bei Multithreading die run()-Methode verwenden.
// public void run() {
// }
}
```

7.5 Die Applet-Klasse

Wir haben jetzt gesehen, dass ein Applet von der Applet-Klasse abgeleitet wird. Die `Applet`-Klasse selbst wiederum erbt von einer ganzen Reihe anderer Klassen. Eine Klasse, die `java.applet.Applet` erweitert, kann Variablen und Methoden (sowie Konstruktoren) aus `java.lang.Object`, `java.awt.Component`, `java.awt.Container` und `java.awt.Panel` verwenden. Schauen wir uns die `Applet`-Klasse noch ein bisschen genauer an.

Die Methoden der `Applet`-Klasse ermöglichen Applets diverse Grundfunktionen. Sie erlauben Applets,

- Parameter zu handhaben,
- Grafik zu handhaben,
- Bilder zu importieren,
- Sound-Clips zu importieren und abzuspielen,
- mit dem Browser oder dem Appletviewer zu interagieren oder
- Lebenszyklen des Applets zu verwalten.

Wir wollen uns ein paar wichtige Methoden ansehen, die diese Vorgänge unterstützen. Dabei muss als Hinweis vorausgeschickt werden, dass diese Vorstellung nur ein Schlaglicht auf die Möglichkeiten eines Applets werfen wird. Insbesondere werden nicht sämtliche Varianten der angesprochenen Methoden aufgeführt, sondern jeweils eine sinnvolle Version ausgewählt.

Selbst ein einfaches Applet besteht also in der Regel aus mehr als nur den genannten vier Grundmethoden. Da sind zum einen die selbst definierten Methoden, die z.B. in der `start()`-Methode aufgerufen werden können. Es gibt jedoch noch ein paar Standardmethoden, die so wichtig sind, dass wir sie hier behandeln wollen. Einige davon kennen Sie bereits. Wir haben sie in unserem ersten Applet verwendet.

7.5.1 Ausgabe in einem Applet mit der paint()-Methode

Erinnern Sie sich an die Ausgabe in unserem ersten Applet? Wir haben dazu nicht die `println()`-Methode, sondern eine Ausgabe innerhalb der `public void paint(Graphics g)`-Methode verwendet. Wir haben unseren Text gezeichnet und nicht geschrieben. Zeichnen bestimmt im Allgemeinen, wie ein Applet etwas auf dem Bildschirm ausgibt. Dies kann beliebig oft während eines Applet-Lebens vorkommen. Um einem Applet das Zeichnen zu ermöglichen, müssen sie diese `paint()`-Methode überschreiben. Diese steht jedem Applet defaultmäßig zur Verfügung und wird automatisch aufgerufen, wenn es notwendig ist (d.h. beim Starten des Applets, wenn das Applet zum Teil verdeckt wurde und wieder ein größerer Bereich angezeigt wird usw.).

Die `paint()`-Methode wird automatisch aufgerufen, wenn Sie das Applet (erneut) starten.

Es wird Ihnen sicher auffallen, dass die `paint()`-Methode einen Parameter hat: eine Instanz der Klasse `java.awt.Graphics`. Deshalb müssen Sie sicherstellen, dass die Graphics-Klasse in Ihren Applet-Code importiert wird, oder den Parameter vollqualifiziert angegeben.

Die Grafik-Ausgabe ist ein so wichtiges Thema, dass wir es hier abbrechen und in einem eigenen Kapitel gründlich durchsprechen wollen. Nichtsdestotrotz wird in den nachfolgenden Beispielen die `paint()`-Methode verwendet werden.

Auch auf eigenständigen Applikationen mit grafischer Oberfläche steht die `paint()`-Methode zur Verfügung.

7.5.2 Übergabewerte für ein Applet

Über die `getParameter()`-Methode können Sie auf jedes Argument zugreifen, das unter HTML in der Applet-Referenz mit dem `name`-Wert spezifiziert wurde. Dieser Wert von `name` ist eine Zeichenkette, die in der `getParameter()`-Methode in den Klammern angeben werden muss. Die Rückgabe der `getParameter()`-Methode ist der mit `value` in der HTML-Datei spezifizierte Wert.

Beispiel:

```
String parameter1 = getParameter("htmlParameterName")
```

Falls Sie einen Parameter spezifizieren, der in der HTML-Datei nicht angegeben wurde, gibt die `getParameter()`-Methode `null` zurück. Dies sollte man immer mit einer Abfrage auswerten und einen Standardwert einstellen.

Beispiel:

```
if (parameter1 == null) parameter1 = "Standardwert1"
```

Der im HTML-Parameter-Tag angegebene Name und der in der `getParameter()`*-Methode spezifizierte Name müssen absolut identisch sein, auch in Bezug auf Groß- und Kleinschreibung.*

Da die Rückgabe der `getParameter()`-Methode immer eine Zeichenkette ist, müssen alle Übergabeparameter eines Java-Applets in Form von Zeichenketten übergeben und dann im Source ebenso behandelt werden. Wenn Sie die Parameter in einem anderen Datentyp verwenden wollen, müssen Sie diese konvertieren. Sie können allerdings (wie im letzten Kapitel besprochen) nicht per Casting vorgehen, sondern müssen die Instanzen der als Ersatz dafür vorhandenen Wrapper-Klassen verwenden. Diese sind für alle primitiven Datentypen vorhanden. Wenn Sie z.B. einen Parameter in einen `long`-Wert umwandeln wollen, kann das so erfolgen:

```
long l = new Long(stringParameter).longValue()
```

Wie ein Applet-Code Standard-Parameter aus einem HTML-Dokument erhält, haben wir also damit gesehen. Wie aber stellt das Applet Informationen über benötigte Parameter bereit? Was von einem Anwender des Applets verlangt wird, ist ja, dass er die Parameter in der HTML-Datei richtig angibt, also den Namen der Übergabewerte, den jeweiligen Datentyp und eine Beschreibung. Diese Informationen kann ein Applet in einer Methode dokumentieren, die ein Applet-Client dann beispielsweise auswerten kann. Es handelt sich um die Methode

```
public String[][] getParameterInfo().
```

Diese gibt ein Zeichenketten-Array wieder, wo alle Parameter aufgelistet werden können, die das Applet benötigt. Diese Zeichenkette spezifiziert für jeden Parameter den Namen, den Typ und eine Beschreibung. Beispiel (mit zwei Parametern):

```
public String[][] getParameterInfo() {
 String[][] pinfo = {
  {"MeinPara1", "String", "Stringübergabewert"},
  {"MeinPara2", "int", "int-Übergabewert"}, };
 return pinfo;
 }
```

Listing 7.12: Beispiel für eine überschriebene getParameterInfo()-Methode

In den gleichen Zusammenhang fällt die Methode

```
public String getAppletInfo().
```

Mit dieser Methode kann man Angaben über das Applet machen, etwa den Autor, die Version, das Erstellungsdatum usw. Auch diese Methode kann in einem Applet-Client abgefragt werden.

Direkt mit dem Applet-Client kommuniziert die Methode

```
public void showStatus(String msg).
```

Diese schreibt einen Text in die Statuszeile des Browsers. Allerdings kollidiert diese Methode in einigen Situationen mit Standardausgaben vom Browser und vor allem dem Appletviewer und ist deshalb oft nicht sehr effektiv.

Wenden wir diese Methoden nun in einem Beispiel an. Zuerst wird eine HTML-Datei mit Übergabe-Parametern erstellt.

Listing 7.13: Die HTML-Datei mit der Applet-Referenz und Parametern

```
<HTML>
<BODY>
<APPLET CODEBASE = "."
  CODE = "UebergabeApplet.class"
  NAME = "TestApplet"
  WIDTH = 400 HEIGHT = 300
  HSPACE = 0  VSPACE = 0  ALIGN = middle>
<PARAM NAME = "MeinPara1"
   VALUE = "Das wird ausgegeben">
<PARAM NAME = "MeinPara2" VALUE = "2">
</APPLET>
</BODY>
</HTML>
```

An das Applet wird ein String übergeben, der dann im Applet ausgegeben werden soll. Der zweite Übergabewert ist ein `int`-Wert, der intern im Applet zum Setzen der Hintergrundfarbe verwendet wird. Die `paint()`-Methode wird zur Ausgabe verwendet.

Beachten Sie, dass wir in dem Beispiel bewusst das Schlüsselwort `this` *einsetzen. An einigen Stellen wird es aus Demonstrationsgründen eingesetzt, um in der gleichen Situation an anderer Stelle darauf zu verzichten.*

Listing 7.14: Das Applet mit der Auswertung von Übergabewerten plus der Bereitstellung von Informationen gegenüber Applet-Clients

```
import java.awt.*;
import java.applet.*;
public class UebergabeApplet extends Applet {
String meinP1;
int meinP2;
 // Laden der Parameterwerte
 // Mit Test, ob der Parameter auch vorhanden ist
public String getParameter(String key, String def) {
 return getParameter(key) != null
```

Abbildung 7.9: Das Applet – in der Statuszeile ist die Reaktion auf die Methode showStatus() zu sehen.

```
? getParameter(key) : def;
}
  //Initialisieren des Applets
public void init() {
meinP1 =
  this.getParameter("MeinPara1", "Vorbelegung");
meinP2 =
Integer.parseInt(getParameter("MeinPara2", "2"));
    // Setze Hintergrundfarbe
switch (meinP2) {
    case 1:
    this.setBackground(Color.cyan);
    break;
    case 2:
    this.setBackground(Color.red);
    break;
    case 3:
    this.setBackground(Color.yellow);
    break;
    default:
    this.setBackground(Color.white);
    }
    // Setze die Groesse des Applets
this.setSize(new Dimension(400,300));
```

```
}
public void paint(Graphics g) {
  g.drawString(meinP1,50,50);
  g.drawString("Farbwahl war: "
    + new Integer(meinP2).toString(),50,150);
  showStatus("Das Applet wurde gestartet");
  }
  //Die Applet-Information
public String getAppletInfo() {
 return "Die Applet Information: (C) RJS, Version 1.0, 11/2000 ";
}
// Parameterinfo
public String[][] getParameterInfo() {
 String[][] pinfo = {
  {"MeinPara1", "String",
    "Text, der ausgeben werden soll"},
  {"MeinPara2", "int", "Farbwahl"}, };
 return pinfo;
 }
}
```

Im Appletviewer kann man über das Menü auf einfache Weise die bereitgestellten Informationen des Applets abgreifen.

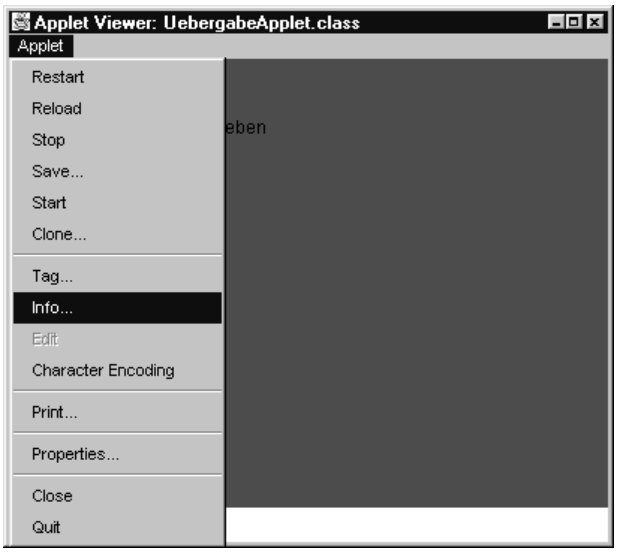


Abbildung 7.10: Im Appletviewer kann man auf die Methoden getAppletInfo() und getParameterInfo() zugreifen.

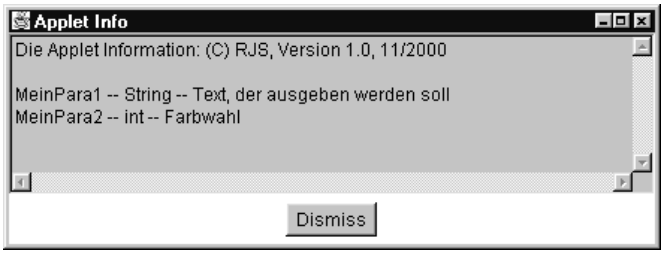


Abbildung 7.11: Die Infos, die das Applet bereitstellt, im Appletviewer

7.5.3 Bilder importieren

Selbstverständlich können Applets Bilder importieren und ausgeben. Dazu können die folgenden Methoden genutzt werden:

```
getImage()
drawImage()
```

Die Methode `public Image getImage(URL url, String name)` lädt Bild-Dateien in ein Applet. Beispiel:

```
bild1 = getImage(getCodeBase(), "bild.gif");
```

Das Beispiel lädt das Bild `bild.gif`. In unserem Beispiel verwenden wir die Methode `getCodeBase()` als Argument für die Ermittelung des URLs, d.h. wir laden das Bild von der gleichen Stelle, wo sich auch die HTML-Datei befindet (gleich noch mehr dazu).

Als Bild können Sie jedes der unter Java erlaubten Formate (etwa GIF oder JPEG) verwenden. Das Thema Bildverarbeitung unter Java ist natürlich mit diesem kurzen Beispiel nicht erschöpft, sondern wird im Rahmen des Kapitels über Grafik und Animation ausführlich erläutert. Dort finden Sie dann auch nähere Erklärungen zu der Methode. Hier soll nur die prinzipielle Möglichkeit eines Applets zum Laden eines Bildes angesprochen werden.

Mit der Methode `public abstract boolean drawImage(Image img, int x, int y, ImageObserver observer)` können Sie ein geladenes Bild anzeigen.

Beispiel:

```
g.drawImage(meinBild, 50, 50, this);
```

Wir wollen ein kleines Applet erstellen, das ein Bild lädt und auf dem Bildschirm wieder ausgibt. Die folgende HTML-Datei dient zum Einbinden.

Listing 7.15: Die HTML-Datei

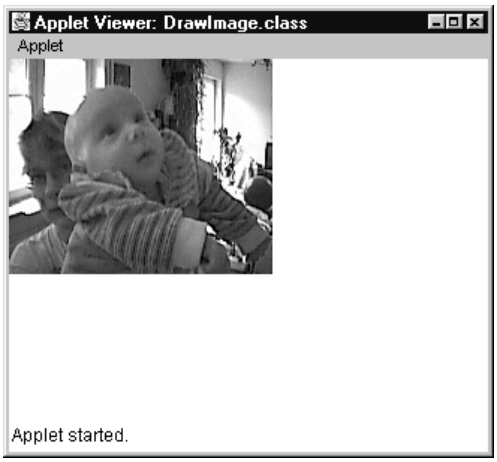
```
<HTML>
<BODY>
<APPLET CODEBASE = "." CODE = "DrawImage.class"
  WIDTH = 400 HEIGHT = 300></APPLET>
</BODY>
</HTML>
```

Das Applet sieht so aus:

Listing 7.16: Ein Applet mit Bild-Ausgabe

```
import java.awt.Image;
import java.awt.Graphics;
public class DrawImage extends java.applet.Applet {
 Image samImage;
 public void init() {
// ein Bild wird geladen
  samImage = getImage(getDocumentBase(), "images/baby.jpg");
  resize(320, 240);  // Größe des Applets
 }
 public void paint(Graphics g) {
// Ausgabe des Bildes
  g.drawImage(samImage, 0, 0, this);
 }
}
```

Abbildung 7.12: Ein Bild in einem Applet



Das Beispiel setzt natürlich voraus, dass im dem Unterverzeichnis `images` die entsprechende Grafik vorhanden ist. In diesem Fall wird in der `init()`-Methode die Grafik geladen und dann die Applet-Größe angepaßt. Nach der Initialisierung des Applets wird als nächstes die `paint()`-Methode aufgerufen. Diese verwendet unsere hier besprochene Methode dann zum Anzeigen des aktuellen Bildes. Mehr dazu finden Sie beim Abschnitt über Grafiken.

7.5.4 Importieren und Abspielen von Sound

Multimedia-Eigenschaften waren ja am Anfang ein Grund für den einschlagenden Erfolg von Java beim ersten Auftauchen im Internet[3]. Dazu zählt natürlich ebenfalls die Akustik. Ab der Java-Version 1.2 können Sie MIDI-Dateien (Typ 0 und Typ 1) sowie RMF-, WAVE-, AIFF-, und AU-Dateien in hoher Tonqualität abspielen und zusammenmischen. Ein Applet kann über die Methode `public AudioClip getAudioClip(URL url, String name)` sowohl Audio-Clips als auch Sound-Clips importieren.

Beispiel:

```
meinSoundClip = getAudioClip(getCodeBase(), "history.wav");
```

Die Methoden `public void play(URL url, String name)` und `public abstract void loop()` dienen zum Abspielen und Wiederholen des Clips.

7.5.5 Die Interaktion des Applets mit der Umgebung und den Lebenszyklus eines Applets verwalten

Jedes Applet kann mit seiner Umgebung in Kontakt treten und auf sie reagieren. Dies ergibt sich in einfacher Form schon allein deshalb, weil es eine Ableitung der Klasse `Applet` ist. Die einfachsten (und grundlegendsten) Funktionalitäten eines Applets sind, dass es sich starten lässt und – hier kommt die Interaktion zur Laufzeit ins Spiel – vernünftig beenden lässt (es reagiert auf Schließbefehle des Fensters). Java besitzt dazu einen Mechanismus, durch den Applets in Schlüsselmomenten, in denen bestimmte Ereignisse eintreten, die Kontrolle übernehmen können. Die schon behandelten Methoden `init()`, `start()`, `stop()` und `destroy()` werden entsprechend aufgerufen, wenn das Applet geladen wird, mit der Ausführung beginnt, mit der Ausführung stoppt und zerstört wird. Wenn ein Applet zu einem dieser Schlüsselmomente etwas ausführen soll, dann müssen Sie nur die entsprechende Methode überschreiben, indem Sie Ihre eigene Implementierung davon in das Applet einbauen.

Interaktionen gehen jedoch selbstverständlich noch weiter. Ein etwas komplizierteres Applet wird auf Buttons, Mausaktionen im Allgemeinen, Menübefehle und/oder Tastatureingaben reagieren. Dies ist sicher nicht überraschend. So etwas nennt man ein Programm mit einer (sinnvollen) Anwenderschnittstelle. Java stellt Ihnen Methoden zur Verfügung, die das System bei bestimmten Ereignissen aufrufen kann. Wenn Sie irgendeine Aufgabe als Reaktion auf ein Ereignis erledigen wollen, müssen Sie nur die entsprechende Methode überschreiben.

[3] Mittlerweile ist das exzessive Multimedia-Bombardement per HTML-Grafiken, JavaScript, Java oder ActiveX leider nur noch als Anti-Kriterium für das Laden von Webseiten zu sehen.

Interaktionen betreffen jedoch auch die Plattform, unter der das Applet läuft (Zugriffe auf Ressourcen, soweit dies erlaubt ist), sowie Kontakte über Netzwerkverbindungen hinweg.

Interaktion ist also ein recht weit gefasster Begriff. Schauen wir uns ein paar Methoden an, mit denen ein Applet auf seine Umwelt reagiert und darauf Einfluss nimmt.

Die Methode getAppletContext()

Die Methode `public AppletContext getAppletContext()` gibt ein Objekt vom Typ `AppletContext` (eine Schnittstelle) zurück, das Sie dazu verwenden können, Informationen und Kontrolle über die Umgebung des Applets zu erlangen. `AppletContext`-Methoden ermöglichen es Ihnen,

- herauszufinden, welche Applets außer dem aktuellen noch auf derselben Seite laufen,
- Videos und Sounds zu importieren,
- eine andere Webseite aus dem Applet heraus zu laden,
- den Kontext eines anderen Applets zu importieren und dessen Methoden aufzurufen oder
- Meldungen an andere Applets weiterzugeben.

Konkret haben Sie über ein Objekt dieses Typs folgende Methoden zur Verfügung:

```
public AudioClip getAudioClip(URL url)
public Image getImage(URL url)
public Applet getApplet(String name)
public Enumeration getApplets()
public void showDocument(URL url)
public void showDocument(URL url, String target)
public void showStatus(String status)
```

Schauen wir uns davon einige in der Praxis an. Das folgende Beispiel lädt aus dem Applet heraus über einen `AppletContext` direkt eine neue Webseite. Dazu verwenden wir die `showDocument()`-Methode. Beachten Sie, dass wir dieser ein `URL`-Objekt übergeben müssen, das wir anonym innerhalb der Parameterklammern erzeugen. Dabei müssen wir eine Ausnahmebehandlung vornehmen. Auf die näheren Details dazu gehen wir noch ein. In dem Beispiel wird die Webseite `Applet2.html` geladen.

Listing 7.17: Laden einer neuen Webseite aus einem Applet heraus

```
import java.awt.*;
import java.applet.*;
import java.net.*;
public class Applet1 extends Applet {
```

```
  public void init()  {
   try  {
   getAppletContext().showDocument(new
URL(getDocumentBase(),"Applet2.html"));
   }
   catch(Exception e)   {
    System.out.println(e.toString());
   }
  }
}
```

Das nächste Beispiel gibt Informationen über andere in der Webseite geladenen Applets aus. In der nachfolgenden Webseite sind zwei Applets geladen.

Listing 7.18: Eine Webseite mit zwei Applets

```
<HTML>
<BODY>
<APPLET
  CODEBASE = "." CODE = "UebergabeApplet.class"
  NAME = "TestApplet"
  WIDTH = 400 HEIGHT = 300
  HSPACE = 0 VSPACE = 0 ALIGN = middle>
<PARAM NAME = "MeinPara1" VALUE = "Das wird ausgegeben">
<PARAM NAME = "MeinPara2" VALUE = "1">
</APPLET>
<br>
<APPLET CODEBASE = "." CODE = "Applet2.class"
  WIDTH = 500 HEIGHT = 200></APPLET>
</BODY>
</HTML>
```

Das zweite Applet greift mittels der Methode `getApplets()` und dort via `nextElement()` auf das erste Applet zu und gibt Informationen darüber aus.

Listing 7.19: Zugriff auf Interna des Nachbar-Applets in der Webseite

```
import java.awt.*;
import java.applet.*;
public class Applet2 extends Applet {
public void paint(Graphics g) {
g.drawString(getAppletContext().getApplets().
nextElement().toString(),50,100);
 } }
```

Die Methoden getCodeBase() und getDocumentBase()

Schon an verschiedenen Stellen in diesem Kapitel ist die Methode `public URL getCodeBase()` verwendet worden. Damit wird der[4] URL des Applets selbst zurückgegeben. Die Methode `public URL getDocumentBase()` funktio-

[4] »Der« URL oder »die« URL ist wieder so ein Fachbegriff, wo sich Duden und Fachleute streiten. Ich verwende gegen die Empfehlung des Duden »der«.

niert ähnlich, nur wird der URL des Verzeichnisses mit dem HTML-Dokument zurückgegeben.

Die Methode isActive()

Die Methode `public boolean isActive()` stellt fest, ob ein Applet aktiv ist. Sie können diese Methode zum Überprüfen des Zustands eines anderen Applets mit dem Aufruf von `AppletContext.getApplets()` im Kontext und durch den Aufruf von `isActive()` für ein bestimmtes Applet benutzen.

Die Methode setStub()

Sie können die Methode `public final void setStub(AppletStub stub)` verwenden, um eine neue `AppletStub`-Schnittstelle zu erstellen. Eine mögliche Verwendung ist die Implementation eines neuen Appletviewers. Normalerweise wird der `AppletStub` automatisch durch das System gesetzt. Wir verfolgen die Methode nicht weiter.

Die Methode resize()

Diese Methode werden Sie in vielen Applets entdecken. Damit kann man die Größe eines Fensters und im Prinzip auch jedes Applets festlegen. Die Methode `public void resize(Dimension d)` bzw. `public void resize(int x, int y)` legt die Größe eines Fenster bzw. des Anzeigebereichs des Applets fest. Dabei ist zu beachten, dass die äußeren Größenangaben der HTML-Datei die angezeigte Größe eines Applets beeinflussen, d.h., in einem Browser gelten die dort festgelegten Größenangaben. Diese Methode kann nicht bewirken, dass ein über HTML-Größenangaben festgelegtes Applet in einer Webseite innerhalb eines Browsers redimensioniert wird. Der Appletviewer hingegen reagiert dementsprechend mit einer Anpassung der Größe. Sinn macht die Methode aber vor allem, wenn ein eigenständiges Folgefenster aktiviert wird.

7.5.6 Methoden zur Ereignisbehandlung in Applets

Natürlich kann ein Applet auch auf Ereignisse wie Mausbewegungen oder Tastatureingaben reagieren. Wir wollen dieses Verhalten noch im Rahmen eines Abschnittes über das prinzipielle Handling von Ereignissen in Java ausführlicher behandeln. Wir werden dort sehen, wie Java konkret auf die nachfolgenden Aktionen reagiert. Hier interessieren uns nur die Möglichkeiten und die Anwendung in einfachen Beispielen.

Die meisten der nachfolgenden Methoden werden von Sun als deprecated bezeichnet. Wie aber schon verschiedentlich besprochen, müssen Sie sich bei Applets in vielen Fällen auf Java 1.0.x beschränken, was insbesondere die Verwendung des Eventmodells 1.0 erzwingt. Dieses unterscheidet sich erheblich von dem mittlerweile verwendeten Eventmodell 1.1. Die nachfol-

genden Methoden gehören allesamt zum Eventmodell 1.0. Wir werden im Abschnitt über die beiden Eventmodelle darauf noch näher eingehen und die beiden Modelle gegenüberstellen.

Mausaktionen

Zu den wichtigsten Interaktionen eines Applets (und natürlich eines gewöhnlichen Programms) zählt die Reaktion auf Mausereignisse. Dabei soll hier nicht die Auswertung von Button und Menüeinträgen im Blickpunkt stehen, sondern die Reaktion auf Mausereignisse, die sich irgendwo im Bereich des Applets abspielen. Obwohl es nicht zwingend ist, beschränken wir uns auf die übliche linke Maustaste.

Es gibt sechs mögliche Mausereignisse:

- Die Maustaste wird gedrückt.
- Die gedrückte Maustaste wird wieder losgelassen.
- Die Maus wird mit gedrückter Maustaste bewegt (Drag).
- Die Maus wird mit losgelassener Maustaste bewegt.
- Der Mauszeiger verlässt den Bereich des Applets.
- Der Mauszeiger kommt (wieder) in den Bereich des Applets.

Für jedes Ereignis stellt Java in dem alten Eventmodell eine Methode zur Verfügung, die zur Applet-Klasse gehört.

Über die Methode

```
public boolean mouseDown(Event evt, int x, int y)
```

können Sie Aktionen auslösen, wenn der Benutzer eine Maustaste drückt.

Dabei ist `Event` die Ereignisklasse, die die Informationen über das Ereignis enthält und `x` und `y` sind die Koordinaten, bei denen die Maustaste gedrückt wurde.

Das Gegenstück zu der Methode `mouseDown()` ist die Methode

```
public boolean mouseUp(Event event, int x, int y),
```

über die Sie Aktionen auslösen können, wenn der Benutzer eine Maustaste losgelassen hat. Die `mouseUp()`-Methode verwendet die gleichen Argumente wie `mouseDown()`.

Die Methode

```
public boolean mouseEnter(Event event, int x, int y)
```

wird immer dann aufgerufen, wenn der Mauszeiger in den Bereich des Applets bewegt wird. Die Maustaste muss nicht extra gedrückt werden, damit die Methode aufgerufen wird. Die Methode verwendet die gleichen Argumente wie die bereits beschriebenen.

Die Methode

```
public boolean mouseExit(Event event, int x, int y)
```

ist das Gegenstück zu `mouseEnter()` und wird immer dann aufgerufen, wenn der Mauspfeil aus den Bereich des Applets bewegt wird. Die Maustaste muss nicht extra gedrückt werden und es gelten wieder die üblichen Argumente.

Die Methode

```
public boolean mouseDrag(Event event, int x, int y)
```

wird immer dann aufgerufen, wenn die Maus mit gedrückter Taste im Bereich des Applets bewegt wird und die Methode

```
public boolean mouseMove(Event event, int x, int y)
```

wird immer dann aufgerufen, wenn die Maus bewegt wird und keine Taste gedrückt ist. Es gelten wieder die üblichen Argumente.

Jedes Mal, wenn die Maustaste gedrückt wird, werden vor dem Aufrufen von `mouseDown()` *die Methoden* `mouseExit()` *und* `mouseEnter()` *aufgerufen. Das bedeutet, dass die Methode* `mouseEnter()` *keinen Code enthalten sollte, der bei der Aktion* `mouseDown()` *schaden kann.*

Wir wollen zum Ende des Kapitels ein etwas komplexeres Applet zusammenbauen, das einige der bisher erarbeiteten Funktionalitäten nutzt. Es soll auf Mausaktionen reagieren, die `getInfo()`- und `getCodeBase()`-Methoden unterstützen und auf Wunsch ausgeben, Übergabeparameter verwenden, auf Arrays zugreifen, ein Bild laden und dieses dann ausgeben. Der Source wird mit Kommentaren dokumentiert, sodass er sicher verständlich bleibt.

Listing 7.20: Ein Applet, das auf verschiedene Mausaktionen reagiert

```
import java.applet.*;
import java.awt.*;
public class AppletReaktionen extends Applet {
/* Variablen zur Aufnahme von Übergabeparameter 1 - 3 */
 private String m_Para1 = "";
```

```
 private String m_Para2 = "";
 private String m_Para3 = "";
/* Die Parameternamen in der HTML-Datei. */
 private final String PARAM_Para1 = "Para1";
 private final String PARAM_Para2 = "Para2";
 private final String PARAM_Para3 = "Para3";
/* Eine Image-Variable*/
 Image samImage;
/* Eine Instanz von Graphics, die für diverse
 Ausgabeaktionen genutzt wird. */
 private Graphics ausgabe;
 public String getAppletInfo() { // Die Info
 return
"Name: AppletReaktionen " + "Autor: Ralph Steyer";
 }
// Die Parameter-Info
 public String[][] getParameterInfo() {
 String[][] info = {
{ PARAM_Para1, "String", "Parameterbeschreibung für unseren ersten
Übergabeparameter" },
{ PARAM_Para2, "String", "Parameterbeschreibung Nummer 2" },
{ PARAM_Para3, "String", "Hier kommt ein dritter Übergabeparameter" }, };
 return info;
 }
 // die Initialisierung
 public void init() {
// lokale Testvariable zum Auswerten der
// Übergabeparameter
 String param;
// Test des 1. Übergabeparameters
// Wenn der Übergabeparameter vorhanden ist,
// Zuweisung des 1. Übergabeparametes in die
// entsprechende Variable
 param = getParameter(PARAM_Para1);
 if (param != null) m_Para1 = param;
// Test des 2. Übergabeparameters
// Wenn der Übergabeparameter vorhanden ist,
// Zuweisung des 2. Übergabeparametes in die
// entsprechende Variable
 param = getParameter(PARAM_Para2);
 if (param != null) m_Para2 = param;
// Test der 3. Übergabeparameters - wie oben
 param = getParameter(PARAM_Para3);
 if (param != null) m_Para3 = param;
// Laden eines Bildes
 samImage = getImage(getDocumentBase(), "images/ba2.jpg");
// Initialisierung der Graphic-Instanz
 ausgabe = getGraphics();
 }
 public void paint(Graphics g) {
/* Automatische Ausgabe des ersten Übergabeparameters */
 g.drawString(m_Para1, 10, 20);
/* Zeige das geladene Bild an */
```

```
 g.drawImage(samImage, 20, 50, this);
 }
public boolean mouseDown(Event evt, int x, int y) {
// Wir verwenden hier einen der Übergabeparameter
// an das Applet.
// Hier wird der Übergabeparameter 3 rechts oben
// im Appletbereich angezeigt, wenn der Mauszeiger
// gedrückt wird.
 ausgabe.drawString(m_Para3, 300, 10);
 return true;
}
 public boolean mouseUp(Event evt, int x, int y) {
// Die Parameterinfo wird eingeholt und damit ein
// 2-dimensionales Zeichen-Array gefüllt.
 String [][] parainfo = getParameterInfo();
// Wenn der Mauszeiger gedrückt und dann wieder
// losgelassen wird, wird die Appletinfo
// angezeigt.
 ausgabe.drawString(getAppletInfo(), 5, 180);
// Zusätzlich wird ein Teil der Parameterinfo
// angezeigt. Es erfolgt der Zugriff auf das Feld
// 3,3
 ausgabe.drawString(parainfo[2][2], 5, 200);
 return true;
 }
public boolean mouseDrag(Event evt, int x, int y) {
// Zeige das geladene Bild an anderer Stelle
// an, wenn die Maus gezogen wird.
 ausgabe.drawImage(samImage, 300, 20, this);
 return true;
 }
public boolean mouseMove(Event evt, int x, int y) {
 return true;
 }
public boolean mouseEnter(Event evt, int x, int y) {
// Hier werden alle in den anderen Mausaktionen
// ausgegebener Zeichenketten wieder gelöscht,
// wenn der Mauszeiger wieder in den Appletbereich
// kommt.
 ausgabe.clearRect(0, 0, 500, 500);
 return true;
 }
public boolean mouseExit(Event evt, int x, int y) {
// Wir verwenden hier einen der Übergabeparameter
// an das Applet.
// Hier wird der Übergabeparameter 2 links unten
// im Appletbereich angezeigt, wenn der Mauszeiger
// den Appletbereich verlässt.
 ausgabe.drawString(m_Para2, 20, 230);
 return true;
 }
}
```

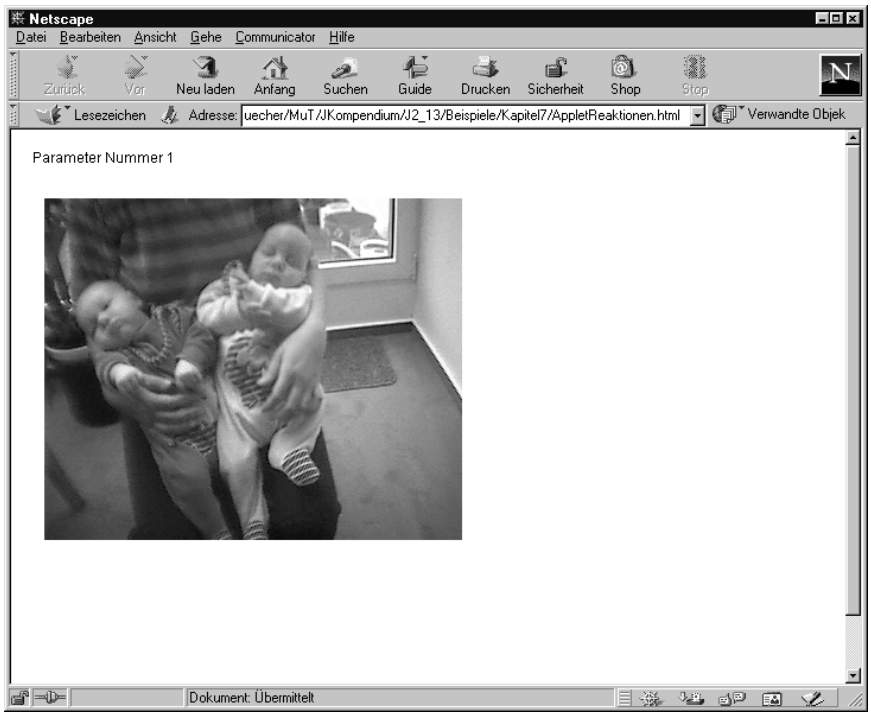


Abbildung 7.13: Das Applet nach dem Start

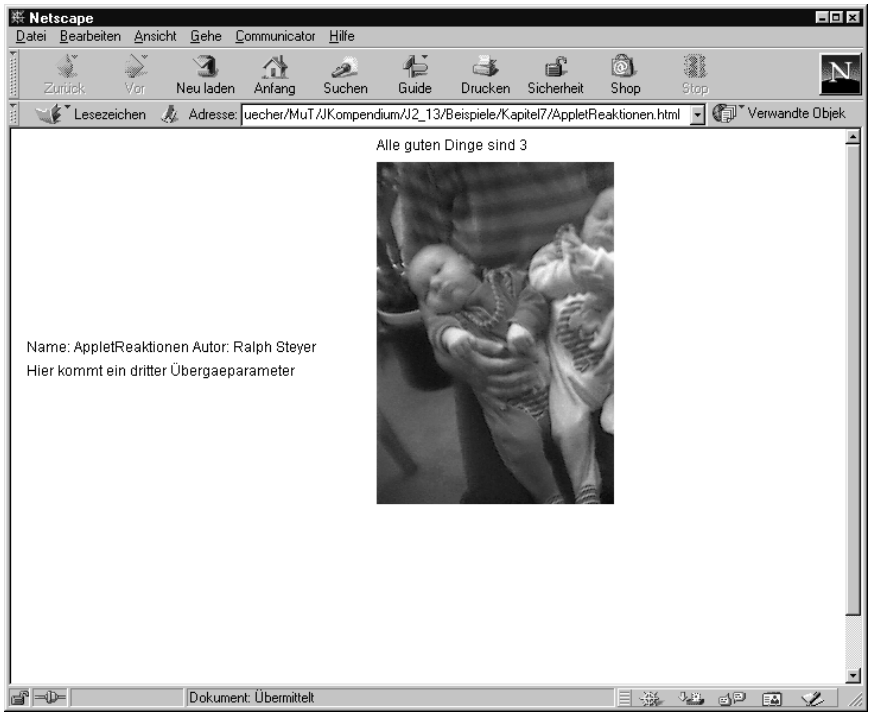


Abbildung 7.14: Das Applet in voller Pracht. Einige der Mausaktionen wurden ausgelöst.

Die HTML-Datei zum Einbinden sieht so aus:

Listing 7.21: Die HTML-Datei mit den Übergabe-Parametern

```
<HTML>
<BODY>
<APPLET CODE = "AppletReaktionen.class"
  WIDTH = 500 HEIGHT = 500>
<param name=Para1 value="Parameter Nummer 1">
<param name=Para2 value="Hier kommt der zweite">
<param name=Para3 value="Alle guten Dinge sind 3">
</APPLET>
</BODY>
</HTML>
```

Tastaturereignisse

Mit Sicherheit zählen auch Tastaturereignisse zu den Grundereignissen der Interaktion zwischen einem Applet (und erst recht einer eigenständigen Applikation) und seiner Umwelt. Jedesmal, wenn ein Anwender eine Taste auf der Tastatur drückt, kann diese ein Ereignis auslösen, das das Applet auswerten kann. Dabei unterscheidet man zwischen verschiedenen Typen von Tastaturereignissen.

- Irgendeine Taste wird gedrückt. Die genaue Taste spielt keine Rolle.
- Eine bestimmte Taste wird gedrückt und ausgewertet.
- Eine so genannte Ergänzungstaste wird gedrückt.

Dennoch haben Tastaturereignisse sicher nicht die Bedeutung wie Mausereignisse, denn das WWW arbeitet ja möglichst selten mit der Tastatur.

Die Methode

```
public boolean keyDown(Event event, int key)
```

wird immer dann aufgerufen, wenn eine Taste gedrückt wird.

Die durch die Methode `keyDown()` erfassten Tastenanschläge sind Ganzzahlen und werden über den Parameter `key` an die Methode übergeben. Die Ganzzahlen stellen den ASCII-Wert des gedrückten Zeichens dar. Es können sämtliche Tastenanschläge ausgewertet werden, die einen ASCII-Code besitzen.

Um die Ganzzahlen aus dem `key`-Parameter als Zeichen verwenden zu können, müssen sie per Casting in Zeichen umgewandelt werden. Die Syntax dafür ist beispielsweise

```
einZeichen = (char)key;
```

Wenn Sie den Parameter `key` nicht auswerten wollen, heißt dies, dass eine beliebige Taste gedrückt werden kann und alleine darauf hin eine Aktion erfolgt.

Wir wollen uns wieder ein einfaches Beispiel zu Gemüte führen, das über die `keyDown()`-Methode Tastenanschläge auswertet und im Anzeigebereich des Applets ausgibt.

Listing 7.22: Das Applet wertet Tastatureingaben aus

```
import java.applet.*;
import java.awt.Event;
import java.awt.*;
public class AppletTaste extends Applet {
  static char taste;
  public void paint(Graphics g) {
    g.drawString(String.valueOf(taste), 50, 30);
   }
  public boolean keyDown(Event evt, int key) {
    taste=(char)key;
    repaint();
    return true;
 } }
```

Beachten Sie im Beispiel die Methode `repaint()`. Damit wird nach jeder Tastatureingabe die `paint()`-Methode aufgerufen. Mehr dazu bei der Behandlung der Grafikmöglichkeiten von Java.

Aufgerufen wird das Applet mit

Listing 7.23: Die HTML-Datei zum Einbinden

```
<HTML><BODY>
<APPLET CODE = "AppletTaste.class"
  WIDTH = 500 HEIGHT = 500></APPLET>
</BODY></HTML>
```

Das Beispiel wird unter Umständen im Appletviewer des JDK 1.2 nicht laufen. Es gibt zwar keinerlei Fehlermeldung, aber die Anzeige der gedrückten Tasten unterbleibt seltsamerweise. Im Navigator oder Explorer gibt es keine Schwierigkeiten. Auch nicht in anderen Versionen des Appletviewers (auch die des JDK 1.3 ist okay).

Standardtasten der Event-Klasse

Die `Event`-Klasse stellt einige Klassenvariablen zur Verfügung, die einige nicht alphanumerische Standardtasten repräsentieren. Dies sind z.B. die Pfeiltasten. Diese Klassenvariablen können Sie anstelle der numerischen ASCII-Werte in der `keyDown()`-Methode verwenden. Damit wird eine bessere Übersichtlichkeit des Sourcecodes gewährleistet.

Folgende Standardvariablen stehen zur Verfügung:

Tabelle 7.4: Standardtasten der Event-Klasse

Taste	Klassenvariable
Funktionstasten F1 bis F12	`Event.F1 - Event.F12`
Pos1	`Event.home`
Ende	`Event.end`
Bild nach oben	`Event.pgup`
Bild nach unten	`Event.pgdn`
Pfeil nach oben	`Event.up`
Pfeil nach unten	`Event.down`
Pfeil nach links	`Event.left`
Pfeil nach rechts	`Event.right`

Die Ergänzungstasten

Es gibt drei wichtige Tasten, welchen kein ASCII-Code direkt zugeordnet ist, die aber für sich so wichtig sind, dass sie dennoch ausgewertet werden müssen. Es handelt sich um:

- die `Shift`-Taste
- die `STRG` oder `Control`-Taste
- die `Meta`-Taste

Das besondere an diesen drei Tasten ist, dass sie keine Tastenereignisse auslösen, die mit der `keyDown()`-Methode ausgewertet werden können.

In einigen Fällen ist es klar, dass die `Shift`-Taste gedrückt wurde, nämlich immer dann, wenn ein Großbuchstabe oder ein sonstiges oberes Zeichen der Tastatur als ASCII-Wert interpretiert werden kann. Dies ist jedoch eine implizite Vorgehensweise.

Es gibt indes noch eine andere Strategie. Wenn ein gewöhnliches Tastatur- oder Mausereignis eintritt, kann über zusätzliche Methoden getestet werden, ob eine der drei Tasten zusätzlich gedrückt wurde.

Die `Event`-Klasse enthält drei Methoden, die testen, ob zusätzlich zu den gewöhnlichen Tastatur- oder Mausereignissen eine der drei Zusatztasten gedrückt wurden. Es handelt sich um die `shiftDown()`-, die `metaDown()`- und die `controlDown()`-Methode.

Alle drei Methoden geben boolesche Werte zurück. Sie können diese Methoden in jeder beliebigen Ereignishandlung (Tastatur- oder Mausereignis) verwenden, indem Sie sie zusätzlich zu dem Ereignis-Objekt, das an die jeweilige Methode weitergegeben wurde, aufrufen.

Ein kurzes Beispiel für jede Ergänzungsmethode soll die Technik verdeutlichen:

Listing 7.24: Skizzierter Einsatz der Ergänzungstasten

```
public boolean keyDown(Event event, int key) {
if (event.shiftDown)
... irgendwelche Anweisungen ...
else
... irgendwelche Anweisungen ...
}
public boolean mouseDrag(Event event, int x, int y) {
if (event.metaDown)
... irgendwelche Anweisungen ...
else
... irgendwelche Anweisungen ...
}
public boolean mouseDown(Event evt, int x, int y) {
if (event.contolDown)
... irgendwelche Anweisungen ...
else
... irgendwelche Anweisungen ...
}
```

Die Standardmethoden zur Handhabung von Ereignissen in Applets gehören eigentlich zum AWT-Event-Handler. Mehr dazu erörtern wir im Kapitel über das AWT und das prinzipielle Eventhandling von Java.

7.6 Multithreading bei Applets

Kommen wir nun zum ersten Mal zu dem Begriff Threads innerhalb von Applets. Das Thema Multithreading in Java soll ebenfalls in einem eigenen Kapitel behandelt werden, aber an dieser Stelle soll zumindest noch die Bedeutung der `run()`-Methode erklärt werden. Java unterstützt Multithreading. Die genauere Erklärung von Multithreading finden Sie in Kapitel 8. Multithreading bedeutet für uns erst einmal nur, dass mehrere Aufgaben oder Prozesse quasi gleichzeitig ausgeführt werden können.

Durch die Verwendung von Threads können Sie Applets so erstellen, dass diverse Prozesse in ihrem eigenen Thread laufen, ohne andere Teile des Systems zu beeinflussen. Zwar werden eventuelle Engpässe in der Kapazität des Rechners die diversen Threads unter Umständen verlangsamen, aber sie sind und bleiben unabhängig voneinander.

Wie wird nun ein Applet Multithreading-fähig? Sie müssen im Wesentlichen vier Schritte durchführen.

1. Erweitern Sie die Unterschrift des Applets um `implements Runnable` (Sie integrieren also diese Schnittstelle).
2. Fügen Sie eine Instanzvariable ein, die den Thread des Applets enthält.
3. Reduzieren Sie die `start()`-Methode so, dass sie außer dem Start des Threads (bzw. ggf. dem Erzeugen eines Thread-Objekts) keinen weiteren Aufruf enthält.
4. Jetzt kommt eine neue Methode ins Spiel – die `run()`-Methode, die ab sofort den eigentlichen Code enthält, den das Applet ausführen soll. Sie ersetzt damit die `start()`-Methode weitgehend.

Schauen wir uns die Schritte anhand eines Beispiels an.

Listing 7.25: Ein Multithreading-Applet

```
/* Datumsausgabe mit Threads*/
import java.awt.Graphics;
import java.awt.Font;
import java.util.Date;
public class DatumThread extends java.applet.Applet implements Runnable {
  Font schriftArt = new Font("TimesRoman",Font.BOLD,24);
  Date Datum1, Datum2;
  Thread MeinThread1, MeinThread2;
  public void start() {
    if (MeinThread1 == null); {
      MeinThread1 = new Thread(this);
      MeinThread1.start();
    }
    if (MeinThread2 == null); {
      MeinThread2 = new Thread(this);
      MeinThread2.start();
    }
  }
  public void stop() {
    if (MeinThread1 != null) {
      MeinThread1.stop();
      MeinThread1 = null;
    }
    if (MeinThread2 != null) {
      MeinThread2.stop();
      MeinThread2 = null;
    }
  }
  public void run() {
    while (true) {
      try {
      Datum1 = new Date();
      MeinThread1.sleep(2017);
```

```
      repaint();
      }
      catch (InterruptedException e) { }
      try {
      Datum2 = new Date();
      MeinThread2.sleep(5029);
      repaint();
      }
      catch (InterruptedException e) { }
    }
  }
  public void paint(Graphics g) {
    g.setFont(schriftArt);
    g.drawString(Datum1.toString(),15,30);
    g.drawString(Datum2.toString(),15,80);
  }
}
```

Wenn Sie nun das Applet mit einer einfachen HTML-Seite einbinden, können wir sehen, was es tut.

```
<HTML><BODY>
<APPLET CODE = "DatumThread.class"
  WIDTH = 500 HEIGHT = 500></APPLET>
</BODY></HTML>
```

Listing 7.26: Die HTML-Seite für das Multithreading-Applet

Das Applet gibt zweimal Datum und Uhrzeit aus. Die Threading-Eigenschaft äußert sich darin, dass die beiden Zeiten nicht identisch sind und asynchron in verschieden langen Intervallen aktualisiert werden. Das liegt daran, dass nach Erzeugung eines jeweiligen Datumsobjekts der gerade aktive Thread »schlafen geht« und deshalb bis zur Erzeugung des zweiten Datumsobjekts ein gewisses Intervall vergeht. Zudem ist der Aufruf der `repaint()`-Methode entsprechend damit gekoppelt, weshalb die Anzeige immer nur eine der beiden Datumsausgaben aktualisieren kann.

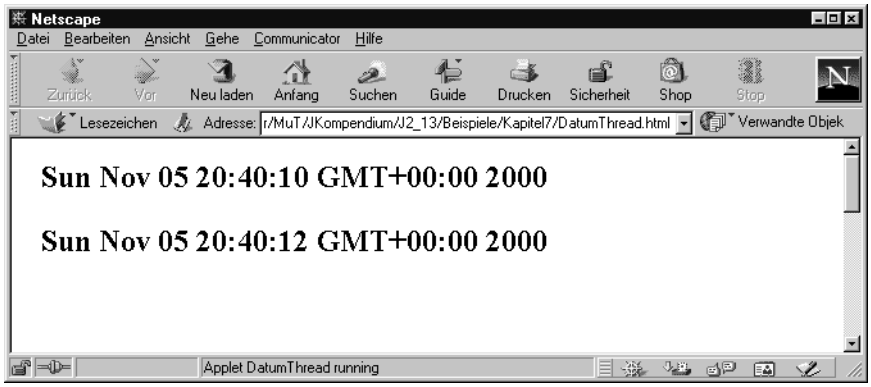


Abbildung 7.15: Wer außer Buchautoren arbeitet schon sonntagabends so spät?

Analysieren wir das Applet ein wenig:

```
public class DatumThread extends java.applet.Applet implements Runnable
```

ist unser Schritt 1 – Implementieren der Schnittstelle `Runnable`.

```
Thread MeinThread1, meinThread2;
```

ist der Schritt 2, das Einfügen der Instanzvariablen, die die Threads des Applets enthalten.

Schritt 3 ist die `start()`-Methode, die eine Abprüfung auf den Thread enthält und nur wenn kein Thread vorhanden ist, einen Thread startet. Sonst tut die `start()`-Methode nichts.

Die eigentliche Aktivität des Applets muss – wie wir in Schritt 4 festgelegt haben – innerhalb der `run()`-Methode stattfinden. Genau das passiert hier: Bis zum Auslösen der `stop()`-Methode wird eine Endlosschleife ausgeführt, die mittels der `repaint()`-Methode (zu dieser gleich mehr) immer wieder eine Ausgabe der aktualisierten Zeit auf dem Bildschirm erzeugt. Zu einer Behandlung von potenziellen Ausnahmen werden einige Anweisungen in einem `try-catch`-Block platziert.

7.7 Zusammenfassung

Applets benötigen immer einen Container, in dem sie zum Leben erweckt werden. In diesen werden sie als eine Referenz in einer HTML-Seiten geladen. Ein Java-Applet in der einfachsten Form (ohne irgendwelche Parameter und optionale Attribute) kann mit folgender HTML-Syntax in eine HTML-Seite eingebunden werden:

```
<APPLET CODE="[classelement]" WIDTH=[Wert] HEIGHT=[Wert]></APPLET>
```

Daneben gibt es noch diverse optionale Angaben (etwa Positionsangaben, Quelltextverweise, Parameterübergabe).

Sie können ein Applet auch mit anderen HTML-Tags in eine Webseite einbinden, die aber komplizierter zu handhaben sind und durch ihre Einschränkungen auf spezielle Voraussetzungen das `<APPLET>`-Tag noch nicht abgelöst haben.

Um ein Applet zu erstellen, müssen Sie immer eine Subklasse der Klasse `Applet` erzeugen. Diese Klasse beinhaltet sämtliche Eigenschaften, die die Kooperation mit dem Container sicherstellen, die Fähigkeiten des AWT für Oberflächenprogrammierung nutzbar machen und auch die lebenswichtigen Grundmethoden für ein Applet.

Der Einstieg in ein Applet-Leben ist die Initialisierung. Diese erfolgt immer mit der Methode `init()`, die automatisch aufgerufen wird, sobald das Applet in den Browser (oder einen anderen Container wie den Appletviewer) geladen wird. Die Methode `start()` startet das eigentliche Applet direkt nach der Initialisierung. Während die Initialisierung nur einmal erfolgt, kann (und wird normalerweise) der Start beliebig oft wiederholt werden. Die Methode `stop()` stoppt ein Applet, das dann mit `start()` wieder zu neuem Leben erweckt werden kann. Engültig zerstört wird ein Applet mit der Methode `destroy()`. Neben diesen Grundmethoden gibt es noch eine Vielzahl von Methoden, um die Funktion eines Applets sinnvoll zu erweitern.