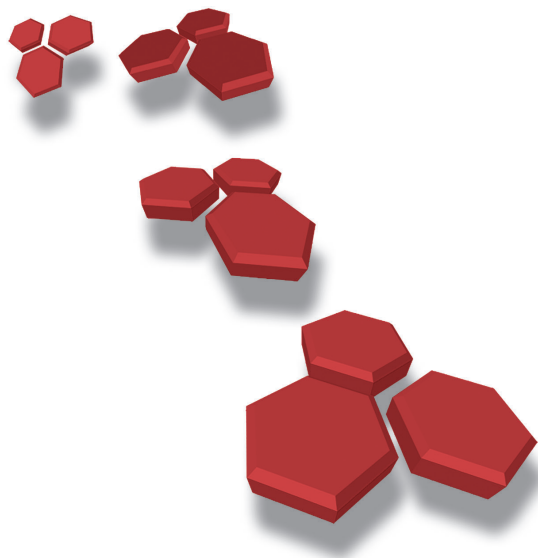


Thomas Stark, Karsten Samaschke

Das J2EE-Codebook



 ADDISON-WESLEY

An imprint of Pearson Education

München • Boston • San Francisco • Harlow, England
Don Mills, Ontario • Sydney • Mexico City
Madrid • Amsterdam

Netzwerk, Streams und Co.

Der Zugriff auf das Netzwerk und die Arbeit mit Streams erlauben es Applikationen, mit anderen Applikationen zu kommunizieren, Informationen zu versenden und abzulegen. Die wenigsten Applikationen kommen heutzutage ohne Datei-Handling oder Datenbankabfragen aus. Und noch seltener werden Sie auf Applikationen treffen, die auf den Einsatz von Streams gänzlich verzichten können. Ebenso wichtig wie das Ablegen und Laden von Daten ist der Zugriff auf das Netzwerk. Dabei können sowohl die Inhalte von Ressourcen abgerufen als auch Informationen an den User gesendet werden.

155 Wie können Dateiinformationen ausgelesen werden?

Für den Umgang mit Dateien stellt Java die Klasse `java.io.File` zur Verfügung. Mit deren Hilfe kann plattformunabhängig auf das tatsächliche Dateisystem zugegriffen werden, ohne dass den Entwickler dessen tatsächliche Beschaffenheit interessieren müsste.

Die Klasse `File` stellt viele Methoden zur Verfügung, mit deren Hilfe Dateiinformationen eingelesen und manipuliert werden können. Sie erlaubt jedoch keinen direkten Zugriff auf die Dateiinhalte – dafür sind die verschiedenen Stream-Implementierungen gedacht.

Im Folgenden soll gezeigt werden, wie mit Hilfe einer `File`-Instanz verschiedene Informationen über die referenzierte Datei erfasst werden können:

```
package codebook.j2ee.io;

import javax.servlet.http.*;
import javax.servlet.ServletException;
import java.io.*;

public class FileInfo extends HttpServlet {

    protected void service(
        HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {

        PrintWriter out = response.getWriter();
        out.print("<html><head><title>File-Info</title><head><body>");
        out.print("<h3>File-Info</h3>");
        out.print("<ul>");
```

Listing 382: Auslesen von Dateiinformationen

```
// File-Instanz erzeugen und auf Datei zeigen lassen
File file = new File("c:/temp/test.txt");

// Datei existiert?
out.print(String.format("<li>File existst: %s</li>",
    file.exists()));

if(file.exists()) {
    // Dateigrösse
    out.print(String.format("<li>File length: %s</li>",
        file.length()));

    // Lesezugriff erlaubt?
    out.print(String.format("<li>Can read: %s</li>",
        file.canRead()));

    // Schreibzugriff
    out.print(String.format("<li>Can write: %s</li>",
        file.canWrite()));

    // Absoluter Pfad
    out.print(String.format("<li>Absolute path: %s</li>",
        file.getAbsolutePath()));

    // Versteckte Datei?
    out.print(String.format("<li>Is hidden: %s</li>",
        file.isHidden()));

    // Letzte Änderung
    out.print(String.format("<li>Last modified: %s</li>",
        file.lastModified()));
}

out.print("</ul></html></body>");
}
```

Listing 382: Auslesen von Dateiinformatioren (Forts.)

Die hier verwendeten Eigenschaften der File-Instanz geben Auskunft über folgende Informationen:

Eigenschaft	Bedeutung
exists()	Gibt an, ob die angegebene Datei tatsächlich existiert.
length()	Datei-Größe in Bytes.

Tabelle 60: Dateieigenschaften einer File-Instanz

Eigenschaft	Bedeutung
canRead()	Gibt an, ob aus der Datei gelesen werden kann.
canWrite()	Gibt an, ob in die Datei geschrieben werden kann.
getAbsolutePath()	Gibt den absoluten Pfad der referenzierten Datei zurück.
isHidden()	Gibt an, ob die referenzierte Datei versteckt ist.
lastModified()	Gibt den Zeitpunkt der letzten Änderung an der Datei als Unix-Zeitstempel (Sekunden seit dem 01.01.1970) zurück.

Tabelle 60: Dateieigenschaften einer File-Instanz (Forts.)

Im Browser ausgeführt, kann dies für die Datei `c:\temp\test.txt` so aussehen:

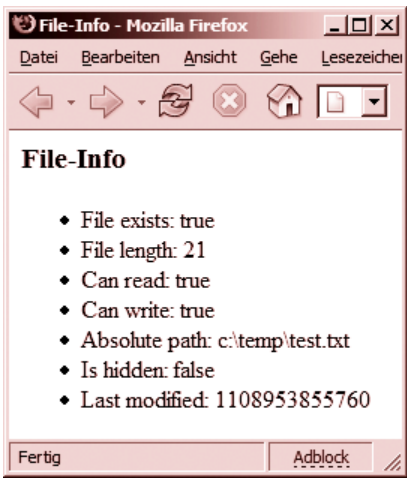


Abbildung 112: Anzeige der Informationen einer Datei

Hinweis

Es ist sinnvoll, bei Pfadangaben auf Windows-Systemen stets vorwärts gerichtete Schrägstriche (/) statt der sonst üblichen Back-Slashes einzusetzen, denn einerseits erleichtert dies die Lesbarkeit durch Nicht-Windows-Programmierer, und andererseits (und das ist viel wichtiger) muss die Zeichenkette nicht escaped werden, denn Back-Slashes (\) dienen bei Java der Kennzeichnung von nicht druckbaren Sonderzeichen und speziellen Zeichen. Bei einer Pfadangabe müssten sie verdoppelt werden – aus `c:\temp\test.txt` würde `c:\\temp\\test.txt` werden müssen, anderenfalls gäbe es eine Exception. Schreiben Sie stattdessen also lieber `c:/temp/test.txt`.

156 Den absoluten Pfad einer Datei innerhalb der aktuellen Applikation ermitteln

Innerhalb einer Webapplikation möchte man in den seltensten Fällen absolute und fest verdrahtete Pfade verwenden, da dies die Wiederverwendbarkeit und Nutzbarkeit der Applikation stark einschränken kann.

In der Regel werden Ressourcen (von der Applikation referenzierte Dateien) innerhalb von deren Verzeichnis abgelegt – entweder unterhalb der Wurzel der Applikation oder innerhalb des */WEB-INF*-Verzeichnisses. Leider ist es nötig, für den Zugriff auf eine Datei deren absoluten Pfad zu kennen, da die Datei sonst innerhalb des Servlet-Containers und nicht innerhalb der aktuellen Applikation gesucht werden würde.

Um dieses Problem zu lösen, gibt es die Methode `getRealPath()` der aktuellen `ServletContext`-Instanz. Dieser `ServletContext` kann aus Servlets und JSPs mit Hilfe der Methode `getServletContext()` referenziert werden. Die Methode `getRealPath()` nimmt als Parameter den relativen Pfad zu einer Datei oder einem Verzeichnis entgegen und gibt deren absolutes Pendant zurück.

Der absolute Dateiname der Datei *test.txt*, die sich im Verzeichnis */resources* innerhalb der Applikation befindet, kann so ermittelt werden:

```
ServletContext ctx = getServletContext();
String path = ctx.getRealPath("/resources/test.txt");
```

Je nach Konfiguration Ihres Systems kann die Rückgabe von `getRealPath()` dann so aussehen:

```
D:\jboss-4.0.0\server\default\tmp\deploy\tmp4111001_File-
exp.war\resources\test.txt
```

Das ist ein Dateiname, den niemand fest verdrahten möchte; zumal er sich beim nächsten Deployment-Prozess oder beim nächsten Server-Neustart durchaus wieder ändern kann.

157 Dateien anlegen, löschen und umbenennen

Es gibt zwei Möglichkeiten, Dateien anzulegen: entweder explizit mit Hilfe der Methode `createNewFile()` oder implizit durch Erzeugen einer neuen `FileOutputStream`-Instanz. Mit Hilfe der Methode `delete()` kann die so erzeugte Datei wieder gelöscht werden. Und die Methode `renameTo()` erlaubt das Umbenennen einer vorhandenen Datei.

Um eine neue Datei anzulegen, erzeugen Sie eine neue `File`-Instanz und übergeben dieser im Konstruktor den absoluten Namen der anzugebenden Datei. Anschließend rufen Sie die seit Java 1.2 vorhandene Methode `createNewFile()` auf, die einen booleschen Wert zurückgibt, der `true` ist, wenn die Datei noch nicht existierte und angelegt werden konnte:

```
String fileName = "c:/temp/test.txt";
```

```
// Neue File-Instanz erzeugen
File file = new File(fileName);

// Leere Datei anlegen
if(file.createNewFile()) {
    // Datei angelegt, weitere Verarbeitung vornehmen
    // ...
} else {
    // Datei existiert schon und ist auch nicht gelöscht worden!
    throw new Exception(
        String.format("Unable to create new file %s", fileName));
}
```

Alternativ können Sie die Datei implizit mit Hilfe einer `FileOutputStream`-Instanz erzeugen. Dies funktioniert mit jeder Java-Version:

```
String fileName = "c:/temp/test.txt";

// File-Instanz erzeugen
File file = new File(file);

// FileOutputStream erzeugen
FileOutputStream fos = new FileOutputStream(file);

// In Datei schreiben
// ...

// FileOutputStream-Instanz schliessen, WICHTIG!!!
fos.close();
```

Achtung

Schließen Sie die `FileOutputStream`-Instanz am Ende immer explizit! Sollten Sie dies nicht machen, können Sie zwar problemlos in die Datei schreiben, jedoch wird diese am Ende nicht physisch erzeugt. Ein abschließendes `close()` sorgt dafür, dass die Inhalte des Streams gespeichert werden.

Um eine Datei zu löschen, erzeugen Sie eine neue `File`-Instanz und rufen anschließend deren Methode `delete()` auf. Diese gibt einen booleschen Wert zurück, der `true` ist, wenn die Datei gelöscht werden konnte:

```
String fileName = "c:/temp/test.txt";

// File-Instanz erzeugen
File file = new File(file);
// Datei löschen
if(file.delete()) {
    // Datei ist gelöscht
    // ...
} else {
    // Datei ist nicht gelöscht worden
```

```
    throw new Exception(String.format("Unable to delete File %s", fileName));
}
```

Das Umbenennen einer Datei geschieht mit Hilfe der Methode `renameTo()` einer `File`-Instanz. Diese nimmt als Parameter eine `File`-Instanz entgegen, die mit dem neuen Namen erzeugt worden ist. Wenn das Umbenennen funktioniert hat, wird `renameTo()` den booleschen Wert `true` zurückgegeben, anderenfalls ist die Rückgabe `false`:

```
String fileName = "c:/temp/test.txt";
String newFileName = "c:/temp/example.txt";

// File-Instanz erzeugen
File file = new File(fileName);

// Neue File-Instanz für das Umbenennen
File newFile = new File(newFileName);

// Umbenennen
if(file.renameTo(newFile)) {
    // Datei ist umbenannt worden
    // ...
} else {
    // Datei konnte nicht umbenannt werden
    throw new Exception(String.format("Unable to rename file %s to %s",
        fileName, newFileName));
}
```

158 Verzeichnisse anlegen

Das Anlegen von Verzeichnissen – die ebenso wie Dateien durch eine `File`-Instanz repräsentiert werden – geschieht mit Hilfe der Methoden `mkdir()` und `mkdirs()`. Der Unterschied zwischen beiden Varianten besteht darin, dass `mkdir()` voraussetzt, dass alle übergeordneten Verzeichnisse bereits existieren, während `mkdirs()` diese bei Bedarf mit anlegt.

Um ein neues Verzeichnis anzulegen, verwenden Sie die Methode `mkdir()`:

```
String dirName = "c:/temp/test";

// File-Instanz, die das Verzeichnis repräsentiert, erzeugen
File directory = new File(dirName);

// Verzeichnis anlegen
if(directory.mkdir()) {
    // Verzeichnis konnte erstellt werden
    // ...
} else {
    // Verzeichnis konnte nicht erstellt werden, etwa weil ein übergeordnetes
    // Verzeichnis nicht existierte
}
```

```

    throw new Exception(String.format(
        "Unable to create directory %s!", dirName));
}

```

Wenn Sie nicht sicher sind, ob die übergeordneten Verzeichnisse tatsächlich existieren, oder Sie eine tiefe Verzeichnisstruktur auf einen Rutsch erzeugen wollen, verwenden Sie die Methode `makedirs()`:

```

String dirName = "c:/temp/test/of/the/makedirs/method/in/java";

// File-Instanz, die das Verzeichnis repräsentiert, erzeugen
File directory = new File(dirName);

// Verzeichnisse anlegen
if(directory.mkdirs()) {
    // Verzeichnisse konnten erstellt werden
    // ...
} else {
    // Verzeichnisse konnten nicht erstellt werden
    throw new Exception(String.format(
        "Unable to create directories in path %s!", dirName));
}

```

159 Verzeichnisse auflisten

Das Auflisten von Dateien eines Verzeichnisses geschieht mit Hilfe der verschiedenen `list()`- und `listFiles()`-Methoden. Um die Auflistung etwas einzuschränken, können Sie eine `FileFilter`-Implementierung einsetzen, welche die Dateinamen nach Ihren Kriterien filtert:

```

package codebook.j2ee.io;

import java.io.FileFilter;
import java.io.File;

public class CustomFileFilter implements FileFilter {

    public boolean accept(File file) {
        // Dateinamen extrahieren
        String fn = file.getName().toLowerCase();

        // Ergebnis ist nur dann true, wenn der Datei-
        // name mit der Endung .txt endet und das
        // angegebene Objekt eine Datei ist
        return (file.isFile() && fn.endsWith(".txt"));
    }
}

```

Listing 383: FileFilter-Implementierung, die nur Textdateien zulässt

Diese `FileFilter`-Implementierung kann nun den Methoden `list()` und `listFiles()` als Parameter übergeben werden. Der Unterschied zwischen den Methoden liegt im Rückgabe-Datentyp: Die überladene `list()`-Methode gibt ein `String`-Array zurück, das aus den Dateinamen der Dateien besteht. Die überladene Methode `listFiles()` gibt dagegen ein Array aus `File`-Instanzen zurück.

Um alle Dateinamen eines Verzeichnisses auszugeben, können Sie diesen Code verwenden:

```
String directoryName = "c:\\temp";

// File-Instanz erzeugen, die ein Verzeichnis repräsentiert
File directory = new File(directoryName);

// Dateien ermitteln und ausgeben
String[] files = directory.list();
for(String file : files) {
    // Dateinamen ausgeben
    out.println(file);
}
```

Um nur Textdateien auszugeben, erzeugen Sie eine `CustomFileFilter`-Instanz und übergeben diese als Parameter an die `list()`- oder `listFiles()`-Methode:

```
String directoryName = "c:\\temp";

// File-Instanz erzeugen, die ein Verzeichnis repräsentiert
File directory = new File(directoryName);

// FileFilter-Instanz erzeugen
CustomFileFilter filter = new CustomFileFilter();

// Dateien ermitteln und ausgeben
File[] files = directory.listFiles(filter);
for(File file : files) {
    // Dateinamen ausgeben
    out.println(file.getName());
}
```

160 Eine Datei kopieren

Das Kopieren einer Datei erfolgt nicht über die `File`-Klasse, sondern mit Hilfe von `FileInputStream`- und `FileOutputStream`-Instanzen. Dazu wird ein `Byte`-Array als Puffer verwendet, in das Daten aus dem Quell-Stream geschrieben werden. Der Puffer wird direkt im Anschluss an das Lesen in den Ziel-Stream entleert. Dieser Prozess läuft so lange ab, bis keine Daten mehr aus der Quelldatei gelesen werden können. Zuletzt werden die verwendeten Streams geschlossen und somit die Zieldatei tatsächlich erzeugt:

```
String sourceFile = "c:/temp/test.txt";
String targetFile = "c:/temp/copy_of_test.txt";

FileInputStream in = null;
FileOutputStream out = null;

try {
    // File-Instanzen erzeugen, welche die Dateien repräsentieren
    File source = new File(sourceFile);
    File target = new File(targetFile);

    // Input- und OutputStreams erzeugen, welche die Daten lesen und speichern
    in = new FileInputStream(source);
    out = new FileOutputStream(target);

    // Puffer definieren: 8192 Bytes sollten ausreichen
    int bufferSize = 8192;
    byte[] buffer = new byte[bufferSize];

    // Daten einlesen, solange mehr als 0 Bytes gelesen werden können
    int bytes = 0;
    while((bytes = in.read(buffer)) > 0) {
        // Eingelesene Daten direkt wieder speichern
        out.write(buffer, 0, bytes);
    }
} catch (FileNotFoundException e) {
    e.printStackTrace();
} catch (IOException e) {
    e.printStackTrace();
} finally {
    // Streams schliessen (besonders bei OutputStream wichtig!)
    try {
        in.close();
        out.close();
    } catch (IOException e) { }
}
```

Listing 384: Kopieren einer Datei

161 Eine Datei in einen String einlesen

Das Einlesen einer Datei in einen String ist auf vielfältigste Art und Weise möglich. Die einfachste Variante ist sicherlich die Verwendung eines `FileReaders` und einer `StringBuffer`-Instanz. Dabei wird ein `Char-Array` zur Pufferung der geladenen Daten verwendet, und alle eingelesenen Daten werden im `StringBuffer` gehalten, der später weiterverwendet werden kann:

```

String fileName = "c:/temp/test.txt";

// File-Instanz erzeugen
File file = new File(fileName);

// FileReader zum Einlesen der Daten
FileReader reader = null;
try {
    reader = new FileReader(file);

    // StringBuffer zum Halten der Daten
    StringBuffer buffer = new StringBuffer();
    int bytes = 0;

    // Char-Array zum Zwischenpuffern der Daten
    char[] chars = new char[1024];

    // Daten einlesen
    while((bytes = reader.read(chars)) > 0) {
        buffer.append(chars);
    }

    // Daten ausgeben
    System.out.println(buffer.toString());
} catch (FileNotFoundException e) {
    e.printStackTrace();
} catch (IOException e) {
    e.printStackTrace();
} finally {
    try {
        reader.close();
    } catch (IOException e) {
        e.printStackTrace();
    }
}

```

Listing 385: Einlesen einer Textdatei

162 Binäre Daten lesen und speichern

Das Laden und Speichern von binären Daten geschieht mit Hilfe von `DataInputStream`- und `DataOutputStream`-Instanzen. Diese implementieren die Interfaces `DataInput` und `DataOutput`, die Methoden für das Laden und Speichern aller primitiven Datentypen und von Strings im UTF-8-Format definieren. Dabei legt das speichernde Programm das Format und die Reihenfolge der Daten selbst fest und muss auch selbst für das Auflösen der gespeicherten Daten sorgen.

Dieses Beispiel schreibt einige Daten per `DataOutputStream`:

```
String fileName = "c:/temp/data.txt";

// File-Instanz erzeugen
File file = new File(fileName);

try {
    // DataOutputStream zum Speichern der Daten
    DataOutputStream out = new DataOutputStream(
        new BufferedOutputStream(
            new FileOutputStream(file)));

    // booleschen Wert ausgeben
    out.writeBoolean(true);

    // String ausgeben
    out.writeUTF("Hello world!");

    // Integer-Wert ausgeben
    out.writeInt(2004);

    // String ausgeben
    out.writeUTF(
        "This is an UTF-8 encoded String " +
        "containing German Umlauts: äöüß");

    // Puffer leeren
    out.flush();

    // DataOutputStream schliessen
    out.close();
} catch (FileNotFoundException e) {
    e.printStackTrace();
} catch (IOException e) {
    e.printStackTrace();
}
```

Listing 386: Speichern von Daten per `DataOutputStream`

Wenn das Beispiel ausgeführt worden ist und die Datei *data.txt* in einem Editor geöffnet wird, ergibt sich das Bild aus Abbildung 113.

Das erneute Einlesen der Daten geschieht nun per `DataInputStream`-Instanz. Dabei werden die einzelnen Schritte des Speichervorgangs in genau der gleichen Reihenfolge wiederholt – nur wird hier gelesen statt geschrieben:



Listing 387: Einlesen von Daten per `DataInputStream`

163 Objekte serialisieren und deserialisieren

Für gewöhnlich existieren Objekte so lange im Speicher, bis sie vom Garbage Collector zerstört werden oder die JVM heruntergefahren wird. Wenn sie einmal zerstört sind, sind die von ihnen repräsentierten Informationen unwiderruflich verloren – es sei denn, die Informationen wurden zuvor gespeichert. Technisch ist es kein Problem, alle Informationen aus Objektinstanzen auszulesen und zu speichern. Praktisch ist es das auch nicht – nur ist es sehr ermüdend, dies für jede Art von Objekt einzeln zu implementieren.

Sinnvoller ist es, gleich komplette Objektinstanzen zu speichern, was mit Hilfe der Klasse `ObjectOutputStream` umgesetzt werden kann. Diese Klasse nimmt im Konstruktor eine `OutputStream`-Instanz entgegen, die das serialisierte Objekt speichert. Das eigentliche Speichern des Objekts geschieht mit Hilfe der Methode `writeObject()`:

```
// Hashtable zum Halten von Daten
Hashtable<String, String> data = new Hashtable<String, String>();
data.put("Karsten", "Samaschke");
data.put("Thomas", "Stark");
data.put("Christian", "Wenz");

String fileName = "c:/temp/object.bin";

// File-Instanz für den Zugriff auf Daten
File file = new File(fileName);

try {
    // FileOutputStream zum Speichern der Daten
    FileOutputStream fos = new FileOutputStream(file);

    // ObjectOutputStream zum Serialisieren der Daten
    ObjectOutputStream out = new ObjectOutputStream(fos);

    // Speichern des Objekts
    out.writeObject(data);

    // Schliessen der Streams
    out.close();
} catch (FileNotFoundException e) {
    e.printStackTrace();
} catch (IOException e) {
    e.printStackTrace();
}
```

Listing 388: Serialisieren eines Objekts

Das Einlesen eines serialisierten Objekts und dessen Deserialisierung gestaltet sich ebenso einfach: Mit Hilfe einer `FileInputStream`-Instanz wird eine `ObjectInput-`

Stream-Instanz erzeugt, deren Methode `readObject()` die deserialisierte Object-Instanz zurückgibt. Diese muss noch in den korrekten Typ gecastet und kann danach weiter verwendet werden:

```
String fileName = "c:/temp/object.bin";

// File-Instanz für den Zugriff auf Daten
File file = new File(fileName);

try {
    // FileInputStream zum Laden der Daten
    FileInputStream fin = new FileInputStream(file);

    // ObjectInputStream zum Deserialisieren
    ObjectInputStream in = new ObjectInputStream(fin);

    // Objekt deserialisieren
    Object object = in.readObject();

    // Stream schliessen
    in.close();

    if(null != object && object instanceof Hashtable) {
        // Object-Instanz in den richtigen Typ casten
        Hashtable<String, String> data =
            (Hashtable<String, String>)object;

        // Deserialisiertes Objekt weiter verarbeiten
        // ...
    }
} catch (FileNotFoundException e) {
    e.printStackTrace();
} catch (IOException e) {
    e.printStackTrace();
} catch (ClassNotFoundException e) {
    e.printStackTrace();
}
```

Listing 389: Deserialisieren eines Objekts

164 Ein ZIP-Archiv entpacken

ZIP-Archive sind gerade im Webumfeld weit verbreitet, um Daten austauschen zu können. Java selbst verwendet verschiedene Archive, die alle ebenfalls auf dem ZIP-Algorithmus beruhen. Was also lag näher, als in Java die Möglichkeit des Umgangs mit ZIP-Archiven zu integrieren?

Das Paket `java.util.zip` enthält einige Klassen, mit deren Hilfe ZIP-Archive gelesen und entpackt werden können. Repräsentiert wird ein derartiges Archiv durch die Klasse `ZipFile`, die auch den Zugriff auf alle enthaltenen Elemente erlaubt. Mit Hilfe der Enumeration `entries` können die einzelnen Bestandteile des Archivs durchlaufen werden.

Jedes einzelne dieser Elemente wird durch eine Instanz der Klasse `ZipEntry` repräsentiert. Dabei muss unterschieden werden, ob es sich bei der aktuellen Instanz um die Repräsentation eines Verzeichnisses oder einer Datei handelt. Wenn das Archiv entpackt werden soll, muss das Verzeichnis angelegt werden. Sollte es sich um eine Datei handeln, muss sie extrahiert werden.

Dieses Extrahieren findet per `FileOutputStream` und einer `InputStream`-Instanz statt. Die `FileOutputStream`-Instanz speichert dabei die aus dem `InputStream` ausgelesenen Daten. Eine `InputStream`-Instanz wird durch die `ZipFile`-Instanz, die das gesamte Archiv repräsentiert, erzeugt – dabei muss die derzeit behandelte `ZipEntry`-Instanz als Parameter übergeben werden.

Der Rest ist dann reine Routine: Die `InputStream`-Instanz wird so lange ausgelesen, wie sie Daten zurückliefert. Dabei wird stets ein Puffer gefüllt, dessen Inhalt dann in den `OutputStream` geschrieben wird:

```
String fileName = "c:/temp/test.zip";

// File-Instanzen zur Repräsentation von
// ZIP-Archiv und Pfad
File file = new File(fileName);
File path = new File(file.getParent());

try {
    // Neue ZipFile-Instanz erzeugen
    ZipFile zip = new ZipFile(file);

    // Elemente des ZIP-Archivs durchlaufen
    Enumeration entries = zip.entries();
    while(entries.hasMoreElements()) {
        // Aktuelles Element abrufen
        ZipEntry current = (ZipEntry)entries.nextElement();

        // File-Instanz zum Speichern des Eintrags oder
        // zur Repräsentation eines Unterverzeichnisses
        File currentFile = new File(path, current.getName());

        // ZIP-Element entpacken
        if(current.isDirectory()) {
            // Unterverzeichnisse anlegen
            currentFile.mkdirs();
        } else {
```

Listing 390: Entpacken eines ZIP-Archivs


```

// Inhalt entpacken und speichern
FileOutputStream fout =
    new FileOutputStream(currentFile);

// Datei per InputStream auslesen
InputStream zin = zip.getInputStream(current);
byte[] buffer = new byte[8192];
int bytes = 0;

// Auslesen der Daten in den Puffer
while((bytes = zin.read(buffer)) > 0) {
    // Schreiben der Daten in den OutputStream
    fout.write(buffer, 0, bytes);
}

// Streams schliessen
fout.close();
zin.close();
}
} catch (IOException e) {
    e.printStackTrace();
}
}

```

Listing 390: Entpacken eines ZIP-Archivs (Forts.)

165 Eine Datei zum Browser senden, obwohl kein direkter Zugriff möglich ist

Nicht immer ist ein direkter Zugriff auf Dateien möglich oder sinnvoll. Stattdessen befinden sich manche Dateien oftmals außerhalb des direkt per Browser erreichbaren Bereichs einer Webapplikation, was insbesondere Sinn macht, wenn der Zugriff auf Dateien kontrolliert werden soll.

Mit Hilfe der bereits vorgestellten Techniken ist es problemlos möglich, auch Dateien zum Download bereitzustellen, die sich nicht in einem per Browser erreichbaren Bereich befinden. Angenommen, Sie wollten das PDF-Dokument `c:\temp\download.pdf` zum Download bereitstellen, dann könnten Sie dies mit einem Servlet umsetzen, das den Inhalt der Datei per `FileInputStream` lädt und über die `Standard-ServletOutputStream`-Instanz ausgibt. Wichtig dabei ist, dass der korrekte Inhaltstyp und die korrekte `Content-Disposition` gesetzt sind. Ersteres teilt dem Browser mit, was ihn erwartet, und letztere Information sagt ihm, wie er die empfangenen Daten behandeln soll:

```
package codebook.j2ee.io;

import javax.servlet.ServletException;
import javax.servlet.ServletOutputStream;
import javax.servlet.http.*;
import java.io.*;

public class DownloadServlet extends HttpServlet {

    protected void service(
        HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {

        String sourceFile = "c:/temp/download.pdf";

        // OutputStream ist der OutputStream
        // der HttpServletResponse-Instanz
        ServletOutputStream out = response.getOutputStream();

        // Inhaltstyp setzen
        response.setContentType("application/pdf");

        // File-Instanz zur Repräsentation der Datei
        File file = new File(sourceFile);

        // FileInputStream zum Einlesen der Daten
        FileInputStream in = new FileInputStream(file);
        byte[] buffer = new byte[16384];
        int bytes = 0;

        // Daten einlesen und direkt in den OutputStream schreiben
        while((bytes = in.read(buffer)) > 0) {
            out.write(buffer, 0, bytes);
        }

        // InputStream schliessen
        in.close();

        // Content-Disposition setzen
        response.addHeader(
            "Content-Disposition",
            String.format("inline;filename=%s", file.getName()));

        // Ausgabepuffer leeren
        response.flushBuffer();
    }
}
```

Listing 391: Ausgabe einer Datei per Servlet

Wenn das Servlet aufgerufen wird, sollte der Nutzer eine Ausgabe ähnlich dieser erhalten:

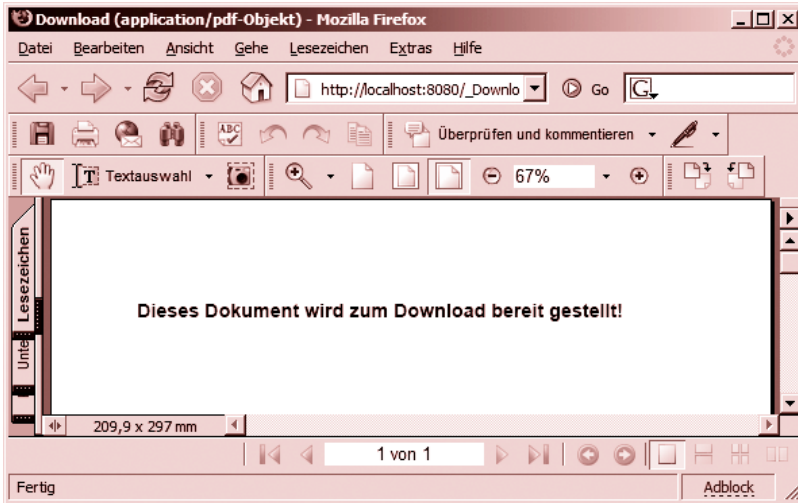


Abbildung 114: Das Dokument ist zum Browser gesendet worden, obwohl kein direkter Zugriff per Browser möglich ist

166 Auf Applikationsressourcen zugreifen

Der Zugriff auf Ressourcen einer Webapplikation kann direkt per `File`-Instanz und einem entsprechenden `FileInputStream` erfolgen. Viel eleganter ist jedoch, über den `ServletContext` einer Webapplikation zu gehen und die dort zur Verfügung gestellte Methode `getResourceAsStream()` zu nutzen.

Vorteil dieses Ansatzes: Der Name der Ressource kann relativ zur Wurzel der Applikation angegeben werden. Der Servlet-Container übernimmt das Erzeugen der benötigten Reader-Instanzen und gibt entweder eine initialisierte `StreamReader`-Instanz oder den Inhalt der angeforderten Ressource als String zurück.

Um auf den Inhalt einer Textdatei zuzugreifen, können Sie die Methode `getResourceAsStream()` verwenden und mit Hilfe eines `InputStreamReader`s einen `BufferedReader` erzeugen, der das Einlesen der Daten übernimmt. Dies kann zeilenweise geschehen, was das Handling deutlich erleichtert:

```
package codebook.j2ee.io;

import javax.servlet.ServletContext;
import javax.servlet.ServletException;
import javax.servlet.http.*;
import java.io.*;
```

Listing 392: Zugriff auf Ressourcen mit Hilfe der Methode `getResourceAsStream()`

```

public class ResourceAsString extends HttpServlet {

    protected void service(
        HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {

        String resourceName = "/resources/test.txt";

        // ServletContext referenzieren
        ServletContext ctx = getServletContext();

        // Ressourceninhalt per BufferedReader laden
        BufferedReader br = new BufferedReader(
            new InputStreamReader(
                ctx.getResourceAsStream(resourceName)));

        // StringBuffer zum Halten der geladenen Daten
        StringBuffer result = new StringBuffer();

        // Daten einlesen und direkt dem
        // StringBuffer für die weitere Verarbeitung
        // zuweisen
        String line = "";
        while((line = br.readLine()) != null) {
            result.append(line);
            result.append("\n");
        }

        // Stream schliessen
        br.close();

        PrintWriter out = response.getWriter();
        out.print("<html><head><title>Content</title><head>");
        out.print(
            String.format(
                "<body><h3>Content</h3><pre>%s</pre></html>",
                result.toString()));
    }
}

```

Listing 392: Zugriff auf Ressourcen mit Hilfe der Methode `getResourceAsStream()` (Forts.)

Analog zum Zugriff auf eine Textressource kann auch der Zugriff auf eine binäre Ressource erfolgen. Statt eines `BufferedReader`s samt `InputStreamReader` kommt hier eine `BufferedStreamReader`-Instanz zum Einsatz. Die geladenen Daten werden direkt in den Ausgabestrom geschrieben, um im Browser angezeigt zu werden:

```

package codebook.j2ee.io;

import javax.servlet.http.*;
import javax.servlet.*;
import java.io.*;

public class ResourceAsStream extends HttpServlet {

    protected void service(
        HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {

        String resourceName = "/resources/download.pdf";

        // ServletContext referenzieren
        ServletContext ctx = getServletContext();

        // Ressourceninhalt per BufferedReader laden
        BufferedInputStream bin =
            new BufferedInputStream(
                ctx.getResourceAsStream(resourceName));

        // OutputStream referenzieren
        ServletOutputStream out = response.getOutputStream();
        byte[] buffer = new byte[16382];
        int bytes = 0;

        // Inhaltstyp setzen
        response.setContentType("application/pdf");

        // Content-Disposition setzen
        response.addHeader(
            "Content-Disposition",
            "inline;filename=download.pdf");

        // Daten einlesen und direkt wieder ausgeben
        while((bytes = bin.read(buffer)) > 0) {
            out.write(buffer, 0, bytes);
        }

        // Stream schliessen
        bin.close();
    }
}

```

Listing 393: Binäre Ressource ausgeben

167 Die einzelnen Informationen zu einem URL auslesen

Die Klasse `java.net.URL` kapselt Zeiger auf Ressourcen. Diese Zeiger können auf Ressourcen im Internet verweisen (Webseiten, Bilder etc), können aber ebenso gut auf lokale Ressourcen (Dateien) zeigen.

Die URL-Klasse selbst ist komplett ohne Verbindung zu einer derart bezeichneten Ressource – sie repräsentiert sie nur. Sie bietet einige Methoden an, mit deren Hilfe die Ressource und ihre Eigenschaften genauer bestimmt werden kann. Außerdem verfügt sie über die Methoden `getConnection()` und `getStream()`, die für das Herstellen einer Verbindung mit der bezeichneten Ressource genutzt werden können.

Die URL-Klasse verfügt unter anderem über folgende Methoden:

Methode	Beschreibung
<code>getHost()</code>	Gibt den repräsentierten Hostnamen oder die repräsentierte IP-Adresse zurück.
<code>getQuery()</code>	Gibt die enthaltene QueryString-Komponente oder null zurück.
<code>getPath()</code>	Gibt die enthaltene Pfadkomponente zurück.
<code>getPort()</code>	Gibt den repräsentierten Port zurück.
<code>getProtocol()</code>	Gibt das enthaltene Protokollkürzel zurück.
<code>getRef()</code>	Gibt die enthaltene Referenz (der Teil eines URL, der nach der Raute kommt und auf einen Teil des Dokuments verweist) zurück.
<code>getUserInfo()</code>	Gibt die enthaltene User-Information (in der Regel ist dies der für den Zugriff auf die Ressource zu verwendende Benutzername) oder null zurück.

Tabelle 61: Methoden der URL-Klasse

Um Informationen über eine Ressource zu erhalten, könnte das folgende Codefragment eingesetzt werden:

```
try {
    URL url = new URL("http://www.ksamaschke.de");

    // Protokoll ausgeben
    out.println(url.getProtocol());

    // Host ausgeben
    out.println(url.getHost());

    // Port
    out.println(url.getPort());

    // Pfad
```

Listing 394: Informationen über einen URL ausgeben

```
out.println(url.getPath());

// QueryString
out.println(url.getQuery());

// Referenz
out.println(url.getRef());

// Benutzer-Info
out.println(url.getUserInfo());

} catch (MalformedURLException e) {
    e.printStackTrace();
}
```

Listing 394: Informationen über einen URL ausgeben (Forts.)

168 Die IP-Adresse zu einem URL ermitteln

Die Ermittlung der IP-Adresse, auf die ein URL verweist, wird von der Klasse `java.net.InetAddress` übernommen. Deren statische Methode `getByName()` nimmt einen Hostnamen oder eine IP-Adresse entgegen und gibt eine `InetAddress`-Instanz zurück. Diese `InetAddress`-Instanz verfügt über drei Methoden, die mehr über den Host verraten:

Methode	Beschreibung
<code>getHostname()</code>	Gibt den Host-Namen zurück
<code>getHostAddress()</code>	Gibt die IP-Adresse zurück
<code>isReachable()</code>	Gibt an, ob eine Verbindung zum angegebenen Host in der als Parameter übergebenen Zeitspanne in Millisekunden hergestellt werden konnte.

Tabelle 62: Methoden einer `InetAddress`-Instanz

Um mehr Informationen über einen Host zu gewinnen, könnte etwa dieses Code-Fragment eingesetzt werden:

```
InetAddress address =
    InetAddress.getByName(url.getHost());

out.println(String.format("Hostname: %s",
    address.getHostName()));
out.println(String.format("Host-Address: %s",
    address.getHostAddress()));
out.println(String.format("Is reachable: %s",
    address.isReachable(2000)));
```

169 Inhalt von einem URI abrufen

Der Abruf von durch eine URL-Instanz bezeichneten Inhalten geschieht unter Verwendung einer `InputStreamReader`-Instanz. Diese nimmt im Konstruktor die `InputStream`-Instanz entgegen, welche die URL-Instanz durch ihre Methode `getStream()` zurückgibt. Die zurückgegebenen Zeilen können so lange durchlaufen und verarbeitet werden, bis die `InputStreamReader`-Instanz keine Inhalte mehr zurückliefert:

```
try {
    URL url = new URL("http://www.ksamaschke.de");
    BufferedWriter bw =
        new BufferedWriter(response.getWriter())

    BufferedReader rdr = new BufferedReader(
        new InputStreamReader(url.openStream()));

    String line = null;
    while((line = rdr.readLine()) != null) {
        bw.write(line + "\n");
    }

    bw.close();
    rdr.close();
} catch (MalformedURLException e) {
    e.printStackTrace();
} catch (IOException e) {
    e.printStackTrace();
}
```

170 Binären Inhalt von einem URI abrufen

Der Abruf von binären Inhalten erfolgt ähnlich dem Abruf von Textinhalten, nur wird hier keine `BufferedReader`-Instanz verwendet, um die Daten zu laden, sondern eine `BufferedInputStream`-Instanz kommt zum Einsatz. Die Daten werden durch eine `BufferedOutputStream`-Instanz, die mit Hilfe eines `FileOutputStreams` erzeugt worden ist, gespeichert.

Der eigentliche Vorgang des Abrufens geschieht analog zum Laden einer Datei: Es werden so lange Daten aus den `InputStream` in den Puffer geschrieben, bis keine Daten mehr zurückgegeben werden. Der Puffer wird dabei direkt in den Ausgabestrom entleert. Sobald der Ausgabestrom geschlossen worden ist, ist die abgerufene Datei lokal verfügbar und kann verwendet werden.

Um die abgerufenen Daten als Datei zu speichern, können Sie dieses Codefragment einsetzen:


```

try {
    // File-Instanz, welche die zu speichernde Datei repräsentiert
    File file = new File(
        "c:/temp/downloaded.pdf");

    // URL-Instanz, welche die zu ladende Ressource repräsentiert
    java.net.URL url = new java.net.URL(
        "http://downloads.aspectra.de/download.pdf");

    // OutputStream(s) zum Speichern des Downloads
    BufferedOutputStream bos =
        new BufferedOutputStream(
            new FileOutputStream(file));

    // InputStream(s) zum Laden des Downloads
    BufferedInputStream bin =
        new BufferedInputStream(url.openStream());

    byte[] buffer = new byte[16382];
    int bytes = 0;

    // Einlesen und Speichern der Datei
    while((bytes = bin.read(buffer)) > 0) {
        bos.write(buffer, 0, bytes);
    }

    // Aufräumen
    bos.close();
    bin.close();
} catch (MalformedURLException e) {
    e.printStackTrace();
} catch (IOException e) {
    e.printStackTrace();
}

```

Listing 395: Speichern von binärem Content aus einer externen Ressource

171 Daten an eine Ressource senden

Mit Hilfe der `URLConnection`-Klasse können an eine Ressource Daten gesendet werden, etwa um das Ausfüllen von Formularfeldern zu simulieren. Eine Instanz dieser Klasse erhält man durch die Methode `openConnection()` einer `URL`-Instanz. Per `setDoOutput(true)` kann der `URLConnection`-Instanz mitgeteilt werden, dass Parameter übertragen werden sollen.

Diese Parameter werden als Name-Wert-Paare via `PrintWriter` an den anderen Webserver gesendet. Einzelne Name-Wert-Paare werden durch Ampersands (&) getrennt.

Die Antwort kann mit Hilfe eines `InputStreamReader`s und einer `BufferedReader`-Instanz abgerufen und weiter verarbeitet werden:

```
try {
    // Repräsentation der Ressource, zu der connected werden soll
    java.net.URL url = new java.net.URL("...");

    // URLConnection instanziiieren
    URLConnection conn = url.openConnection();

    // Output zulassen
    conn.setDoOutput(true);

    // PrintWriter erzeugen, mit dem in den Output
    // geschrieben werden kann
    PrintWriter out = new PrintWriter(
        new OutputStreamWriter(conn.getOutputStream()));

    // Parameter übergeben
    out.print("first-name=karsten&last-name=samaschke");

    // OutputStream schliessen
    out.close();

    // BufferedReader zum Einlesen der Rückgabe erzeugen
    BufferedReader in = new BufferedReader(
        new InputStreamReader(conn.getInputStream()));

    // Rückgabe einlesen und ausgeben
    String line = null;
    while((line = in.readLine()) != null) {
        System.out.println(line);
    }

    // Aufräumen
    in.close();
} catch (MalformedURLException e) {
    e.printStackTrace();
} catch (IOException e) {
    e.printStackTrace();
}
```

Listing 396: Senden von Daten an eine Ressource

172 Auf per Basic-Authentication geschützte Ressourcen zugreifen

Basic-Authentication ist eine häufige Form der Authentifizierung, die vom Benutzer die Eingabe von Benutzernamen und Kennwort verlangt. Sämtliche Webserver beherrschen diese Art der Authentifizierung, die durch den Statuscode 401 ausgelöst wird. Der Webserver erwartet die Benutzername-Kennwort-Kombination als Base-64-codiertes Name-Wert-Paar im Header der Anforderung.

Soweit die Theorie. Glücklicherweise muss man sich bei Java nicht mit derartigen Details auseinander setzen. Um sich an derart geschützten Seiten per Code anzumelden, reicht es aus, eine eigene Authenticator-Ableitung zu schreiben, welche die Methode `getPasswordAuthentication()` überschreibt. Im Sinne einer besseren Wiederverwendbarkeit sollen Benutzername und Kennwort im Konstruktor der Klasse übergeben werden:

```
package codebook.j2ee.io;

import java.net.*;

public class GenericAuthenticator extends Authenticator {

    private String username;
    private String password;

    public GenericAuthenticator(
        String username, String password) {

        // Zuweisung von Benutzernamen und Kennwort an
        // die Instanz-Member
        this.username = username;
        this.password = password;
    }

    protected PasswordAuthentication
    getPasswordAuthentication() {

        PasswordAuthentication result =
            new PasswordAuthentication(
                username, password.toCharArray());

        return result;
    }
}
```

Listing 397: Implementierung einer Authenticator-Ableitung

Wenn Sie nun auf eine geschützte Ressource zugreifen wollen, können Sie der statischen Methode `setDefault()` der `java.net.Authenticator`-Klasse eine Instanz des

`GenericAuthenticators` übergeben, dessen Konstruktor die Angabe von Benutzername und Kennwort erwartet.

Die so übergebenen Credentials für den Zugriff auf geschützte Ressourcen werden nun so lange verwendet, bis andere Credentials zugewiesen werden. Sollte die Benutzername-Kennwort-Kombination nicht korrekt sein, wird eine Exception geworfen:

```
try {
    // Repräsentation der Ressource, zu der connected werden soll
    java.net.URL url = new java.net.URL("...");

    // URLConnection instanziiieren
    URLConnection conn = url.openConnection();

    // GenericAuthenticator-Instanz erzeugen
    GenericAuthenticator auth =
        new GenericAuthenticator("Test", "test");

    // Authenticator aktivieren
    java.net.Authenticator.setDefault(auth);

    // Output zulassen
    conn.setDoOutput(true);

    // PrintWriter erzeugen, mit dem in den Output
    // geschrieben werden kann
    PrintWriter out = new PrintWriter(
        new OutputStreamWriter(conn.getOutputStream()));

    // Parameter übergeben
    out.print("first-name=karsten&last-name=samaschke");

    // OutputStream schliessen
    out.close();

    // BufferedReader zum Einlesen der Rückgabe erzeugen
    BufferedReader in = new BufferedReader(
        new InputStreamReader(conn.getInputStream()));

    // Rückgabe einlesen und ausgeben
    String line = null;
    while((line = in.readLine()) != null) {
        System.out.println(line);
    }

    // Aufräumen
    in.close();
}
```

Listing 398: Verwendung der Klasse `GenericAuthenticator`

```

} catch (MalformedURLException e) {
    e.printStackTrace();
} catch (IOException e) {
    e.printStackTrace();
}

```

Listing 398: Verwendung der Klasse GenericAuthenticator (Forts.)

173 Eine E-Mail senden

E-Mails werden aus Java heraus in der Regel über das von Sun bereitgestellte API `JavaMail` versendet. Dieses API, das den Namensraum `javax.mail` verwendet und deren aktuelle Version derzeit 1.3.2 ist, können Sie unter der Adresse <http://java.sun.com/products/javamail/downloads/index.html> herunterladen. Zusätzlich benötigen Sie noch `Suns Java Beans Activation Framework (JAF)`, das Sie unter der Adresse <http://java.sun.com/products/javabeans/glasgow/jaf.html> finden können.

Das Versenden einer E-Mail per `JavaMail` geschieht immer nach dem gleichen Muster:

1. Erzeugen einer `Mail-Session`-Instanz, die Informationen zur Art des Versandes per `Properties`-Instanz zugewiesen bekommt
2. Erzeugen einer `Mail`-Instanz über die `Mail-Session`
3. Festlegen der Eigenschaften zu Absender, Empfänger, Betreff und Nachricht
4. Versenden der E-Mail über eine `Transport`-Instanz

In Code umgesetzt, kann der Versand einer E-Mail so aussehen:

```

try {
    // Server-Namen in Properties erfassen
    Properties settings = new Properties();
    settings.put("java.mail.host", "...");

    // Mail-Session erzeugen
    Session session = Session.getDefaultInstance(settings);

    // Nachricht erzeugen
    Message message = new MimeMessage(session);

    // Absender
    message.setFrom(new InternetAddress("info@ksamaschke.de"));

    // Empfänger
    message.setRecipient(Message.RecipientType.TO,

```

Listing 399: Versand einer E-Mail per JavaMail

```
new InternetAddress("info@aspextra.de"));

// Betreff
message.setSubject("Test-Email per JavaMail");

// Nachrichtentext
message.setText("Hallo,\n\n\n"
    + "Diese Nachricht ist per JavaMail generiert worden");

// Versenden
Transport.send(message);
} catch (MessagingException e) {
    e.printStackTrace();
}
```

Listing 399: Versand einer E-Mail per JavaMail (Forts.)

Die hier erzeugte E-Mail wird über den mit Hilfe des Schlüssels `java.mail.host` angegebenen Server versendet. Der Absender wird mit Hilfe der Methode `setFrom()` festgelegt. Seine E-Mail-Adresse wird durch eine Instanz der Klasse `javax.mail.InternetAddress` repräsentiert.

Mit Hilfe der Methode `setRecipient()` kann der Empfänger festgelegt werden. Deren erster Parameter nimmt einen Integer-Wert entgegen, der die Art des Empfängers (TO, CC oder BCC) kennzeichnet. Der zweite Parameter ist eine `InternetAddress`-Instanz, welche die E-Mail-Adresse des Empfängers aufnimmt.

Nach dem Setzen von Betreff per `setSubject()` und des Mail-Textes per `setText()` kann die E-Mail versendet werden. Dies geschieht über die statische Methode `send()` der `Transport`-Klasse.

Die generierte E-Mail sieht dann so aus:

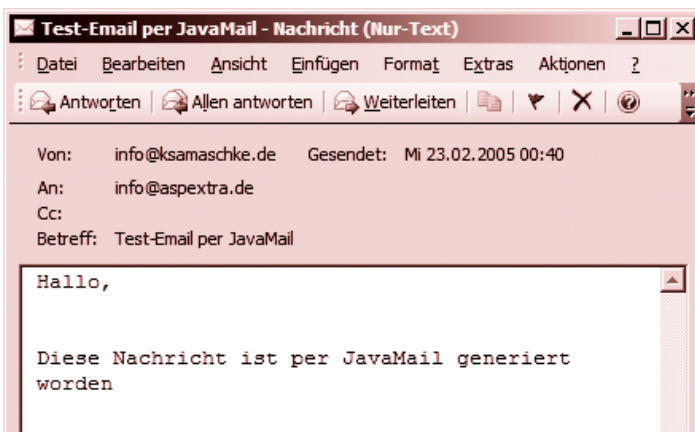


Abbildung 115: Per JavaMail generierte E-Mail

174 Eine E-Mail über einen Server senden, der Authentifizierung erfordert

Der Versand über einen E-Mail-Server, der eine Authentifizierung des Absenders per Benutzername-Kennwort-Kombination erfordert, setzt voraus, dass eine Ableitung der Klasse `javax.mail.Authenticator` erzeugt wird.

Diese ähnelt weitestgehend der weiter oben gezeigten `java.net.Authenticator`-Instanz. Der einzige Unterschied ist der Konstruktor der Klasse `javax.mail.PasswordAuthentication`: Dieser erlaubt im Gegensatz zum Konstruktor von `java.net.PasswordAuthentication` die Übergabe eines Strings für das Passwort:

```
package codebook.j2ee.io;

import javax.mail.Authenticator;
import javax.mail.PasswordAuthentication;

public class MailCredentials extends Authenticator {

    private String username;
    private String password;

    public MailCredentials(String username, String password) {

        // Zuweisung von Benutzernamen und Kennwort an
        // die Instanz-Member
        this.username = username;
        this.password = password;
    }

    // Rückgabe der Credentials
    protected PasswordAuthentication getPasswordAuthentication() {

        PasswordAuthentication result =
            new PasswordAuthentication(username, password);

        return result;
    }
}
```

Listing 400: Die Klasse `MailCredentials` gibt die Informationen zurück, die zur Authentifizierung am Server benötigt werden

Beim Versand einer E-Mail über den gesicherten Server kann beim Erzeugen der Referenz auf die Session eine Instanz der `MailCredentials`-Klasse als Parameter übergeben werden:

```
try {
    // Server-Namen in Properties erfassen
    Properties settings = new Properties();
    settings.put("java.mail.host", "...");

    // Authenticator-Ableitung instanziiieren, welche die Zugriffsinformationen
    // auf den Mail-Server hält
    MailCredentials auth = new MailCredentials("test", "test");

    // Mail-Session erzeugen
    Session session = Session.getDefaultInstance(settings, auth);

    // Nachricht erzeugen
    Message message = new MimeMessage(session);

    // Absender
    message.setFrom(new InternetAddress("info@ksamaschke.de"));

    // Empfänger
    message.setRecipient(
        Message.RecipientType.TO,
        new InternetAddress("info@aspextra.de"));

    // Betreff
    message.setSubject("Test-Email per JavaMail");

    // Nachrichtentext
    message.setText("Hallo,\n\n\n"
        + "Diese Nachricht ist per JavaMail generiert worden");

    // Versenden
    Transport.send(message);
} catch (MessagingException e) {
    e.printStackTrace();
}
```

Listing 401: Versand von E-Mails über einen gesicherten Mailserver

175 Eine HTML-E-Mail senden

HTML-E-Mails können relativ einfach per JavaMail versendet werden. Das Prinzip dahinter ist, dass eine E-Mail mit dem Inhaltstyp `multipart/alternative` erzeugt werden muss. Dieser Inhaltstyp gibt an, dass die E-Mail sowohl Text- als auch HTML-Code enthält und das E-Mail-Programm selbst entscheiden kann, welche Repräsentation es darstellt.

Der Multipart-Inhalt einer E-Mail wird mit Hilfe der Klasse `javax.mail.Multipart` repräsentiert. Der erste Teil einer derartigen Multipart-E-Mail ist dabei die reine Textform, die auch von älteren E-Mail-Programmen angezeigt werden kann und in der Regel als Fall-Back-Möglichkeit dient, falls die HTML-Anzeige nicht erwünscht oder möglich ist. Der zweite Teil ist der HTML-Code.

Die einzelnen Teile werden durch `javax.mail.BodyPart`-Instanzen repräsentiert. Deren Methode `setContent()` nimmt den Inhalt und den Inhaltstyp als Parameter entgegen. Nach dem Hinzufügen der einzelnen `BodyParts` an den `Multipart` kann dieser an die Nachricht angehängt und die Nachricht versendet werden:

```
try {
    // Server-Namen in Properties erfassen
    Properties settings = new Properties();
    settings.put("java.mail.host", "...");

    // Mail-Session erzeugen
    Session session = Session.getDefaultInstance(settings);

    // Nachricht erzeugen
    Message message = new MimeMessage(session);

    // Absender
    message.setFrom(new InternetAddress("info@ksamaschke.de"));

    // Empfänger
    message.setRecipient(
        Message.RecipientType.TO,
        new InternetAddress("info@aspextra.de"));

    // Betreff
    message.setSubject("HTML-Email per JavaMail");

    // Multipart-Instanz vom Typ alternative erzeugen
    Multipart mp = new MimeMultipart("alternative");

    // BodyParts erzeugen
    // 1. Textteil
    BodyPart part = new MimeBodyPart();
    part.setContent(
        "Hallo,\n\nDiese Nachricht ist "
        + "per JavaMail generiert worden",
        "text/plain");
    mp.addBodyPart(part);

    // 2. HTML-Teil erzeugen
```

Listing 402: Erzeugen einer HTML-E-Mail

```

part = new MimeBodyPart();
part.setContent(
    "<html><body><h3>Hallo,</h3>"
    + "<p>Diese Email ist per JavaMail "
    + "generiert worden.</p>"
    + "</body></html>",
    "text/html");
mp.addBodyPart(part);

// Multipart-Instanz zuweisen
message.setContent(mp);

// Versenden
Transport.send(message);
} catch (MessagingException e) {
    e.printStackTrace();
}

```

Listing 402: Erzeugen einer HTML-E-Mail (Forts.)

Der obige Code erzeugt diese HTML-E-Mail:

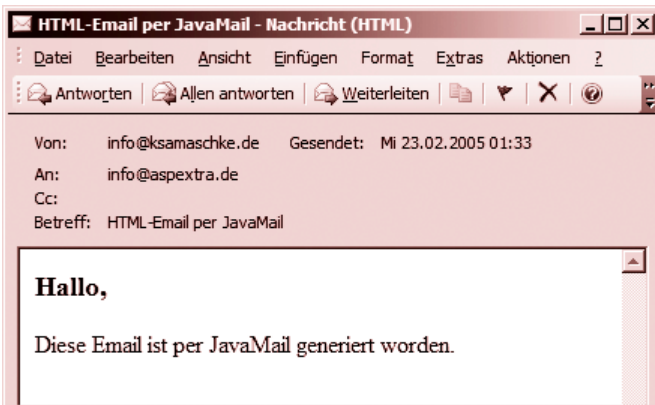


Abbildung 116: HTML-E-Mail, die per JavaMail generiert worden ist

176 Eine E-Mail mit Anhang versenden

Der Versand einer E-Mail mit Anhang erfolgt fast genau so wie der Versand einer HTML-E-Mail, jedoch mit dem Unterschied, dass der E-Mail-Typ nicht multipart/alternative ist. Einzelne Dateien können über Instanzen der Klasse `javax.activation.FileDataSource` abgebildet und den einzelnen `BodyParts`, welche die Dateianhänge repräsentieren, zugewiesen werden. Der erste Teil einer derartigen Nachricht ist der anzuzeigende Text, der übrigens auch gerne HTML sein kann:

```

try {
    // Server-Namen in Properties erfassen
    Properties settings = new Properties();
    settings.put("java.mail.host", "...");

    // Mail-Session erzeugen
    Session session = Session.getDefaultInstance(settings);

    // Nachricht erzeugen
    Message message = new MimeMessage(session);

    // Absender
    message.setFrom(new InternetAddress("info@ksamaschke.de"));

    // Empfänger
    message.setRecipient(
        Message.RecipientType.TO,
        new InternetAddress("info@aspextra.de"));

    // Betreff
    message.setSubject("Attachment per JavaMail");

    // Multipart-Instanz erzeugen
    Multipart mp = new MimeMultipart();

    // BodyParts erzeugen
    // 1. HTML-Teil
    BodyPart part = new MimeBodyPart();
    part.setContent(
        "<html><body><h3>Hallo,</h3>"
        + "<p>Anbei finden Sie einen Datei-Anhang!</p>"
        + "</body>",
        "text/html");
    mp.addBodyPart(part);

    // 2. Datei anhängen
    part = new MimeBodyPart();
    FileDataSource fds = new FileDataSource(
        new File("c:/Temp/download.pdf"));
    part.setDataHandler(new DataHandler(fds));
    part.setFileName(fds.getName());
    mp.addBodyPart(part);

    // Multipart-Instanz zuweisen
    message.setContent(mp);
}

```

Listing 403: Erzeugen einer E-Mail mit Dateianhang

```
// Versenden
Transport.send(message);
} catch (MessagingException e) {
    e.printStackTrace();
}
```

Listing 403: Erzeugen einer E-Mail mit Dateianhang (Forts.)

So sieht die generierte E-Mail aus:

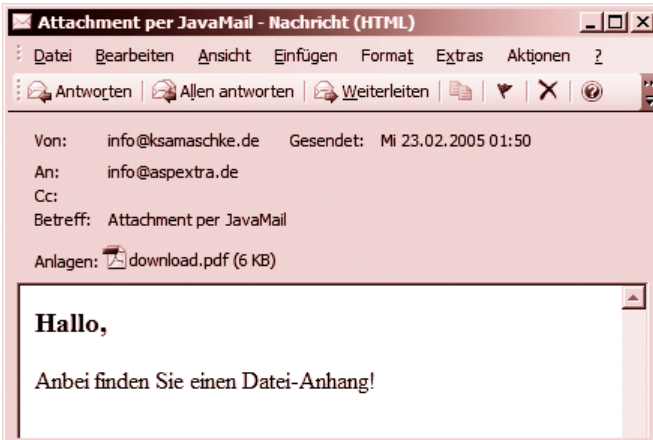


Abbildung 117: Die generierte E-Mail hat einen Dateianhang